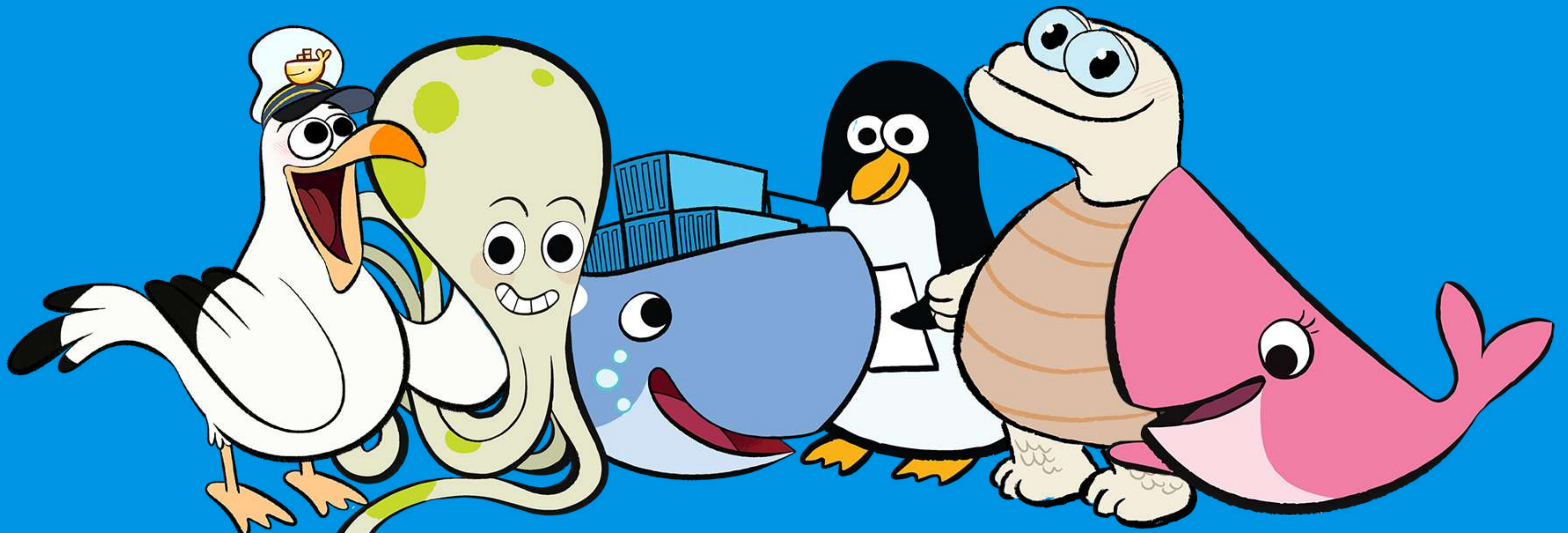


How Docker is Changing IT

5 YEARS CREATING AN INDUSTRY OF INNOVATION



**This is it. Containers are the next
once-in-a-decade shift in infrastructure
and process that make or break you.**

SAID BY ME. JUST NOW.

Let's Recap

MAJOR INFRASTRUCTURE SHIFTS



Mainframe to PC

90'S

Baremetal to Virtual

00'S

Datacenter to Cloud

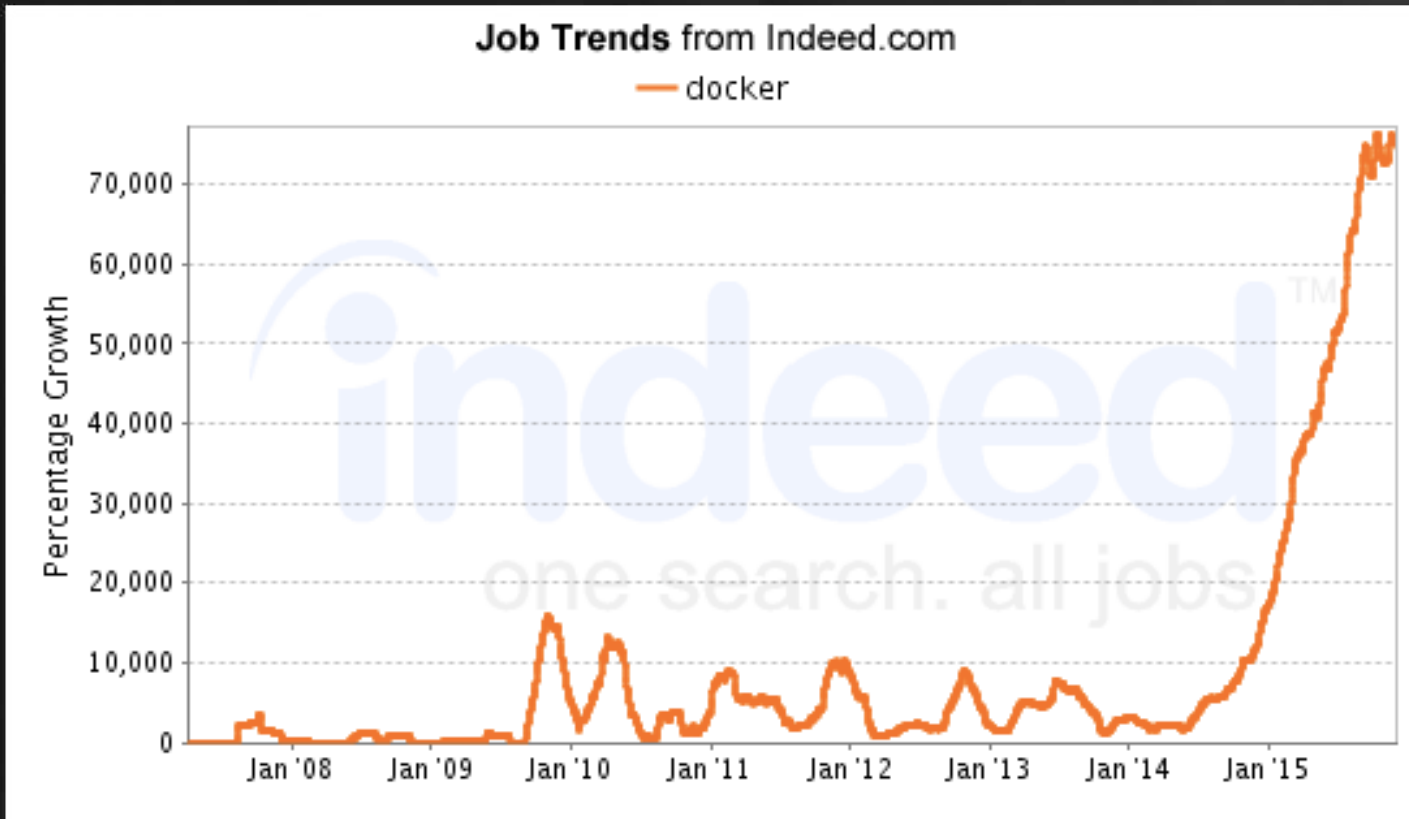
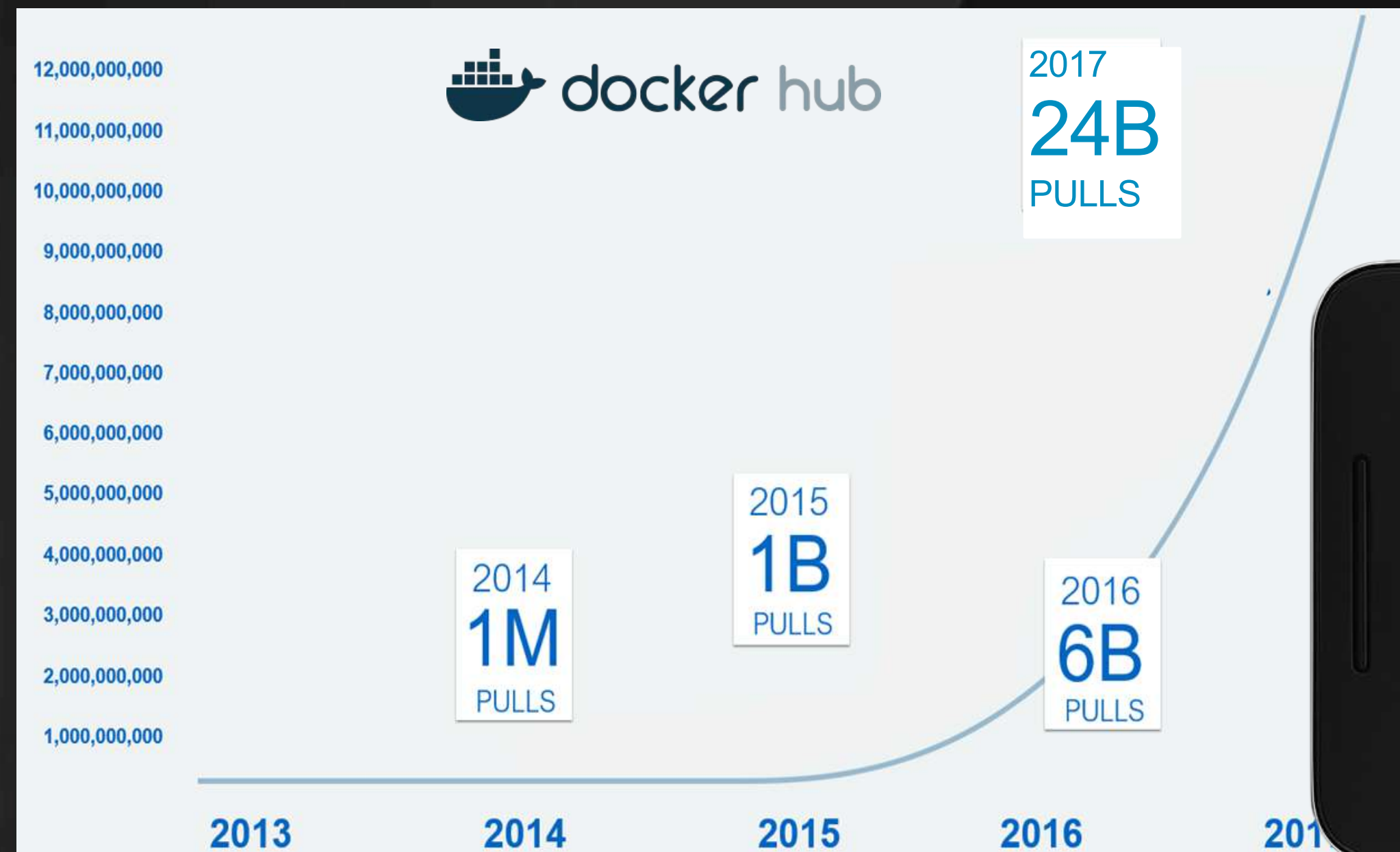
10'S

**Host to Container
(Serverless)**

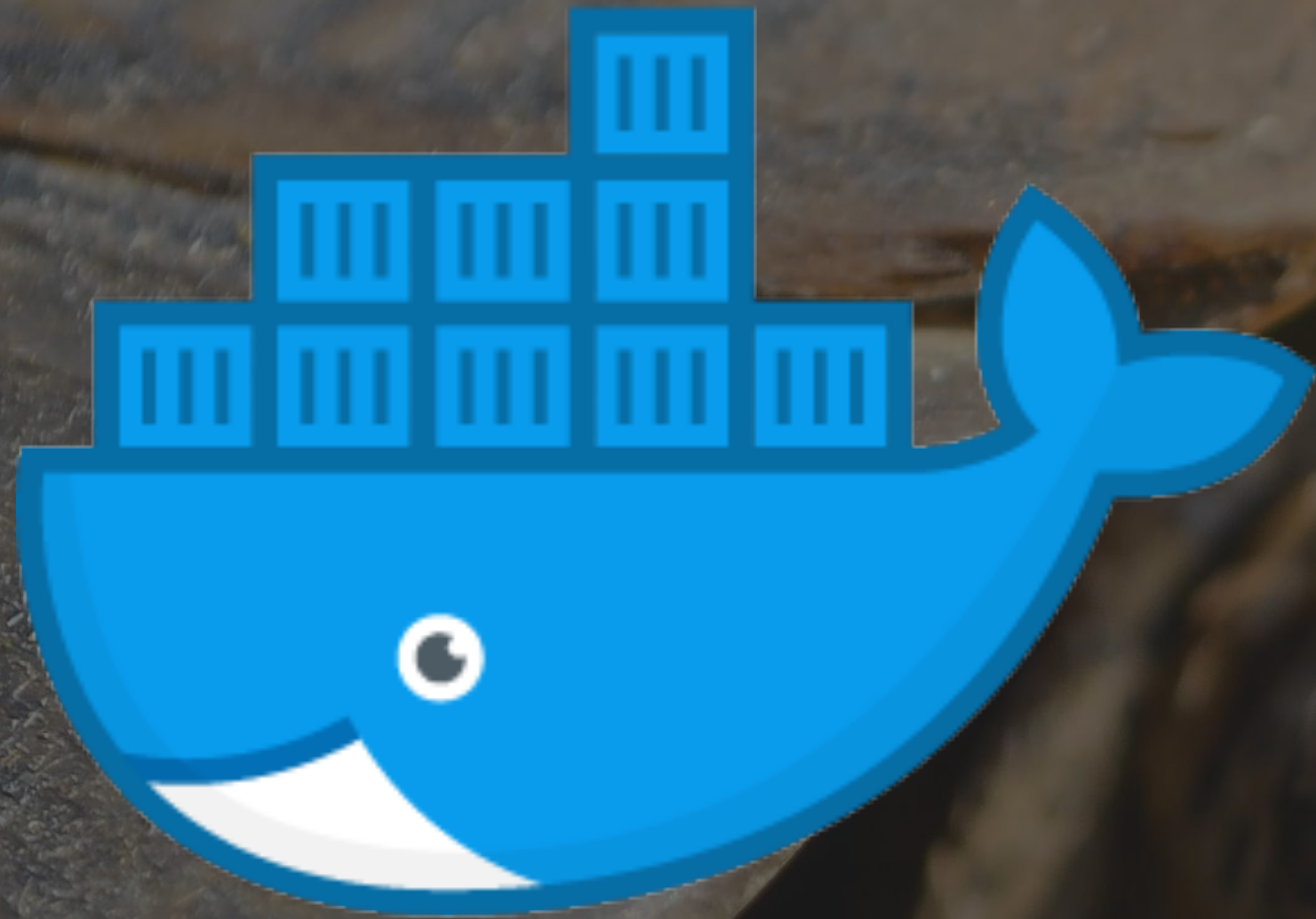
Virtuali keeps shifts, shifts, and docks together
for a seamless migration experience.

Containers are the “Fastest Growing Cloud Enabling Technology”

“By 2020, more than 50% of global organizations will be running containers in production.”
-Gartner



@docker



@gordonTheTurtle



What will Docker do for me?

Docker is all about speed.

Develop Faster.

Build Faster.



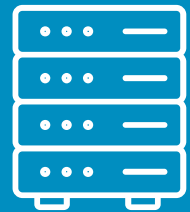

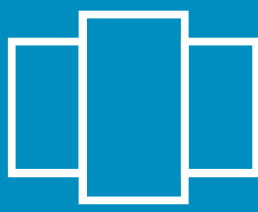



Test Faster.

Deploy Faster.

Update Faster.

Recover Faster.

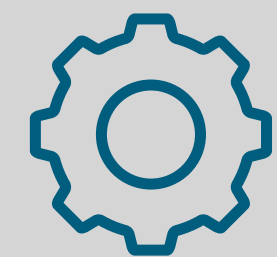
The “Matrix from Hell” Breeds Complexity

Static Website	?	?	?	?	?	?	?	?
Web Frontend	?	?	?	?	?	?	?	?
Background Workers	?	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?	?
	Desktop 	Test/QA Cluster 	Production Cluster 	Public Cloud 	Data Center 	Mainframe 	Windows Server 	Edge Device 

Containers Reduce Complexity



Maintenance and Complexity Drains Budgets, So Innovation Suffers



80%
MAINTENANCE



20%
INNOVATION



IT demands
increasing!



<20% server
utilization***



8+
IT Silos*



1000's
of apps

Stalled Initiatives

- Modernizing legacy apps
- Cloud migration
- Server consolidation
- Faster s/w time to market

60% of CIOs say
***“We’re behind in digital
transformation”*****




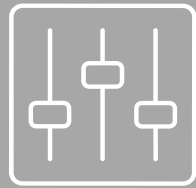
























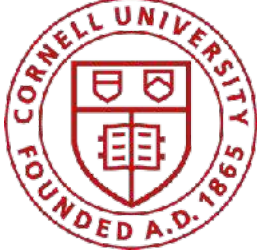

* Average number infrastructures in the enterprise (on-prem and cloud)

** Forrester, CIO and the emerging Digital Crisis, 2018

*** CloudPhysics, Global IT Data Lake Report, Q4, '16

Sources: IDC “The Cost of Retaining Aging IT Infrastructure”, RightScale 2017 State of the Cloud Report

500+ Build Their Containerization Strategy with Docker Enterprise Edition

 Oil & Gas / Energy	 Healthcare & Science	 Financial Services	 Tech	 Insurance	 Public Sector
   	   	   	   	   	    INDIANA UNIVERSITY



18 Month project
Migrated 700+ apps
Now 150,000 containers
50% dev productivity boost



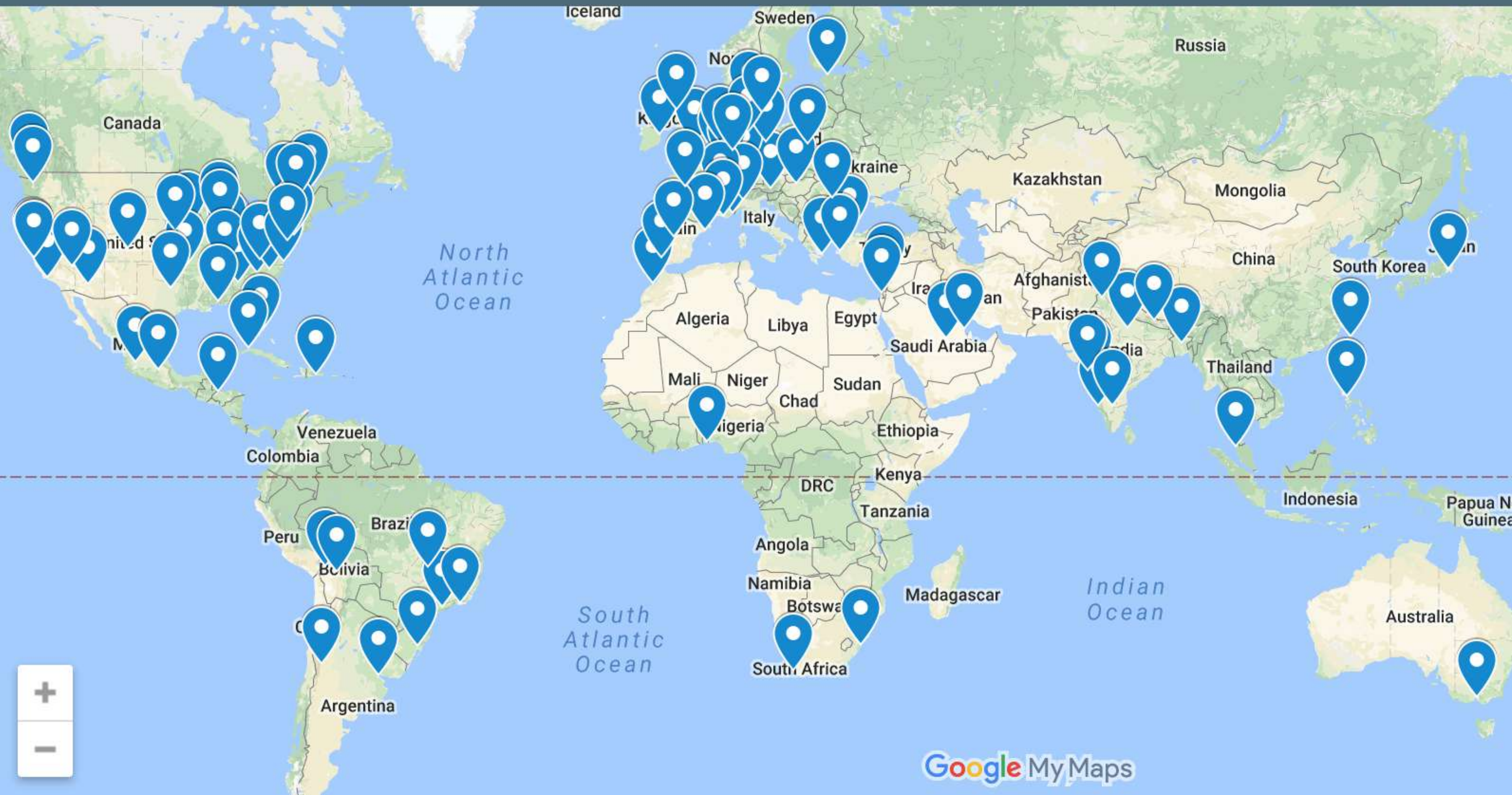
70% reduction in VM costs
67% fewer CPU's
10x average CPU utilization
66% cost reduction



docker



Docker Birthday #5





Cloud Native Landscape

v2.1

See the interactive landscape at landscape.cncf.io

Greyed logos are not open source

App Definition & Development

Database & Data Warehouse

Streaming

Source Code Management

Application Definition

Continuous Integration / Continuous Delivery (CI/CD)

Orchestration & Management

Scheduling & Orchestration

Coordination & Service Discovery

Service Management

Runtime

Cloud-Native Storage

Container Runtime

Cloud-Native Network

Provisioning

Host Management / Tooling

Infrastructure Automation

Container Registries

Secure Images

Key Management

Cloud

Public

Private

github.com/cncf/landscape

This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.

Platforms

Observability & Analysis

Serverless

Kubernetes Certified Service Provider

Special

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

Docker Editions at store.docker.com

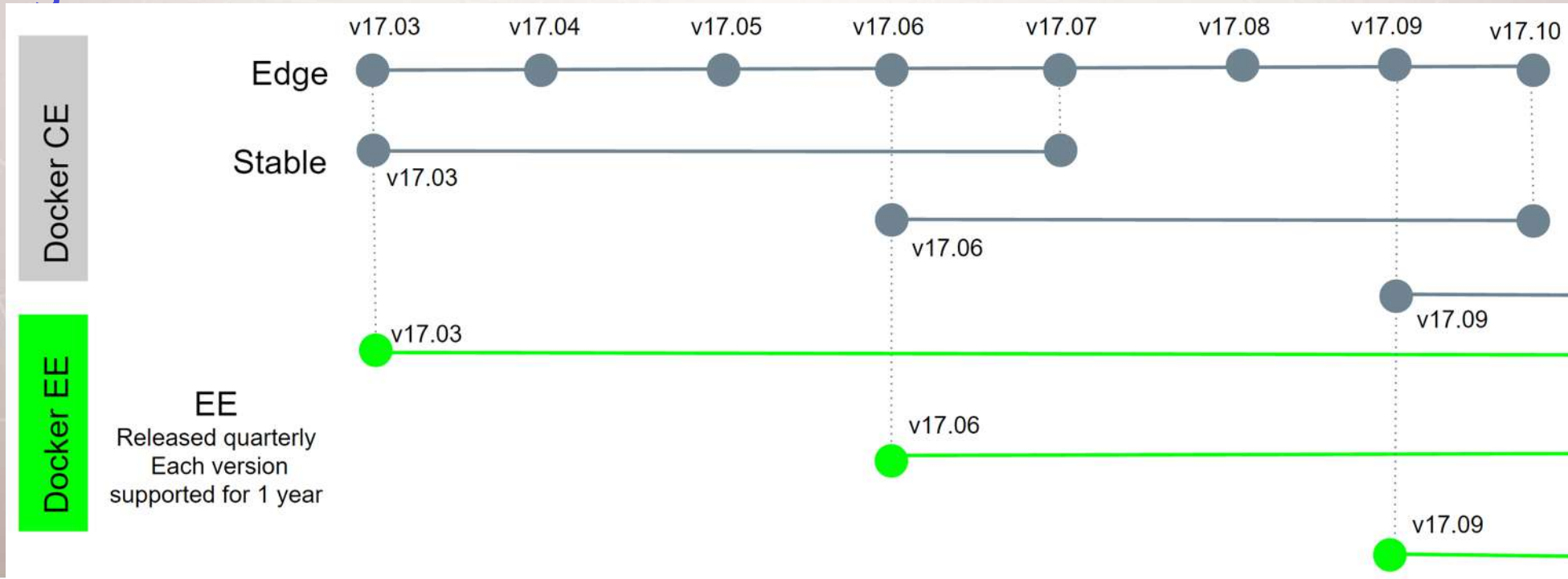
- Docker is no longer just a "container runtime"
- Docker moves fast, it matters how you install it
- Linux (different per distro, don't use default package)
- Docker Desktop (Recommended for Windows and Mac)
- Docker Toolbox (Legacy for Win10 Home/Win7/Win8)

Community vs. Enterprise

- Docker Community (free) vs. Docker Enterprise (paid)
- Old Name "EE" = Enterprise Edition (same thing)
- Windows Server 2016 comes with Enterprise Basic
- Linux servers are a target for Enterprise
- Enterprise also gets you GUI apps like DTR and UCP
- Developers and Sysadmins use Community locally

Stable vs. Edge

- 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



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

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

Docker on Windows 10 Pro/Ent

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

Docker for Mac

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

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
- Must put all your code in C:\Users\<your-username>

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

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)

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: cmdr
 - <http://cmdr.net/>
- Great info and troubleshooting/FAQ
 - <https://docs.docker.com/docker-for-windows/>

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)

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: cmdr
 - <http://cmdr.net/>

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

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)

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>

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.)

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)

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

What's In This Section

- Check versions of our docker cli and engine
- Create a Nginx (web server) container
- Learn common container management commands
- Learn Docker networking basics
- Requirements: Have latest Docker installed from last Section

What We Covered

- command: `docker version`
 - verified cli can talk to engine
- command: `docker info`
 - most config values of engine
- docker command line structure
 - old (still works): `docker <command> (options)`
 - new: `docker <command> <sub-command> (options)`

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`

This Lecture

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

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)

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

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

Example Of Changing The Defaults

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

change version of image



change host listening port



change CMD run on start



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 `--detach` (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 `--env` 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

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

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

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

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

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 than 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...

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`

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

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

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

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 😬

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 run --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`

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

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 `--network-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

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

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

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

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"

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)`

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)

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

Docker Compose

- Why: configure relationships between containers
- Why: save our docker container run settings in easy-to-read file
- Why: create one-liner developer environment startups
- Comprised of 2 separate but related things
- 1. YAML-formatted file that describes our solution options for:
 - containers
 - networks
 - volumes
- 2. A CLI tool `docker-compose` used for local dev/test automation with those YAML files

docker-compose.yml

- Compose YAML format has its own versions: 1, 2, 2.1, 3, 3.1
- YAML file can be used with `docker-compose` command for local docker automation or..
- With `docker` directly in production with Swarm (as of v1.13)
- `docker-compose --help`
- `docker-compose.yml` is default filename, but any can be used with `docker-compose -f`

docker-compose CLI

- CLI tool comes with Docker for Windows/Mac, but separate download for Linux
- Not a production-grade tool but ideal for local development and test
- Two most common commands are
 - `docker-compose up` # setup volumes/networks and start all containers
 - `docker-compose down` # stop all containers and remove cont/vol/net
- If all your projects had a `Dockerfile` and `docker-compose.yml` then "new developer onboarding" would be:
 - `git clone github.com/some/software`
 - `docker-compose up`

Assignment: Writing A Compose File

- Build a basic compose file for a Drupal content management system website. Docker Hub is your friend
- Use the `drupal` image along with the `postgres` image
- Use `ports` to expose Drupal on 8080 so you can localhost:8080
- Be sure to set `POSTGRES_PASSWORD` for postgres
- Walk through Drupal setup via browser
- Tip: Drupal assumes DB is `localhost`, but it's service name
- Extra Credit: Use volumes to store Drupal unique data

Using Compose to Build

- Compose can also build your custom images
- Will build them with `docker-compose up` if not found in cache
- Also rebuild with `docker-compose build`
 - or all in one: `docker-compose up --build`
- Great for complex builds that have lots of vars or build args

Assignment: Build and Run Compose

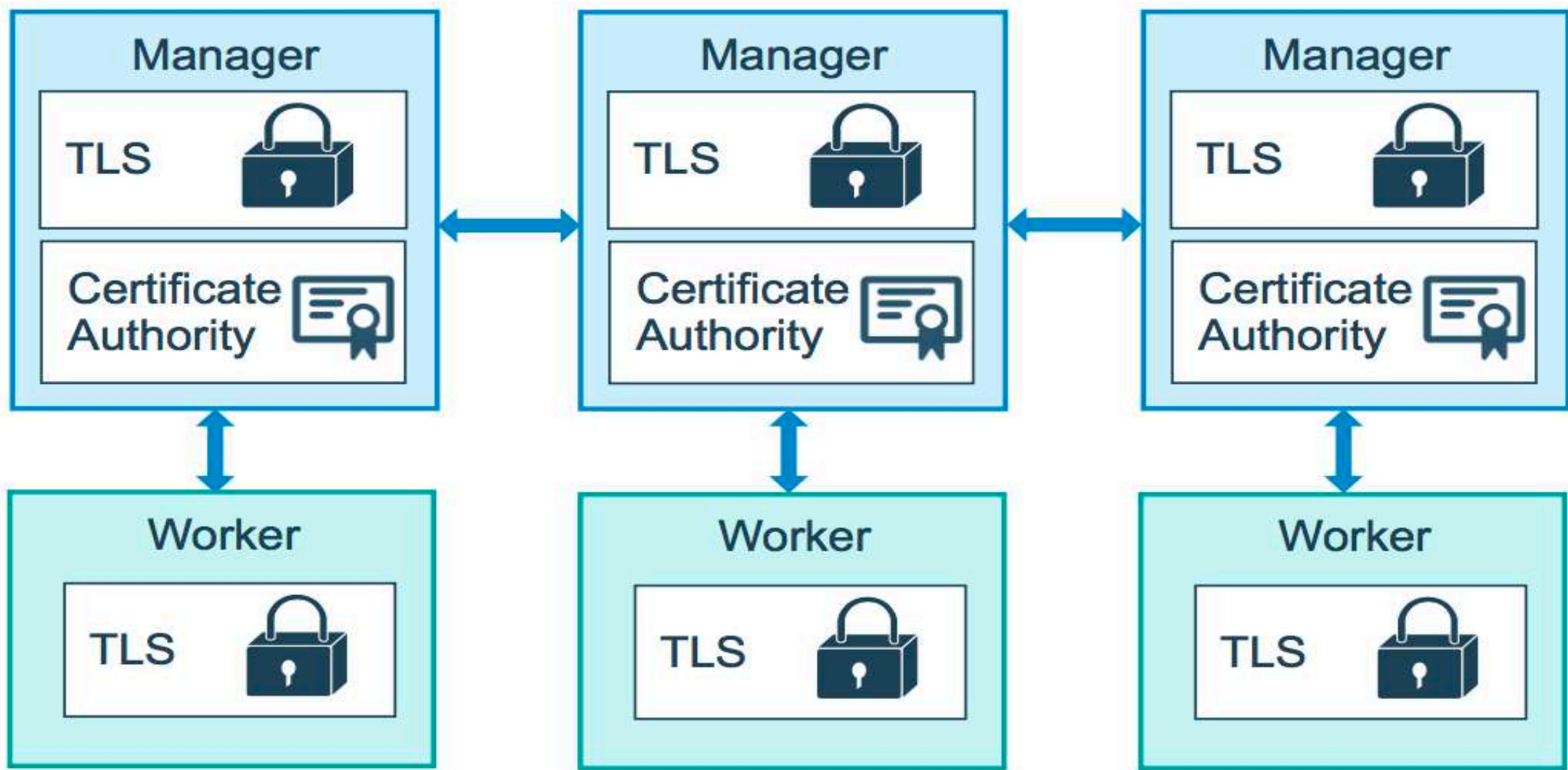
- "Building custom `drupal` image for local testing"
- Compose isn't just for developers. Testing apps is easy/fun!
- Maybe your learning Drupal admin, or are a software tester
- Start with Compose file from previous assignment
- Make your `Dockerfile` and `docker-compose.yml` in dir `compose-assignment-2`
- Use the `drupal` image along with the `postgres` image as before
- Use `README.md` in that dir for details

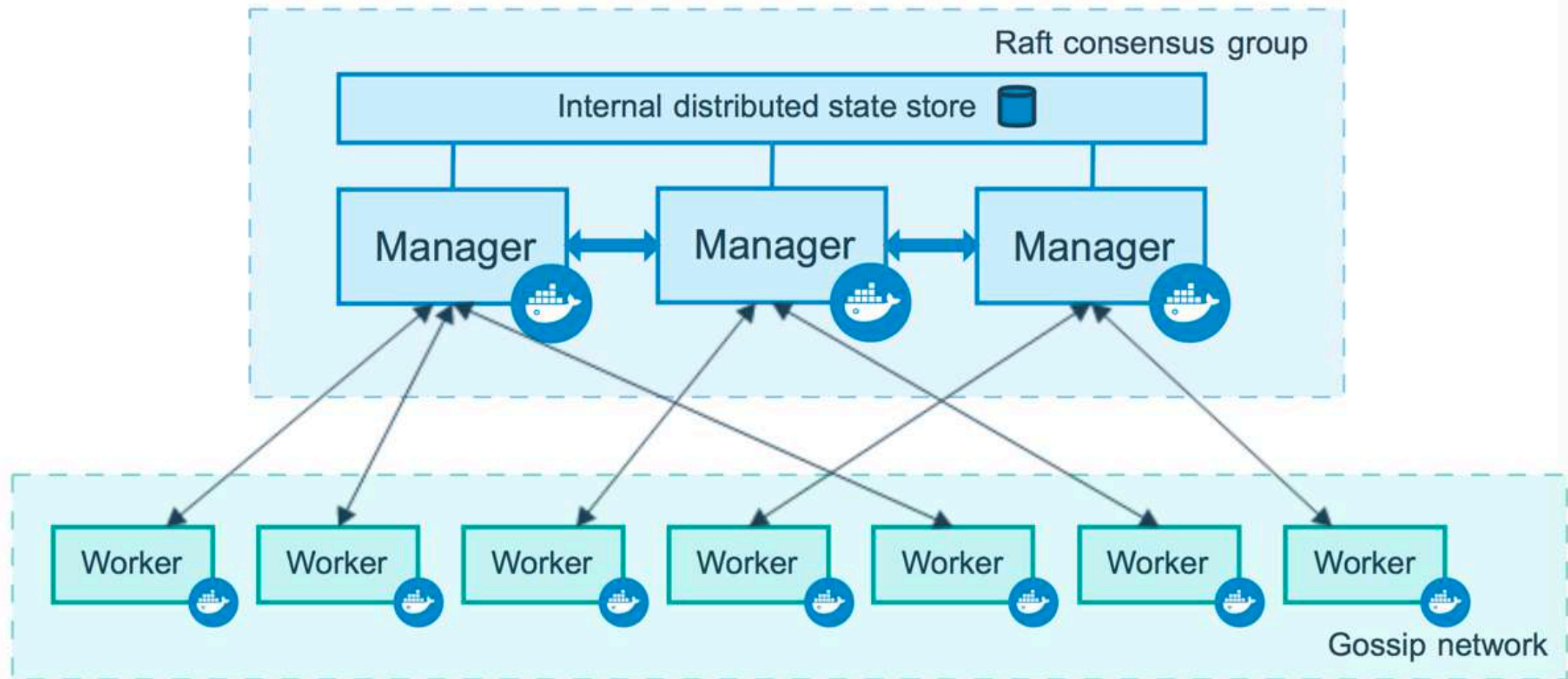
Containers Everywhere = New Problems

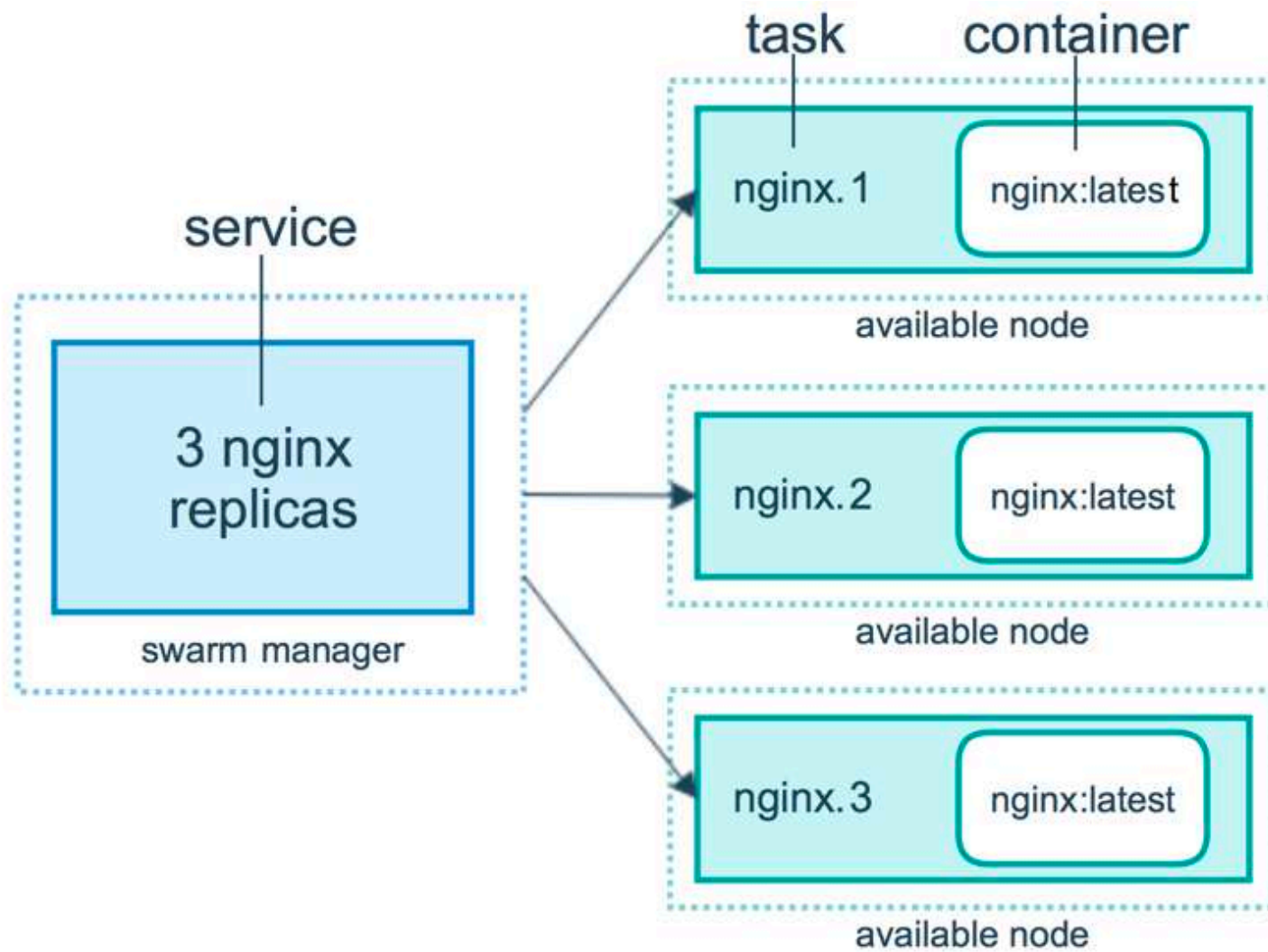
- How do we automate container lifecycle?
- How can we easily scale out/in/up/down?
- How can we ensure our containers are re-created if they fail?
- How can we replace containers without downtime (blue/green deploy)?
- How can we control/track where containers get started?
- How can we create cross-node virtual networks?
- How can we ensure only trusted servers run our containers?
- How can we store secrets, keys, passwords and get them to the right container (and only that container)?

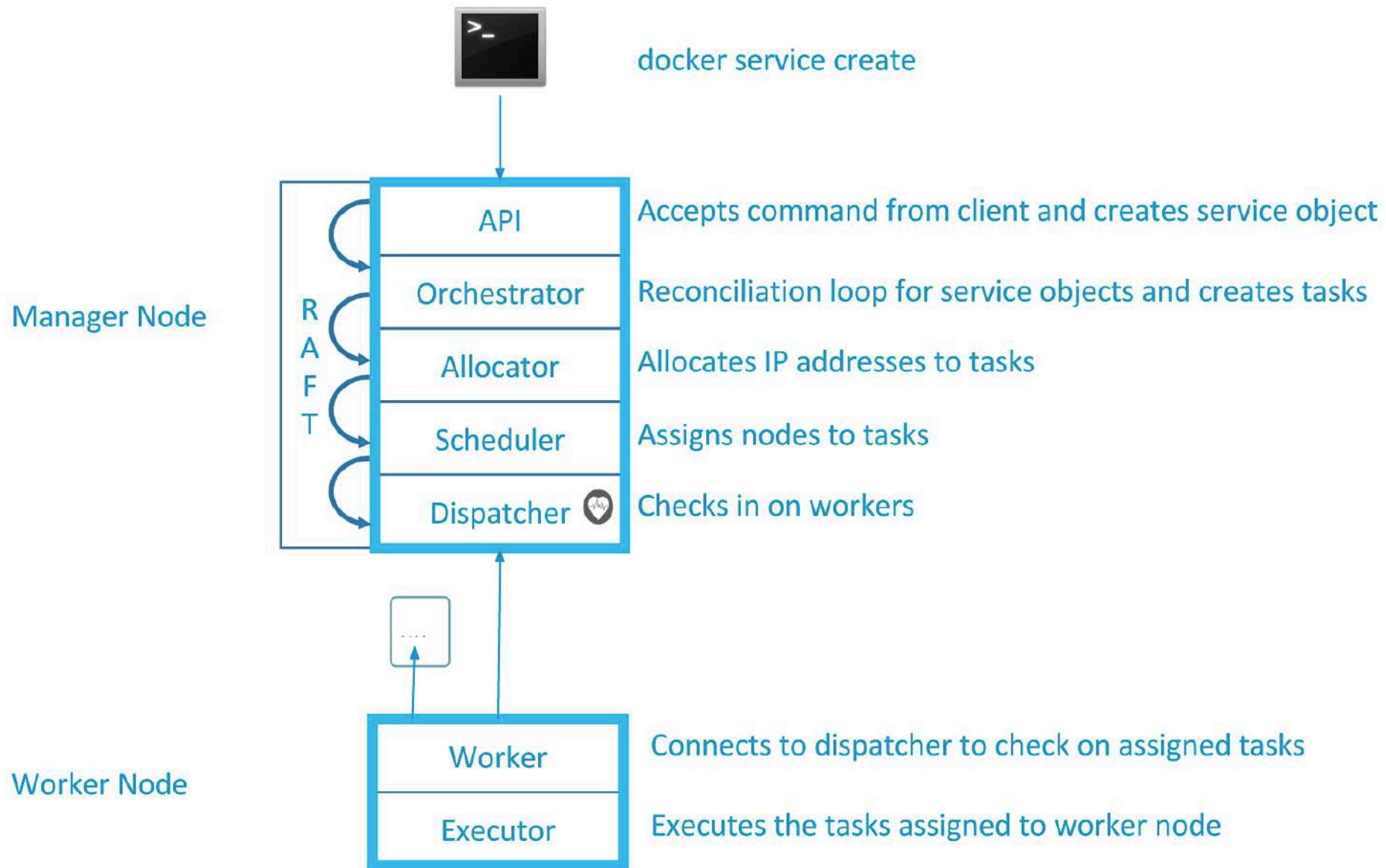
Swarm Mode: Built-In Orchestration

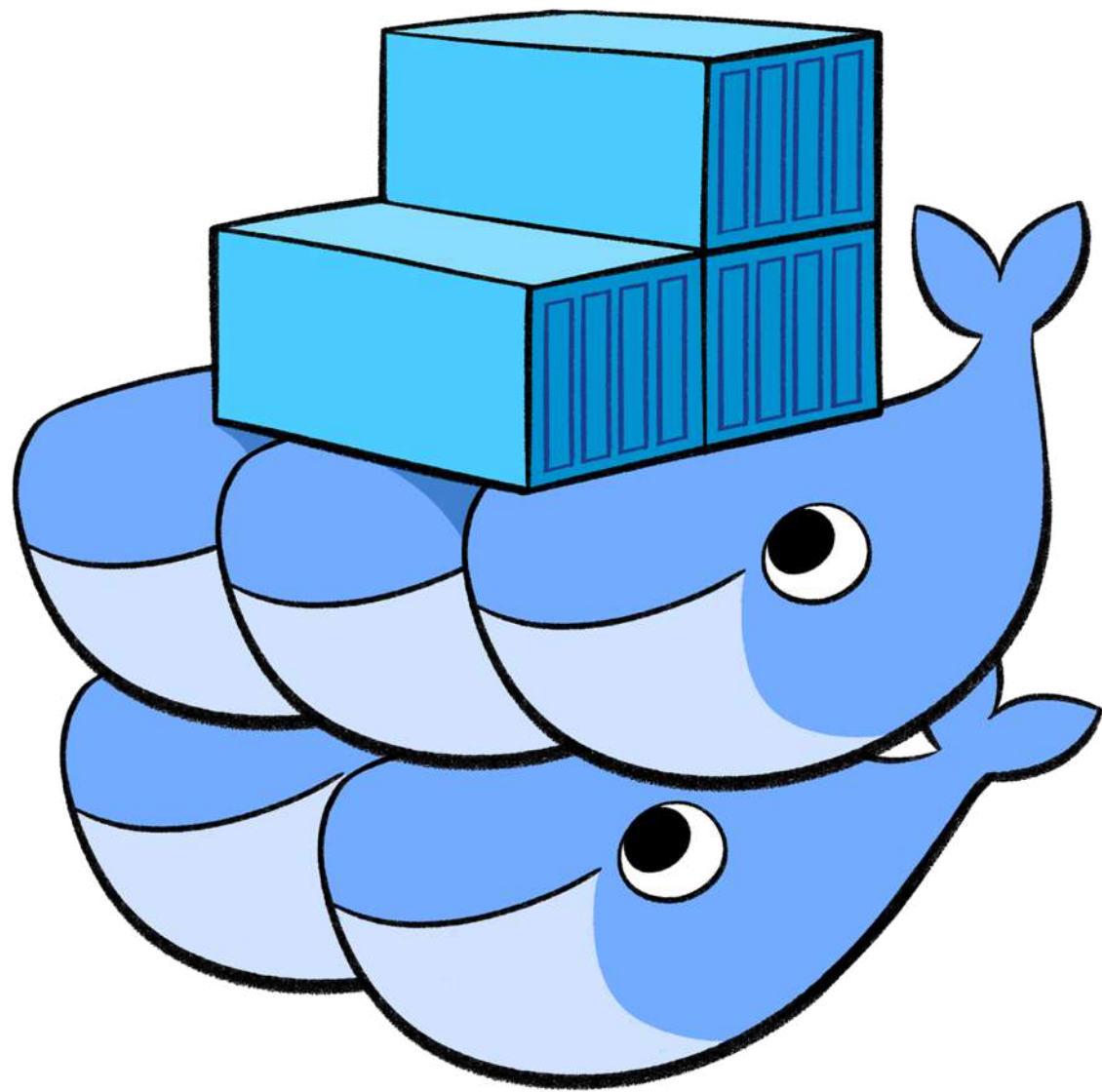
- Swarm Mode is a clustering solution built inside Docker
- Not related to Swarm "classic" for pre-1.12 versions
- Added in 1.12 (Summer 2016) via SwarmKit toolkit
- Enhanced in 1.13 (January 2017) via Stacks and Secrets
- Not enabled by default, new commands once enabled
 - docker swarm
 - docker node
 - docker service
 - docker stack
 - docker secret











docker swarm init: What Just Happened?

- Lots of PKI and security automation
 - Root Signing Certificate created for our Swarm
 - Certificate is issued for first Manager node
 - Join tokens are created
- Raft database created to store root CA, configs and secrets
 - Encrypted by default on disk (1.13+)
 - No need for another key/value system to hold orchestration/secrets
 - Replicates logs amongst Managers via mutual TLS in "control plane"

Creating 3-Node Swarm: Host Options

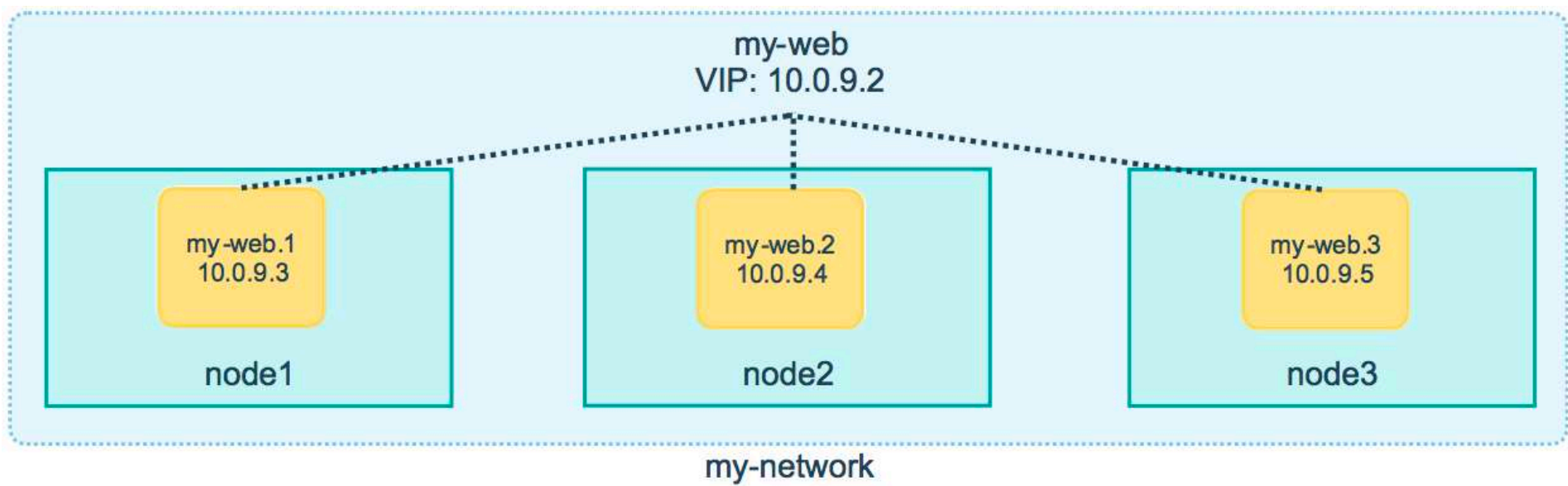
- A. play-with-docker.com
 - Only needs a browser, but resets after 4 hours
- B. docker-machine + VirtualBox
 - Free and runs locally, but requires a machine with 8GB memory
- C. Digital Ocean + Docker install
 - Most like a production setup, but costs \$5-10/node/month while learning
 - Use my referral code in section resources to get \$10 free
- D. Roll your own
 - docker-machine can provision machines for Amazon, Azure, DO, Google, etc.
 - Install docker anywhere with get.docker.com

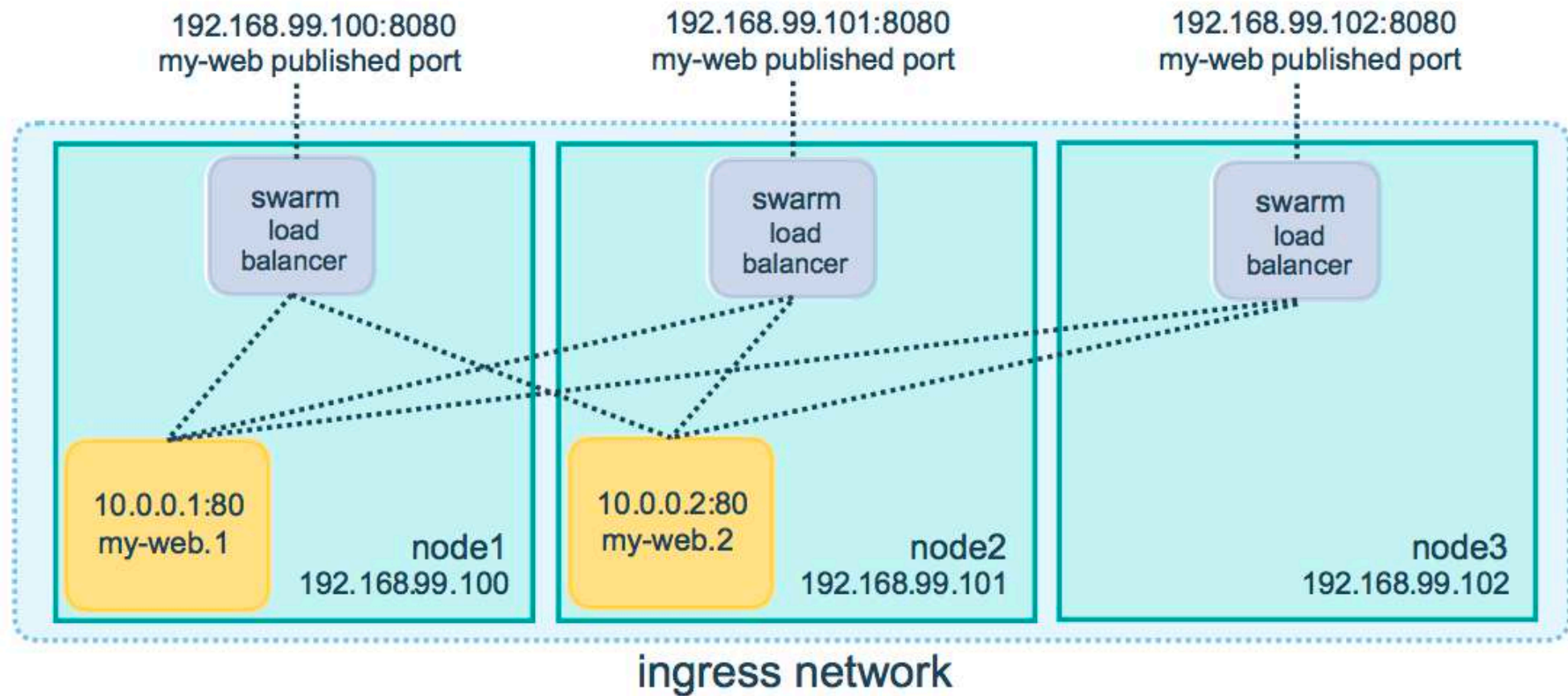
Overlay Multi-Host Networking

- Just choose `--driver overlay` when creating network
- For container-to-container traffic inside a single Swarm
- Optional IPsec (AES) encryption on network creation
- Each service can be connected to multiple networks
 - (e.g. front-end, back-end)

Routing Mesh

- Routes ingress (incoming) packets for a Service to proper Task
- Spans all nodes in Swarm
- Uses IPVS from Linux Kernel
- Load balances Swarm Services across their Tasks
- Two ways this works:
 - Container-to-container in a Overlay network (uses VIP)
 - External traffic incoming to published ports (all nodes listen)



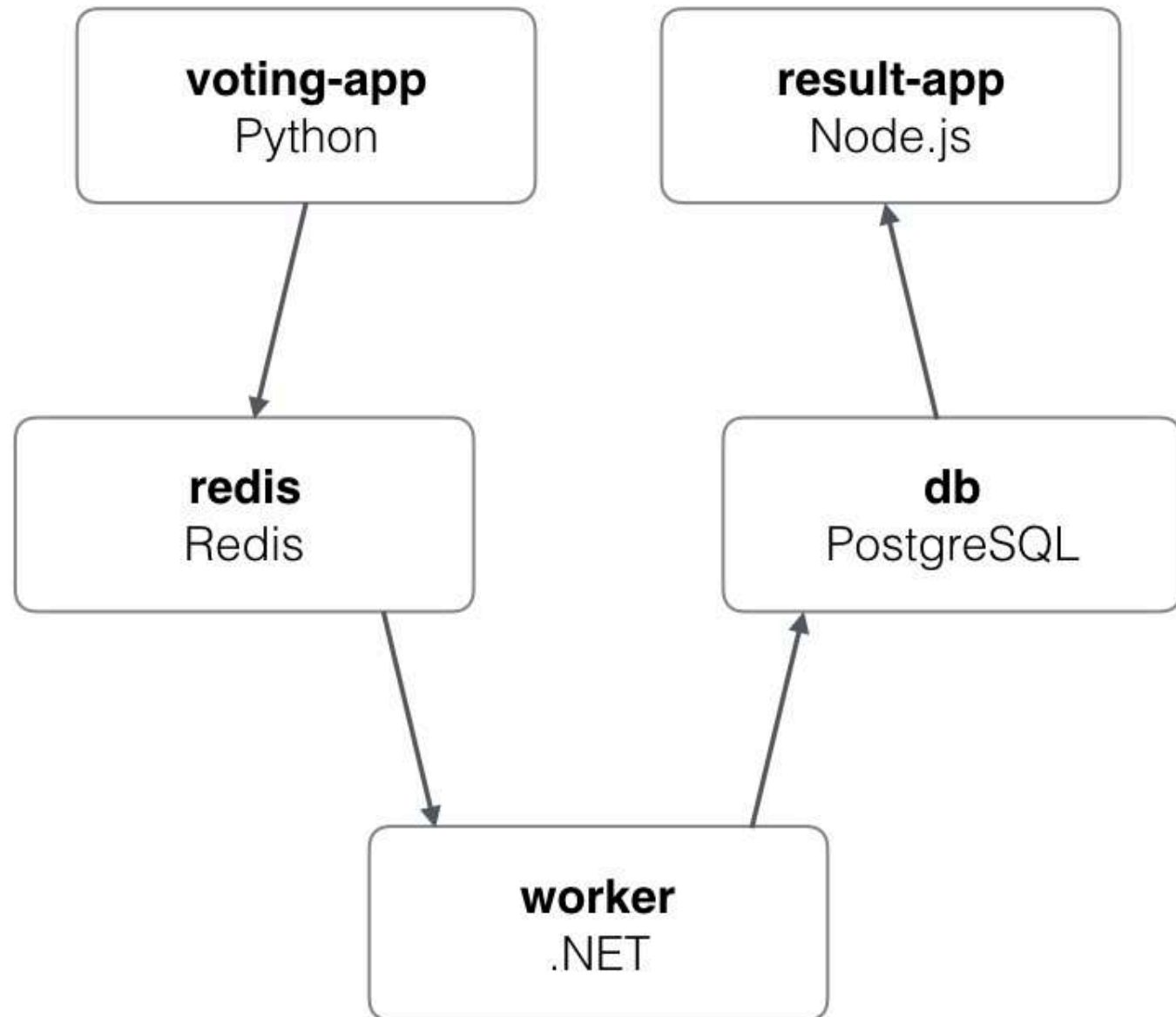


Routing Mesh Cont.

- This is stateless load balancing
- This LB is at OSI Layer 3 (TCP), not Layer 4 (DNS)
- Both limitation can be overcome with:
- Nginx or HAProxy LB proxy, or:
- Docker Enterprise Edition, which comes with built-in L4 web proxy

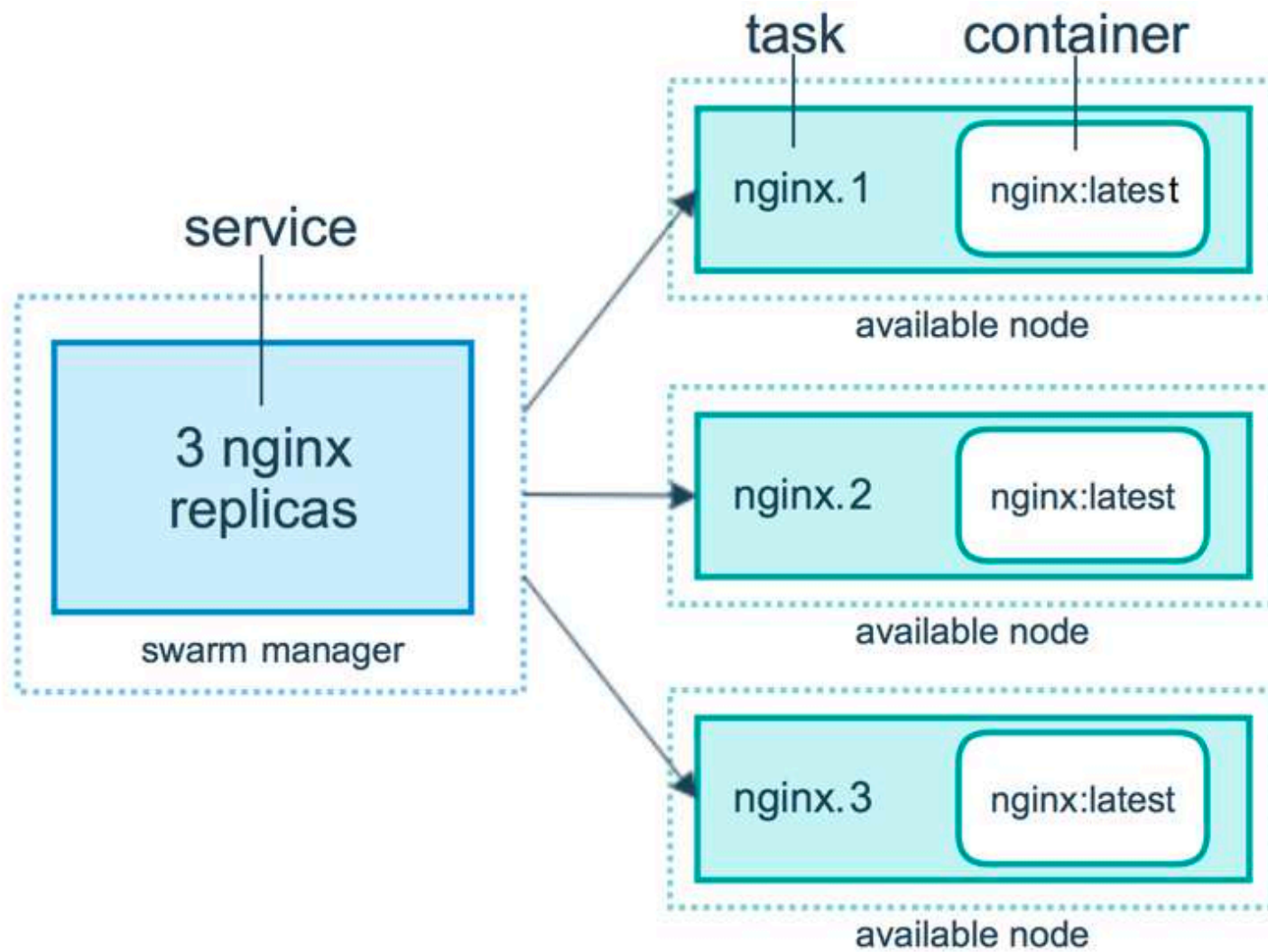
Assignment: Create Multi-Service App

- Using Docker's Distributed Voting App
- use `swarm-app-1` directory in our course repo for requirements
- 1 volume, 2 networks, and 5 services needed
- Create the commands needed, spin up services, and test app
- Everything is using Docker Hub images, so no data needed on Swarm
- Like many computer things, this is ½ art form and ½ science

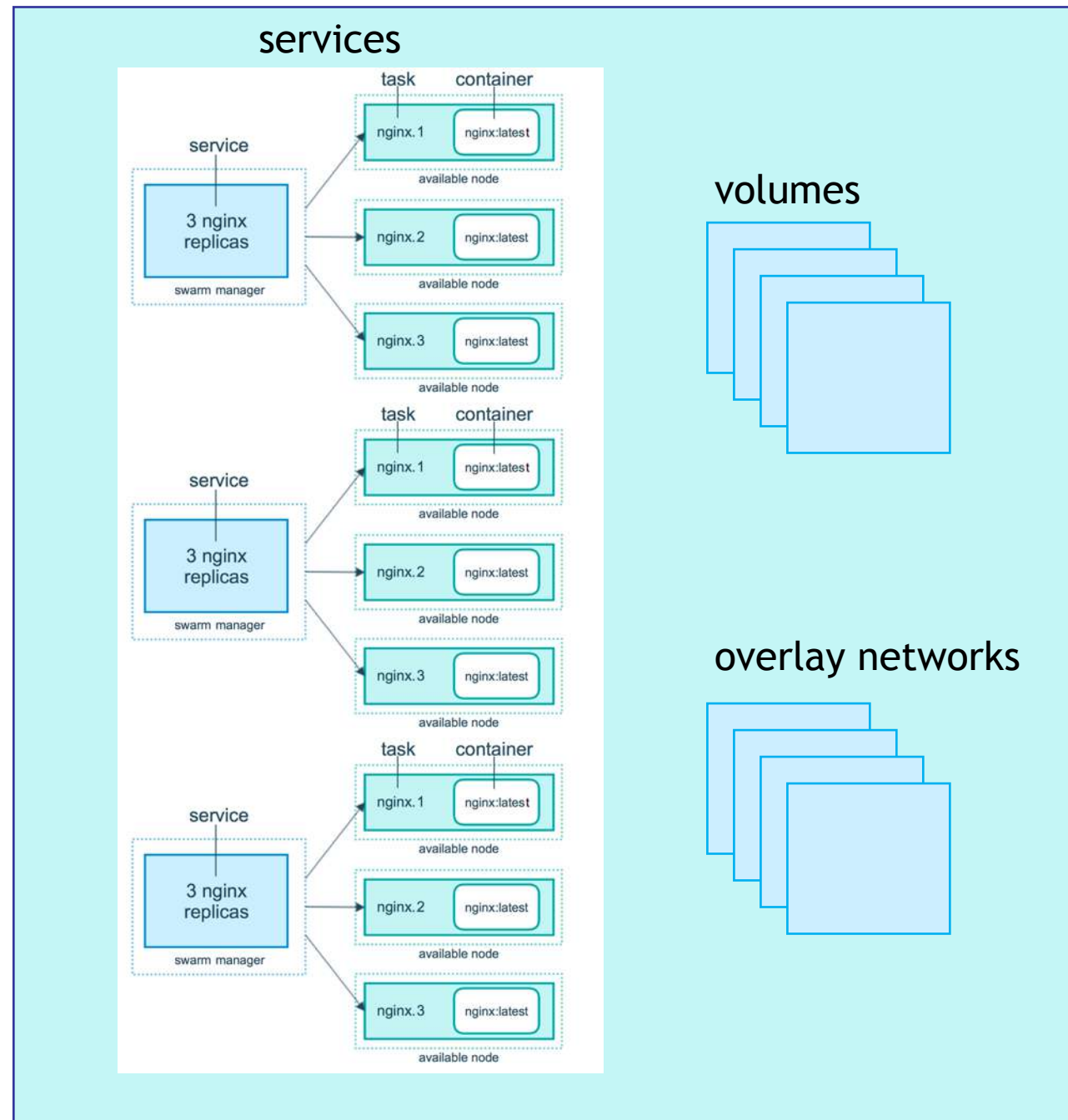


Stacks: Production Grade Compose

- In 1.13 Docker adds a new layer of abstraction to Swarm called Stacks
- Stacks accept Compose files as their declarative definition for services, networks, and volumes
- We use `docker stack deploy` rather than `docker service create`
- Stacks manages all those objects for us, including overlay network per stack. Adds stack name to start of their name
- New `deploy:` key in Compose file. Can't do `build:`
- Compose now ignores `deploy:`, Swarm ignores `build:`
- `docker-compose` cli not needed on Swarm server



stack!



Secrets Storage

- Easiest "secure" solution for storing secrets in Swarm
- What is a Secret?
 - Usernames and passwords
 - TLS certificates and keys
 - SSH keys
 - Any data you would prefer not be "on front page of news"
- Supports generic strings or binary content up to 500Kb in size
- Doesn't require apps to be rewritten

Secrets Storage Cont.

- As of Docker 1.13.0 Swarm Raft DB is encrypted on disk
- Only stored on disk on Manager nodes
- Default is Managers and Workers "control plane" is TLS + Mutual Auth
- Secrets are first stored in Swarm, then assigned to a Service(s)
- Only containers in assigned Service(s) can see them
- They look like files in container but are actually in-memory fs
- `/run/secrets/<secret_name>` or `/run/secrets/<secret_alias>`
- Local docker-compose can use file-based secrets, but not secure

Assignment: Create Stack w/ Secrets

- Let's use our Drupal compose file from last assignment
 - (`compose-assignment-2`)
- Rename image back to official `drupal:8.2`
- Remove `build:`
- Add secret via `external:`
- use environment variable `POSTGRES_PASSWORD_FILE`
- Add secret via cli `echo "<pw>" | docker secret create psql-pw -`
- Copy compose into a new yml file on you Swarm node1

Full App Lifecycle With Compose

- Live The Dream!
- Single set of Compose files for:
- Local `docker-compose up` development environment
- Remote `docker-compose up` CI environment
- Remote `docker stack deploy` production environment
- Note: `docker-compose -f a.yml -f b.yml config` mostly works
- Note: Compose `extends`: doesn't work yet in Stacks
- Fast moving part of toolset. Expect this to change/fix.

Service Updates

- Provides rolling replacement of tasks/containers in a service
- Limits downtime (be careful with "prevents" downtime)
- Will replace containers for most changes
- Has many, many cli options to control the update
- Create options will usually change, adding -add or -rm to them
- Includes rollback and healthcheck options
- Also has scale & rollback subcommand for quicker access
 - `docker service scale web=4` and `docker service rollback web`
- A stack deploy, when pre-existing, will issue service updates

Swarm Update Examples

- Just update the image used to a newer version
 - `docker service update --image myapp:1.2.1 <servicename>`
- Adding an environment variable and remove a port
 - `docker service update --env-add NODE_ENV=production --publish-rm 8080`
- Change number of replicas of two services
 - `docker service scale web=8 api=6`

Swarm Updates in Stack Files

Same command. Just edit the YAML file, then

```
docker stack deploy -c file.yml <stackname>
```


Lecture Cleanup

- Remove the service we created

> `docker service rm web`

Docker Healthchecks

- **HEALTHCHECK** was added in 1.12
- Supported in Dockerfile, Compose YAML, docker run, and Swarm Services
- Docker engine will **exec**'s the command in the container
 - (e.g. **curl localhost**)
- It expects **exit 0** (OK) or **exit 1** (Error)
- Three container states: starting, healthy, unhealthy
- Much better then "is binary still running?"
- Not a external monitoring replacement

Docker Healthchecks Cont.

- Healthcheck status shows up in `docker container ls`
- Check last 5 healthchecks with `docker container inspect`
- Docker run does nothing with healthchecks
- Services will replace tasks if they fail healthcheck
- Service updates wait for them before continuing

Healthcheck Docker Run Example

```
docker run \  
  --health-cmd="curl -f localhost:9200/_cluster/health || false" \  
  --health-interval=5s \  
  --health-retries=3 \  
  --health-timeout=2s \  
  --health-start-period=15s \  
  elasticsearch:2
```

Healthcheck Dockerfile Examples

- Options for healthcheck command
 - `--interval=DURATION` (default: 30s)
 - `--timeout=DURATION` (default: 30s)
 - `--start-period=DURATION` (default: 0s) (17.09+)
 - `--retries=N` (default: 3)
- Basic command using default options
 - `HEALTHCHECK curl -f http://localhost/ || false`
- Custom options with the command
 - `HEALTHCHECK --timeout=2s --interval=3s --retries=3 \`
 - `CMD curl -f http://localhost/ || exit 1`

Healthcheck in Nginx Dockerfile

Static website running in Nginx, just test default URL

FROM nginx:1.13

HEALTHCHECK --interval=30s --timeout=3s \
CMD curl -f http://localhost/ || exit 1

Healthcheck in PHP Nginx Dockerfile

PHP-FPM running behind Nginx, test the Nginx and FPM status URLs

FROM your-nginx-php-fpm-combo-image

- # don't do this if php-fpm is another container
- # must enable php-fpm ping/status in pool.ini
- # must forward /ping and /status urls from nginx to php-fpm

```
HEALTHCHECK --interval=5s --timeout=3s \
  CMD curl -f http://localhost/ping || exit 1
```

Healthcheck in postgres Dockerfile

Use a PostgreSQL utility to test for ready state

FROM postgres

specify real user with -U to prevent errors in log

```
HEALTHCHECK --interval=5s --timeout=3s \  
  CMD pg_isready -U postgres || exit 1
```

Healthcheck in Compose/Stack Files

version: "2.1" (minimum for healthchecks)

services:

web:

image: nginx

healthcheck:

test: ["CMD", "curl", "-f", "http://localhost"]

interval: 1m30s

timeout: 10s

retries: 3

start_period: 1m #version 3.4 minimum

Lecture Cleanup

- Remove the containers and services we created

> docker container rm -f p1 p2

> docker service rm p1 p2

Container Registries

- An image registry needs to be part of your container plan
- More Docker Hub details including auto-build
- How Docker Store (store.docker.com) is different then Hub
- How Docker Cloud (cloud.docker.com) is different then Hub
- Use new Swarms feature in Cloud to connect Mac/Win to Swarm
- Install and use Docker Registry as private image store
- 3rd Party registry options

Docker Hub: Digging Deeper

- The most popular public image registry
- It's really Docker Registry plus lightweight image building
- Let's explore more of the features of Docker Hub
- Link GitHub/BitBucket to Hub and auto-build images on commit
- Chain image building together

Docker Store: What Is It For?

- Download Docker "Editions"
- Find certified Docker/Swarm plugins and commercial certified images

Docker Cloud: CI/CD and Server Ops

- Web based Docker Swarm creation/management
- Uses popular cloud hosters and bring-your-own-server
- Automated image building, testing, and deployment
- More advanced than what Docker Hub does for free
- Includes an image security scanning service

Running Docker Registry

- A private image registry for your network
- Part of the docker/distribution GitHub repo
- The de facto in private container registries
- Not as full featured as Hub or others, no web UI, basic auth only
- At its core: a web API and storage system, written in Go
- Storage supports local, S3/Azure/Alibaba/Google Cloud, and OpenStack Swift

Running Docker Registry Cont.

- Look in section resources for links to:
- Secure your Registry with TLS
- Storage cleanup via Garbage Collection
- Enable Hub caching via "--registry-mirror"

Run a Private Docker Registry

- Run the registry image on default port 5000
- Re-tag an existing image and push it to your new registry
- Remove that image from local cache and pull it from new registry
- Re-create registry using a bind mount and see how it stores data

Registry and Proper TLS

- "Secure by Default": Docker won't talk to registry without HTTPS
- Except, localhost (127.0.0.0/8)
- For remote self-signed TLS, enable "insecure-registry" in engine

Run a Private Docker Registry Recap

- Run the registry image
 - `docker container run -d -p 5000:5000 --name registry registry`
- Re-tag an existing image and push it to your new registry
 - `docker tag hello-world 127.0.0.1:5000/hello-world`
 - `docker push 127.0.0.1:5000/hello-world`
- Remove that image from local cache and pull it from new registry
 - `docker image remove hello-world`
 - `docker image remove 127.0.0.1:5000/hello-world`
 - `docker pull 127.0.0.1:5000/hello-world`
- Re-create registry using a bind mount and see how it stores data
 - `docker container run -d -p 5000:5000 --name registry -v $(pwd)/registry-data:/var/lib/registry registry`

Remember To Cleanup!

- No containers created in this Lecture are required for future Lectures

Private Docker Registry with Swarm

- Works the same way as localhost
- Because of Routing Mesh, all nodes can see 127.0.0.1:5000
- Remember to decide how to store images (volume driver)
- NOTE: All nodes must be able to access images
- ProTip: Use a hosted SaaS registry if possible

What is Kubernetes?



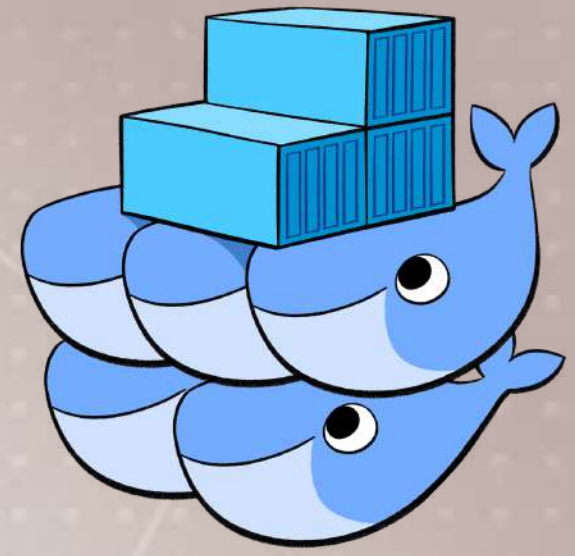
- Kubernetes = popular container orchestrator
- Container Orchestration = Make many servers act like one
- Released by Google in 2014, maintained by large community
- Runs on top of Docker (usually) as a set of APIs in containers
- Provides API/CLI to manage containers across servers
- Many clouds provide it for you
- Many vendors make a "distribution" of it

Why Kubernetes?



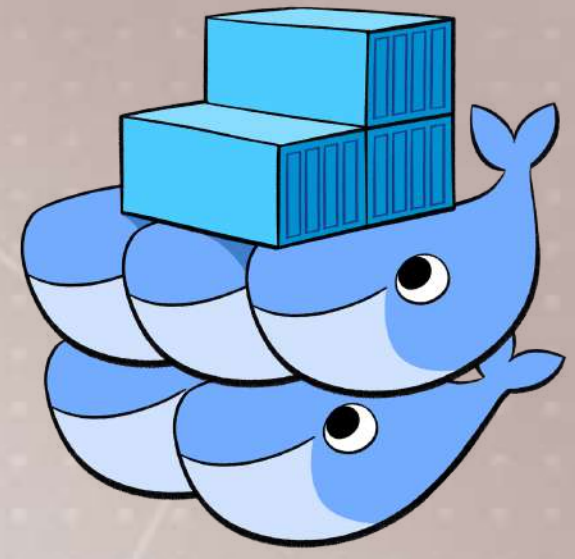
- Review "Swarm Mode: Built-In Orchestration"
- Orchestration: Next logical step in journey to faster DevOps
- First, understand why you *may* need orchestration
- Not every solution needs orchestration
- Servers + Change Rate = Benefit of orchestration
- Then, decide which orchestrator
- If Kubernetes, decide which distribution
 - cloud or self-managed (Docker Enterprise, Rancher, OpenShift, Canonical, VMWare PKS)
 - Don't usually need pure upstream

Kubernetes or Swarm?



- Review "Swarm Mode: Built-In Orchestration"
- Kubernetes and Swarm are both container orchestrators
- Both are solid platforms with vendor backing
- Swarm: Easier to deploy/manage
- Kubernetes: More features and flexibility
- What's right for you? Understand both and know your requirements

Advantages of Swarm



- Comes with Docker, single vendor container platform
- Easiest orchestrator to deploy/manage yourself
- Follows 80/20 rule, 20% of features for 80% of use cases
- Runs anywhere Docker does:
 - local, cloud, datacenter
 - ARM, Windows, 32-bit
- Secure by default
- Easier to troubleshoot

Advantages of Kubernetes



- Clouds will deploy/manage Kubernetes for you
- Infrastructure vendors are making their own distributions
- Widest adoption and community
- Flexible: Covers widest set of use cases
- "Kubernetes first" vendor support
- "No one ever got fired for buying IBM"
 - Picking solutions isn't 100% rational
 - Trendy, will benefit your career
 - CIO/CTO Checkbox



Basic Terms: System Parts

- Kubernetes: The whole orchestration system
 - K8s "k-eights" or Kube for short
- Kubectl: CLI to configure Kubernetes and manage apps
 - Using "cube control" official pronunciation
- Node: Single server in the Kubernetes cluster
- Kubelet: Kubernetes agent running on nodes
- Control Plane: Set of containers that manage the cluster
 - Includes API server, scheduler, controller manager, etcd, and more
 - Sometimes called the "master"

Install Kubernetes Locally



- Kubernetes is a series of containers, CLI's, and configurations
- Many ways to install, lets focus on easiest for learning
- Docker Desktop: Enable in settings
 - Sets up everything inside Docker's existing Linux VM
- Docker Toolbox on Windows: MiniKube
 - Uses VirtualBox to make Linux VM
- Your Own Linux Host or VM: MicroK8s
 - Installs Kubernetes right on the OS

Kubernetes In A Browser



- Try <http://play-with-k8s.com> or katacoda.com in browser
- Easy to get started
- Doesn't keep your environment

Docker Desktop



- Runs/configures Kubernetes Master containers
- Manages kubectl install and certs
- Easily install, disable, and remove from Docker GUI

MiniKube



- Download Windows Installer from GitHub
- minikube-installer.exe
- minikube start
- Much like the docker-machine experience
- Creates a VirtualBox VM with Kubernetes master setup
- Doesn't install kubectl

MicroK8s



- Installs Kubernetes (without Docker Engine) on localhost (Linux)
- Uses snap (rather than apt or yum) for install
- Control the MicroK8s service via **microk8s.** commands
- kubectl accessible via **microk8s.kubectl**
- Add CoreDNS for services to work
 - **microk8s.enable dns**
- Add an alias to your shell (**.bash_profile**)
 - **alias kubectl=microk8s.kubectl**

Kubernetes Container Abstractions



- **Pod:** one or more containers running together on one Node
 - Basic unit of deployment. Containers are always in pods
- **Controller:** For creating/updating pods and other objects
 - Many types of Controllers inc. Deployment, ReplicaSet, StatefulSet, DaemonSet, Job, CronJob, etc.
- **Service:** network endpoint to connect to a pod
- **Namespace:** Filtered group of objects in cluster
- **Secrets, ConfigMaps, and more**

Kubernetes Run, Create, and Apply



- Kubernetes is evolving, and so is the CLI
- We get three ways to create pods from the kubectl CLI
 - > `kubectl run` (changing to be only for pod creation)
 - > `kubectl create` (create some resources via CLI or YAML)
 - > `kubectl apply` (create/update anything via YAML)
- For now we'll just use `run` or `create` CLI
- Later we'll learn YAML and pros/cons of each



Creating Pods with kubectl

- Are we working?
 - > `kubectl version`
- Two ways to deploy Pods (containers): Via commands, or via YAML
- Let's run a pod of the nginx web server!
 - > `kubectl create deployment my-nginx --image nginx`
- Let's list the pod
 - > `kubectl get pods`
- Let's see all objects
 - > `kubectl get all`

Pods -> ReplicaSet -> Deployment



Cleanup



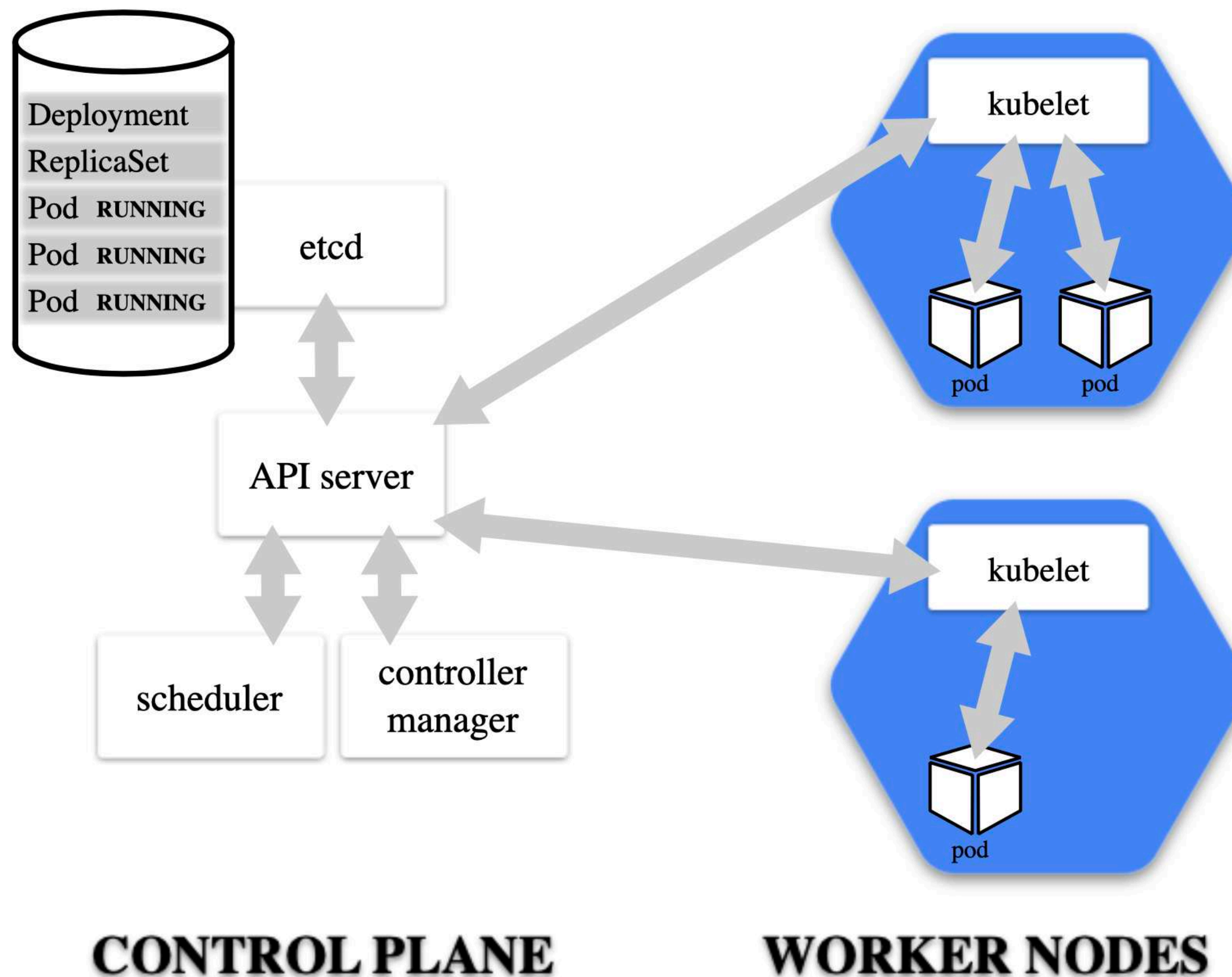
- Let's remove the Deployment
 - > `kubectl delete deployment my-nginx`



Scaling ReplicaSets

- Start a new deployment for one replica/pod
 - > `kubectl create deployment my-apache --image httpd`
- Let's scale it up with another pod
 - > `kubectl scale deploy/my-apache --replicas 2`
 - > `kubectl scale deployment my-apache --replicas 2`
- those are the same command
- `deploy = deployment = deployments`

What Just Happened? kubectl scale



Don't Cleanup



- We'll use these httpd containers in the next lecture



Inspecting Deployment Objects

- > `kubectl get pods`
- Get container logs
 - > `kubectl logs deployment/my-apache --follow --tail 1`
- Get a bunch of details about an object, including events!
 - > `kubectl describe pod/my-apache-xxxx-yyyy`
- Watch a command (without needing `watch`)
 - > `kubectl get pods -w`
- In a separate tab/window
 - > `kubectl delete pod/my-apache-xxxx-yyyy`
- Watch the pod get re-created

Cleanup



- Let's remove the Deployment
 - > `kubectl delete deployment my-apache`

Exposing Containers



- **kubectl expose** creates a **service** for existing pods
- A **service** is a stable address for pod(s)
- If we want to connect to pod(s), we need a **service**
- CoreDNS allows us to resolve **services** by name
- There are different types of **services**
 - ClusterIP
 - NodePort
 - LoadBalancer
 - ExternalName



Basic Service Types

- **ClusterIP** (default)
 - Single, internal virtual IP allocated
 - Only reachable from within cluster (nodes and pods)
 - Pods can reach service on apps port number
- **NodePort**
 - High port allocated on each node
 - Port is open on every node's IP
 - Anyone can connect (if they can reach node)
 - Other pods need to be updated to this port
- **These services are always available in Kubernetes**

More Service Types



- **LoadBalancer**
 - Controls a LB endpoint external to the cluster
 - Only available when infra provider gives you a LB (AWS ELB, etc)
 - Creates NodePort+ClusterIP services, tells LB to send to NodePort
- **ExternalName**
 - Adds CNAME DNS record to CoreDNS only
 - Not used for Pods, but for giving pods a DNS name to use for something outside Kubernetes
- **Kubernetes Ingress:** We'll learn later

Creating a ClusterIP Service



- Open two shell windows so we can watch this
 - > `kubectl get pods -w`
- In second window, lets start a simple http server using sample code
 - > `kubectl create deployment httpenv --image=bretfisher/httpenv`
- Scale it to 5 replicas
 - > `kubectl scale deployment/httpenv --replicas=5`
- Let's create a ClusterIP service (default)
 - > `kubectl expose deployment/httpenv --port 8888`

Inspecting ClusterIP Service



- Look up what IP was allocated
 - > `kubectl get service`
- Remember this IP is cluster internal only, how do we curl it?
- If you're on Docker Desktop (Host OS is not container OS)
 - > `kubectl run --generator=run-pod/v1 tmp-shell --rm -it --image bretfisher/netshoot -- bash`
 - > `curl httpenv:8888`
- If you're on Linux host
 - > `curl [ip of service]:8888`

Cleanup



- Leave the deployment there, we'll use it in the next Lecture

Create a NodePort Service



- Let's expose a NodePort so we can access it via the host IP (including localhost on Windows/Linux/macOS)
 - > `kubectl expose deployment/httpenv --port 8888 --name httpenv-np --type NodePort`
- Did you know that a NodePort service also creates a ClusterIP?
- These three service types are additive, each one creates the ones above it:
 - ClusterIP
 - NodePort
 - LoadBalancer

Add a LoadBalancer Service



- If you're on Docker Desktop, it provides a built-in LoadBalancer that publishes the `--port` on localhost
 - > `kubectl expose deployment/httpenv --port 8888 --name httpenv-lb --type LoadBalancer`
 - > `curl localhost:8888`
- If you're on kubeadm, minikube, or microk8s
 - No built-in LB
 - You can still run the command, it'll just stay at "pending" (but its NodePort works)

Cleanup



- Let's remove the Services and Deployment
 - > `kubectl delete service/httpenv service/httpenv-np`
 - > `kubectl delete service/httpenv-lb deployment/httpenv`



Kubernetes Services DNS

- Starting with 1.11, internal DNS is provided by CoreDNS
- Like Swarm, this is DNS-Based Service Discovery
- So far we've been using hostnames to access Services
 - > `curl <hostname>`
- But that only works for Services in the same Namespace
 - > `kubectl get namespaces`
- Services also have a FQDN
 - > `curl <hostname>.<namespace>.svc.cluster.local`

Assignment: Explore run get and logs



- Dry Run
 - > `kubectl create deployment nginx --image nginx --dry-run`
- Run does different things based on options
 - > `kubectl create deployment nginx --image nginx --dry-run --port 80 --expose`
- Only create a simple Pod, not a Deployment, ReplicaSet, etc.
 - > `kubectl run nginx-pod --generator=run-pod/v1 --image nginx`
- Get a shell in new Pod, remove on exit
 - > `kubectl run shell --generator=run-pod/v1 --rm -it --image busybox`

Assignment: Explore run get and logs



- Create a Deployment and ClusterIP Service in one line
 - > `kubectl run nginx2 --image nginx --replicas 2`
- Get multiple resources in one line
 - > `kubectl get deploy,pods`
- Get all pods, in wide format (gives more info)
 - > `kubectl get pods -o wide`
- Get all pods and show labels
 - > `kubectl get pods --show-labels`
-

Assignment: Explore run get and logs



- Better log viewing with stern
 - github.com/wercker/stern
 - > `kubectl run mydate --image bretfisher/date --replicas 3`
 - > `kubectl logs deployment/mydate`
 - > `stern mydate`

Cleanup



- Let's remove everything but the service/kubernetes
 - > kubectl get all
 - > kubectl delete deployment/nginx2 pod/nginx-pod

Run, Create, and Expose Generators



- These commands use helper templates called "generators"
- Every resource in Kubernetes has a specification or "spec"
 - > `kubectl create deployment sample --image nginx --dry-run -o yaml`
- You can output those templates with `--dry-run -o yaml`
- You can use those YAML defaults as a starting point
- Generators are "opinionated defaults"

Generator Examples



- Using dry-run with yaml output we can see the generators
 - > `kubectl create deployment test --image nginx --dry-run -o yaml`
 - > `kubectl create job test --image nginx --dry-run -o yaml`
 - > `kubectl expose deployment/test --port 80 --dry-run -o yaml`
 - You need the deployment to exist before this works

Cleanup



- Let's remove the Deployment
 - > `kubectl delete deployment test`



The Future of kubectl run

- Right now (1.12-1.15) run is in a state of flux
- The goal is to reduce its features to only create Pods
 - Right now it defaults to creating Deployments (with the warning)
 - It has lots of generators but they are all deprecated
 - The idea is to make it easy like **docker run** for one-off tasks
- It's not recommended for production
- Use for simple dev/test or troubleshooting pods

Old Run Confusion



- The generators activate different Controllers based on options
- Using dry-run we can see which generators are used
 - > `kubectl run test --image nginx --dry-run`
 - > `kubectl run test --image nginx --port 80 --expose --dry-run`
 - > `kubectl run test --image nginx --restart OnFailure --dry-run`
 - > `kubectl run test --image nginx --restart Never --dry-run`
 - > `kubectl run test --image nginx --schedule "*/1 * * * *" --dry-run`



Imperative vs. Declarative

- Imperative: Focus on *how* a program operates
- Declarative: Focus on *what* a program should accomplish
- Example: "I'd like a cup of coffee"
- Imperative: I boil water, scoop out 42 grams of medium-fine grounds, pour over 700 grams of water, etc.
- Declarative: "Barista, I'd like a a cup of coffee".
(Barista is the engine that works through the steps, including retrying to make a cup, and is only finished when I have a cup)

Kubernetes Imperative



- Examples: `kubectl run`, `kubectl create deployment`, `kubectl update`
 - We start with a state we know (no deployment exists)
 - We ask `kubectl run` to create a deployment
- Different commands are required to change that deployment
- Different commands are required per object
- Imperative is easier when you know the state
- Imperative is easier to get started
- Imperative is easier for humans at the CLI
- Imperative is NOT easy to automate

Kubernetes Declarative



- Example: `kubectl apply -f my-resources.yaml`
 - We don't know the current state
 - We only know what we want the end result to be (yaml contents)
- Same command each time (tiny exception for delete)
- Resources can be all in a file, or many files (apply a whole dir)
- Requires understanding the YAML keys and values
- More work than `kubectl run` for just starting a pod
- The easiest way to automate
- The eventual path to GitOps happiness

Three Management Approaches



- **Imperative commands:** `run`, `expose`, `scale`, `edit`, `create deployment`
 - Best for dev/learning/personal projects
 - Easy to learn, hardest to manage over time
- **Imperative objects:** `create -f file.yml`, `replace -f file.yml`, `delete...`
 - Good for prod of small environments, single file per command
 - Store your changes in git-based yaml files
 - Hard to automate
- **Declarative objects:** `apply -f file.yml` or `dir\`, `diff`
 - Best for prod, easier to automate
 - Harder to understand and predict changes

Three Management Approaches



- **Most Important Rule:**
 - Don't mix the three approaches
- **Bret's recommendations:**
 - Learn the Imperative CLI for easy control of local and test setups
 - Move to `apply -f file.yml` and `apply -f directory\` for prod
 - Store yaml in git, git commit each change before you apply
 - This trains you for later doing GitOps (where git commits are automatically applied to clusters)

kubectl apply



- Remember the three management approaches?
- Let's skip to full Declarative objects
- > `kubectl apply -f filename.yml`
- Why skip `kubectl create`, `kubectl replace`, `kubectl edit`?
- What I recommend \neq all that's possible



Using kubectl apply

- create/update resources in a file
 - > `kubectl apply -f myfile.yaml`
- create/update a whole directory of yaml
 - > `kubectl apply -f myyaml/`
- create/update from a URL
 - > `kubectl apply -f https://bret.run/pod.yaml`
- Be careful, lets look at it first (browser or curl)
 - > `curl -L https://bret.run/pod`
 - Win PoSH? `start https://bret.run/pod.yaml`

Kubernetes Configuration YAML



- Kubernetes configuration file (YAML or JSON)
- Each file contains one or more manifests
- Each manifest describes an API object (deployment, job, secret)
- Each manifest needs four parts (root key:values in the file)

apiVersion:

kind:

metadata:

spec:



Building Your YAML Files

- **kind:** We can get a list of resources the cluster supports
> `kubectl api-resources`
- Notice some resources have multiple API's (old vs. new)
- **apiVersion:** We can get the API versions the cluster supports
> `kubectl api-versions`
- **metadata:** only name is required
- **spec:** Where all the action is at!



Building Your YAML spec

- We can get all the keys each kind supports
 - > `kubectl explain services --recursive`
- Oh boy! Let's slow down
 - > `kubectl explain services.spec`
- We can walk through the spec this way
 - > `kubectl explain services.spec.type`
- `spec:` can have sub `spec:` of other resources
 - > `kubectl explain deployment.spec.template.spec.volumes.nfs.server`
- We can also use docs
 - kubernetes.io/docs/reference/#api-reference



Dry Runs With Apply YAML

- New stuff, not out of beta yet (1.15)
- dry-run a create (client side only)
 - > `kubectl apply -f app.yml --dry-run`
- dry-run a create/update on server
 - > `kubectl apply -f app.yml --server-dry-run`
- see a diff visually
 - > `kubectl diff -f app.yml`



Labels and Annotations

- Labels goes under **metadata**: in your YAML
- Simple list of **key: value** for identifying your resource later by selecting, grouping, or filtering for it
- Common examples include **tier: frontend, app: api, env: prod, customer: acme.co**
- Not meant to hold complex, large, or non-identifying info, which is what **annotations** are for
- filter a get command
 - > **kubectl get pods -l app=nginx**
- apply only matching labels
 - > **kubectl apply -f myfile.yaml -l app=nginx**

Label Selectors



- The "glue" telling Services and Deployments which pods are theirs
- Many resources use Label Selectors to "link" resource dependencies
- You'll see these match up in the Service and Deployment YAML
- Use Labels and Selectors to control which pods go to which nodes
- Taints and Tolerations also control node placement

Cleanup



- Let's remove anything you created in this section
 - > `kubectl get all`
 - > `kubectl delete <resource type> / <resource name>`

Storage in Kubernetes



- Storage and stateful workloads are harder in all systems
- Containers make it both harder and easier than before
- **StatefulSets** is a new resource type, making Pods more sticky
- Bret's recommendation: avoid stateful workloads for first few deployments until you're good at the basics
 - Use db-as-a-service whenever you can



Volumes in Kubernetes

- Creating and connecting Volumes: 2 types
- **Volumes**
 - Tied to lifecycle of a Pod
 - All containers in a single Pod can share them
- **PersistentVolumes**
 - Created at the cluster level, outlives a Pod
 - Separates storage config from Pod using it
 - Multiple Pods can share them
- CSI plugins are the new way to connect to storage

Ingress



- None of our Service types work at OSI Layer 7 (HTTP)
- How do we route outside connections based on hostname or URL?
- Ingress Controllers (optional) do this with 3rd party proxies
- Nginx is popular, but Traefik, HAProxy, F5, Envoy, Istio, etc.
- Note this is still beta (in 1.15) and becoming popular
- Implementation is specific to Controller chosen

CRD's and The Operator Pattern



- You can add 3rd party Resources and Controllers
- This extends Kubernetes API and CLI
- A pattern is starting to emerge of using these together
- Operator: automate deployment and management of complex apps
- e.g. Databases, monitoring tools, backups, and custom ingresses

Higher Deployment Abstractions



- All our **kubectl** commands just talk to the Kubernetes API
- Kubernetes has limited built-in templating, versioning, tracking, and management of your apps
- There are now over 60 3rd party tools to do that, but many are defunct
- **Helm** is the most popular
- "**Compose on Kubernetes**" comes with Docker Desktop
- Remember these are optional, and your distro may have a preference
- Most distros support **Helm**

Templating YAML



- Many of the deployment tools have templating options
- You'll need a solution as the number of environments/apps grow
- Helm was the first "winner" in this space, but can be complex
- Official Kustomize feature works out-of-the-box (as of 1.14)
- **docker app** and compose-on-kubernetes are Docker's way

Kubernetes Dashboard



- Default GUI for "upstream" Kubernetes
 - github.com/kubernetes/dashboard
- Some distributions have their own GUI (Rancher, Docker Ent, OpenShift)
- Clouds don't have it by default
- Let's you view resources and upload YAML
- Safety first!

Kubectl Namespaces and Context



- Namespaces limit scope, aka "virtual clusters"
- Not related to Docker/Linux namespaces
- Won't need them in small clusters
- There are some built-in, to hide system stuff from **kubectl** "users"
 - > **kubectl get namespaces**
 - > **kubectl get all --all-namespaces**
- Context changes **kubectl** cluster and namespace
- See **~/.kube/config** file
 - > **kubectl config get-contexts**
 - > **kubectl config set***

Future of Kubernetes



- More focus on stability and security
 - 1.14, 1.15, largely dull releases (a good thing!)
 - Recent security audit has created backlog
- Clearing away deprecated features like kubectl run generators
- Improving features like server-side dry-run
- More and improved Operators
- Helm 3.0 (easier deployment, chart repos, libs)
- More declarative-style features
- Better Windows Server support
- More edge cases, kubeadm HA clusters

Related Projects



- Kubernetes has become the "differentencing and scheduling engine backbone" for so many new projects
- Knative - Serverless workloads on Kubernetes
- k3s - mini, simple Kubernetes
- k3OS - Minimal Linux OS for k3s
- Service Mesh - New layer in distributed app traffic for better control, security, and monitoring