MASTER KUBERNETES

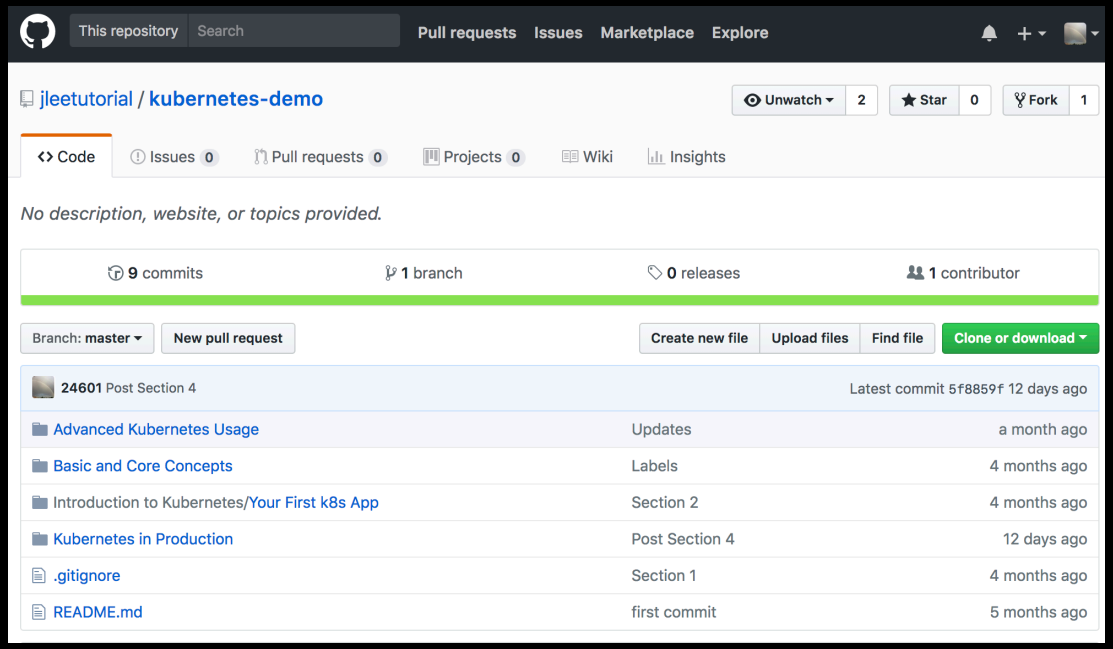
COURSE INFO

A FEW HELPFUL THINGS

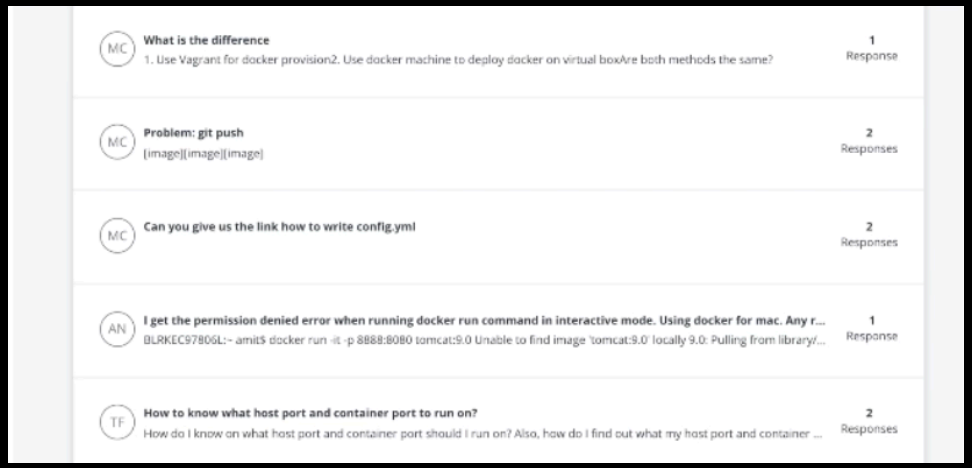
- Source Code Examples in GitHub @ https://github.com/jleetutorial/kubernetes-demo
  - We'll reference files in this repository for your convenience so you may cut & paste
  - Consider replicating the course by downloading the zip available at the URL above or replicating the repository
- Support & Community
  - Q&A Discussion Page in Udemy
  - Facebook Group

GITHUB

- Source Code Examples in GitHub @ https://github.com/jleetutorial/kubernetes-demo

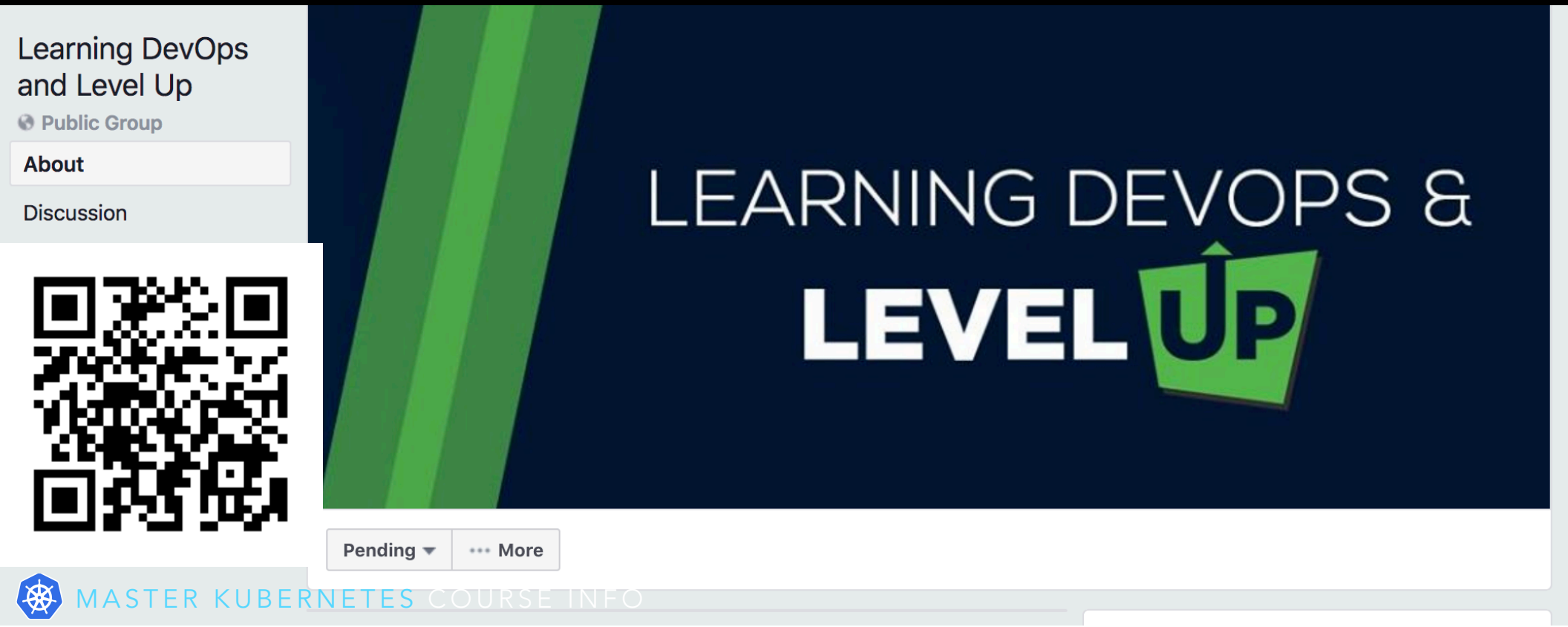


Q&A DISCUSSION PAGE IN UDEMY



FACEBOOK GROUP

- https://www.facebook.com/groups/1911219079195863/



MASTER KUBERNETES

INTRODUCTION

WHAT & WHY - CONTAINERS, ORCHESTRATION

WHAT & WHY - CONTAINERS, ORCHESTRATION

MASTER KUBERNETES INTRODUCTION

WHAT & WHY - CONTAINERS, ORCHESTRATION

	CONTAINERS	CONTAINER ORCHESTRATION
FUNCTION	KEEP SOFTWARE SEPARATED INTO ITS OWN "CLEAN" VIEW OF AN OPERATING SYSTEM	DEFINE RELATIONSHIPS BETWEEN CONTAINERS, WHERE THEY COME FROM, HOW THEY SCALE, AND HOW THEY CONNECT TO THE WORLD AROUND THEM
PREDECESSORS/ ALTERNATIVES	•VIRTUAL MACHINES •DIRECT INSTALLATION	•HOMEGROWN SCRIPTS •MANUAL, BESPOKE STATIC CONFIGURATION BETWEEN CONTAINERS
PACKAGES/VENDORS	•DOCKER •RKT •GARDEN •LXC •MESOS	•KUBERNETES •DOCKER SWARM •AMAZON ECS •MESOS

THE CONTAINER "WAR" MAYBE COMING TO A CLOSE

- Docker is by-and-far the leader
- Defines:
  - A Container Format (Dockerfile)
  - Registry to host Containers (public and private)

THE ORCHESTRA(S) - ORCHESTRATION TECHNOLOGY

- A much newer space
- Kubernetes has quite a lead, and emerging as the clear leader & winner, it's closest competitor, Docker Swarm, recently announced support for Kubernetes-style configuration & compatibility
- Cooperation & formal standards bodies are more prevalent in the growth of orchestration

REVIEW & USING KUBERNETES ORCHESTRATING CONTAINERS

- Containers have helped simplify & standardize how software is modularized & deployed
- Container orchestration has arisen to help equally simplify & standardize how these containers come together to make a usable software system
- Kubernetes as a container orchestration standard

MASTER KUBERNETES

K8S OVERVIEW

KUBERNETES

- Born from a Google internal project in mid-2014 (Google “Borg”)
- 1.0 Release in July 2015
- Google partnered with the Linux Foundation to form the Cloud Native Computing Foundation (CNCF) to offer Kubernetes as an open standard
- Frequently abbreviated “k8s” - Greek for “helmsman” or “pilot”

ADVANTAGES OF KUBERNETES

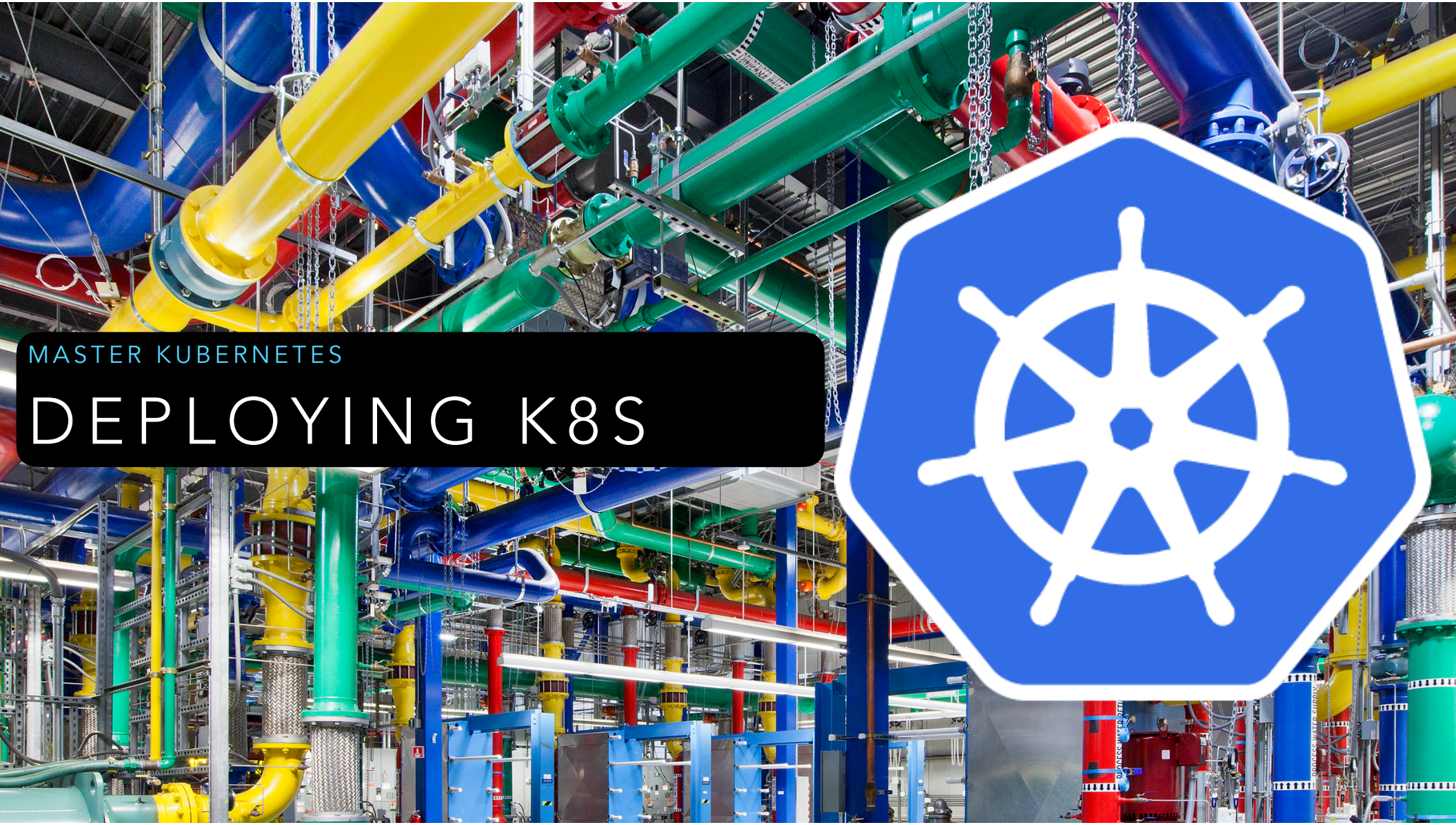
- Based on extensive experience from Google, over a long period of time.
- Large open source community & project, mature governing organization (CNCF)
- Auto-scaling, cloud-agnostic-yet-integratable technologies

MAKING KUBERNETES YOURS

- Kubernetes runs anywhere Linux does
  - Your Laptop
  - Globally distributed data centers
  - Major cloud providers
  - Anywhere in between & nearly any combination (not that you'd want to necessarily)

MASTER KUBERNETES

DEPLOYING K8S



WHERE DOES K8S LIVE?

- Anywhere you need it to
- But, where would you *want* it to live?
- That depends on who you are & what you're using it for
  - Evaluation/Training - minikube (local version of k8s)
  - Development - minikube, dev cluster on a cloud provider
  - Deployment - cloud provider or bare metal

K8S ON AMAZON WEB SERVICES (AWS)

- AWS' premier container offering, Elastic Container Service (ECS), is a proprietary orchestration system - not yet k8s compatible
- AWS has joined the CNCF - support for k8s in some form (ECS or a new offering) is all-but-guaranteed and expected soon
- In the mean time, k8s on AWS can be set up directly on EC2 instances by hand or using kops (preferable)
- kops is a tool for production grade k8s installation, upgrades, and management - maintained by the CNCF

K8S ON GOOGLE CLOUD PLATFORM (GCP)

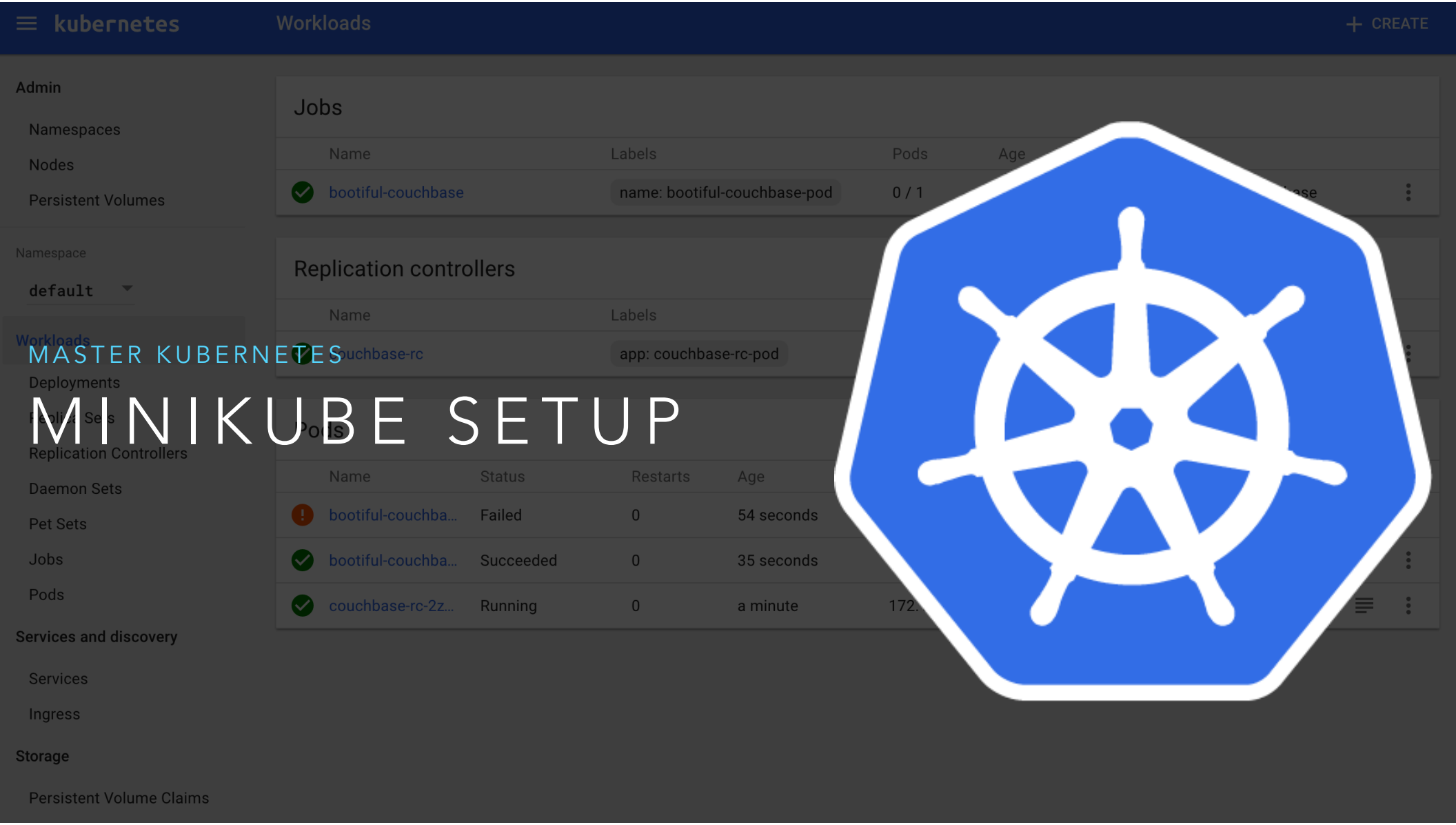
- GCP has high-level support for Kubernetes through its Google Container Engine (GKE) offering
  - No additional fee for clusters <= 5 nodes (user still pays price for any running VMs in cluster)
  - Automates cluster setup, maintenance, monitoring, scaling, health checking
- Still possible to use kops or manual configuration to bring-up a cluster on Google Compute Engine VMs, of course

K8S ON YOUR WORKSTATION

- GCP has high-level support for Kubernetes through its Google Container Engine (GKE) offering
  - No additional fee for clusters <= 5 nodes (user still pays price for any running VMs in cluster)
  - Automates cluster setup, maintenance, monitoring, scaling, health checking
- Still possible to use kops or manual configuration to bring-up a cluster on Google Compute Engine VMs, of course

MASTER KUBERNETES

MINIKUBE SETUP



OVERVIEW OF MINIKUBE

- An all-in-one install of Kubernetes
- Takes all the distributed components of Kubernetes and packages them into a single virtual machine to run locally
- A Few (Important) Caveats:
  - Minikube does not support Cloud Provider specific features such as Load Balancers, Persistent Volumes, Ingress (Don't worry about these yet if you don't know what these mean!)
  - Minikube requires Virtualization - *on Linux Virtualbox must be installed and on OS X either Virtualbox or VMWare Fusion*

OVERVIEW OF INSTALLING MINIKUBE

- Minikube will be a Kubernetes cluster running on your machine
- We'll start minikube and then connect to it from the kubernetes command line tools (kubectl), just as you will when connecting to kubernetes clusters running remotely

DEMO: INSTALLING MINIKUBE OVERVIEW

- Start with a Linux or Mac OS X machine with a hypervisor installed
- Install kubectl
- Install minikube

A FEW CONCEPTS TO NOTE

- Kubernetes "deployments" are the high-level construct that define an application
- "Pods" are instances of a container in a deployment
- "Services" are endpoints that export ports to the outside world
- You can create, delete, modify, and retrieve information about any of these using the *kubectl* command (as well as the Kubernetes local UI we will show later)

INSTALLING KUBECTL

1. Download the latest package for your OS**

Linux:
```
$ curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
```

macOS:
```
$ curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/darwin/amd64/kubectl
```

Windows:
```
curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.8.0/bin/windows/amd64/kubectl.exe
```

2. Make the binary executable (macOS and Linux only)

```
$ chmod +x ./kubectl
```

3. Add the binary to your PATH

Linux:
```
$ sudo mv ./kubectl /usr/local/bin/kubectl
```

macOS:
```
$ sudo mv ./kubectl /usr/local/bin/kubectl
```

Windows: *Use Explorer to move the kubectl binary to a directory in your path of your choice.*

- Latest installation instruction for macOS*, Linux, and Windows are always available online:
  - "Install and Set Up kubectl" in the Kubernetes Documentation
  - *macOS users should consider using the "homebrew" package manager to install and manager kubectl
  - **Google Cloud Platform users may also use the Google Cloud Platform tools to install kubectl, consult Google's SDK for information

TESTING KUBECTL

```
kubectl version
```

```
overpriced-strip-of-glass:Downloads basitmustafa$ kubectl version
Client Version: version.Info{Major:"1", Minor:"8", GitVersion:"v1.8.2", GitCommit:"bdaeafa71f6c7c04636251031f93464384d5496
3", GitTreeState:"clean", BuildDate:"2017-10-24T21:08:42Z", GoVersion:"go1.9.1", Compiler:"gc", Platform:"darwin/amd64"}
Server Version: version.Info{Major:"1", Minor:"8+", GitVersion:"v1.8.1-gke.1", GitCommit:"aba494e68a76583d2d7d1b9c97e4a97a
19c3a920", GitTreeState:"clean", BuildDate:"2017-10-27T23:54:39Z", GoVersion:"go1.8.3b4", Compiler:"gc", Platform:"linux/a
md64"}
```

INSTALLING MINIKUBE

1. Download the latest package for your OS*

Linux:
```
$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.23.0/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

macOS:
```
$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.23.0/minikube-darwin-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

Windows: *Use browser download link from releases link below*

- Latest installation instruction for macOS*, Linux, and Windows are always available online:
  - "Minikube Releases" at the Minikube Github repository
  - *Debian-based Linux & Windows users should consider using the installer packages listed at the above URL

DEMO: BASIC MINIKUBE COMMANDS

- Start minikube

```
$ minikube start
```

- Deploy a sample Kubernetes "deployment" to your local minikube

```
$ kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
```

- Expose this deployment to an external network

```
$ kubectl expose deployment hello-minikube  --type=NodePort
```

- List the "pods" of this deployment

```
$ kubectl get pod
```

- Access the sample service

```
$ curl $(minikube service hello-minikube --url)
```

- Delete the deployment

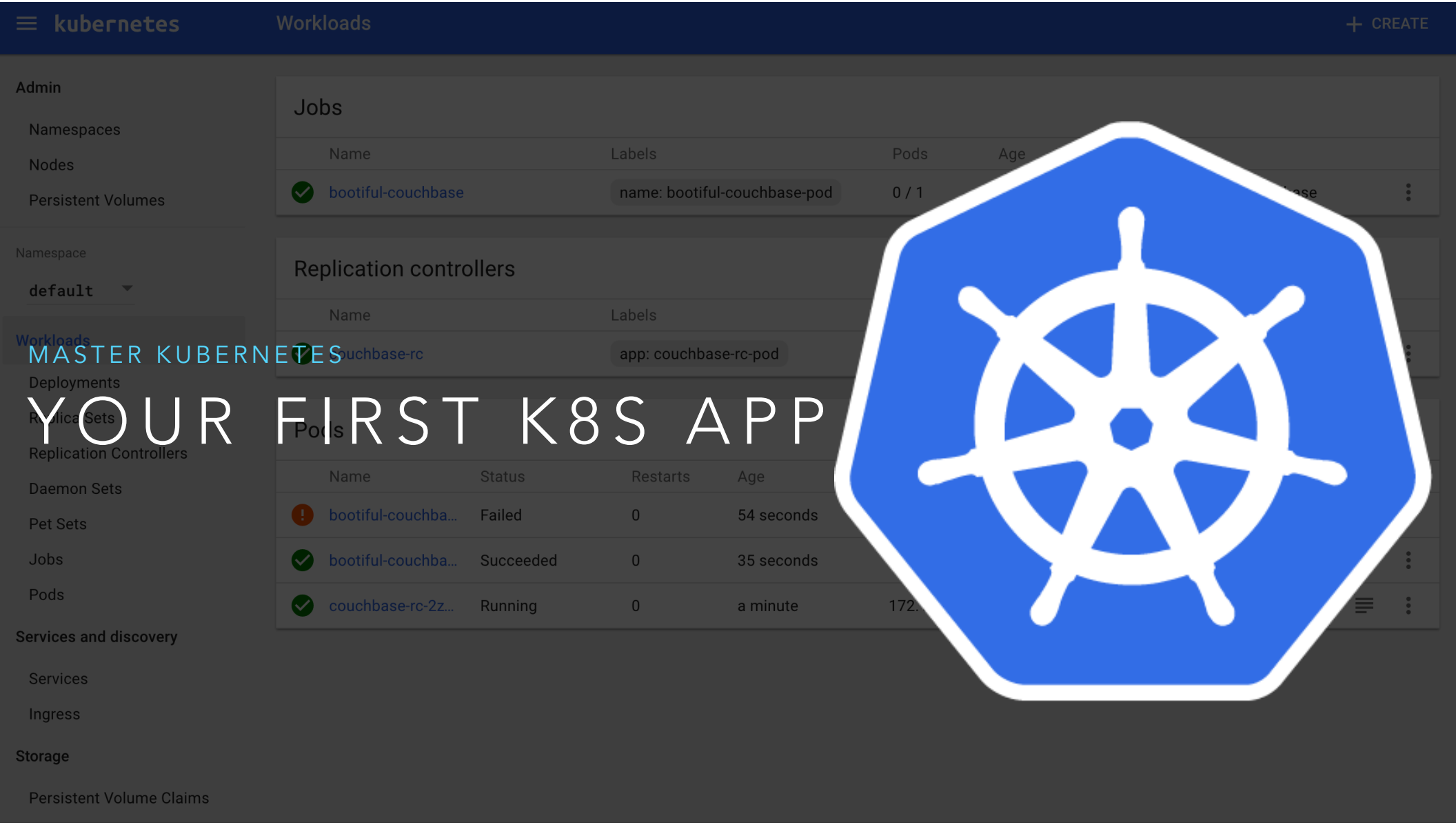
```
$ kubectl delete deployment hello-minikube
```

- Stop minikube

```
$ minikube stop
```

MASTER KUBERNETES

YOUR FIRST K8S APP



WHAT DOES A K8S "APP" LOOK LIKE?

- "Deployments" are the central metaphor for what we'd consider "apps" or "services"
- Deployments are described as a collection of resources and references
- Deployments take many forms based on the type of services being deployed
- Typically described in YAML format

PRACTICAL: A TOMCAT DEPLOYMENT

- We'll deploy the Tomcat App Server using the official docker image
- Key Tasks:
  - Define the deployment
  - Expose its services
  - Deploy it to our cluster

PRACTICAL: A TOMCAT DEPLOYMENT

- First, let's define the deployment
- The most simple deployment in Kubernetes is a single pod
- A Pod is an instance of a container, Deployments can have any number of Pods required to get the job done
- The most simple deployment simply has one pod - no redundancy, no separation of services, just one pod deployed to Kubernetes

PRACTICAL: A TOMCAT DEPLOYMENT

- Let's start with a text editor open to a blank *deployment.yaml* file to define our deployment so we can send it off to our Kubernetes cluster

PRACTICAL: A TOMCAT DEPLOYMENT

deployment.yaml

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  selector:
    matchLabels:
      app: tomcat
  replicas: 1
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat:9.0
        ports:
        - containerPort: 8080
```

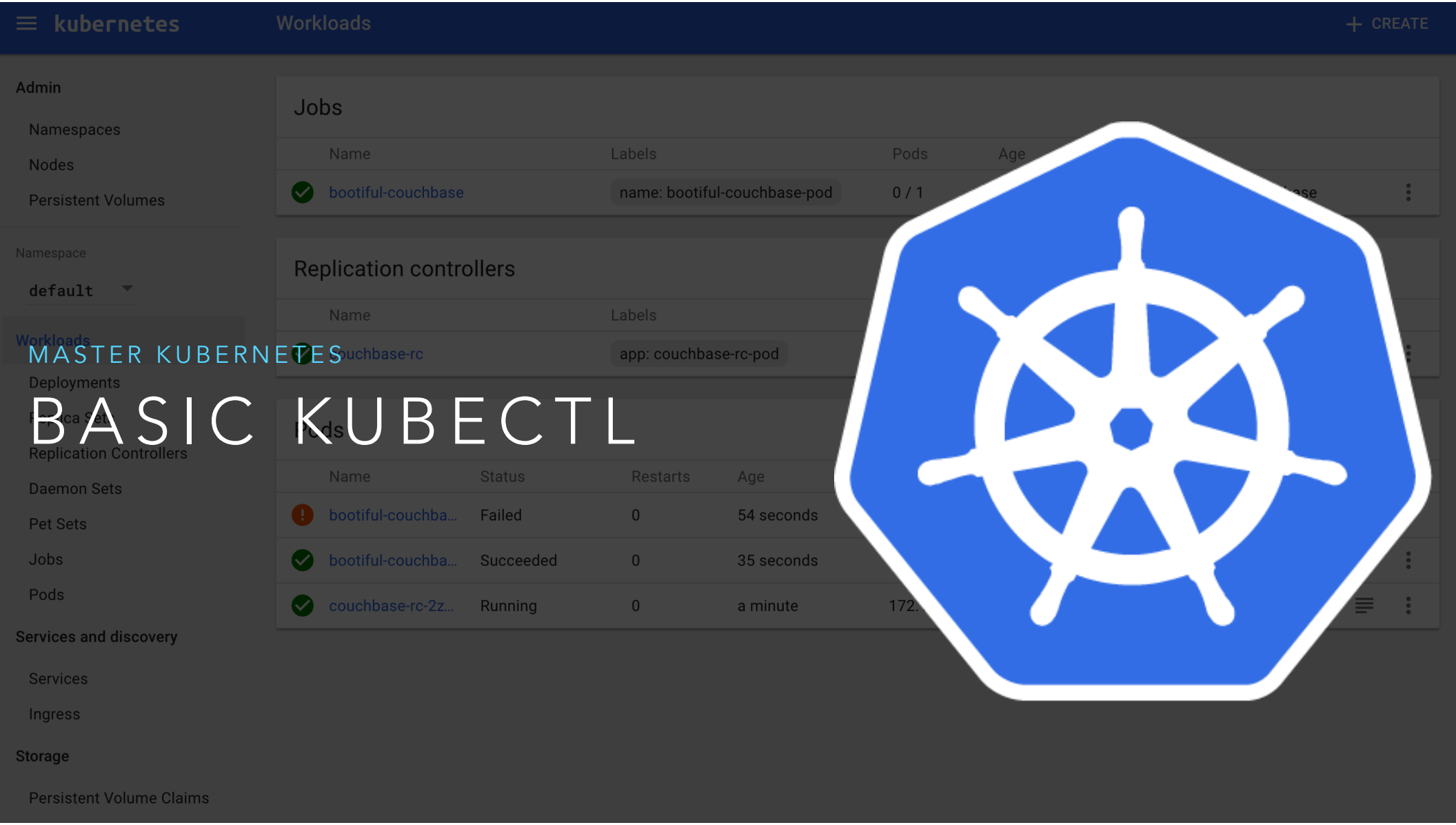
PRACTICAL: A TOMCAT DEPLOYMENT

- Expose the pod to the external world as a service

```
$ kubectl expose deployment tomcat-deployment --type=NodePort
```

MASTER KUBERNETES

BASIC KUBECTL



KUBECTL

- kubectl provides access to nearly every Kubernetes
- Primary command line access tool

KUBECTL GET PODS

```
$ kubectl get pods
```

- Lists all pods in all namespaces
- Provides the pod name, how many instances of the pod are running & ready, its status, how many times they have restarted, and their age

Sample Output

```
root@vt-es-1:~# kubectl get pods
NAME                                  READY     STATUS    RESTARTS   AGE
tomcat-deployment-7bd7889564-7l7n2    1/1       Running   0          15m
root@vt-es-1:~#
```

KUBECTL DESCRIBE POD

```
$ kubectl get pods [pod name]
```

- Describes detailed information about all pods or a specified pod (optional pod name argument)

Sample Output

```
root@vt-es-1:~# kubectl describe pod tomcat-deployment-7bd7889564-717n2
Name:           tomcat-deployment-7bd7889564-717n2
Namespace:      default
Node:           vt-es-1/10.240.0.2
Start Time:     Tue, 07 Nov 2017 05:04:30 +0000
Labels:         app=tomcat
                pod-template-hash=3683445120
Annotations:    kubernetes.io/created-by={"kind":"SerializedReference","apiVersion":"v1","reference":{"kind":"ReplicaSet","namespace":"default","name":"tomcat-deployment-7bd7889564","uid":"22e6f547-c379-11e7-9196-420...
Status:         Running
IP:             172.17.0.4
Created By:     ReplicaSet/tomcat-deployment-7bd7889564
Controlled By:  ReplicaSet/tomcat-deployment-7bd7889564
Containers:
  tomcat:
    Container ID:   docker://97ed5ccb28deaa0ea780b8a99a10a648cdfdff185eb785113589c337481a7414
    Image:          tomcat:9.0
    Image ID:       docker-pullable://tomcat@sha256:50cc2d5cd7959cb893e553f9b25d6337d9549b0c2850e383beecd010156d4a84
    Port:           8080/TCP
    State:          Running
      Started:      Tue, 07 Nov 2017 05:04:31 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-ck2l7 (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
Volumes:
  default-token-ck2l7:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-ck2l7
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     <none>
Events:
  Type    Reason                 Age   From               Message
  ----    ------                 ----  ----               -------
  Normal  Scheduled              19m   default-scheduler  Successfully assigned tomcat-deployment-7bd7889564-717n2 to vt-es-1
  Normal  SuccessfulMountVolume  19m   kubelet, vt-es-1   MountVolume.SetUp succeeded for volume "default-token-ck2l7"
  Normal  Pulled                 19m   kubelet, vt-es-1   Container image "tomcat:9.0" already present on machine
  Normal  Created                19m   kubelet, vt-es-1   Created container
  Normal  Started                19m   kubelet, vt-es-1   Started container
root@vt-es-1:~# kubectl get pods
NAME                                   READY     STATUS    RESTARTS   AGE
tomcat-deployment-7bd7889564-717n2   1/1       Running   0          19m
root@vt-es-1:~#
```

KUBECTL EXPOSE PORT

```
$ kubectl expose <type name> <identifier/name> [—port=external port] [—target-port=container-port [—type=service-type]
```

- Exposes a port (TCP or UDP) for a given deployment, pod, or other resource

Sample Output

```
root@vt-es-1:~# kubectl expose deployment tomcat-deployment --type=NodePort
service "tomcat-deployment" exposed
```

KUBECTL PORT-FORWARD

```
$ kubectl port-forward <pod name> [LOCAL_PORT:]REMOTE_PORT]
```

- Forwards one or more local ports to a pod

Sample Output

```
root@vt-es-1:~# kubectl port-forward tomcat-deployment-7bd7889564-7l7n2 5000:6000
Forwarding from 127.0.0.1:5000 -> 6000
```

KUBECTL ATTACH

```
$ kubectl attach <pod name> -c <container>
```

- Attaches to a process that is already running inside an existing container

Sample Output

```
root@vt-es-1:~# kubectl attach tomcat-deployment-7bd7889564-7l7n2
If you don't see a command prompt, try pressing enter.
```

KUBECTL EXEC

```
$ kubectl exec  [-it] <pod name> [-c CONTAINER] — COMMAND [args…]
```

- Execute a command in a container
- -i option will pass stdin to the container
- -t option will specify stdin is a TTY

Sample Output

```
root@vt-es-1:~# kubectl exec -it tomcat-deployment-7bd7889564-7l7n2 bash
root@tomcat-deployment-7bd7889564-7l7n2:/usr/local/tomcat# whoami
root
root@tomcat-deployment-7bd7889564-7l7n2:/usr/local/tomcat#
```

KUBECTL LABEL PODS

```
$ kubectl label [—overwrite] <type> KEY_1=VAL_1 ….
```

- Updates the labels on a resource

Sample Output

```
root@vt-es-1:~# kubectl label pods tomcat-deployment-7bd7889564-7l7n2 healthy=false
pod "tomcat-deployment-7bd7889564-7l7n2" labeled
root@vt-es-1:~#
```

KUBECTL RUN

```
$ kubectl run <name> —image=image
```

- Run a particular image on the cluster

Sample Output

```
root@vt-es-1:~# kubectl run hazelcast --image=hazelcast --port=5701
deployment "hazelcast" created
root@vt-es-1:~#
```

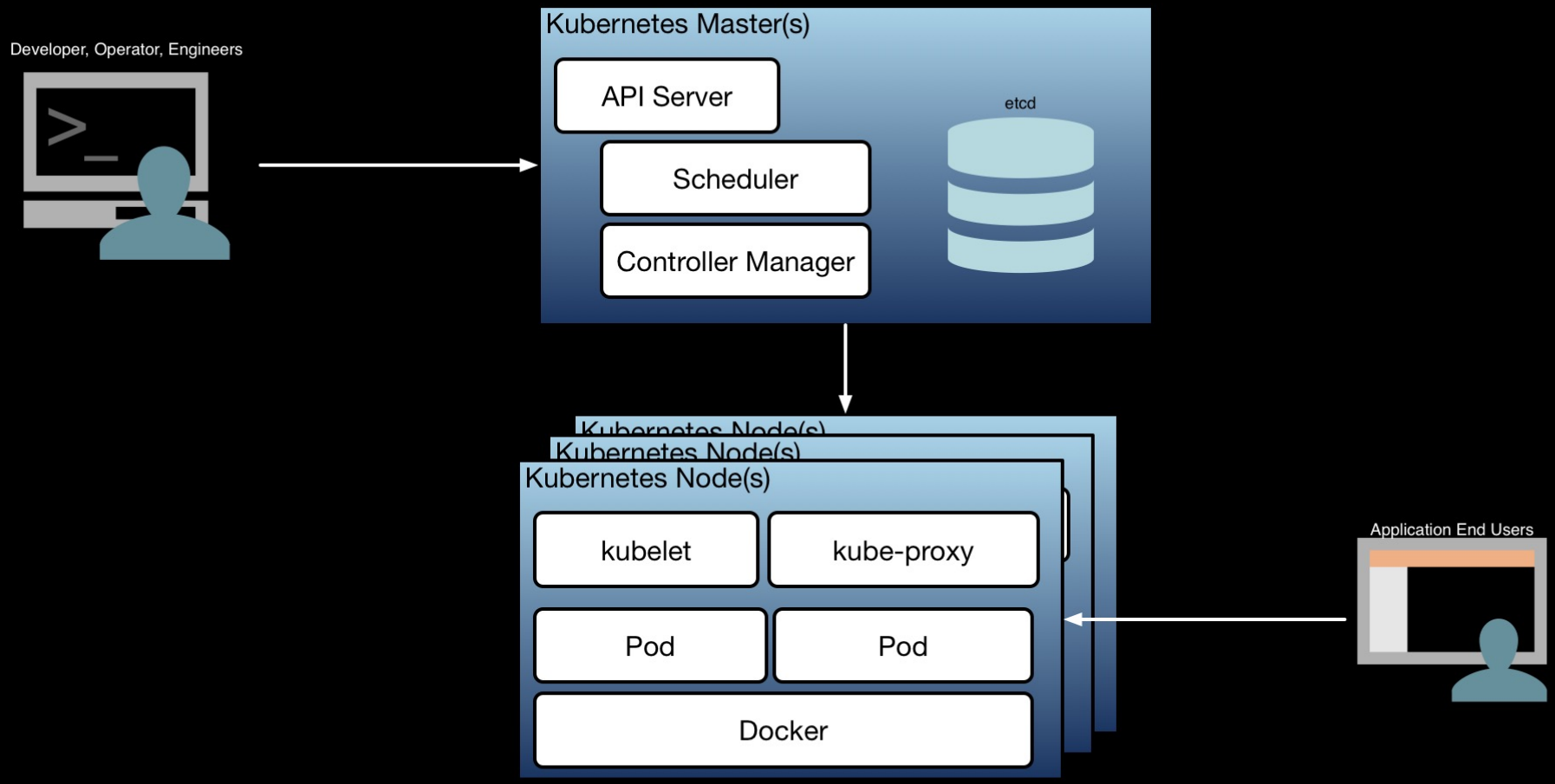
FULL KUBECTL REFERENCE & CHEAT SHEET

- kubectl reference: https://kubernetes.io/docs/user-guide/kubectl/v1.8/
- kubectl cheat sheet: https://kubernetes.io/docs/user-guide/kubectl-cheatsheet/

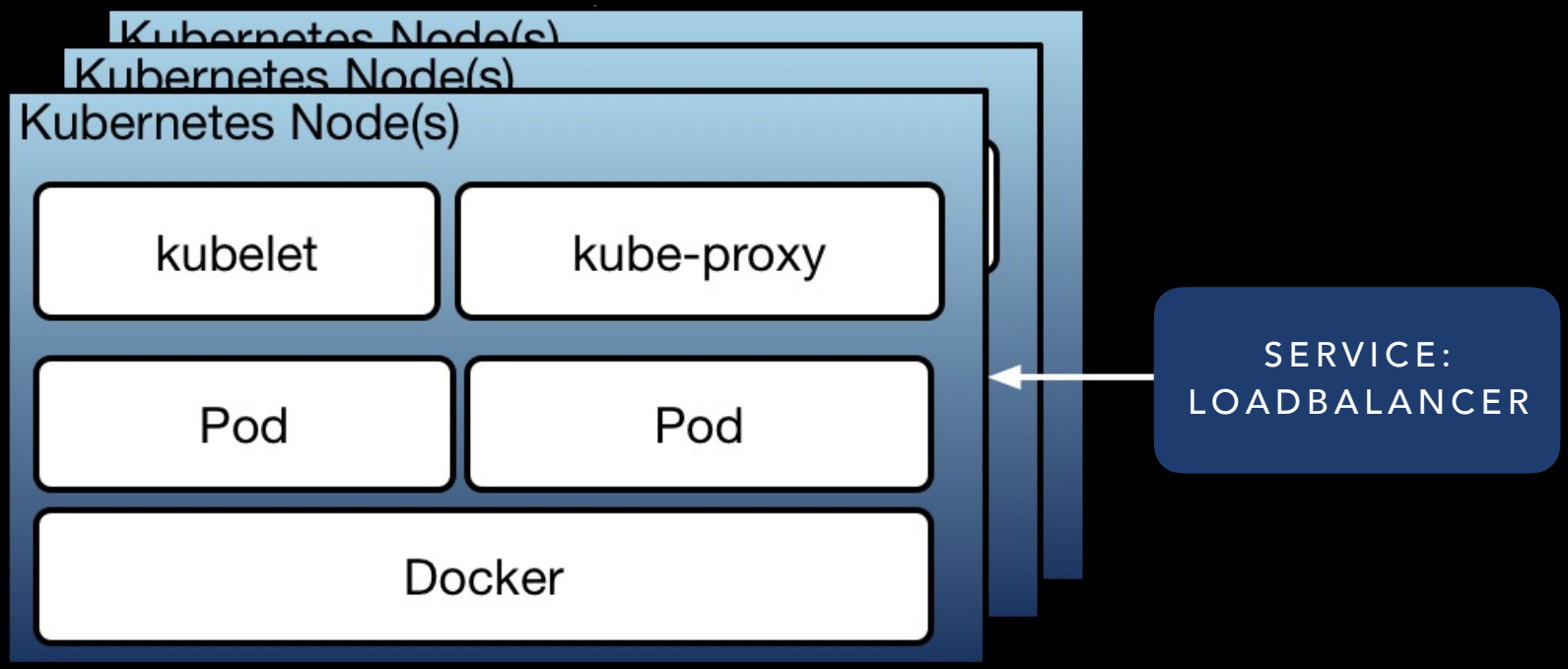
MASTER KUBERNETES

ARCHITECTURE OVERVIEW

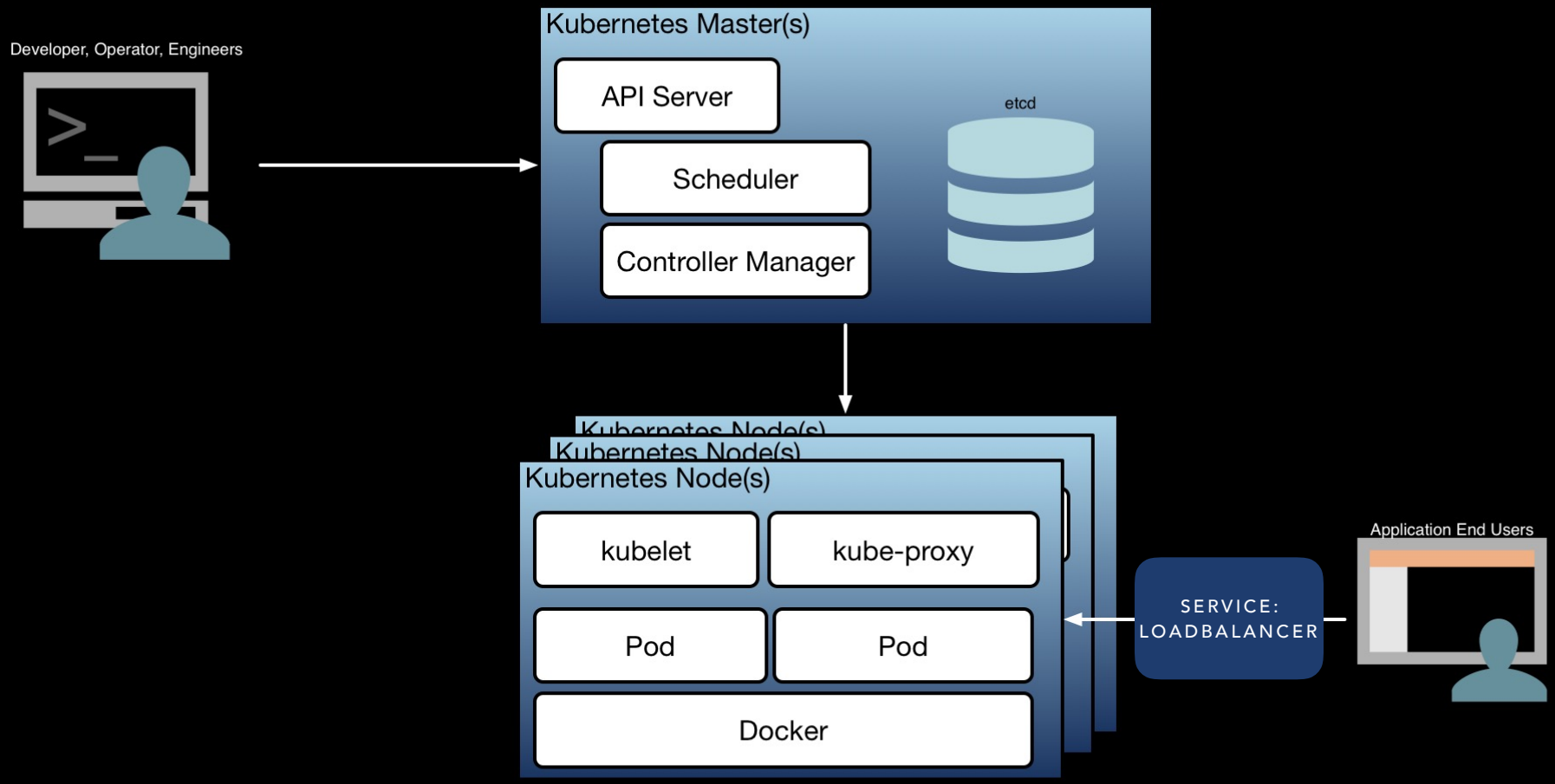
KUBERNETES FROM ABOVE (SIMPLIFIED)



A COMMON STARTING POINT: NODES



REVIEW


Developer, Operator, Engineers
Kubernetes Master(s)
API Server
Scheduler
Controller Manager
etcd
Kubernetes Node(s)
Kubernetes Node(s)
Kubernetes Node(s)
kubelet
kube-proxy
Pod
Pod
Docker
SERVICE:
LOADBALANCER
Application End Users

MASTER KUBERNETES

SCALING & REPLICATION

SCALING APPLICATIONS

- Stateful vs Stateless Applications
- How state is handled can either enable or interfere with scaling, it most certainly dictates how you'll scale

REPLICAS

- Kubernetes allows you to define replicas when you deploy an application - you have a few options
  - Setting “replica” in your Deployment (recommended)
  - Defining a ReplicaSet
  - Bare Pods
  - Job
  - DaemonSet

INTRODUCTION TO REPLICAS

- Kubernetes allows you to define replicas when you deploy an application - you have a few options
  - Setting replicas configuration in your Deployment (recommended)
  - Defining a ReplicaSet
  - Bare Pods
  - Job
  - DaemonSet

SCALING - A PRACTICAL EXAMPLE

deployment.yaml

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  selector:
    matchLabels:
      app: tomcat
  replicas: 1
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat:9.0
        ports:
        - containerPort: 8080
```

SCALING - A PRACTICAL EXAMPLE

deployment.yaml

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  selector:
    matchLabels:
      app: tomcat
  replicas: 4
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat:9.0
        ports:
        - containerPort: 8080
```

SCALING - A PRACTICAL EXAMPLE

```
$ kubectl scale —replicas=4 deployment/tomcat-deployment
```

UPDATING THE SERVICE

- Previously, we defined a "NodePort" service for the Tomcat pod

```
$ kubectl expose deployment tomcat-deployment --type=NodePort
```

UPDATING THE SERVICE

- Previously, we defined a "NodePort" service for the Tomcat pod

```
$ kubectl expose deployment tomcat-deployment --type=NodePort
```

- Now, let's define a LoadBalancer service, instead

```
$ kubectl expose deployment tomcat-deployment —type=LoadBalancer —
port=8080 —target-port=8080 —name tomcat-load-balancer
```

- Finally, let's see what IP address was assigned for the service

```
$ kubectl describe services tomcat-load-balancer
```

SEEING IT IN ACTION

- Open a web browser to the IP from the previous command on port 8080

```
$ kubectl describe services tomcat-load-balancer
```

MASTER KUBERNETES

DEPLOYMENTS

WHAT ARE DEPLOYMENTS?

- Deployments are a high-level object in Kubernetes, they define a desired state of an application
  - They consist of Pods
  - Optionally, they can also include ReplicaSets (that are automatically managed by the Deployment based on its replica configuration)

WITH DEPLOYMENT OBJECTS YOU CAN:

- Create a new deployment
- Update an existing deployment
- Apply rolling updates to Pods running on your cluster
- Rollback to a previous version
- Pause & Resume a deployment

WORKING WITH DEPLOYMENTS

- kubectl is your gateway to working with deployments
- Useful Commands & What They Do:
  - List deployments

    ```
    $ kubectl get deployments
    ```

  - View status of deployment roll outs

    ```
    $ kubectl rollout status
    ```

  - Set the image of a deployment

    ```
    $ kubectl set image
    ```

  - View the history of a rollout, including previous revisions

    ```
    $ kubectl rollout history
    ```

LISTING DEPLOYMENTS

```
$ kubectl get deployments
```

VIEWING ROLLOUT STATUS

```
$ kubectl rollout status
```

SETTING/UPDATING THE IMAGE

```
$ kubectl set image
```

VIEWING ROLLOUT HISTORY

```
$ kubectl rollout history
```

MAKING IT REAL

deployment.yaml

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  selector:
    matchLabels:
      app: tomcat
  replicas: 4
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat:9.0
        ports:
        - containerPort: 8080
      nodeSelector:
        storageType: ssd
```

MASTER KUBERNETES

LABELS & SELECTORS

WHAT ARE LABELS & SELECTORS?

- A method to keep things organized, and to help you (a human) and Kubernetes (a machine) identify resources to act upon
- Labels are key/value pairs that you can attach to objects like pods
  - They are for users to help describe meaningful and relevant information about an object
  - They do not affect the semantics of the core system
- Selectors are a way of expressing how to select objects based on their labels
  - Selectors are a simple language to define what labels match and which ones done
  - You can specify if a label equals a given criteria or if it fits inside a set of criteria
    - *Equality-based*
    - *Set-based*

MAKING IT REAL

- You can label nearly anything in the Kubernetes world
  - Deployments
  - Services
  - Nodes
- Let's use labels to label a node that it has SSD storage and then use a selector to tell the deployment that our app should only ever go onto a node with SSD storage

MAKING IT REAL

- For this example we're going to use "nodeSelector"
- nodeSelector is a property on a deployment that uses labels and selectors to choose which nodes the master decides to run a given pod on
- To accomplish our goal of running our deployment only on nodes with SSD we will:
  - Label a node as having an SSD
  - Define the nodeSelector on our deployment to match only nodes having the label we just defined

MASTER KUBERNETES

HEALTH CHECKS

HEALTH CHECKS

- Kubernetes has two types of health checks to ascertain two different things
  - *Readiness Probes*: To determine when a Pod is "ready" (e.g. after it has started to see when it's ready and has loaded what it needs to internally in the image and is ready to take requests from external services)
  - *Liveness Probes:* To determine when a Pod is "healthy" or "unhealthy" after it has become ready

HEALTH CHECKS

- Whether they are a readiness or liveness probe they can use a variety of methods to ascertain a container's status
  - Successful HTTP or TCP request to the Pod
  - Successful command execution on the Pod
- Probes are defined on the container in a Deployment or Pod specification

MAKING IT REAL: A HEALTHY TOMCAT

- A *Readiness Probe* will check to make sure the Pod has started and is ready to begin taking requests
- A *Liveness Probe* on the Tomcat deployment will help us ensure the containers continue to be able to accept and service requests without error in a reasonable amount of time

MAKING IT REAL: A HEALTHY TOMCAT

deployment.yaml

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  selector:
    matchLabels:
      app: tomcat
  replicas: 4
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat:9.0
        ports:
        - containerPort: 8080
        livenessProbe:
          httpGet:
            path: /
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 30
        readinessProbe:
          httpGet:
            path: /
            port: 8080
          initialDelaySeconds: 15
          periodSeconds: 3
```

MASTER KUBERNETES

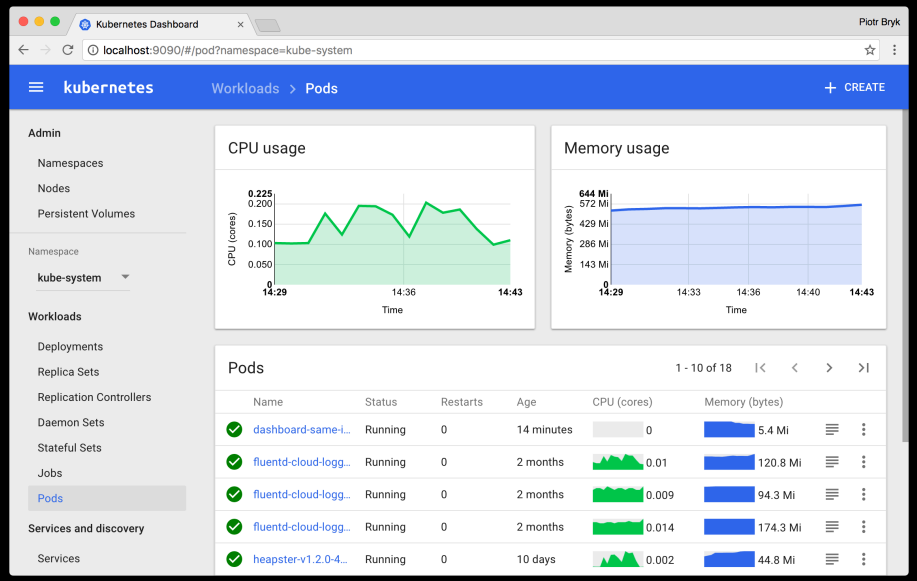
WEB INTERFACE

KUBERNETES WEB UI

- kubectl is great, but sometimes a web UI is useful, too
  - Called the "Dashboard UI"
- Runs on your Kubernetes master(s)
- Accessible directly if you have direct network connectivity directly to your cluster/master(s) (unlikely in production situations)
  - kubectl can create a proxy/tunnel for you in situations you do not

```
$ kubectl proxy
```

KUBERNETES WEB UI



- Provides a variety of views for nearly anything in your Kubernetes cluster
- Allows update, deletion, and creation of nearly anything in your Kubernetes cluster
- Accesses the same APIs as kubectl

USING THE WEB UI

- On some Kubernetes clusters the Dashboard UI is pre-installed, on some it is not (many cloud providers' Kubernetes services include it)
- Installing the Dashboard

```
$ kubectl create -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml
```

- Accessing the dashboard (the most fool-proof way)

```
$ kubectl proxy
```

- Navigate to "http://localhost:8001/ui" in your web browser

TOURING THE DASHBOARD

Kubernetes Dashboard

Piotr Bryk

localhost:9090/#/pod?namespace=kube-system

kubernetes

Workloads > Pods

+ CREATE

Admin

Namespaces

Nodes

Persistent Volumes

Namespace

kube-system

Workloads

Deployments

Replica Sets

Replication Controllers

Daemon Sets

Stateful Sets

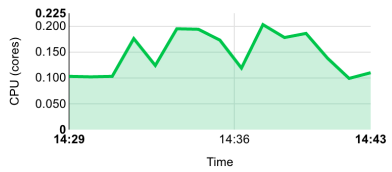
Jobs

Pods

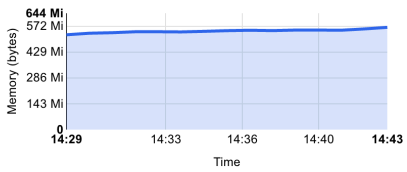
Services and discovery

Services

CPU usage



Memory usage



Pods

1 - 10 of 18

Name	Status	Restarts	Age	CPU (cores)	Memory (bytes)
dashboard-same-i...	Running	0	14 minutes	0	5.4 Mi
fluentd-cloud-logg...	Running	0	2 months	0.01	120.8 Mi
fluentd-cloud-logg...	Running	0	2 months	0.009	94.3 Mi
fluentd-cloud-logg...	Running	0	2 months	0.014	174.3 Mi
heapster-v1.2.0-4...	Running	0	10 days	0.002	44.8 Mi

MASTER KUBERNETES

EXERCISE: DEPLOYING & SCALING

WHAT YOU'VE LEARNED

- How to define a deployment in Kubernetes
- How to deploy this deployment to Kubernetes
- How to scale this deployment in Kubernetes

YOUR TASK

- Use what you've learned to deploy and scale MongoDB (a popular NoSQL datastore)
- Success criteria:
  - The current version of MongoDB is running on your Kubernetes cluster with four replicas
- A few hints:
  - MongoDB listens on port 27017
  - Officially supported MongoDB Docker images are available on Docker Hub

YOUR TURN!

- Come back when you're done and see if your solution matched ours (it's OK if it doesn't: there is more than one way to complete this!)

A POSSIBLE SOLUTION

- Use kubectl run to run the default mongo image

```
$ kubectl run mongo-exercise-1 —image=mongo —port=27017
```

- Use kubectl scale to scale the deployment to 4 replicas

```
$ kubectl scale —replicas=4 deployment/mongo-exercise-1
```

OTHER POSSIBLE SOLUTIONS

- Write a deployment.yaml file, use kubectl apply, & use kubectl expose to expose a service
- Write a deployment.yaml file & a service.yaml file and use kubectl apply on both
- Use a Kubernetes package manager like helm to handle the work for you

MASTER KUBERNETES

DNS & SERVICE DISCOVERY

DNS IN KUBERNETES

- DNS, or the Domain Name Service, is a cornerstone service of the Internet
- It translates names into IP addresses
- Kubernetes has a built-in DNS service that is launched (and configured) automatically
- Kubernetes configures kubelets to tell individual containers to use the DNS service's IP to resolve DNS names

DNS AS SERVICE DISCOVERY IN KUBERNETES

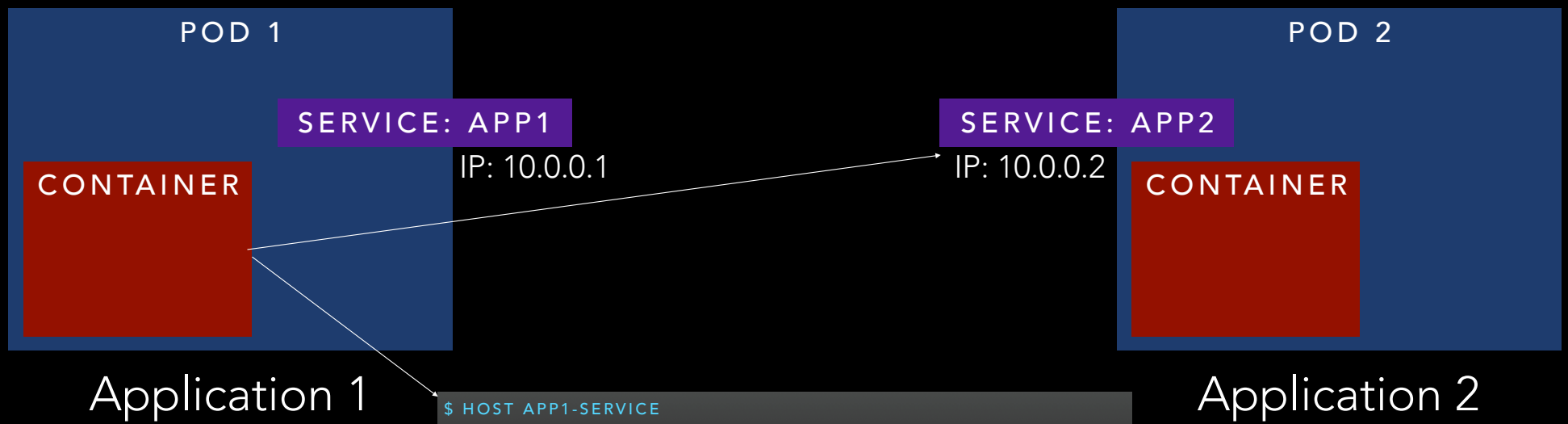
- Every Service in your Kubernetes cluster gets a DNS name
- Kubernetes has a specific & consistent nomenclature for deciding what this DNS name is

```
<my-service-name>.<my-namespace>.svc.cluster.local
```

General From of DNS Name for Non-Headless Services

- But, you can use this well-known pattern to design loosely-coupled services or at least set "sane-defaults"
  - Use could use environment variables to set the DNS name of other services your app may need
  - Or you could expect services at set DNS names
  - Or, combine the two above and expect services at well-known DNS names and allow environment variables to override ("convention over configuration")

HOW APP1 COULD REACH APP2 USING DNS



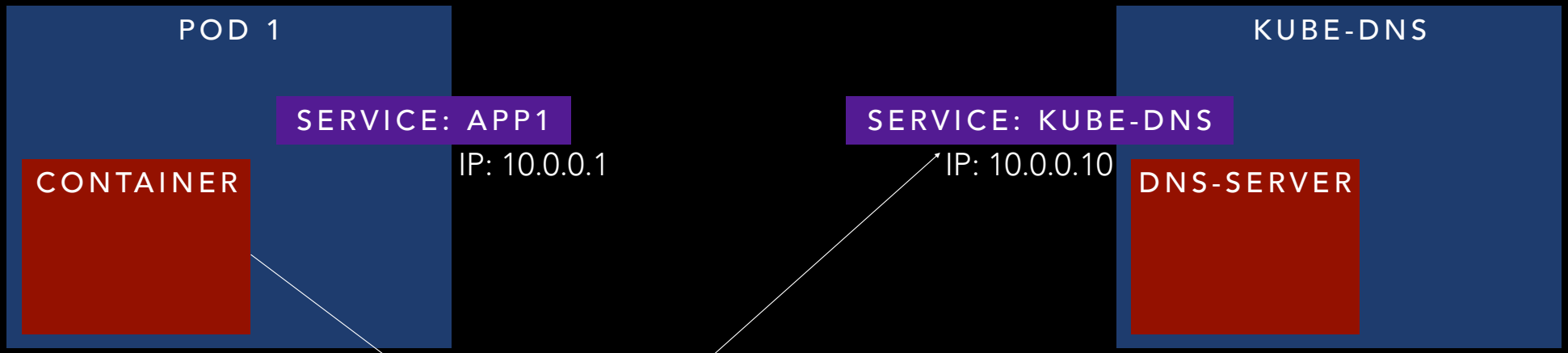
Application 1

Application 2

```
$ HOST APP1-SERVICE
APP1-SERVICE HAS ADDRESS 10.0.0.1
$ HOST APP2-SERVICE
APP2-SERVICE HAS ADDRESS 10.0.0.2
$ HOST APP2-SERVICE.DEFAULT
APP2-SERVICE.DEFAULT HAS ADDRESS 10.0.0.2
$ HOST APP2-SERVICE.DEFAULT.SVC.CLUSTER.LOCAL
APP2-SERVICE.DEFAULT.SVC.CLUSTER.LOCAL HAS ADDRESS 10.0.0.2
```

"DEFAULT" STANDS FOR THE DEFAULT NAMESPACE PODS & SERVICES CAN BE LAUNCHED IN DIFFERENT NAMESPACES TO LOGICALLY SEPARATE YOUR CLUSTER

DNS - HOW DOES IT WORK?



Application 1

```
$ KUBECTL RUN -I -TTY BUSY BOX —IMAGE=BUSYBOX —RESTART=NEVER SH

# CAT /ETC/RESOLV.CONF
SEARCH DEFAULT.SVC.CLUSTER.LOCAL.SVC.LOCAL CLUSTER.LOCAL LAN
NAMESERVER 10.0.0.10
```

kube-dns

(provided by k8s)

A NOTE ON NAMESPACES

- We'll discuss namespaces in depth soon in their own lecture, but for now, just know the following
- Kubernetes allows you to define "namespaces" - they allow you to separate your cluster into smaller logical clusters
  - By default, everything you do is in the "default" namespace (and everything we've done so far has been)
  - Kubernetes DNS names will include this namespace in the assigned DNS name
- Namespaces are helpful mostly for large clusters with many users across many teams & projects - using the default namespace is fine if this logical separation is not required

A PRACTICAL EXAMPLE

- Setting up a blog using a system called WordPress (a very popular blog/website management system)
- WordPress uses a database called MySQL to store its information to host the website, blog posts, and other content the website may have

A PRACTICAL EXAMPLE

- Consider deploying WordPress & MySQL to a Kubernetes cluster
- WordPress & MySQL are both separate deployments
  - However, to function Wordpress needs MySQL
  - Specifically, it must know a TCP/IP hostname at which where to connect to MySQL
- Let's use DNS to create a WordPress deployment that expects MySQL to be running on the "wordpress-mysql" hostname in the same namespace
  - We will allow this value to be overridden by environmental variable as a "best practice"
  - This follows the "convention over configuration" best practice

MYSQL

- We'll deploy MySQL in a deployment named "wordpress-mysql"
- Once deployed, it will run MySQL 5.6 on port 3306 on the wordpress-mysql hostname

WORDPRESS

- We'll deploy WordPress named as "wordpress"
- We'll configure MySQL to be available on the "wordpress-mysql" hostname and tell Wordpress to expect it there

PROCESS

1. Deploy MySQL
2. Deploy WordPress

BEFORE WE BEGIN

- In previous examples, the deployment.yaml files were small enough to fit on screen - they're a bit larger now and we'll review them during the demo in a text editor
- They're available in the GitHub repository for this lecture series in the "Advanced Kubernetes Usage/DNS and Service Discovery" directory

PROCESS

1. **Deploy MySQL**
2. Deploy WordPress

PROCESS

1. Deploy MySQL
2. **Deploy WordPress**

MASTER KUBERNETES

VOLUMES

LIKE DISKS, BUT WITH A BIT MORE

- Volumes can be considered just a directory, with some data, which containers in a pod can access
- Kubernetes supports multiple types of volumes that take care of how that data is stored, persisted, and made available
  - Support for a variety of cloud providers' block store products
  - Support for SAN-type hardware, file systems, etc
  - Support for local volumes (for testing/minikube only! Not production)
- Certain types of Volumes can also provide sharing of files between Pods by being mounted to multiple Pods simultaneously

TYPES OF VOLUMES

- An incomplete, non-comprehensive list
- Cloud Provider
  - Azure Disk & Azure File
  - AWS EBS
  - Google Compute Engine Persistent Disk
- SAN/File System/Hardware
  - CephFS
  - Fibre Channel
  - GlusterFS
  - NFS
  - iSCSI
  - Local *(For development/minikube only - not supported in production)*

USING VOLUMES

- Pods can specify what volumes they need and where to mount them
  - Using the spec.volumes field (what volumes they need)
  - Using the spec.containers.volumeMounts field (where to mount them)
- Processes in the container then see a filesystem view of the data in that Volume
- Using Volumes lets us separate stateless portions of our application (the code) from stateful data
  - The infrastructure can be scaled, maintained, and live separately from the data it works on/with
  - Also may ease portability, backup, recovery, and other management tasks in well-architected systems

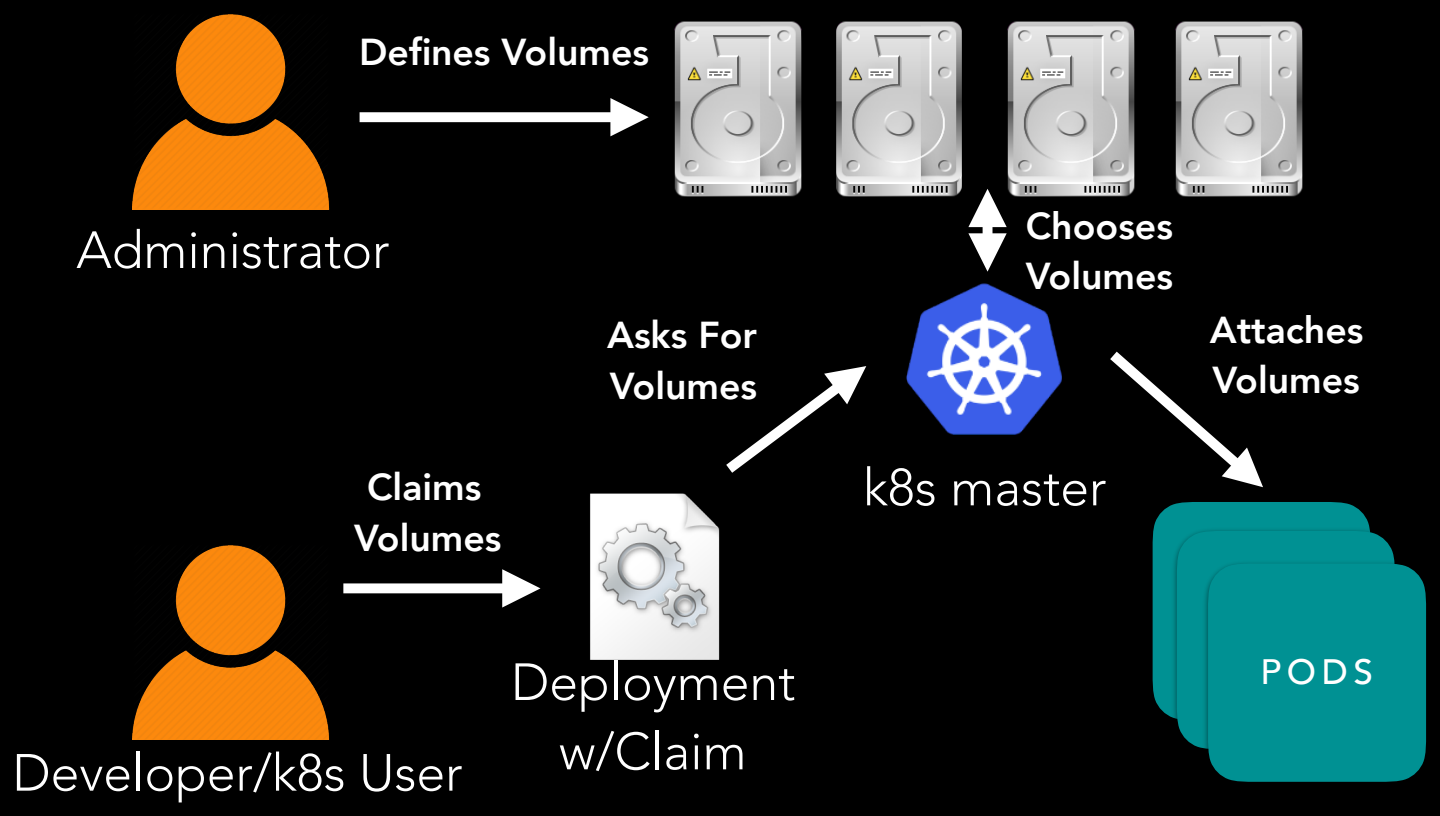
USING VOLUMES

- Pods can specify what volumes they need and where to mount them
  - Using the spec.volumes field (what volumes they need)
  - Using the spec.containers.volumeMounts field (where to mount them)
- Processes in the container then see a filesystem view of the data in that Volume
- Using Volumes lets us separate stateless portions of our application (the code) from stateful data
  - The infrastructure can be scaled, maintained, and live separately from the data it works on/with
  - Also may ease portability, backup, recovery, and other management tasks in well-architected systems

USING PERSISTENTVOLUMES

- PersistentVolumes are a Volumes designed to provide persistent disk-like functionality
- Using them involves:
  - Provisioning a PersistentVolume (akin to creating/installing a disk in a virtual machine or hardware server)
  - Establishing a PersistentVolumeClaim (it is a request for storage by a user/Pod)
- By examining available PersistentVolumes and demands from PersistentVolumeClaims by running Pods, Kubernetes binds available volumes to Pods based on the options specified by the PersistentVolumeClaims to matching PersistentVolumes (e.g. by name, storage size, storage class, etc asked for in the Claims)

HOW CLAIMS & VOLUMES INTERACT



CREATING A VOLUME

- PersistentVolumes are defined using a "PersistentVolume" definition that specifies their type, size, and how they can be accessed
- Their type and access type is highly dependent on the underlying media
  - Local Disk
  - Network Mount
  - Cloud Block Storage Service
  - Directory on the Host (testing only, not for production!)

task-pv-volume.yaml

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

CLAIMING A VOLUME

- Pods use PersistentVolumeClaims to request physical storage defined by PersistentVolumes
- Kubernetes uses the Claim to look for a PersistentVolume that satisfies the claim's requirements
  - If it finds a match, it binds the claim to the volume
  - If it cannot find a match, it results in an error

task-pv-claim.yaml

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

PRACTICAL PROOF

- Let's make a few changes to our Wordpress & MySQL deployments so they have persistent volumes attached to them
- Specifically, Wordpress & MySQL use specific directories to store user data that we'll need to attach to PersistentVolumes so the data is saved between container restarts
  - Wordpress uses the directory **/var/www/html** to store static documents like web pages, logos, art, or other file-like assets you may upload and are a part of your website that don't belong in a database like MySQL (Wordpress stores configuration, posts, and text-like content in MySQL while this directory is used for static file-like content)
  - MySQL uses the directory **/var/lib/mysql** to store files that contain the user data of data entered into the database

PRACTICAL PROOF

- Then let's see how they make persistent data a reality across something like recreating a Pod, where otherwise data would have been lost
- We'll build on our work from the previous lecture by:
  - Setting up WordPress from the vanilla install
  - Creating a Post
  - Deleting the Deployment
  - Recreating the Deployment
  - Seeing if our data persisted!

MASTER KUBERNETES

SECRETS

SECRETS

- Secrets contain small amounts of data
- Secrets can be delivered to a pod in the form of:
  - A file placed on a volume at runtime containing the secret data (useful for certificates)
  - An environment variable referenced by the Pod and inserted at runtime into the environment by the kubelet running the Pod - just like any other environment variable

SECRETS

- Secrets contain small amounts of data
- Secrets can be delivered to a pod in the form of:
  - A file placed on a volume at runtime containing the secret data (useful for certificates)
  - An environment variable referenced by the Pod and inserted at runtime into the environment by the kubelet running the Pod - just like any other environment variable

HOW SECRETS ARE STORED

- Kubernetes provides separation for secrets, it does not provide strong encryption
- Secrets are typically Base64 encoded strings stored separately from configuration and injected at runtime
- You can encode it manually or use Kubernetes' tools to do it for you

A NOTE ON BASE64

- Base64 encoding is a method that stores binary data in a "safe" string by translating it using a "radix-64" representation
  - Helps store small pieces of text or non-text data that would otherwise be difficult to handle in systems or data structures that expect text
  - Small files, images, or even text can be Base64 encoded - the resulting string is typically "safe" in the sense it does not contain characters many popular systems/protocols such as HTTP or the like do not allow as parameters
- The string "Kubernetes" is encoded in Base64 as "S3ViZXJuZXRlcw=="

HOW SECRETS ARE STRUCTURED

- Secrets are key:value pairs, both the key and the value are arbitrary strings

CREATING A SECRET

- You guessed it: kubectl will help you here!
- Creating a secret using kubectl
  - From a file (creates a secret named generic db-user-pass with the username from the username.txt file and password from the password.txt file)

```
$ kubectl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.txt
```

  - From the a literal on the command line

```
$ kubectl create secret generic mysql-pass --from-literal=password=YOUR_PASSWORD
```

USING A SECRET - AS AN ENVIRONMENT VARIABLE

```yaml
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
```

USING A SECRET - AS A FILE

```
<Truncated for length>
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
        - name: secret-storage
          mountPath: /etc/secretStore
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
      - name: secret-storage
        secret:
          secretName: mysql-pass
```

A PRACTICAL EXAMPLE

- Let's remedy a weak-point in our WordPress & MySQL stack: a hard-coded plain-text password
- Let's replace the plain-text value in the deployment with a "secret" in both the MySQL deployment & the WordPress deployment

A PRACTICAL EXAMPLE

- We'll replace the existing environment variable in plain text with an environment variable to reference a secret

```yaml
      tier: mysql
  spec:
    containers:
    - image: mysql:5.6
      name: mysql
      env:
      - name: MYSQL_ROOT_PASSWORD
        value: PASSWORDS_IN_PLAIN_TEXT_ARE_BAD_WE_WILL_SHOW_SOMETHING_MORE_SECURE_LATER
      ports:
      - containerPort: 3306
        name: mysql
```

Current "mysql-deployment.yaml"

```yaml
  spec:
    containers:
    - image: mysql:5.6
      name: mysql
      env:
      - name: MYSQL_ROOT_PASSWORD
        valueFrom:
          secretKeyRef:
            name: mysql-pass
            key: password
```

New "mysql-deployment.yaml"

A PRACTICAL EXAMPLE

- We'll do the same for WordPress, and we'll reference the same secret - one secret, in one place, referenced by both deployments - makes password management easy, too!

```yaml
spec:
  containers:
  - image: wordpress:4.8-apache
    name: wordpress
    env:
    - name: WORDPRESS_DB_HOST
      value: wordpress-mysql
    - name: WORDPRESS_DB_PASSWORD
      value: PASSWORDS_IN_PLAIN_TEXT_ARE_BAD_WE_WILL_SHOW_SOMETHING_MORE_SECURE_LATER
```

Current "wordpress-deployment.yaml"

```yaml
spec:
  containers:
  - image: wordpress:4.8-apache
    name: wordpress
    env:
    - name: WORDPRESS_DB_HOST
      value: wordpress-mysql
    - name: WORDPRESS_DB_PASSWORD
      valueFrom:
        secretKeyRef:
          name: mysql-pass
          key: password
```

New "wordpress-deployment.yaml"

A PRACTICAL EXAMPLE

1. We'll create the secret using kubectl command line
2. We'll apply the modified deployments for both MySQL and WordPress that have the new environment variable declaration referring to our secret for the value instead of including it inline

MASTER KUBERNETES

USAGE & RESOURCE MONITORING

MANY SOLUTIONS

- Usage & resource monitoring is a large field
- Many free & paid packages and services both for Kubernetes and beyond
- We'll explore k8s' "native" stack with a few open source additions
  - Heapster
  - InfluxDB
  - Grafana

HEAPSTER

- Heapster is Kubernetes' Container Cluster Monitoring solution
- Heapster is maintained by the CNCF

ENABLING HEAPSTER ON MINIKUBE

- Heapster comes enabled on most cloud-based platforms
- On Minikube, we'll have to enable the add on

```
$ minikube addons enable heapster
```

- Grab a snack or a beer, it may take a few minutes or longer - check influx and Heapster pod status with

```
$ kubectl get pods --namespace=kube-system
```

- When pods are up access the Grafana Dashboard

```
$ minikube addons open heapster
```

MASTER KUBERNETES

NAMESPACES & RESOURCE QUOTAS

WHAT ARE NAMESPACES? WHY USE THEM?

- Namespaces create multiple virtual clusters on the same physical clusters, these virtual clusters are called namespaces
- Namespaces provide separation, when you start to need them, start using them
- Until then, using “default” is just fine

RESOURCE LIMITS IN NAMESPACES

- Namespaces can be assigned ResourceQuota objects
- Each namespace should have at most one (but it is not required)
- This will limit the amount of usage allowed by the objects in that namespace
- You can limit:
  - Compute
  - Storage
  - Memory
  - How many objects can exist

RESOURCE LIMITS IN NAMESPACES

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    pods: "4"
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

USEFUL COMMANDS FOR NAMESPACES

- Similar to Kubernetes commands for other types of objects
- Use kubectl
  - Create a namespace

```
$ kubectl create namespace <namespace name>
```

  - List namespaces

```
$ kubectl get namespace
```

PRACTICAL EXAMPLE

- Let's examine a deployment that requests a certain amount of CPU (200m per container with 3 replicas, for a total of 600m of CPU) in a namespace that only has 400m of CPU allocated using resource limits
- Let's examine how Kubernetes handles this
- Our Steps Will Include
  1. Create a namespace
  2. Assign the namespace a 400m CPU resource limit
  3. Deploy Tomcat (as we have in the past) into this new namespace but this time with 3 replicas requesting 200m CPU each
  4. Use kubectl to examine deployment status

PRACTICAL EXAMPLE - CREATE NAMESPACE

```
$ kubectl create namespace cpu-limited-tomcat
```

PRACTICAL EXAMPLE - ASSIGN LIMIT

```
$ kubectl create -f ./cpu-limits.yaml —namespace=cpu-limited-tomcat
```

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    limits.cpu: "400m"
```

cpu-limits.yaml

PRACTICAL EXAMPLE - DEPLOY TOMCAT W/ RESOURCE REQUESTS & REPLICAS

```
$ kubectl apply -f ./tomcat-deployment.yaml —namespace=cpu-limited-tomcat
```

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  selector:
    matchLabels:
      app: tomcat
  replicas: 3
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat:9.0
        ports:
        - containerPort: 8080
        resources:
          requests:
            cpu: "200m"
```

Tomcat-deployment.yaml

PRACTICAL EXAMPLE - SEE DEPLOYMENT STATUS

```
$ kubectl describe deployment tomcat-deployment —namespace=cpu-limited-tomcat
```

MASTER KUBERNETES

AUTO-SCALING

WHAT IS THE HORIZONTAL POD AUTOSCALER?

- The Horizontal Pod Autoscaler (HPA) is a Kubernetes facility that adjusts the number of replicas of a Pod to match observed average CPU utilization to a target specified by the user

- There are a variety of configurable options - quite a few - the key takeaway is to know that HPA will create new Pods (or remove Pods) from a replica to maintain average CPU utilization across all Pods to a level specified when you create your HPA - subject to conditions you specify

CREATING AN HPA

- kubectl autoscale provides the needed functions to create an HPA
- To create an autoscaler on our "wordpress" deployment that targets CPU utilization to an average of 50% per Pod, within the parameters of a minimum of 1 pod and up to a maximum of 10 Pods

```
$ kubectl autoscale deployment wordpress --cpu-percent=50 --min=1 --max=10
```

PRACTICAL EXAMPLE

- Artificially limit our Wordpress Pod so we can stress it easily

```
$ kubectl apply -f ./wordpress-deployment.yaml
```

- Add an HPA to enable auto-scaling on our WordPress installation to keep CPU at 50% average (or lower), with a minimum of 1 Pod and max of 5 Pods

```
$ kubectl autoscale deployment wordpress --cpu-percent=50 --min=1 —max=5
```

- Simulate load using an infinite HTTP request loop from a worker Pod on our cluster to WordPress to spike our CPU and see how HPA responds

In a separate terminal window/ tab

```
$ kubectl run -i --tty load-generator --image=busybox /bin/sh

    Hit enter for command prompt

$  while true; do wget -q -O- http://wordpress.default.svc.cluster.local; done
```

- Check the HPA status (immediately and after about a minute)

```
$ kubectl get hpa
```

MASTER KUBERNETES

AUDITING

WHAT'S KUBERNETES AUDITING?

- Kubernetes auditing provides a security-relevant chronological set of records documenting the sequence of activities that have affected the system
- Can help answer what happened, why, by whom, on what, where, how it was initiated, and where was it going?
- The kube-apiserver performs auditing according to the *Audit Policy* and sends records to the *Audit Backend*

THE AUDIT POLICY

- The Audit Policy defines *what* should be logged
- The Audit Policy is defined in a .yaml file like any other Kubernetes object *but* since this is auditing, kubectl won't help you much here!
  - Auditing is defined on the kube-apiserver, it's an option that's passed to kube-apiserver when it starts, this varies by cluster
  - Luckily, for minikube, we just pass a few options (like our Audit Policy) when we start minikube
- Nearly anything you can create or do can be audited in Kubernetes

THE AUDIT BACKEND

- Audit Backends export audit events to external storage
- Out of the box, kube-apiserver provides two backends
  - Logs - which writes events to a disk
  - Webhooks - sends events to an external API/system

PRACTICAL EXAMPLE

- Let's set up a simple Audit Policy to log all requests regarding Metadata, this will let us see what Audit data looks like without overload
- We'll configure minikube to log it to a file on our system
- Our Steps:

1. Stop Minikube (if running)
2. Define an Audit Policy
3. Start Minikube with Audit Policy defined and appropriate options

MASTER KUBERNETES

EXERCISE

ON-YOUR-OWN EXERCISE

- Set up a deployment of the "Ghost" open-source publishing platform that maintains its data across restarts of the container (or minikube)
- For extra credit & challenge, set up the deployment to scale when CPU usage is high in the deployment

A FEW HINTS...

- The “ghost” image on Docker Hub is a perfect place to start your deployment
- The documentation for the docker image has some great hints on how to go about persisting its data!

GET TO WORK AND COME BACK!

A POSSIBLE SOLUTION

- Define a PersistentVolume of 10GB - *local-volume.yaml*
- Define a Deployment that uses the ghost Docker image - *ghost-deployment.yaml*
  - With a PersistentVolumeClaim
  - Mounting the PersistentVolume to /var/lib/ghost/content
- Setup Horizontal Pod Auto-scaler to scale when CPU hits 50%

```
$ kubectl autoscale deployment ghost --cpu-percent=50 --min=1 —max=5
```

MASTER KUBERNETES

HIGH AVAILABILITY

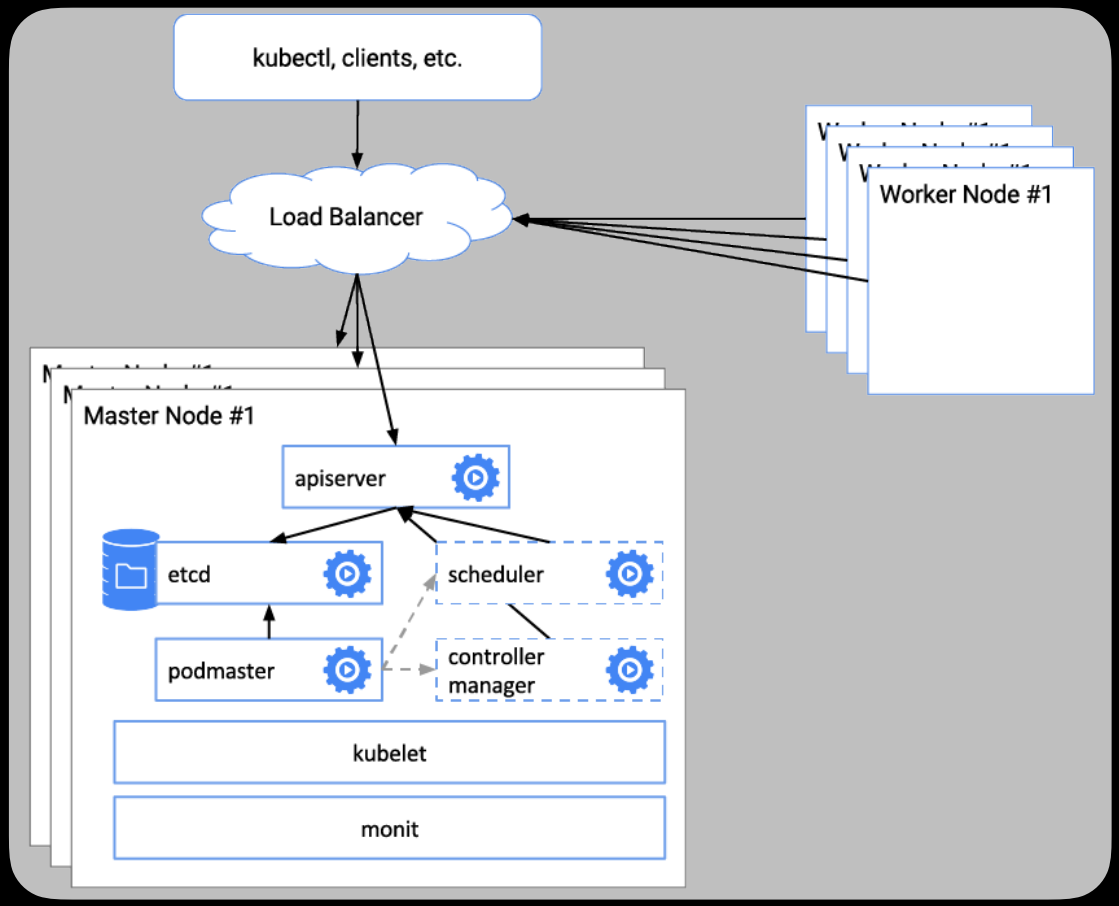
KEY KUBERNETES ARCHITECTURE ITEMS

- Master
  - kube-apiserver
  - kube-controller-manager
  - kube-scheduler
- etcd

SETTING UP A RELIABLE SYSTEM

1. Creating reliable nodes that, together, will form an highly available master implementation in following steps
2. Setting up a redundant, reliable data storage layer with clustered etcd
3. Starting replicated, load balanced API servers (kube-api)
4. Setting up master-elected kube-scheduler and kube-controller-manager daemons
5. Multiple Worker Nodes

SETTING UP A RELIABLE SYSTEM



RELIABLE NODES FOR OUR MASTERS

- Separate, independent Linux machines that will run master processes
- Should run kubelet & monit

A RELIABLE DATA STORAGE LAYER

- etcd is a system that stores key and value pairs that is the data in a Kubernetes cluster
- It should run on every node that will be a master
- Consult the etcd documentation on the variety of options on how etcd can provide even deeper levels of redundancy if needed

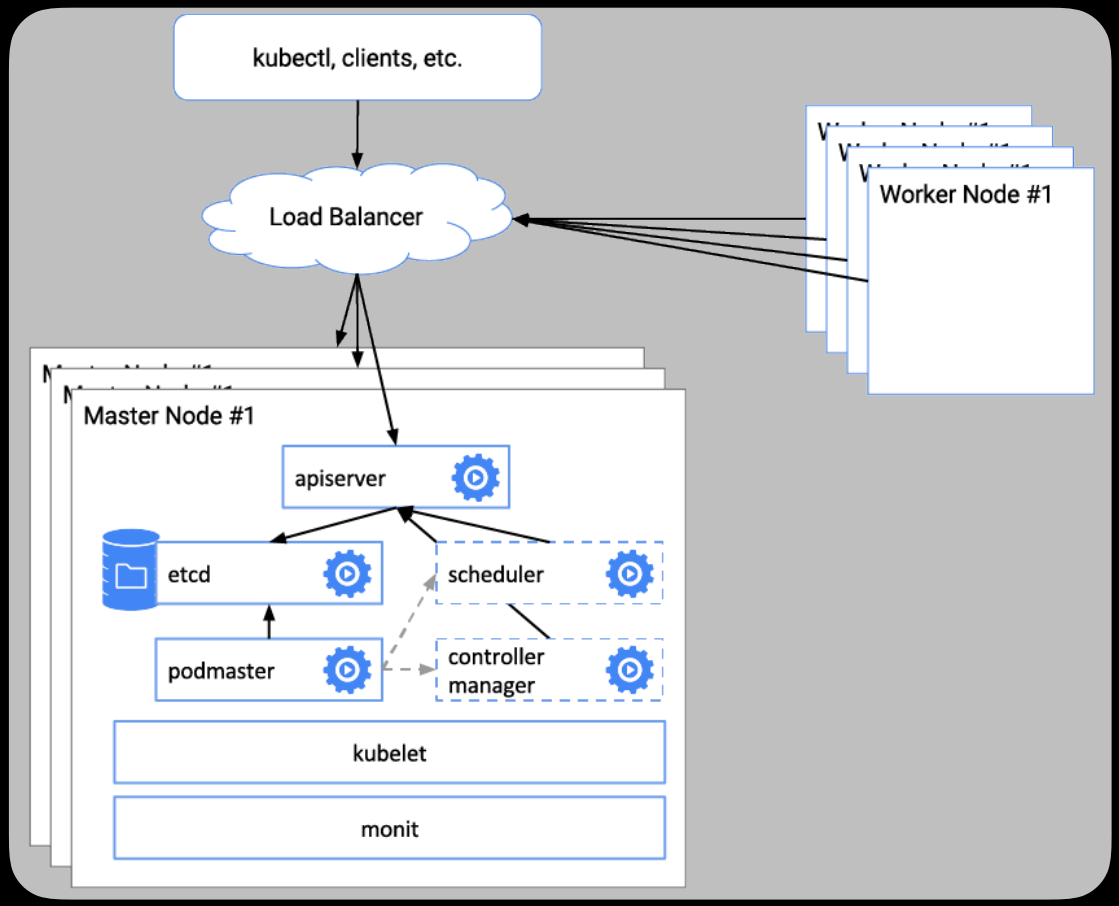
REPLICATED API SERVERS

- kube-api should run on all nodes that will be a master
- kube-api should be behind a network load balancer, this will vary with your environment you are running

MASTER ELECTED COMPONENTS

- Now that items are set up on reliable nodes, we have the pieces in place, but they aren't actually running
- We need to ensure only one actor works on the data at a given time
- Each Scheduler and Controller Manager will be started with a "—leader-elect" flag that will use a lease-lock API between themselves to ensure only one instance of each is running at a given time

KUBERNETES HIGH AVAILABILITY



MASTER KUBERNETES

MASTERS

KUBERNETES MASTERS

- Masters typically run three key processes
  - kube-apiserver
  - kube-controller-manager
  - kube-scheduler
- These processes access and rely on etcd

TO GET A LOOK AT A MASTER...

- We'll need something bigger than minikube
- Let's set up a Kubernetes cluster on AWS

YOU'LL NEED AN AWS ACCOUNT

- AWS is a cloud computing provider that allows access to virtual servers, along with a variety of managed services ranging from object storage, DNS, databases, and beyond
- AWS accounts are free to set up (although they require a credit card), but are **NOT FREE** to use
  - Even though they provide a "Free Tier", setting up a Kubernetes cluster on AWS does not fit inside this usage allowance and it **WILL** incur costs to you depending on how long you leave it running
  - Unless you are prepared to accrue and pay these charges and are confident that you'll be able to properly stop and release all cost-incurring resources, you may want to consider just watching

SETTING UP A KUBERNETES CLUSTER ON AWS

- Amazon offers a managed Kubernetes service called Elastic Kubernetes Service
  - Managed master redundancy/High Availability
  - Highly automated setup
- We'll use "kops", a suite of software provided by Kubernetes maintainers to do a bit more manual work to set up our own cluster

STEPS

1. Download and install “kops”
2. Download, configure, and install the “aws” command line tool suite
3. Create an AWS S3 bucket as a “state store”
4. Create the cluster using kops

STEPS

1. **Download and install "kops"**
2. Download, configure, and install the "aws" command line tool suite
3. Create an AWS S3 bucket as a "state store"
4. Create the cluster using kops

STEPS

1. Download and install “kops”
2. **Download, configure, and install the “aws” command line tool suite**
3. Create an AWS S3 bucket as a “state store”
4. Create the cluster using kops

STEPS

1. Download and install “kops”
2. Download, configure, and install the “aws” command line tool suite
3. **Create an AWS S3 bucket as a “state store”**
4. Create the cluster using kops

STEPS

1. Download and install “kops”
2. Download, configure, and install the “aws” command line tool suite
3. Create an AWS S3 bucket as a “state store”
4. **Create the cluster using kops**

LET'S HAVE A LOOK AT OUR MASTER

- Find the master node
- Find deployments running on it
- Examine the deployments that comprise the master

STOPPING OUR CLUSTER

- Delete the cluster using kops
- To be completely safe, close the AWS account you created to ensure all resources are released and you are not charged (or wait until after you've completed this lecture series if you'd like to follow along in the following lectures)

MASTER KUBERNETES

SETTING UP HIGH AVAILABILITY

KEY HA DESIGN CONSIDERATIONS

- How many masters (an odd number is critical)
- Where the masters are located (across different AWS availability zones)

KOPS CREATE CLUSTER

- The create cluster command we used in the last lecture allows us to specify High Availability options
- How many masters
- In which zones these masters are provisioned

LET'S CREATE A HIGH AVAILABILITY CLUSTER

1. kops create
2. Look at the nodes to see multiple masters
3. Deploy Wordpress & MySQL *without volumes*

MASTER KUBERNETES

VOLUMES ON AWS

DYNAMIC PROVSIOINING

- When an existing Volume does not match a Claim, k8s maybe able to (based on the provider type and configuration) to provision Volumes to fulfill the Claim

- Depends on underlying implementation's ability to create new Volumes (available space, permissions, etc)
  - Example: If underlying system is a cloud provider with enough quota and access levels allow, k8s may be able to create new Volumes
  - Example: If underlying system is a SAN filesystem and there is no more free space, k8s would not be able to create new Volumes (it can't order and install more physical disks on its own, yet! ;))

VOLUMES ON AWS

- Amazon Web Services (AWS) offers persistent disks via its Elastic Block Store (EBS)
- Kubernetes on AWS can use (and even dynamically provision) Volumes on EBS to match or meet PersistentVolumeClaims made by Pods

DEMO

- Edit our existing existing MySQL & WordPress deployments to include PersistentVolumeClaims exactly like we did in the Volumes lecture but this time, we won't create the PersistentVolumes, so this will require Dynamic Provisioning

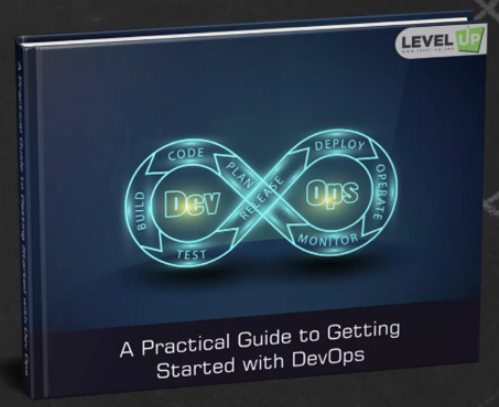
kubernetes
CMA CGM
eco container
CMA CGM
eco container
tex

KEEP
CALM
AND
ENJOY
KUBERNETES

GROW YOUR DEVOPS SKILLS

LEARN HOW TO GET STARTED WITH DEVOPS

Whether you are from a system admin or software developer background, or if you are a recent new graduate or have been in the industry for a while, this book will give you an overview of the DevOps industry and provide you with a practical guide about how to get started with a career in the DevOps domain.



GET YOUR FREE COPY NOW

https://www.level-up.one/devops-pdf-book/

Learning DevOps and Level Up

Public Group

About

Discussion

Members

Events

Photos

Search this group

https://www.facebook.com/groups/1911219079195863/

LEARNING DEVOPS & LEVEL UP

More

Group

Ops and Level Up

a place for folks to discuss general topics about DevOps and
ed helps. Strict policy: No spams and off-topic discussions in

CREATE NEW GROUPS

Groups make it easier than ever to share with friends, family and teammates.

Create Group

Suggested Groups

See All

Apache Spark with Scala - Learn Spark from a Big...

Level Up Expert Program, James...

4.9 (1,063)

~~£150~~ £15

Apache Spark with Java - Learn Spark from a Big...

Tao W., James Lee, Level Up

4.7 (901)

~~£200~~ £15

Master Jenkins CI For DevOps and Developers

Level Up, Tao W., James Lee

4.5 (3,473)

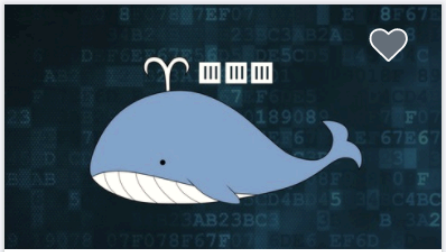
~~£150~~ £15

IntelliJ IDEA Tricks to Boost Productivity for...

Tao W., James Lee

4.5 (243)

~~£100~~ £15



Docker Technologies for DevOps and Developers

Tao W., Level Up, James Lee

4.5 (4,465)

~~£125~~ £15

https://www.level-up.one/join/

Search

LEVEL UP

Created by James Lee and Tao W.

- Home
- Courses
- Blogs
- Free Books
- Who Are We
- Partnership
- Discounts
- Latest Update
- Contact Us

Get 90% Discounts

Do you want to get any of our courses with 90% discount? Simply join our mailing list and we'll immediately send you a link to get any of my courses with 90% discount.

Also, we will periodically send you the latest update in the DevOps and Big Data world and practical tips/tricks.

We will not spam you, and you can 1-click unsubscribe at any time.

GET ALL THE COURSES WITH 90% DISCOUNT.

EMAIL *

FIRST NAME *

SUBSCRIBE