KubeAcademy
from VMware

Introduction to Containers & Kubernetes

VMware-KubeAcademy

Version 2.3.0

KubeAcademy
from VMware

Introduction to Containers & Kubernetes

Welcome! You should be hearing music, if not please try the following:

- Refresh the browser tab
- Click the video box in the top right corner of this browser window
- Make sure you are using either Chrome or Firefox.

About Strigo

- Strigo is a web-based platform that provides the classroom environment for our courses.
- Let's walk through the features of the platform.
- We'll also get your lab environment initialized

Introductions

- About your instructor(s)

- Tell us about yourself
  - Would you classify yourself as a
    - developer
    - systems administrator
    - architect
    - It depends on the day/hour
  - What are your goals for learning and adopting Kubernetes?

Agenda

1. Introduction to Containers
2. Kubernetes Fundamentals
3. Q&A

Course Format

- This is a lab-intensive, hands-on course
- Each section will begin with the introduction of a new concept
- Each section contains a lab exercise where you will explore each new concept

Introduction to Containers

Chapter 01

Agenda

1. Introduction to Containers
2. Kubernetes Fundamentals
3. Q&A

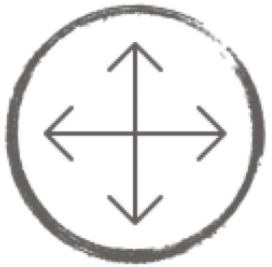
Cloud Native Principles

- Container Packaged
  - isolated unit of work that does not require OS dependencies
- Dynamically Managed
  - actively scheduled and managed by an orchestration process
- Microservice Oriented
  - Loosely coupled with dependencies explicitly described

Why Containers?

Velocity

enables business to develop and roll out new offerings faster



Portability

predictable execution in any linux based environment. Move from your laptop to your datacenter to the cloud

Reliability

hermetically sealed image makes production deployments very simple and less error-prone

Efficiency

maximize resource utilization

Self-Service

developer productivity

Isolation

avoid dependency conflict

VMs ≠ Containers

Virtual Machines

- Full OS on virtual hardware
- Deployed as a unit
- Require a hypervisor
- VM performance critical to cloud performance
- Overhead cannot be removed
- Hotplugging requires support from guest OS

Containers

- Decouple application and OS
- Can be composed together
- Modifies existing OS to provide isolation
- A concept built on kernel namespace feature
- Contain as little as one process
- An application container consumes less RAM
- Operate at a higher abstraction level, offer more insight into behaviors without deploying additional agents

Isolation - All Levels

- Cost to create, store, and run a container is very low

Programmatic Construction

- VM Creation is not guaranteed to be scripted/reproducible
- Container build process enforces reproducibility

```
FROM ubuntu:16.04
RUN apt-get update && apt-get install -y apache2
RUN apt-get clean && rm -rf /var/lib/apt/lists/*
ENV APACHE_RUN_USER www
ENV APACHE_RUN_GROUP www
ENV APACHE_LOG_DIR /var/log/apache2
EXPOSE 80
CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```

Image vs Container

- An image is the result of a build
- A container is a running instance from an image

```
$ docker images
REPOSITORY      TAG         IMAGE ID          CREATED             SIZE
busybox         latest      8c811b4aec35      2 weeks ago         1.15 MB
mysql           latest      a8a59477268d      4 weeks ago         445 MB
foo.com/mysql   latest      a8a59477268d      4 weeks ago         445 MB

$ docker ps
CONTAINER ID  IMAGE  COMMAND    CREATED              STATUS              PORTS      NAMES
Acc5c8f58392  mysql  “mysqld”   About a minute ago   Up About a minute   3306/tcp   myWebsiteDB
```

Image/Container – VM comparison

- Model similar to linux process invocation
  - **image** – one immutable set of bits on disk
  - **container** – running instances launched from the image

  - conceptual like running multiple processes of a command line process at once

- Container filesystem is ephemeral
  - promotes cloud native and immutable deployments

Starting and Stopping Containers

`docker run`

- start a new container from an image

`docker stop`

- stop a running container

`docker rm`

- delete a container (must be stopped)
- add `-f` to both stop and remove a container

Managing Images

`docker images`

- display a list of images on the machine

`docker rmi`

- delete an image

`docker build`

- build an image from a Dockerfile

`docker tag`

- add tags to an image

`docker pull/push`

- pull and push images to/from a registry

Docker Troubleshooting Commands

`docker ps`

- retrieve a list of running containers
- add `-a` to include non-running containers

`docker logs`

- view a container's log output

`docker exec`

- run a command within a container
- can also start a shell within a container (if available)

Container Registry

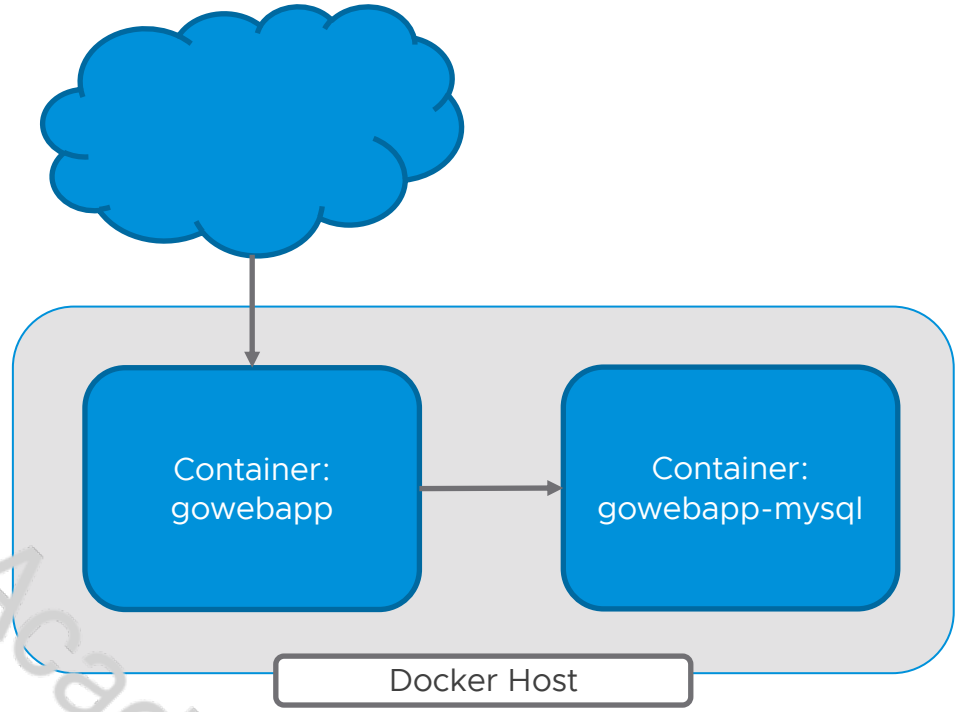
- A container registry provides a central location for container image storage
- Types of registries
  - Hosted: Docker Hub, Google Container Registry
  - Self-Hosted: Artifactory, Harbor, Quay
- Images are built on a build host and pushed to a registry

What We're Building

Deploying Go Web App using Docker

Go Web App:

- A web–based notepad application
- Frontend written in Golang
- MySQL database backend



Lab 01

Containerize Applications

Prepare your lab

Write Dockerfile for frontend web app

Write Dockerfile for backend data store

Build, test, and publish Docker images

Kubernetes Fundamentals

Chapter 02

Agenda

1. Introduction to Containers
2. Kubernetes Fundamentals
3. Q&A

Cloud Native Principles

- Container Packaged
  - isolated unit of work that does not require OS dependencies
- Dynamically Managed
  - actively scheduled and managed by an orchestration process
- Microservice Oriented
  - Loosely coupled with dependencies explicitly described

About Kubernetes

- Open-source cluster management tool
  - Automates: Deploying, managing, scaling applications
- Managed by Cloud Native Computing Foundation (CNCF); originally created at Google
- Under active development by a well-supported community, including former Borg engineers
- Lessons learned from Borg in production for more than a decade
- Can run on both bare metal and on various cloud providers

Imperative vs. Declarative

Imperative	Declarative
defines actions	defines desired state

Kubernetes Concepts

- Pods
- Services
- Deployments

Why Pods?

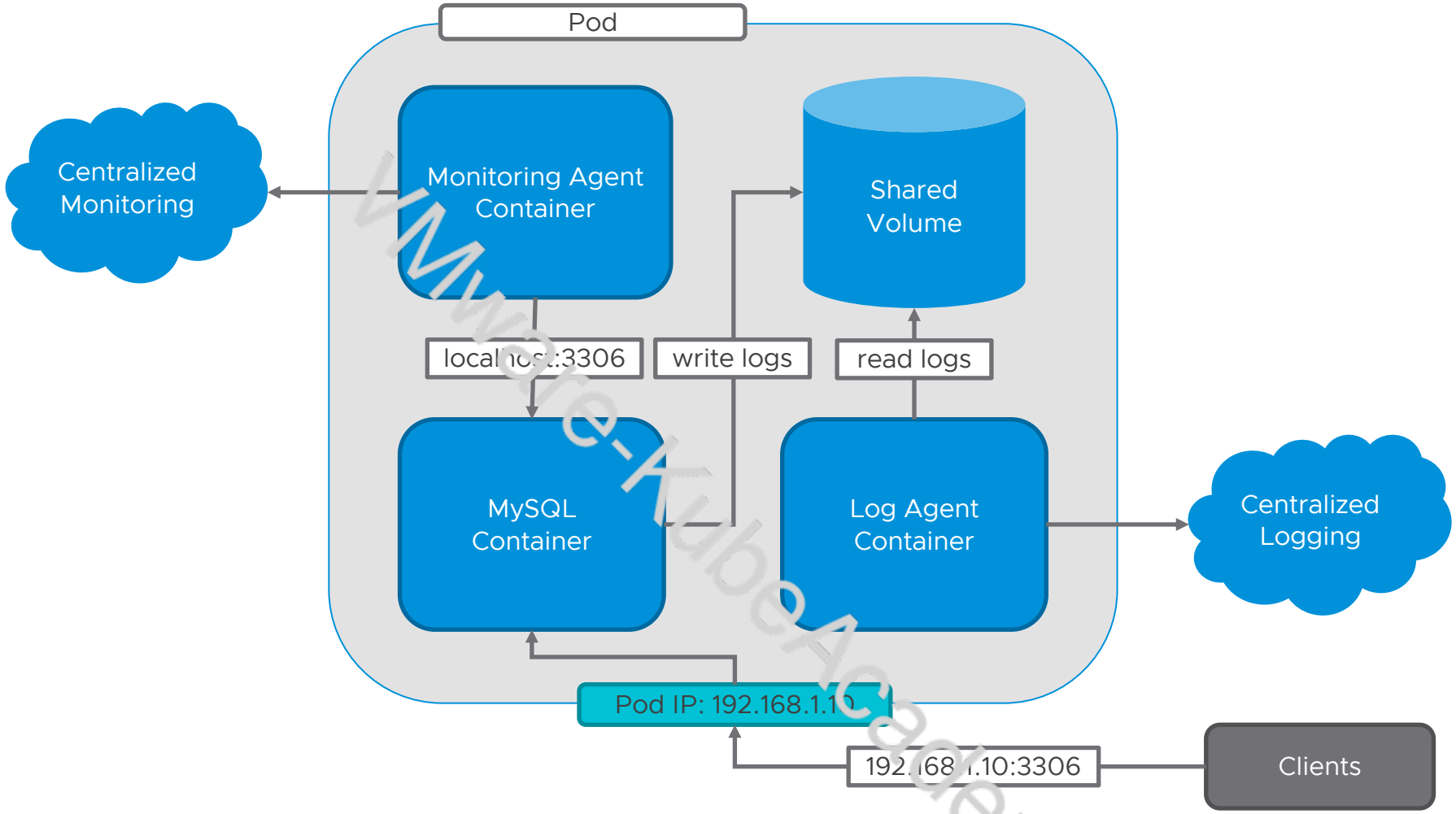
- Legacy applications
  - new apps have luxury of implementing many features within the app
- Separation of Concerns
  - isolate logic and changes to different apps
- Reusability
  - if designed appropriately containers can be re-used by different apps

Pod Characteristics

- One or more containers per pod
  - the pod's existence/purpose is usually because of one particular container

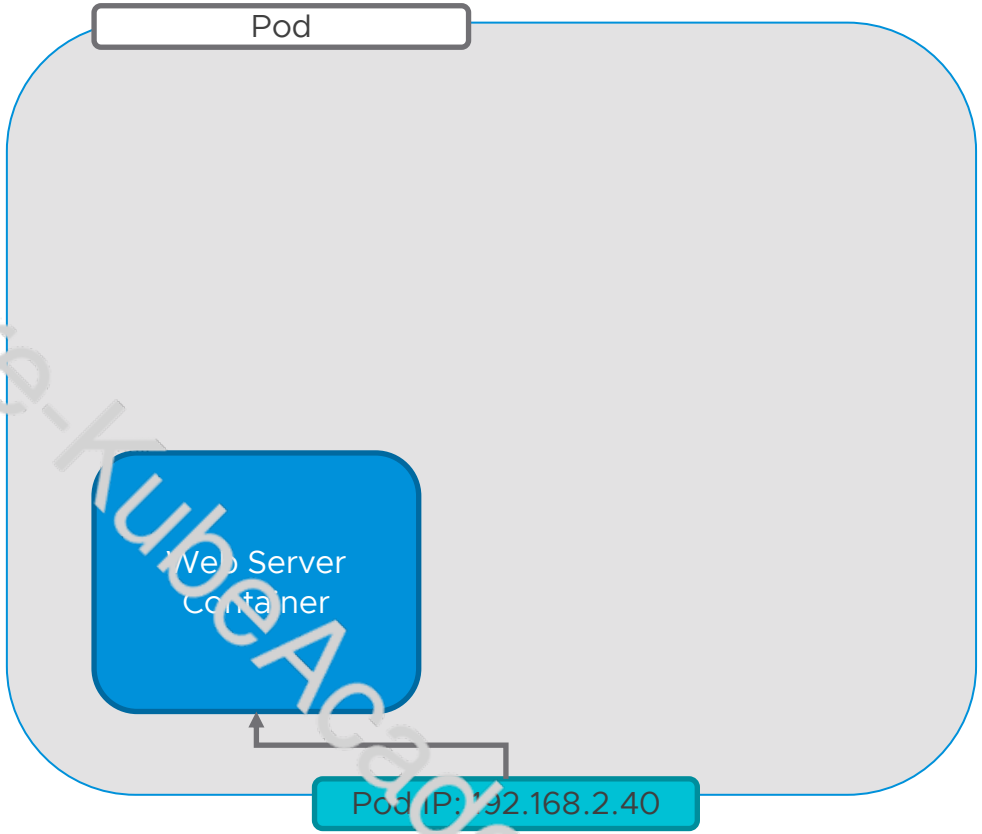
- Assumptions
  - all containers for a pod will run on the same node
  - containers within a pod can talk to each other over localhost
  - containers can share volume resources

Containers in a Pod



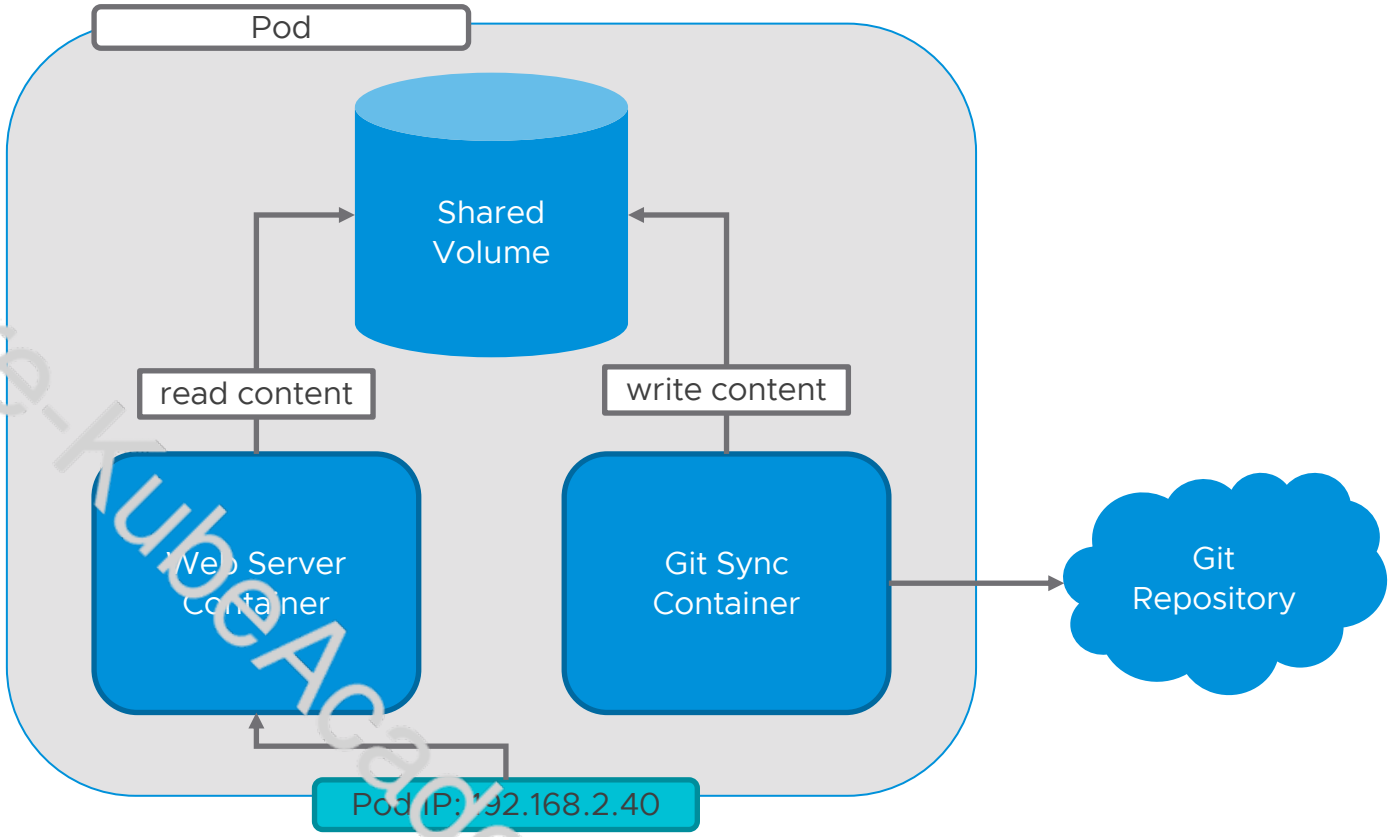
Pod - Main Container

- main container is the reason a Pod exists



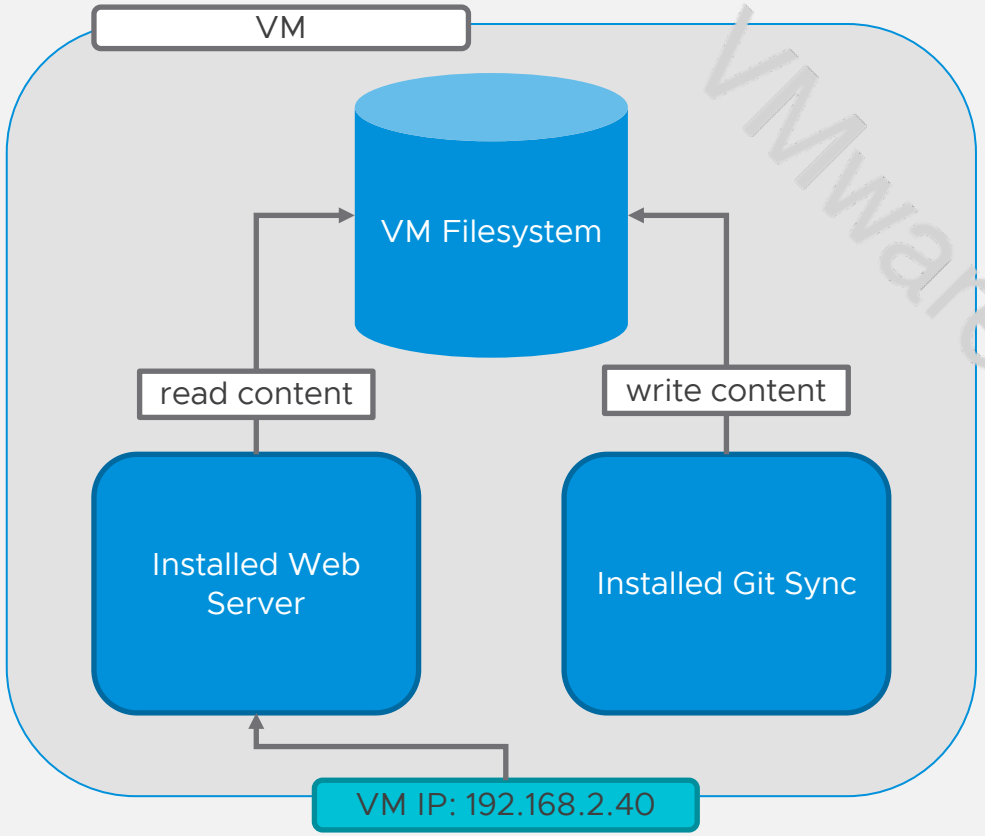
Pod - Sidecar Containers

- sidecars help and assist functions
  - logic could be included in main container
  - separation allows for isolation and reuse
  - each container can have separate resource allocations

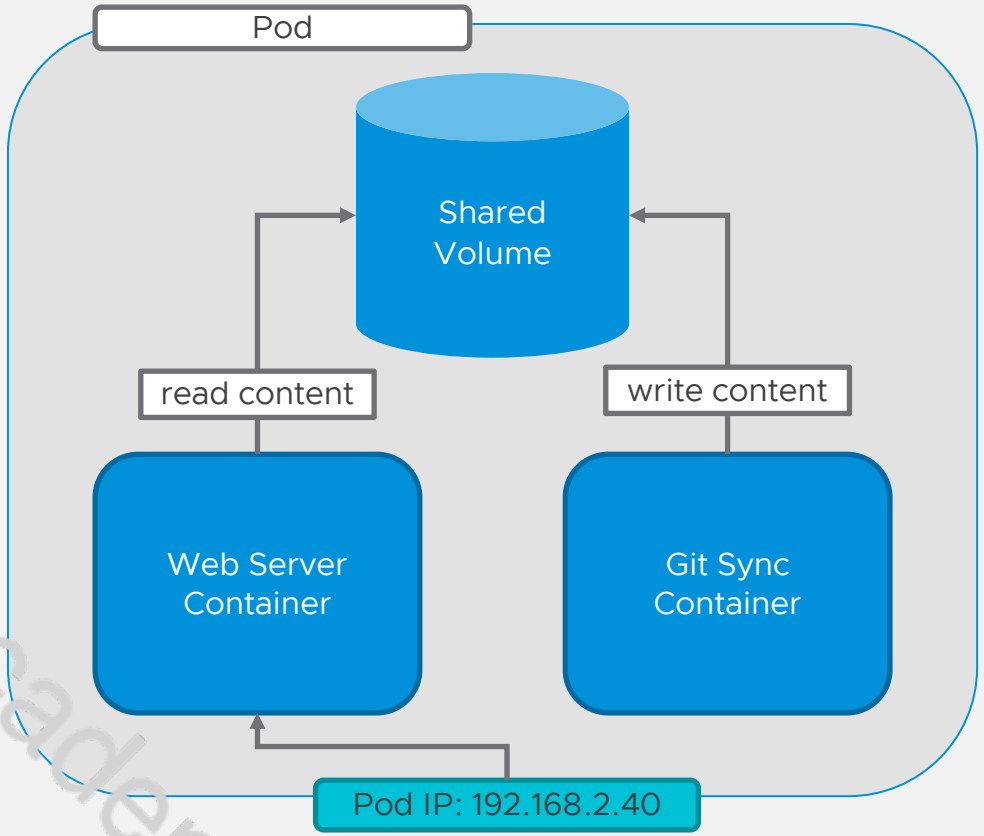


Similar Concepts

Virtualization

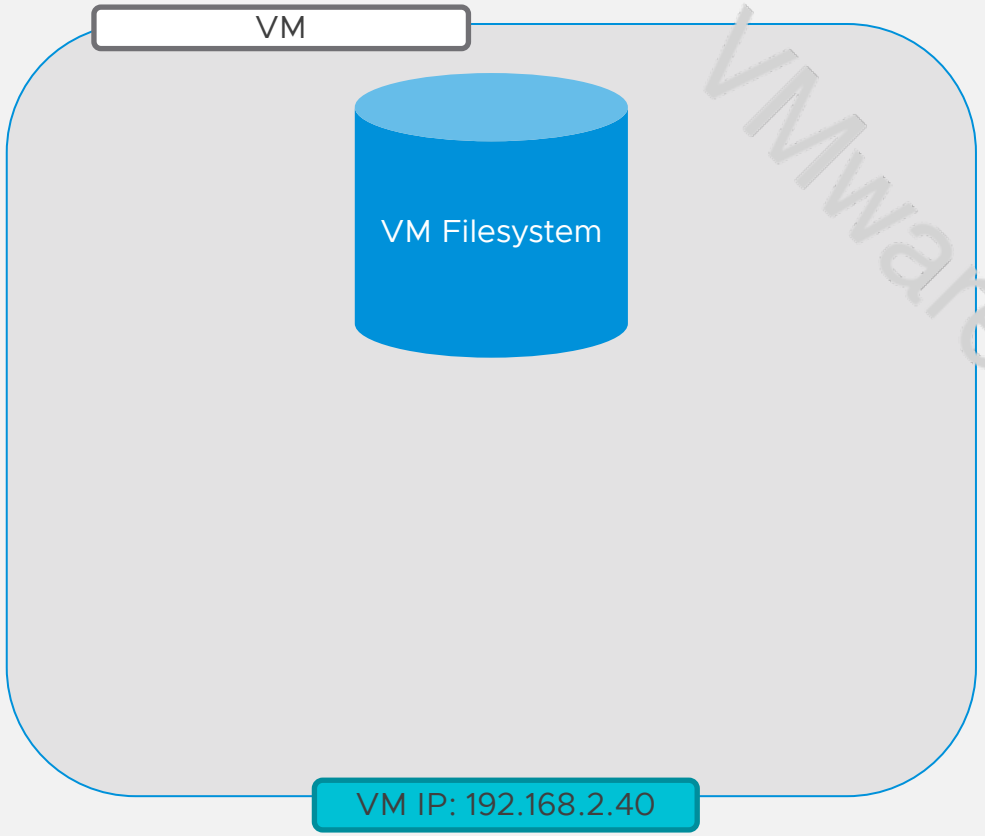


Kubernetes

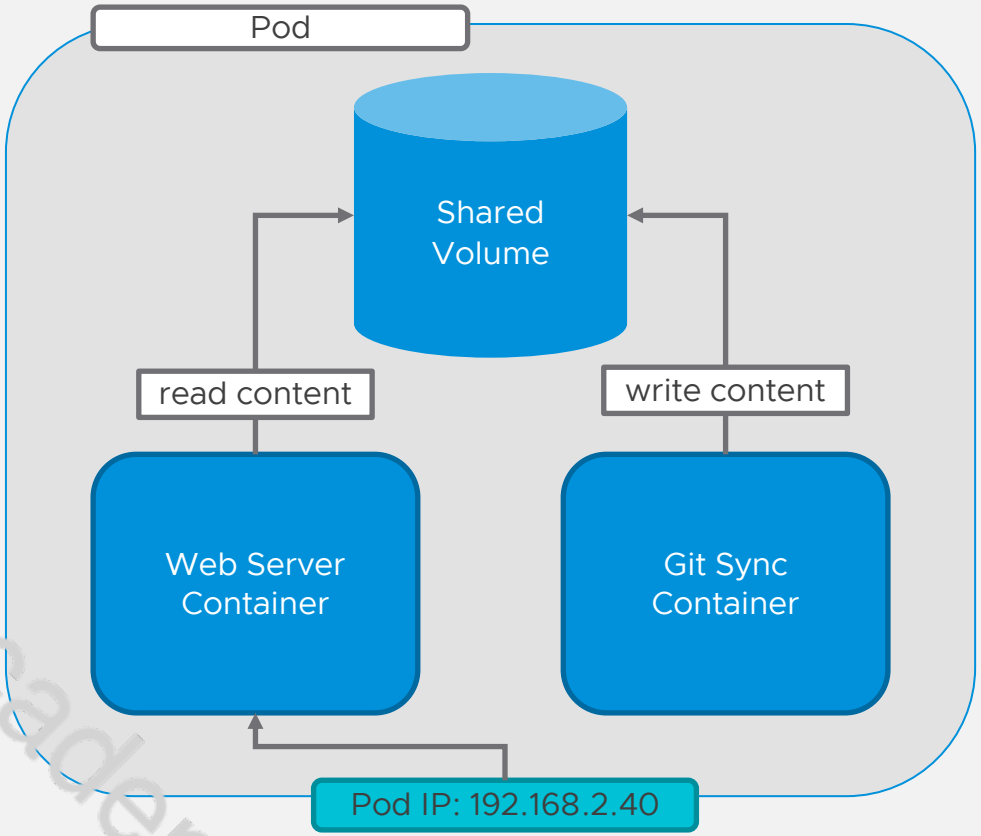


Responsibility and Standardization

Virtualization



Kubernetes



API - REST

- REST / JSON based
  - all internal and external components communicate via this API
  - explicitly versioned to allow breaking changes
  - provides abstraction layer on top of resource storage (etcd)

```
curl http://localhost:8001/api/v1/pods

{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/pods",
    "resourceVersion": "3606680"
  },
  "items": [
```

API - kubectl

- command line interface for the API
  - calls one or more REST API calls for each command line invocation
  - does contain some business logic not in the API

```
$ kubectl get pods

NAME                    READY   STATUS    RESTARTS   AGE
blog-56d5c9dbf7-6j67j   1/1     Running   0          19s
blog-56d5c9dbf7-dh7vb   1/1     Running   0          19s
blog-56d5c9dbf7-zv7d7   1/1     Running   0          19s
```

Yet Another Markup Language (YAML)

```yaml
employeeNumber: 123
name:
  firstName: Jenny
  lastName: Smith
phones:
- label: work
  number: 555-555-5555
- label: mobile
  number: 267-867-5309
```


```json
{
  "employeeNumber": 123,
  "name": {
    "firstName": "Jenny",
    "lastName": "Smith"
  },
  "phones": [
    {
      "label": "work",
      "number": "555-555-555"
    },
    {
      "label": "mobile",
      "number": "267-867-5309"
    }
  ]
}
```

```xml
<employee id="123">
  <name>
      <firstName>Jenny</firstName>
      <lastName>Smith</lastName>
  </name>
  <phones>
    <phone label="work">
        <number>555-1212</number>
    </phone>
    <phone label="mobile">
        <number>867-5309</number>
    </phone>
  </phones>
</employee>
```

API Resource Objects & YAML

- Often represented in YAML (can also use JSON)
- Represent API objects

```
apiVersion: v1
kind: Pod
metadata:
    name: mypod1
    labels:
      app: blog
spec:
    containers:
    - name: nginx
      image: nginx:1.13.1
    - name: www-syncer
      image: syncer:1.2
```

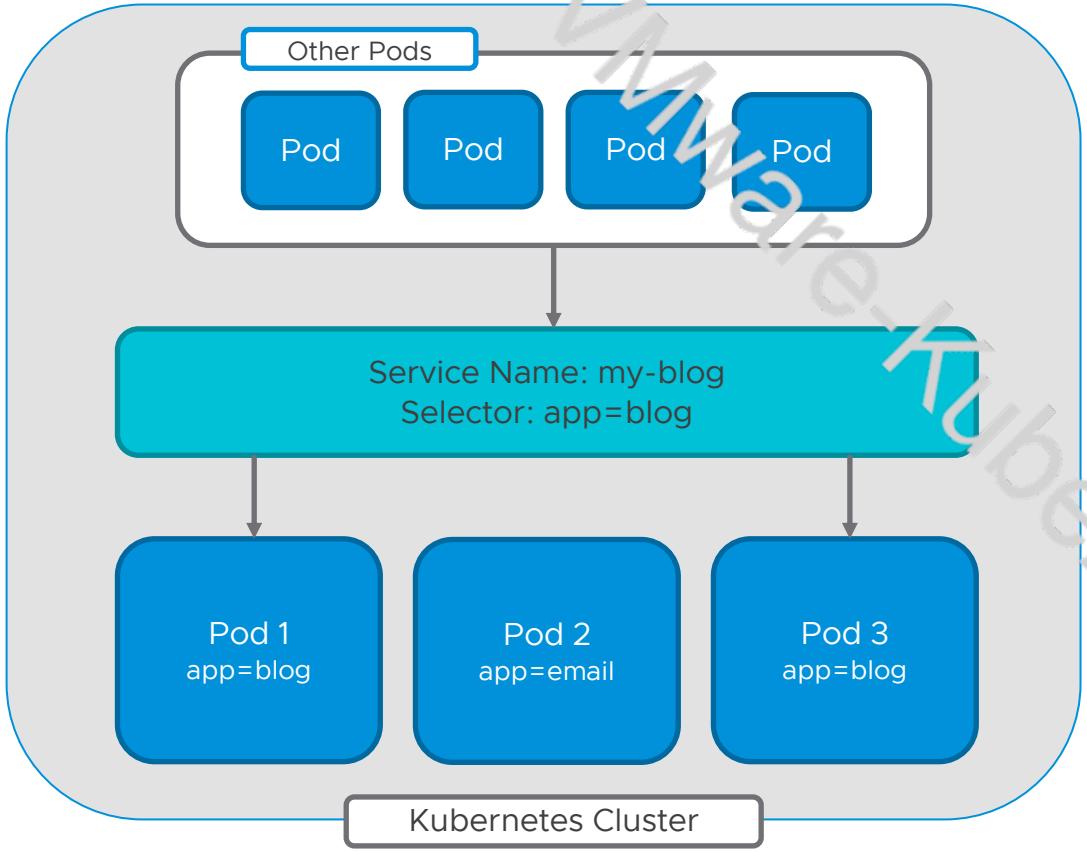
Creating Resources

- Declarative Approach (Preferred)
  - kubectl apply -f [<file> | <directory> | <url>]
    - Create new or update existing resource(s) from file(s)

- Imperative Approach
  - kubectl create - create new resource from a file
  - kubectl replace - update an existing resource from a file
  - kubectl edit - update existing resource using your default editor
  - kubectl patch - update existing resource by merging a code snippet

Services

- Load balancing for pods
- Use labels to determine target pods




```
apiVersion: v1
kind: Service
metadata:
  name: my-blog
  labels:
    app: blog
spec:
  selector:
    app: blog
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Pod Creation & Deployments

- What if I have a website that needs 20 NGINX servers?

Pod Creation & Deployments

- What if I have a website that needs 20 NGINX servers?
  - 20 pod objects sent to Kubernetes
  - each object is exactly the same except for the name

- We need a higher level construct...

Deployments

- Single object that will create other resources
- Pod spec is nested
- Leverages selectors

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-blog
spec:
  replicas: 3
  selector:
    matchLabels:
      app: blog
  template:
    metadata:
      labels:
        app: blog
    spec:
      containers:
        name: nginx
        image: nginx:1.7.9
        ports
        - containerPort: 80
```

Labels

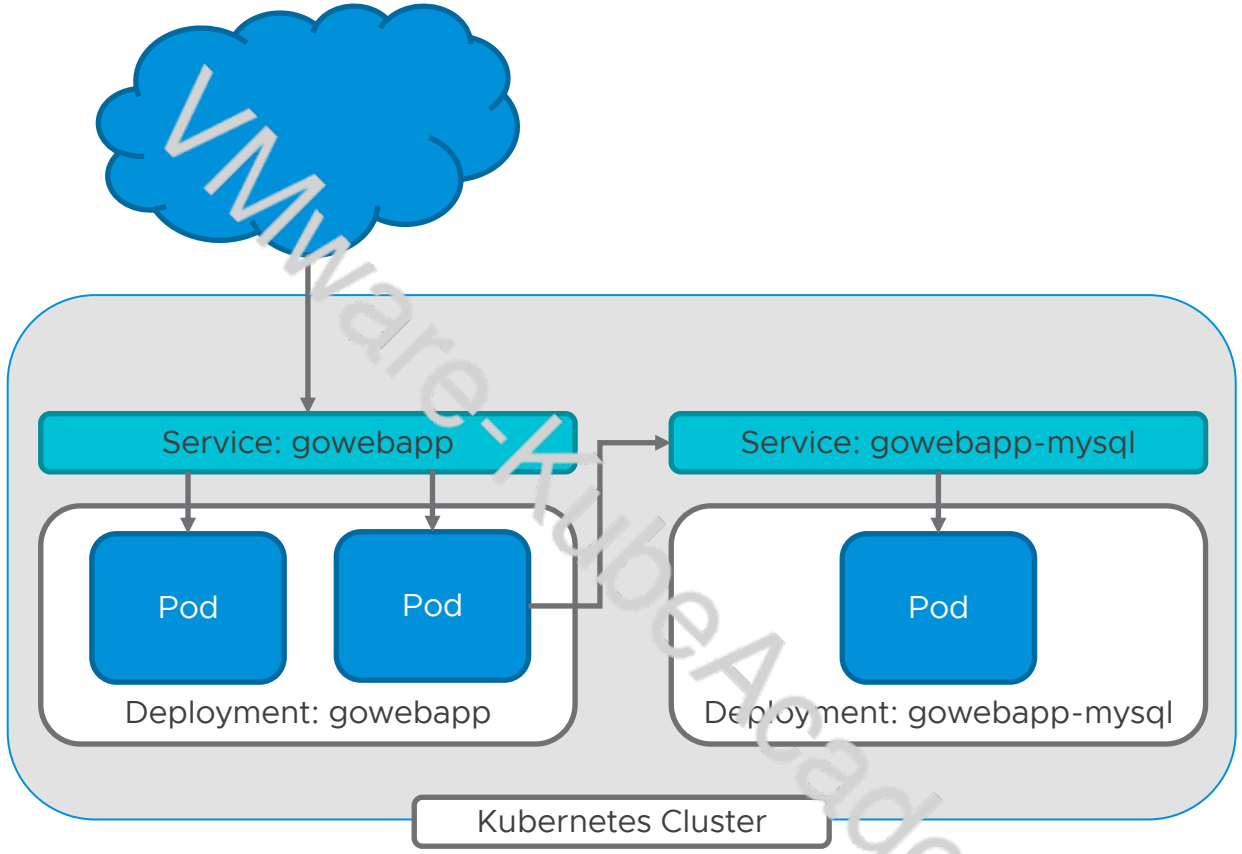
- Characteristics
  - map of key / value pairs
  - both organizational and functional (selectors on Services)
  - indexed and searchable
- Tips
  - avoid compound label values:
    - `app: blog-frontend` vs `app: blog / tier: frontend`
  - try to standardize on key/values across the cluster

Kubernetes Documentation

- Kubernetes Docs
  - https://docs.kubernetes.io
- Kubernetes API Reference
  - https://kubernetes.io/docs/reference/
- kubectl Docs
  - https://kubectl.docs.kubernetes.io/

What We're Building

Deploying Go Web App to Kubernetes



Lab 02

Using Kubectl

Getting familiar with kubectl

Deploy Applications

Write YAML files for Services

Write YAML files for Deployments

Deploy the Applications

Q&A

Thank You