Containerisation with Docker & Kubernetes
Hello!

- I'm Dan!
- Senior Field Engineer at Heptio VMware
- Ex:
  - Heptio
  - Docker
  - Hewlett-Packard Enterprise
  - SkyBet
  - European Space Agency
  - ...
- Still a maintainer and contributor to various Docker and Moby projects (mainly GO, but I sometimes write C (if need be))
- @thebsdbox – for random nonsense
Agenda

• Where did Docker come from?
• Docker Under the Hood
• Using Docker
• Questions?
Where did Docker come from?

In the beginning ...
Where did Docker come from?

Things broke a lot ...
Where did Docker come from?

- FreeBSD Jails expand on Unix chroot to isolate files
- Solaris Zones bring the concept of snapshots
- RedHat adds user namespaces, limiting root access in containers
- Docker provides simple user tools and images. Containers go mainstream

2000
- FreeBSD Jails

2001
- VServer
- FreeBSD Jails expand on Unix chroot to isolate files

2004
- Linux-VServer ports kernel isolation, but requires recompilation
- Google introduces Process Containers, merged as cgroups

2006
- cgroups
- Solaris Zones bring the concept of snapshots

2008
- LXC
- RedHat adds user namespaces, limiting root access in containers
- IBM creates LXC providing user tools for cgroups and namespaces

2013
- Docker
- Docker provides simple user tools and images. Containers go mainstream
Docker under the hood
Docker Installation

• Linux installation(s)
  • `apt-get install -y docker`
  • `yum install -y docker`

• Docker for Mac

• Docker for Windows

• Docker on Windows 10 / 2016 / 2019
Docker Components

Docker CLI

dockerd

containerd

runc
Docker Command Line Interface

• `docker build` build docker image from Dockerfile
• `docker run` run docker image
• `docker logs` show log data for a running or stopped container
• `docker ps` list running containers, `--a` includes stopped containers
• `docker images` list all images on the local volume
• `docker rm` remove a container | `docker rmi` : remove an image
• `docker tag` name a docker image
• `docker login` login to registry
• `docker push/pull` push or pull volumes to/from Docker Registries
• `docker inspect` return info/metadata about a Docker object
Docker Command Line Interface

[dan@docker ~]$ docker run

Docker CLI

Connects to

/var/run/docker.sock

Exposes *

x.x.x.x:<port>

* Configured in /etc/docker/json
Docker Daemon

- Provides the "standard" interaction with a container platform
  - Image download from registries
  - Plugin features to extend the container platform for other vendors
  - In-build orchestration for multiple docker Engines
  - Container build function
containerd

- Manages the full container lifecycle:
  - Container execution
  - Image transfer/storage
  - Presentation of images to the OCI runtime
  - Networking
  - Container supervision
runc (or any OCI specific runtime)

- Lowest level component
- Implements an OCI compatible interface
- Implements the namespace isolation that “defines” a container
- Handles the starting, monitoring and stopping of a container
All tied together

Docker Engine

- Same Docker UI and commands
- User interacts with the Docker Engine
- Engine communicates with containerd
- containerd spins up runc or other OCI compliant runtime to run containers
Docker Image

/bin
/boot
/dev
/etc
/home
/lib64
/mnt
/opt
/var
/usr
Docker Image

/

/bin
/boot
/dev
/etc
/home
/lib64
/mnt
/opt
/var
/usr
/usr/myapp
## Docker Image

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/</code></td>
<td>Root directory</td>
</tr>
<tr>
<td><code>/bin</code></td>
<td>executable files</td>
</tr>
<tr>
<td><code>/boot</code></td>
<td>boot files</td>
</tr>
<tr>
<td><code>/dev</code></td>
<td>device files</td>
</tr>
<tr>
<td><code>/etc</code></td>
<td>system configuration files</td>
</tr>
<tr>
<td><code>/home</code></td>
<td>home directory</td>
</tr>
<tr>
<td><code>/lib64</code></td>
<td>library files</td>
</tr>
<tr>
<td><code>/mnt</code></td>
<td>mounted filesystems</td>
</tr>
<tr>
<td><code>/opt</code></td>
<td>optional packages</td>
</tr>
<tr>
<td><code>/var</code></td>
<td>temporary files</td>
</tr>
<tr>
<td><code>/usr</code></td>
<td>user files</td>
</tr>
<tr>
<td><code>/usr/myapp</code></td>
<td>application directory</td>
</tr>
<tr>
<td><code>/etc/ssl.keys</code></td>
<td>SSL keys</td>
</tr>
</tbody>
</table>

Manifest.json

```
//
/bin
/boot
/dev
/etc
/etc/ssl.keys
/home
/lib64
/mnt
/opt
/var
/usr
/usr/myapp
```
Docker Image
**Container Registry**

* Container registries are now standardized through the Open Containers Initiative.

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Image</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer_100</td>
<td>App1</td>
<td>Latest</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.9</td>
</tr>
<tr>
<td></td>
<td>App2</td>
<td>Latest</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>Web_tier</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>App_tier</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DB_tier</td>
<td>2.0</td>
</tr>
</tbody>
</table>

* If no hostname, defaults to Docker hub.
Pulling an image from a Registry

What happens if no tag is specified?

* Above specifies the project(namespace), the image and the specific tag (version) of the image to be pulled.
What is a container?
What is a container?

PID Namespace

Mount Namespace

Net Namespace

[dan@docker ~]$ docker run -d \
-p 8080:80 \
apache2

:8080
dockerhost01.local
Using Docker
Using Docker

- Docker File
- Docker CLI
- Docker Compose
- Container Orchestration
**FROM** defines the base image that the subsequent layers will build on top of.

The base image **scratch** is the smallest base image and allows (as the name suggests) starting from scratch.

```
FROM golang:1.9.2-alpine3.6
RUN go get github.com/thebsdbox/klippy
WORKDIR /go/src/github.com/thebsdbox/klippy
RUN go build -o /bin/klippy
ENTRYPOINT ["/bin/klippy"]
CMD ["--help"]
```
Dockerfile

**RUN** Creates a new empty layer and will execute a command upon that layer, any filesystem changes will be then stored in the new layer.

In this example `go get` will pull `go` source code from the internet, which will persist in this new layer.

```dockerfile
FROM golang:1.9.2-alpine3.6
RUN go get github.com/thebsdbox/klippy
WORKDIR /go/src/github.com/thebsdbox/klippy
RUN go build -o /bin/klippy
ENTRYPOINT ["/bin/klippy"]
CMD ["--help"]
```
Dockerfile

WORKDIR sets the current working directory for any subsequent commands
Dockerfile

**ENTRYPOINT** specifies the command that will be ran by default when the container is ran.

**CMD** specifies a command that is ran when no arguments are passed to a container.

```dockerfile
FROM golang:1.9.2-alpine3.6
RUN go get github.com/thebsdbox/klippy
WORKDIR /go/src/github.com/thebsdbox/klippy
RUN go build -o /bin/klippy
ENTRYPOINT ["/bin/klippy"]
CMD ["--help"]
```
Efficient Dockerfile(s)

FROM golang:1.9.2-alpine3.6 AS build
RUN go get github.com/thebsdbox/klippy
WORKDIR /go/src/github.com/thebsdbox/klippy
RUN go build -o /bin/klippy

# This results in a single layer image
FROM scratch
COPY --from=build /bin/klippy /bin/klippy

ENTRYPOINT ["/bin/klippy"]
CMD ["--help"]

A MultiPart docker file allows building your image from a number of other images.

Allows the separation of building/compiling images to the final application image, creating smaller images that only have the assets that are needed.
Docker CLI

The CLI is standard across all platforms, I'll be using Docker for Mac to demonstrate some of the more useful commands.

Some CLI commands, such as the `stacks` and `service` commands will only appear once a node or multiple nodes are in a cluster (also only master nodes will be able to manage the cluster)
Docker Compose

Docker compose provides the capability to orchestrate, build and deploy an application that is built from multiple containers.

A compose file is a YAML file that specifies the various containers that make up an application, and how the various containers should interact.
version: "3"
services:
  result:
    build: ./result
    command: nodemon server.js
    volumes:
      - ./result:/app
    ports:
      - "5001:80"
      - "5858:5858"
  redis:
    image: redis:alpine
    container_name: redis
    ports: ["6379"]
  db:
    image: postgres:9.4
    container_name: db
    volumes:
      - "db-data:/var/lib/postgresql/data"
Docker Compose example

https://github.com/dockersamples/example-voting-app
**Container Orchestration**

**Container orchestrators** provide the functionality to manage containers over one or more hosts. When a **scheduling** request is made (i.e. asking the orchestrator to deploy an application) the orchestrator will be able to examine the environment and handle the deployment and placement of containers within the cluster.

Additional features (may) include:

- Provisioning hosts
- Instantiating a set of containers
- Rescheduling failed containers
- Linking containers together through agreed interfaces
- Exposing services to machines outside of the cluster
- Scaling out or down the cluster by adding or removing containers
Container Orchestrators

- Rancher
- Nomad
- Docker Swarm
- Mesos
- Kubernetes
Docker Swarm

Docker Swarm: Produces a single, virtual Docker host by clustering multiple Docker hosts together.

It presents the same Docker API; allowing it to integrate with any tool that works with a single Docker host.

On a Manager node:

docker swarm init
...
  docker swarm join --token aabbcc 192.168.0.1:2377
...

On a Worker node:

docker swarm join --token aabbcc 192.168.0.1:2377
Kubernetes was created by Google and is one of the most feature-rich and widely used orchestration frameworks; its key features include:

- Automated deployment and replication of containers
- Online scale-in or scale-out of container clusters
- Load balancing over groups of containers
- Rolling upgrades of application containers
- Resilience, with automated rescheduling of failed containers
- Controlled exposure of network ports to systems outside of the cluster

Kubernetes is designed to work in multiple environments, including bare metal, on-premises VMs, and public clouds.

Most public clouds now have a managed Kubernetes offering.
Questions?
Agenda

• What is Kubernetes
• Kubernetes Objects
• Kubernetes Networking
• Kubernetes in Action
What is Kubernetes?

- **Container Orchestrator**
  - Provision, manage, scale applications

- Manage infrastructure resources needed by applications
  - Volumes
  - Networks
  - Secrets
  - And much much more ...

- **Declarative model**
  - Provide the "desired state" and Kubernetes will make it happen

- **What's in a name?**
  - Kubernetes (K8s/Kube): ‘helmsman’ in ancient Greek
How was Kubernetes created?
How was Kubernetes created?

• Based on Google’s Borg & Omega

• Open Governance
  • Cloud Native Compute Foundation

• Adoption by Enterprise
  • RedHat, Microsoft, VMware, IBM, and Amazon
At its core, Kubernetes is a database (*etcd*).

- The DB represents the user's desired state
- Watchers & controllers react to changes in the DB.
- Watchers attempt to make reality match the desired state
The API Server is the HTTP/REST frontend to the DB

(more on controllers later ...)
Kubernetes Architecture

• To work with Kubernetes, you use Kubernetes API objects to describe your cluster’s desired state: what applications or other workloads you want to run, what container images they use, the number of replicas, what network and disk resources you want to make available, and more.

• Once you’ve set your desired state, the Kubernetes Control Plane works to make the cluster’s current state match the desired state. (Source: kubernetes.io)
Kubernetes Resource Model

A resource for every purpose

- Config Maps
- Daemon Sets
- **Deployments**
- Events
- Endpoints
- Ingress
- Jobs
- Nodes
- Namespaces
- **Pods**
- Persistent Volumes
- Replica Sets
- Secrets
- Service Accounts
- **Services**
- Stateful Sets, and more...

- Kubernetes aims to have the building blocks on which you build a cloud native platform.

- Therefore, the internal resource model is the same as the end user resource model.

**Key Resources**

- Pod: set of co-located containers
  - Smallest unit of deployment
  - Several types of resources to help manage them
  - Replica Sets, Deployments, Stateful Sets, ...

- Services
  - Define how to expose your app as a DNS entry
  - Query based selector to choose which pods apply
Kubernetes Objects

The big ones:

• Pod
• Service
• Volume
• Namespace
Kubernetes Objects: pods

- The Pod is the core component of Kubernetes
- Collection of 1 or more containers
- Each pod should focus on one container, however sidecar containers can be added to enhance features of the core container

```
spec:
  template:
    spec:
      containers:
      - name: drupal
        image: cr.io/repo/mydrupal:v1
```
Kubernetes Objects: pods

• A pod is a running process on your cluster.
  • the smallest, simplest unit
  • may be one container or multiple (tightly-coupled) containers
  • often Docker containers, but k8s supports other container runtimes

• Each pod:
  • has a unique IP
  • can have storage volumes shared with the whole pod
  • is managed by controllers
  • controllers use pod templates to make pods
Once Kubernetes understands what is in a pod, multiple management features are available:

- **System Performance**
  - Scale up/down the number of pods based on CPU load or other criteria

- **System Monitoring**
  - Probes to check the health of each pod
  - Any unhealthy ones get killed and new pod is put into service

- **Deployments**
  - Deploy new versions of the container
  - Control traffic to the new pods to test the new version
    - Blue/Green deployments
    - Rolling deployments
Kubernetes Objects: services

• A logical set of pods
  • like a microservice
  • how groups of pods find each other (e.g., how frontends find backends)

• Services:
  • have their IPs (sometimes called cluster-IP) managed by Kubernetes (although you can manage it yourself)
  • can provide access to pods, but also to outside services
  • are discoverable, and create environment variables
Kubernetes Objects: services

- Kubernetes Services are used to control communications with the pods
  - Load balance the requests
  - Don’t send traffic to the unhealthy ones
  - Only talk to the correct version

```
apiVersion: v1
kind: Service
metadata:
  name: drupal
spec:
  selector:
    app: drupal
  ports:
  - name: http-port
    port: 80
  type: LoadBalancer
```
Kubernetes Objects: services

- With the Service architecture Kubernetes handles things that you often might have to worry about
  - Service discovery
  - Load balancing
  - Scaling

- Service discovery allows each pod just needs to call the name of the service it wants to talk to

- Services have multiple options
  - Session based load balancing
  - Single port based services
  - External Services

- The Service architecture of Kubernetes can be scaled up to handle as many services as you would like for your system
Kubernetes Objects: volumes

• a directory, usually with data in it, accessible to containers in a pod
• each container in the pod must specify what volumes and where to mount them
• lots of different kinds of volumes, some provider-specific!
Kubernetes Objects: namespaces

- multiple virtual clusters in the same physical cluster
- provide scope for names: names in each namespace are unique
- can be used to divide resources across users
- you don’t need namespaces just to differentiate different versions of the same software: use labels instead
Kubernetes Controllers

- **Deployments**
  - use instead of ReplicaSets
  - describe the desired state, and the controller makes it happen in a controlled way
  - can be versioned for easy rollback
  - give status updates for monitoring *(progressing, complete, fail to progress)*

- **StatefulSets**
  - manages the deployment and scaling of a set of pods, *and provides guarantees about the ordering and uniqueness* of these pods
  - use when you need stable ids, persistent storage, or ordered creation or deletion

- **DaemonSets**
  - runs a copy of a pod in all nodes
  - used for logging, etc.

- **Jobs**
  - creates one or more pods, that run and terminate
Kubernetes Client

• CLI tool to interact with Kubernetes cluster

• Platform specific binary available to download
  • https://kubernetes.io/docs/tasks/tools/install-kubectl

• The user directly manipulates resources via json/yaml

  $ kubectl (create|get|apply|delete) -f myResource.yaml
# Kubernetes Networking

<table>
<thead>
<tr>
<th>Type/Features</th>
<th>L2</th>
<th>L3</th>
<th>Overlay</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>Pods communicate using Layer 2</td>
<td>Pod traffic is routed over underlay network</td>
<td>Pod traffic is encapsulated in an overlay network and uses the underlay network</td>
<td>Pod traffic is routed in a cloud virtual network</td>
</tr>
<tr>
<td><strong>Underlying Technology</strong></td>
<td>L2 ARP, Broadcast</td>
<td>Routing protocols such as BGP</td>
<td>VXLAN</td>
<td>Cloud fabric</td>
</tr>
<tr>
<td><strong>Encapsulation</strong></td>
<td>None</td>
<td>None</td>
<td>Overlay</td>
<td>None</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>Calico</td>
<td>Flannel, Weave</td>
<td>GKS, ACS, AKS</td>
<td></td>
</tr>
</tbody>
</table>
# Kubernetes Networking

<table>
<thead>
<tr>
<th>Features</th>
<th>NodePort</th>
<th>Load Balancer</th>
<th>Ingress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Service is exposed using a reserved port in all nodes of cluster (Default: 32000-32767)</td>
<td>Typically implemented as network load balancer</td>
<td>Typically implemented as http load balancer</td>
</tr>
<tr>
<td>IP Address</td>
<td>Node IP is used for external communication</td>
<td>Each service needs to have own external IP</td>
<td>Many services can share same external IP, uses path based demux</td>
</tr>
<tr>
<td>Use Case</td>
<td>Testing</td>
<td>L3 Services</td>
<td>L7 Services</td>
</tr>
<tr>
<td>Examples</td>
<td>GKE Network Load balancer</td>
<td>Nginx, Contour</td>
<td></td>
</tr>
</tbody>
</table>
NodePort

```
apiVersion: v1
kind: Service
metadata:
  name: productpage
labels:
  app: productpage
spec:
  type: NodePort
  ports:
  - port: 30000
    targetPort: 9080
selector:
  app: productpage
```
Load Balancer

apiVersion: v1
description: Service
metadata:
  name: productpage
labels:
  app: productpage
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 9080
  selector:
    app: productpage
Load Balancer

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gateway
spec:
  backend:
    serviceName: productpage
    servicePort: 9080
  rules:
  - host: mydomain.com
    http:
      paths:
      - path: /productpage
        backend:
          serviceName: productpage
          servicePort: 9080
```
Under the covers

1. User via "kubectli" deploys a new application
2. API server receives the request and stores it in the DB (etcd)
3. Watchers/controllers detect the resource changes and act upon it
4. **ReplicaSet** watcher/controller detects the new app and creates new pods to match the desired # of instances
5. Scheduler assigns new pods to a **kubelet**
6. **Kubelet** detects pods and deploys them via the container running (e.g. Docker)
7. **Kubeproxy** manages network traffic for the pods – including service discovery and load-balancing
Kubernetes in Action!
Where is Kubernetes?

- Main Website - http://kubernetes.io
- Source Code - https://github.com/kubernetes
- YouTube - https://www.youtube.com/channel/UCZ2bu0qutTOM0tHYa_jklwg
Getting Kubernetes

- Azure Kubernetes Service
- Amazon Elastic Container Service for Kubernetes
- Google Kubernetes Engine
- Digital Ocean Kubernetes
Cluster-API

• A management framework to handle day 1 and 2 operations for Kubernetes cluster
  • Day 1) Bringing up a cluster
    • Solves from 0 to Kubernetes
  • Day 2) Managing a cluster
    • Managing in an idiomatic Kubernetes way
    • Upgrades
    • Scaling
  • Standardizing a fragmented ecosystem
    • Many tools, all with varying scope and user experience
• Experimenting still!
Kubevirt

Leverage KubeVirt and Kubernetes to manage virtual machines for impractical-to-containerize apps.

Combine existing virtualized workloads with new container workloads on the one platform.

Support development of new microservice applications in containers that interact with existing virtualized applications.
CNCF

* Cloud Native Computing Foundation
Questions?
Thankyou !