heptio

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

heptio

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

Where did Docker come from?

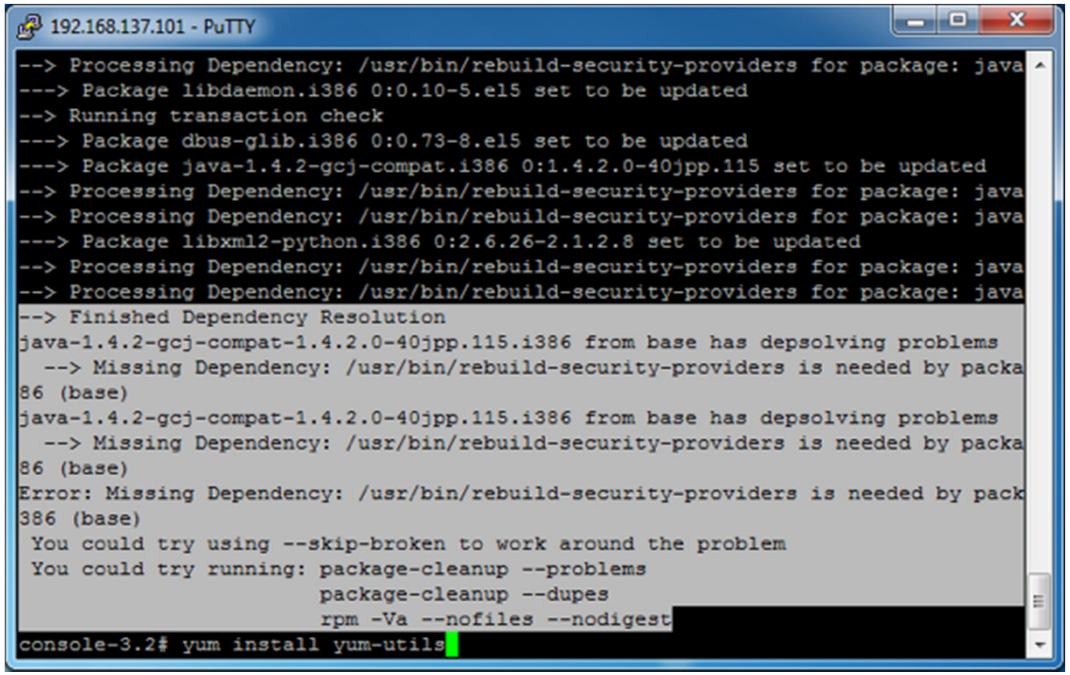
heptio

In the beginning ...

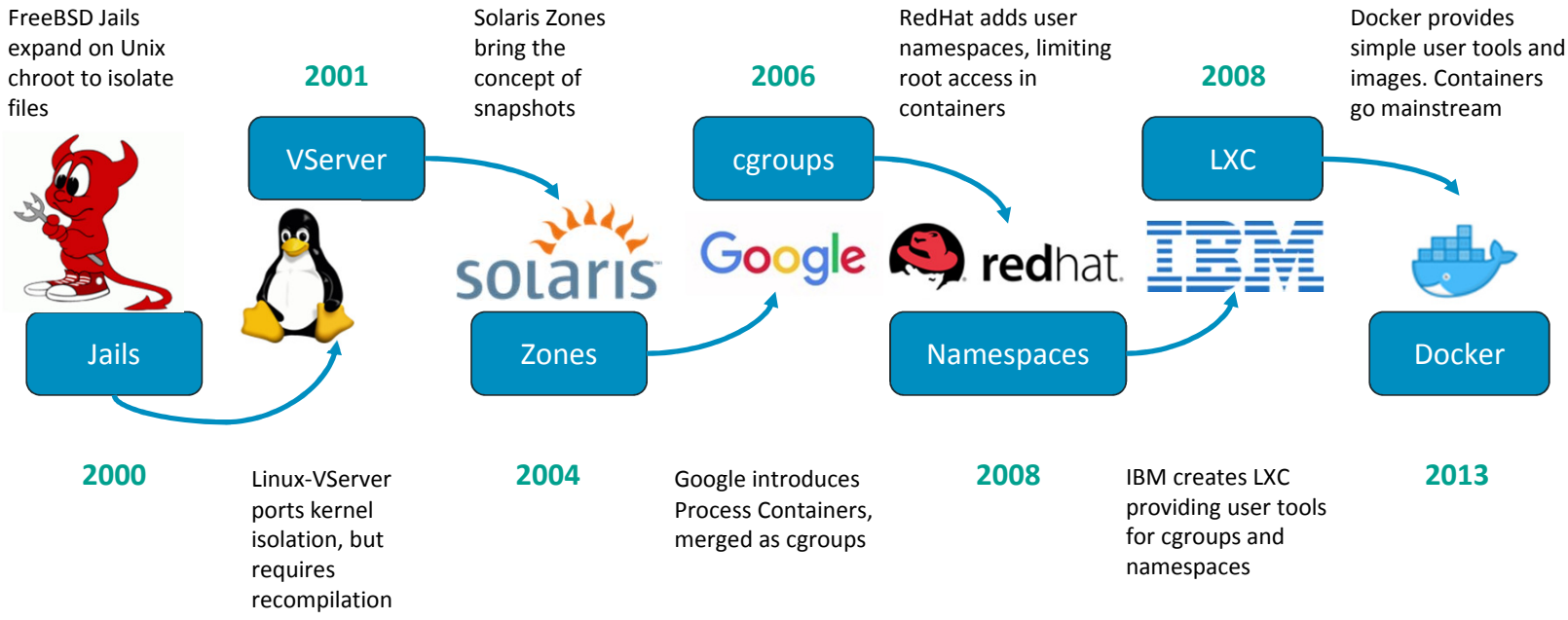
Where did Docker come from?

heptio

Things broke a lot ...

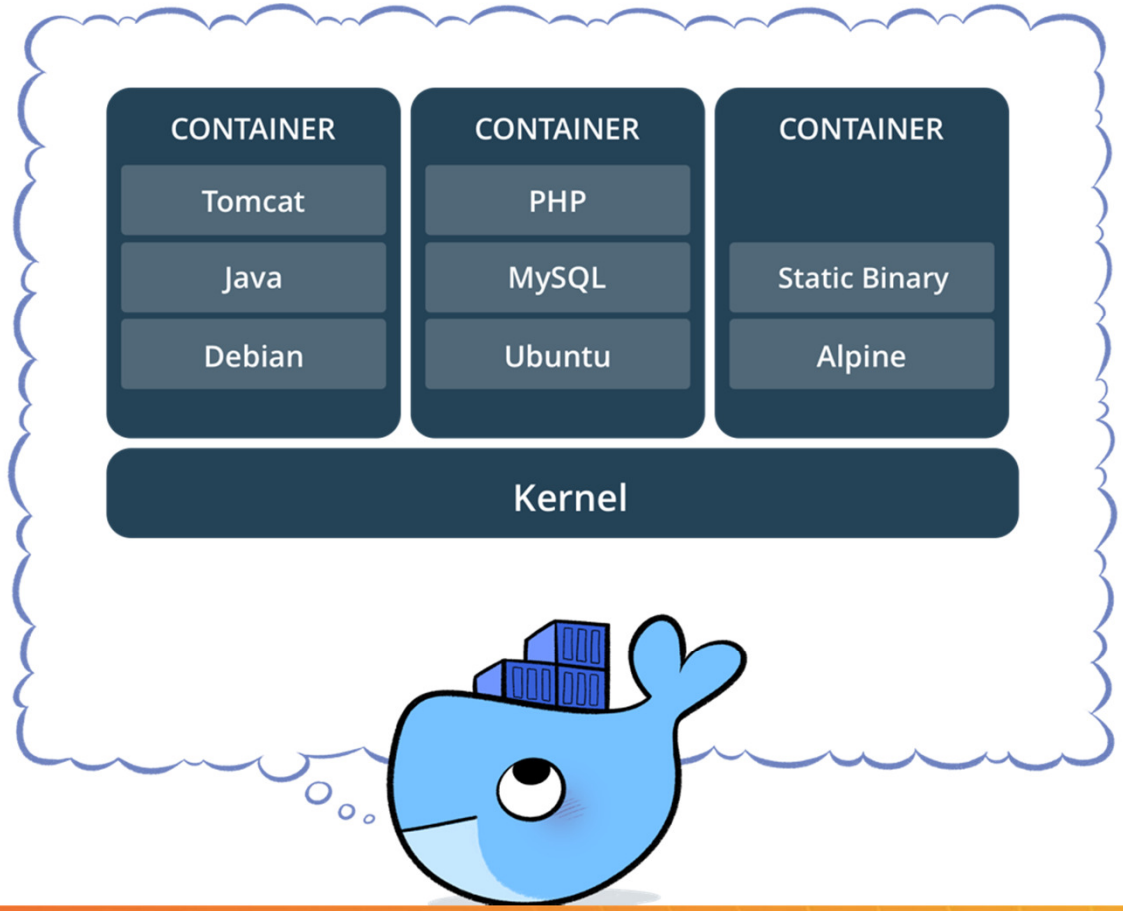


Where did Docker come from?



Docker under the hood

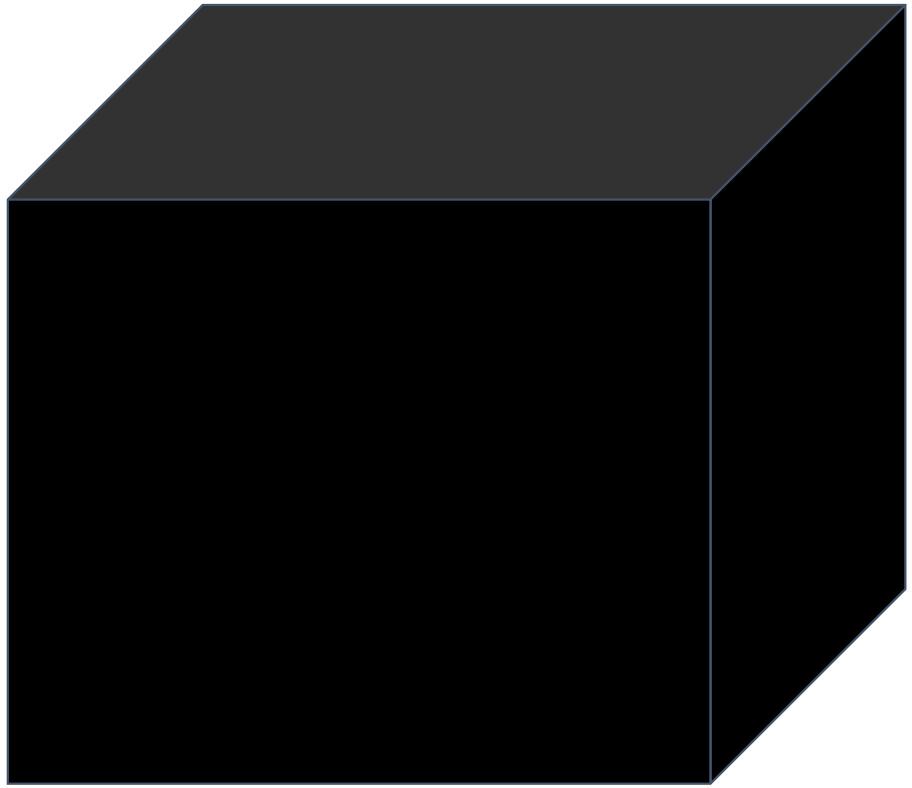
heptio



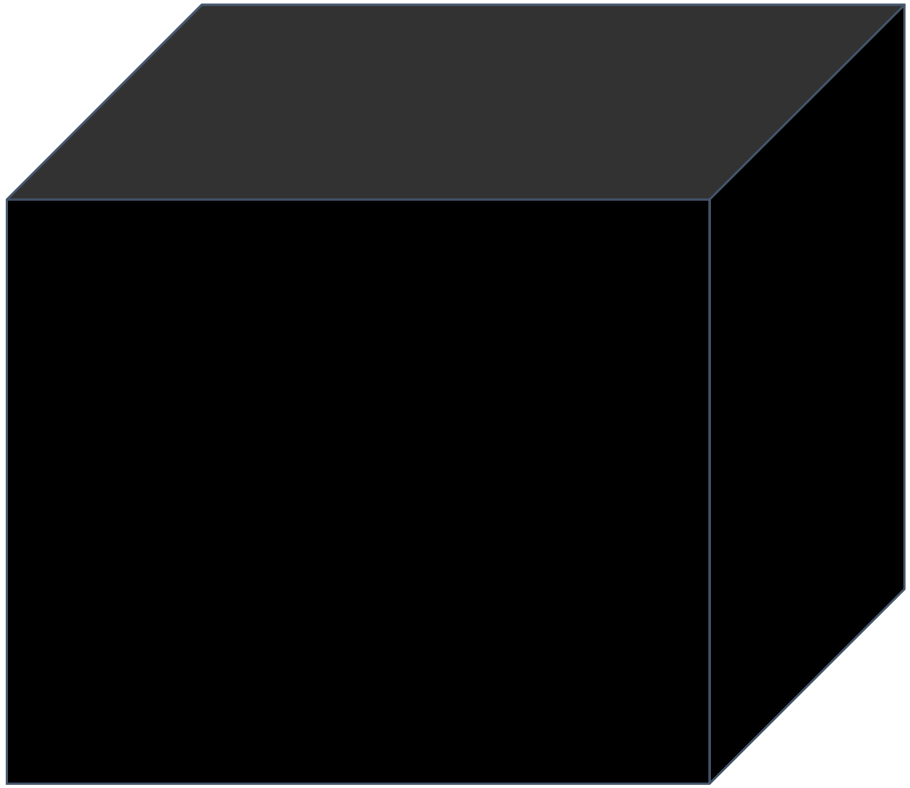
Docker Installation

heptio

- 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`

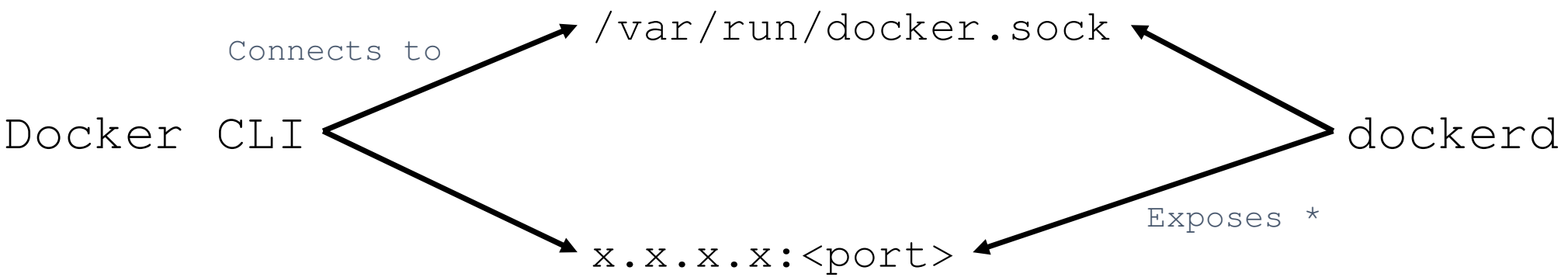
Docker Command Line Interface

- `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

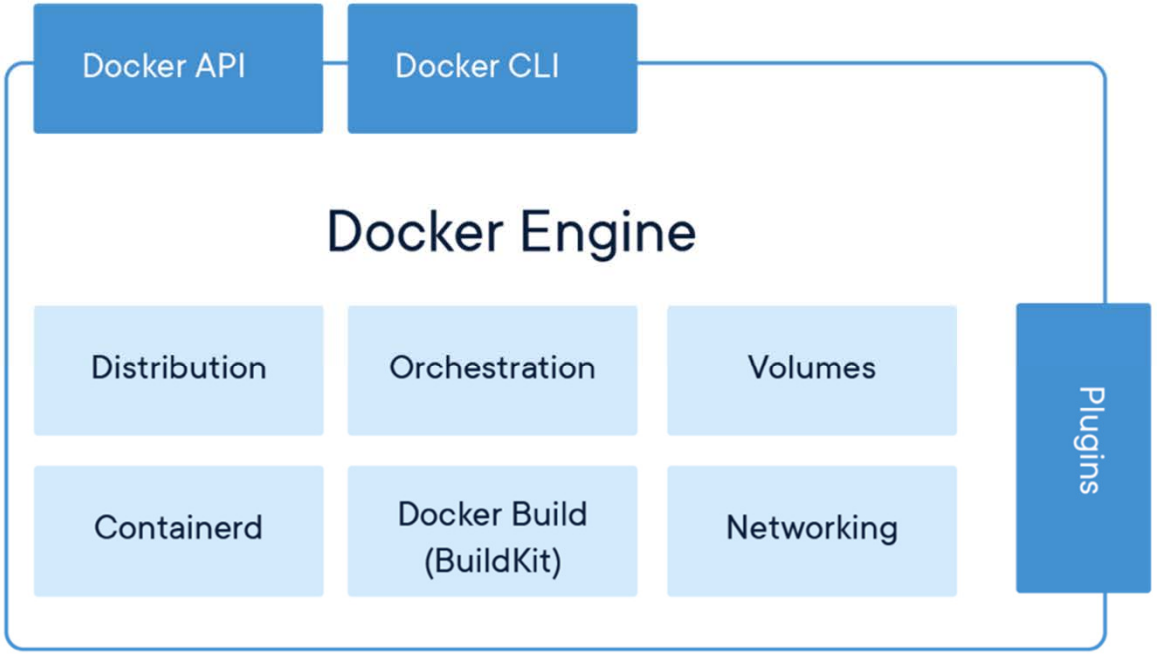
heptio

```
[dan@docker ~]$ docker run
```



* Configured in /etc/docker/json

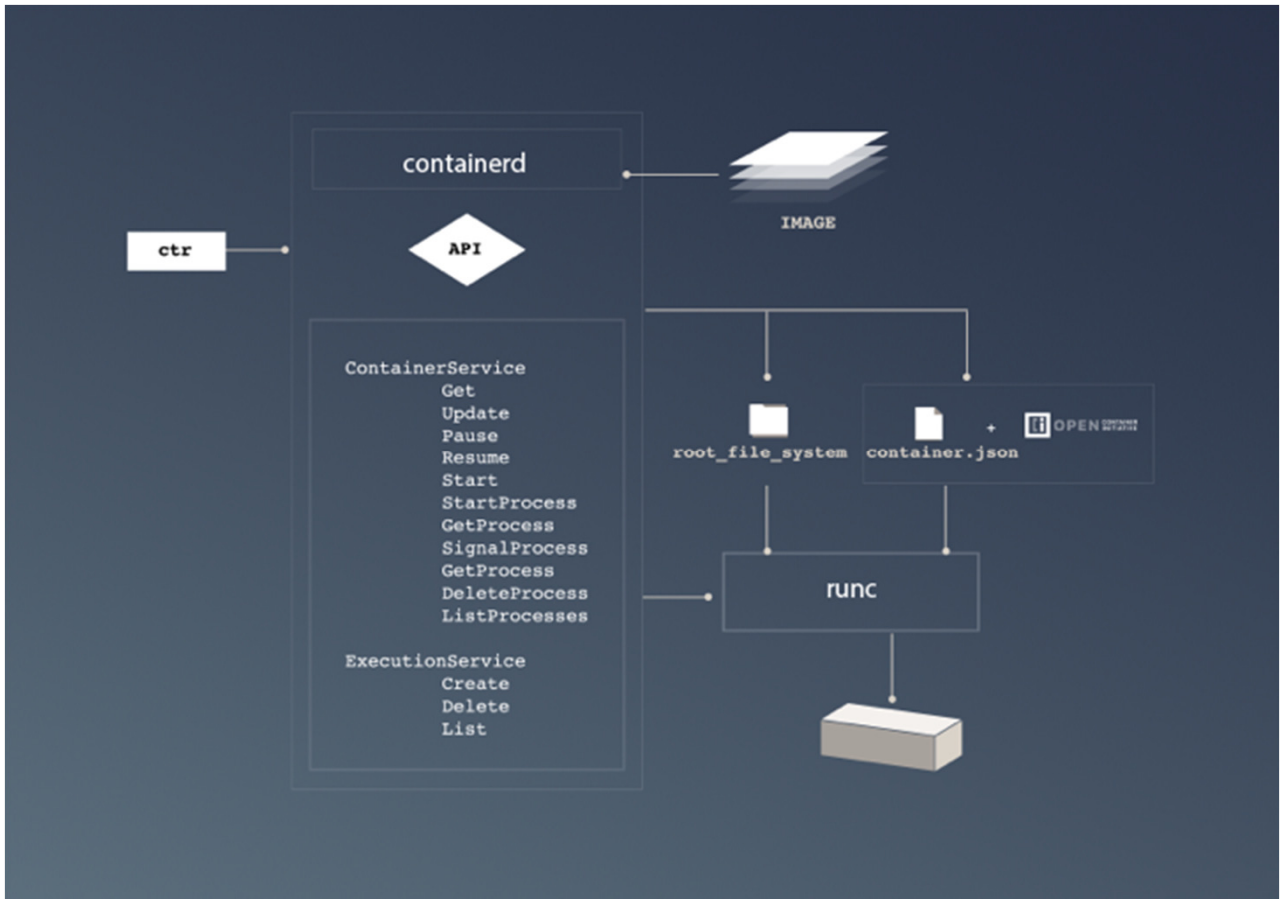
Docker Daemon



- 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

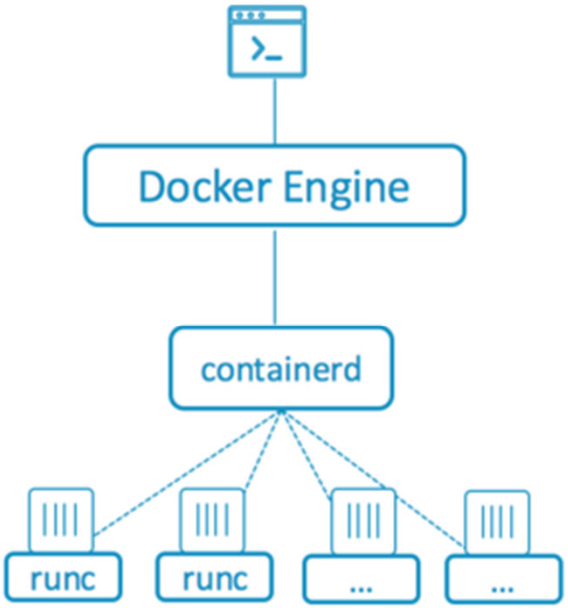


runc (or any OCI specific runtime)

heptio

- 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



Same Docker UI and commands

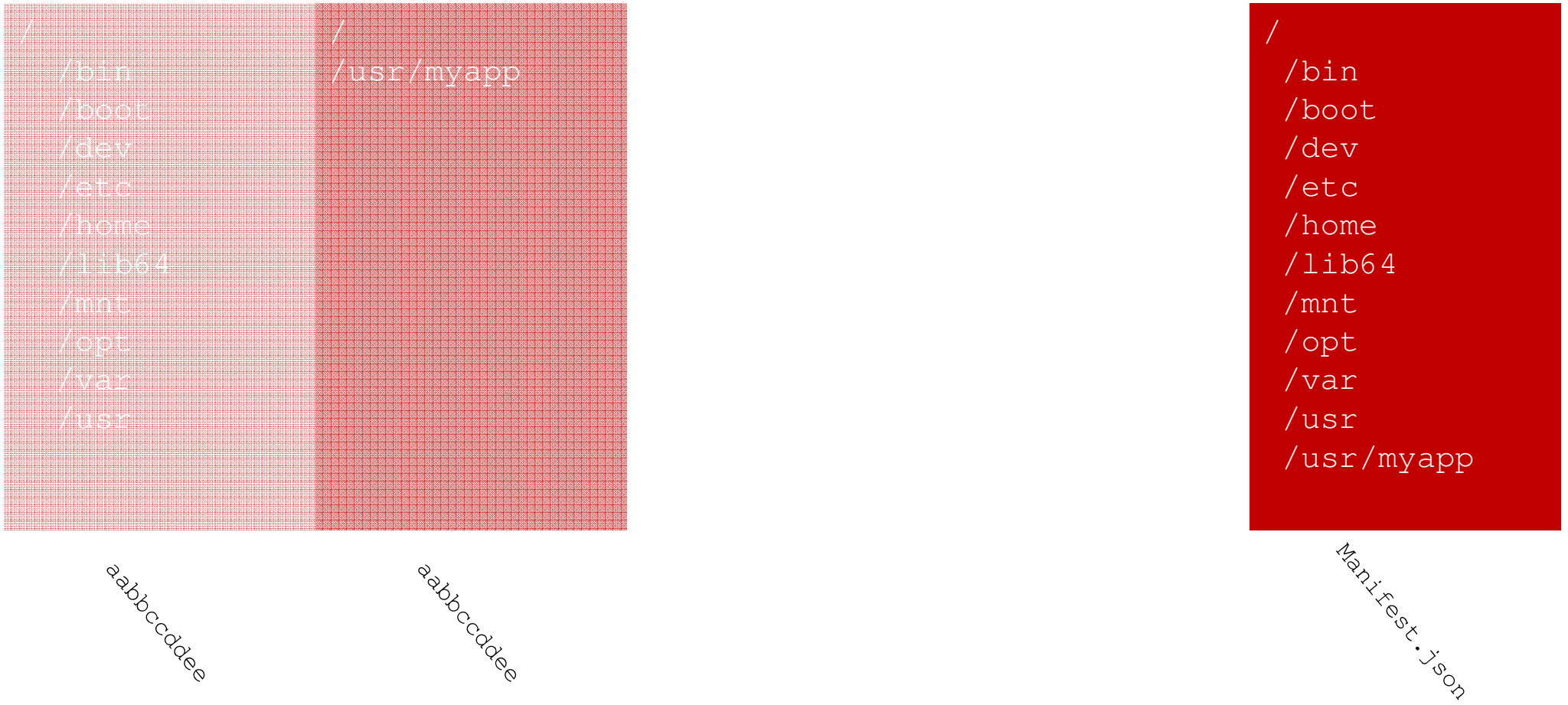
User interacts with the Docker Engine

Engine communicates with containerd

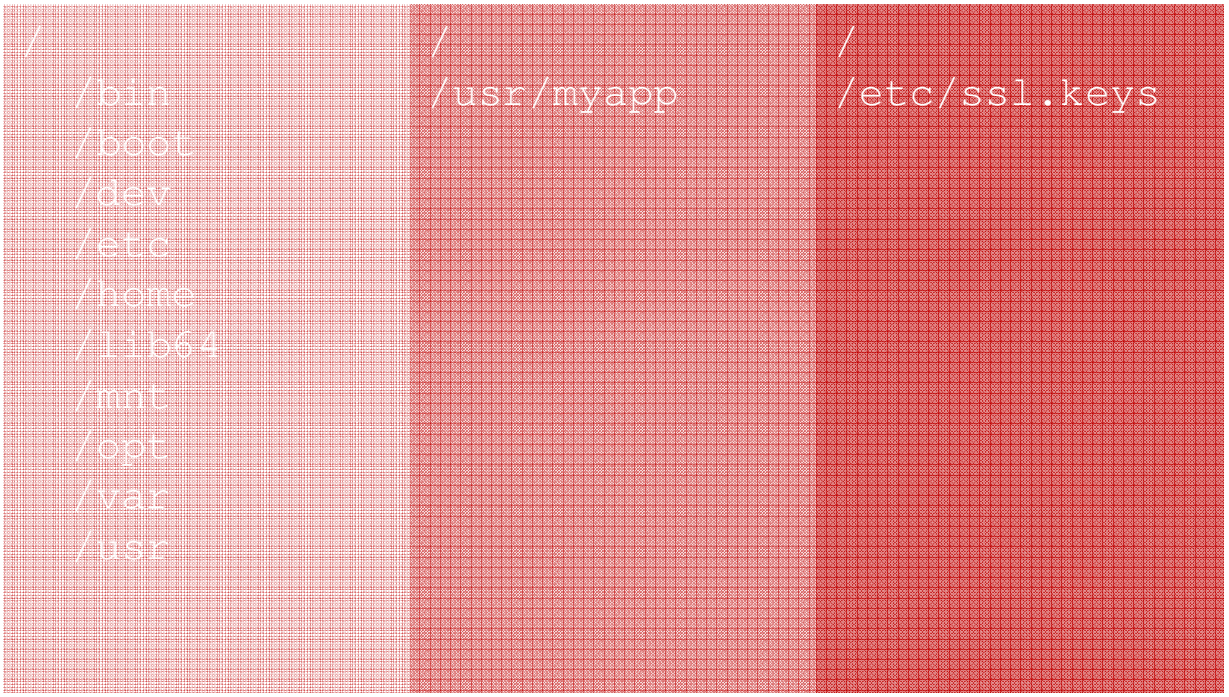
containerd spins up runc or other OCI compliant runtime to run containers

Docker Image

Docker Image



Docker Image



aabbccddee

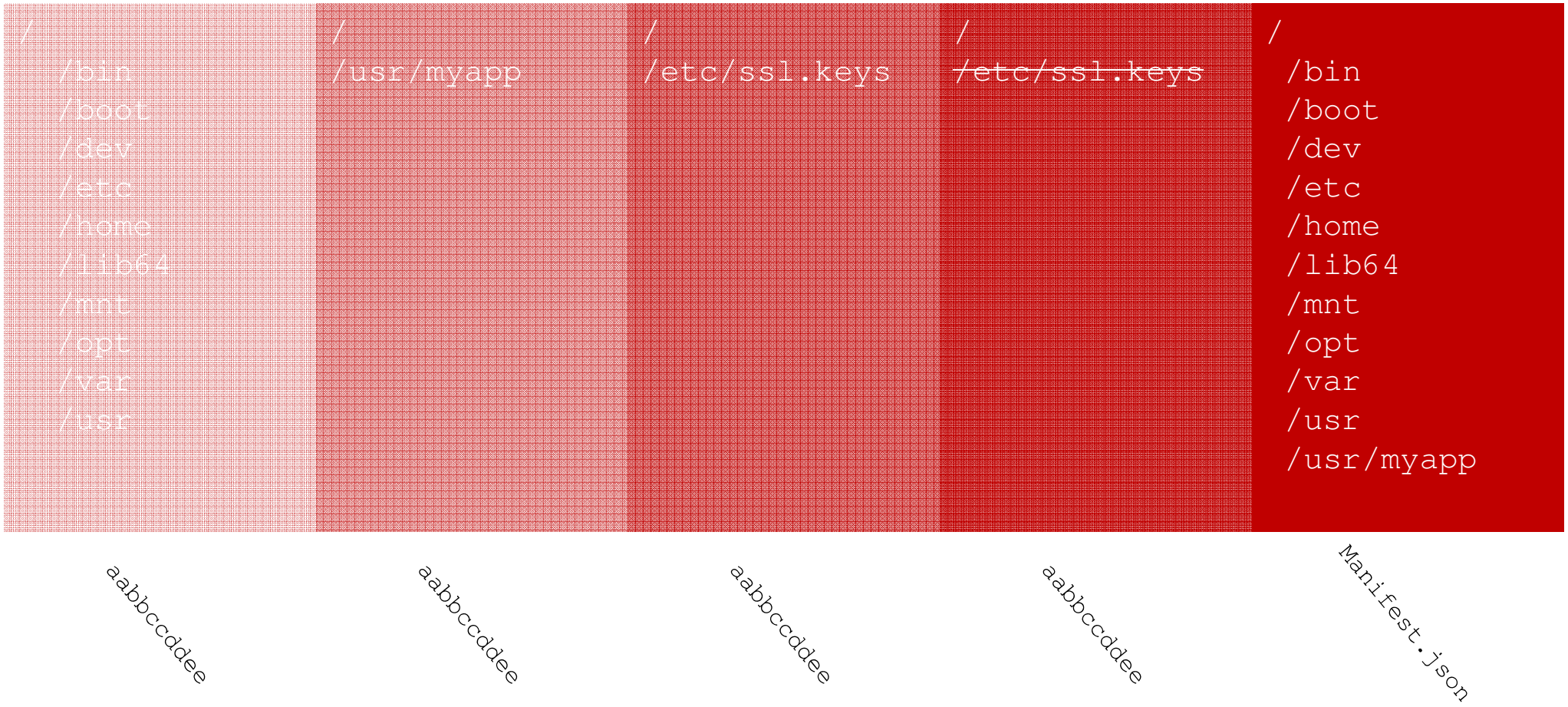
aabbccddee

aabbccddee

```
/
 /bin
 /boot
 /dev
 /etc
 /etc/ssl.keys
 /home
 /lib64
 /mnt
 /opt
 /var
 /usr
 /usr/myapp
```

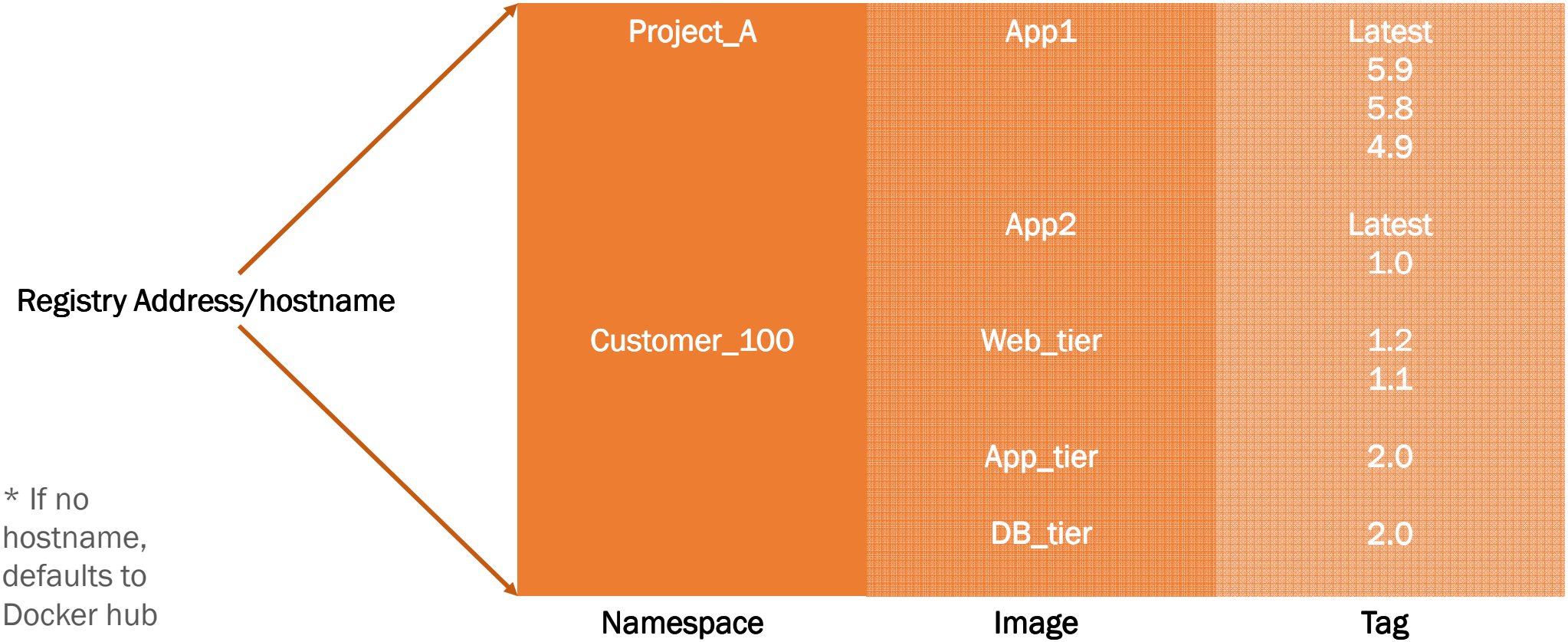
Manifest.json

Docker Image



Container Registry

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



* If no hostname, defaults to Docker hub

Pulling an image from a Registry

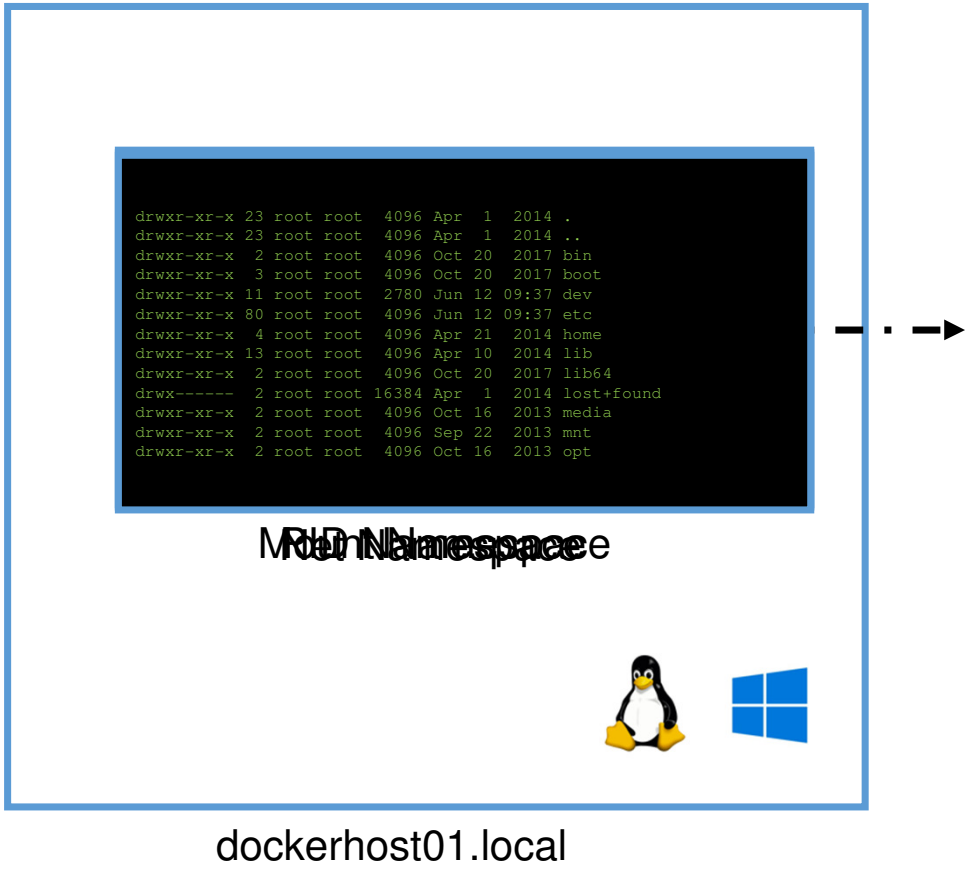
heptio

```
[dan@docker ~]$ docker pull quay.io/customer_a/web_tier:1.0

[dan@docker ~]$ docker pull customer_b/app_tier:2.0
```

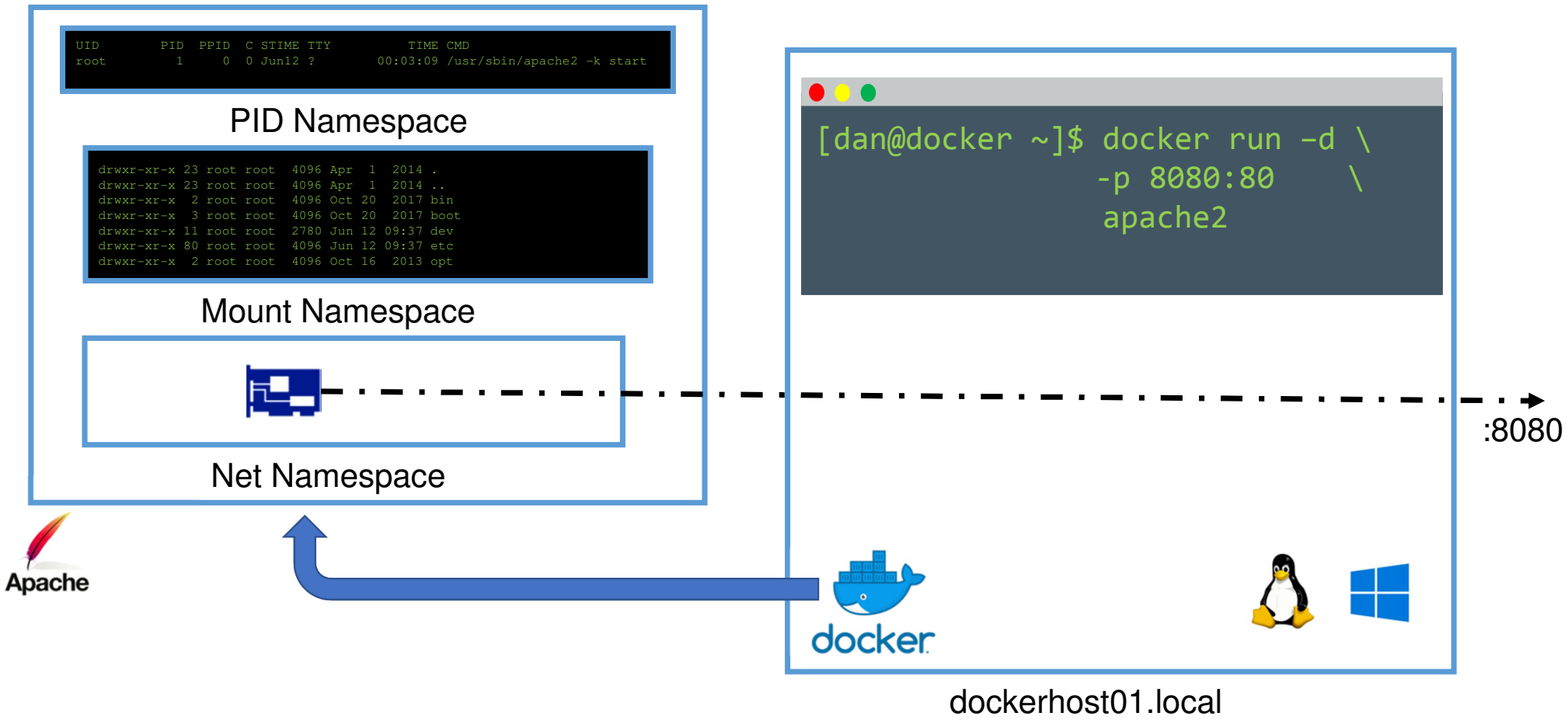
* Above specifies the project(namespace), the image and the specific tag (version) of the image to be pulled.

What happens if no tag is specified?

What is a container?



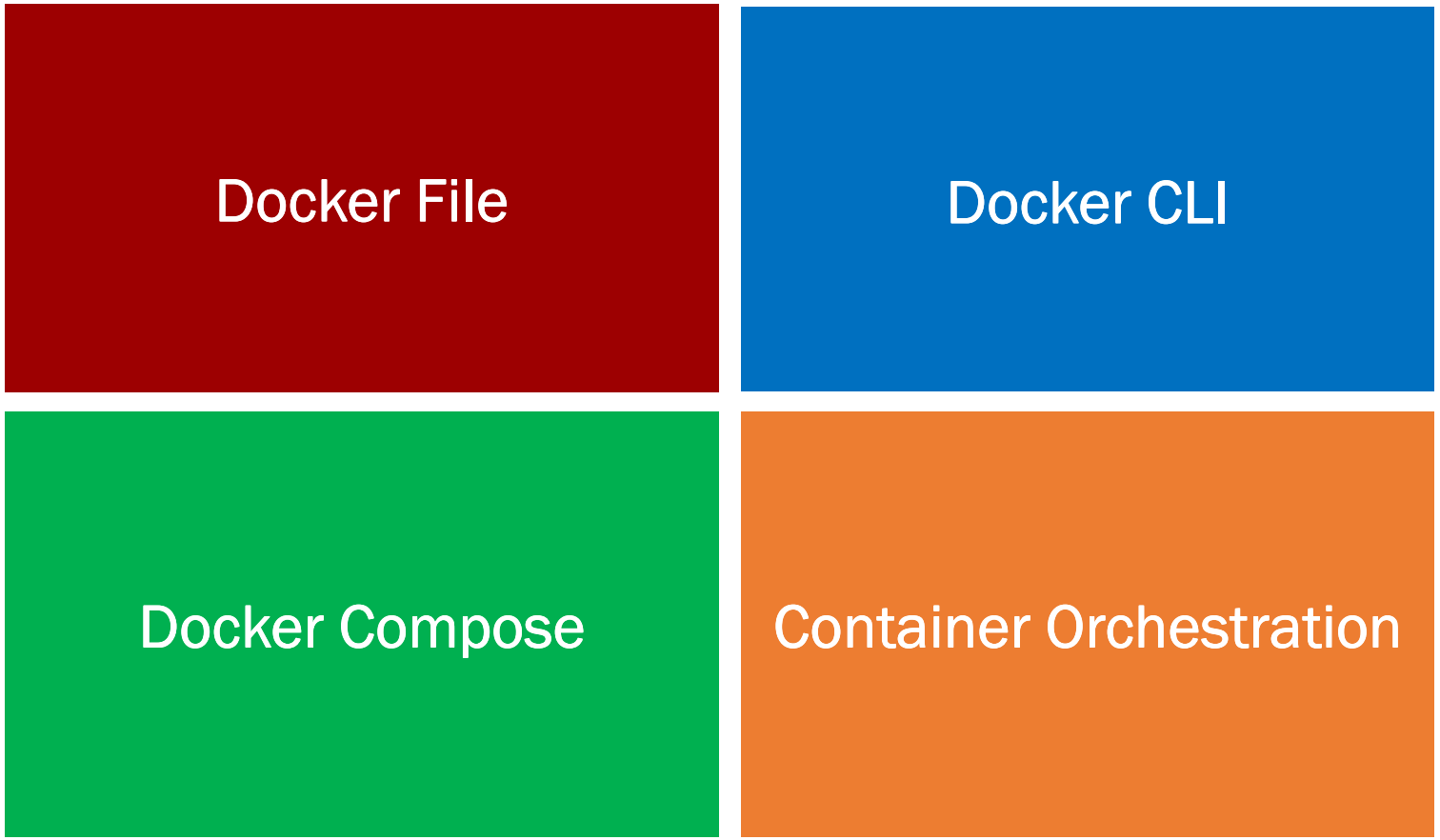
What is a container?



Using Docker

heptio

Using Docker



Dockerfile

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

heptio

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

```
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

heptio

WORKDIR sets the current working directory for any subsequent commands

```
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

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.

```
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)

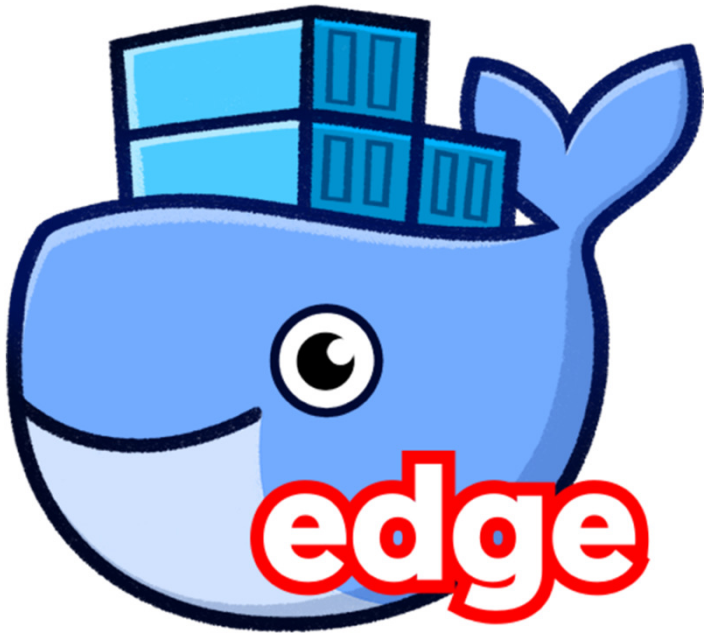
heptio

```
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.

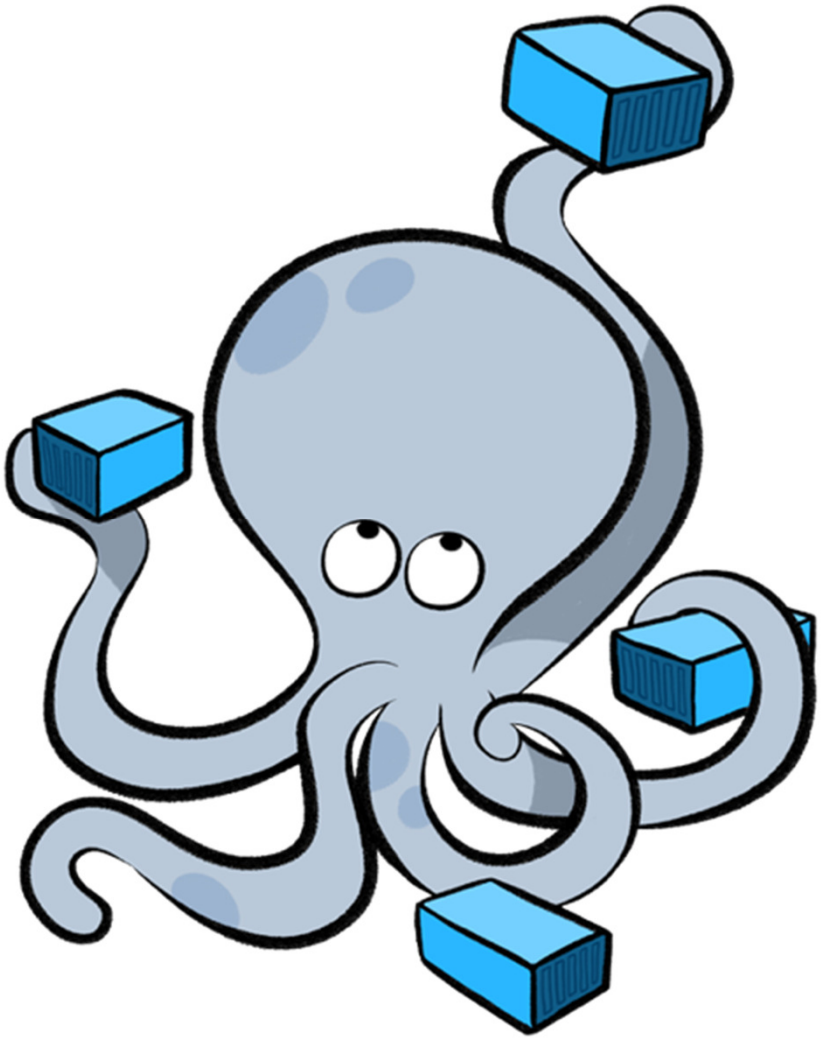
Some **CLI** commands, such as the **stacks** and **service** commands will only appear once a node or multiple nodes are in a cluster (also only master nodes will be able to manage the cluster)

Docker Compose

heptio

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.



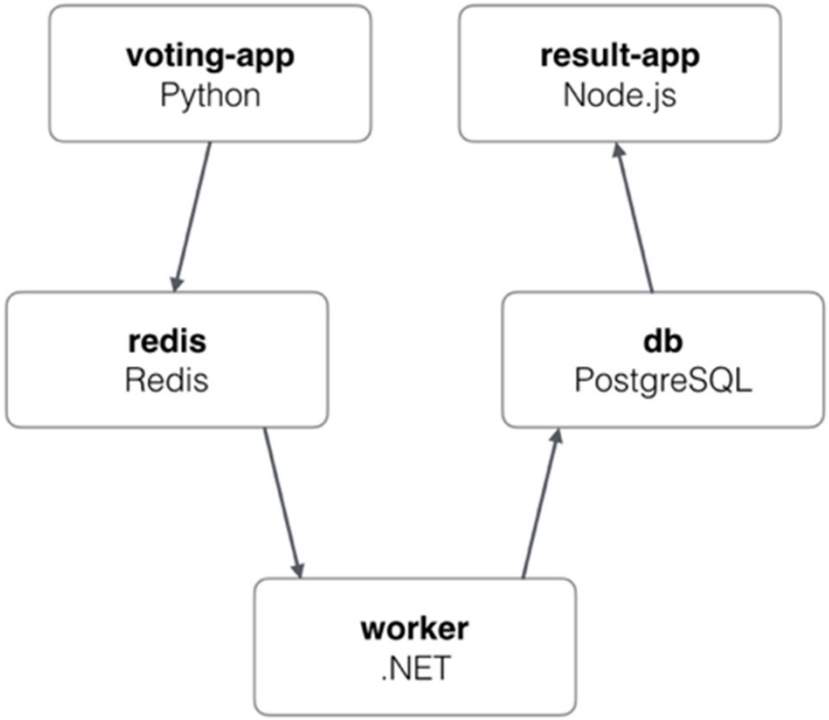
Docker Compose

heptio

```
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

heptio



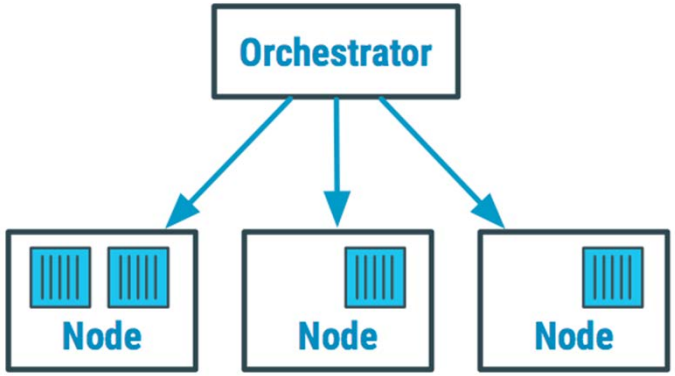
https://github.com/dockersamples/example-voting-app

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

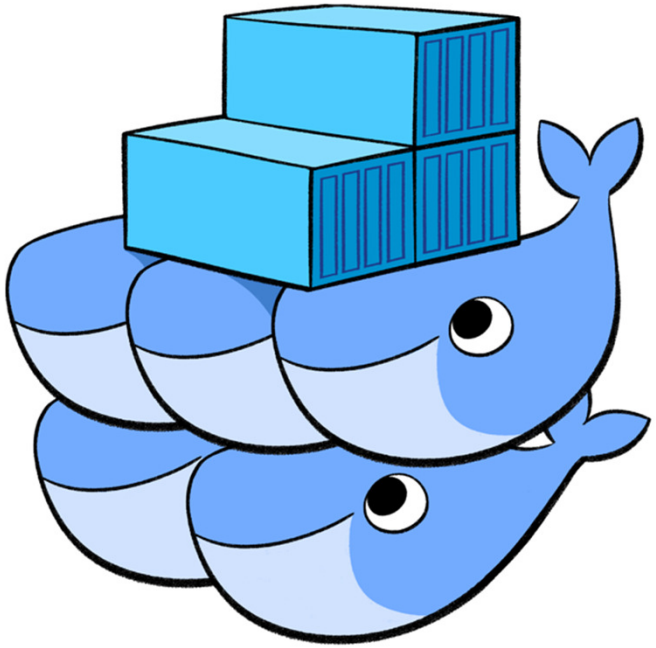


Container Orchestrators

heptio

Docker Swarm

heptio



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

heptio

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.

heptio

Questions?

Agenda

heptio

- What is Kubernetes
- Kubernetes Objects
- Kubernetes Networking
- Kubernetes in Action

What is Kubernetes?

heptio

- 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?

heptio

- Based on Google's Borg & Omega
- Open Governance
  - Cloud Native Compute Foundation
- Adoption by Enterprise
  - RedHat, Microsoft, VMware, IBM, and Amazon

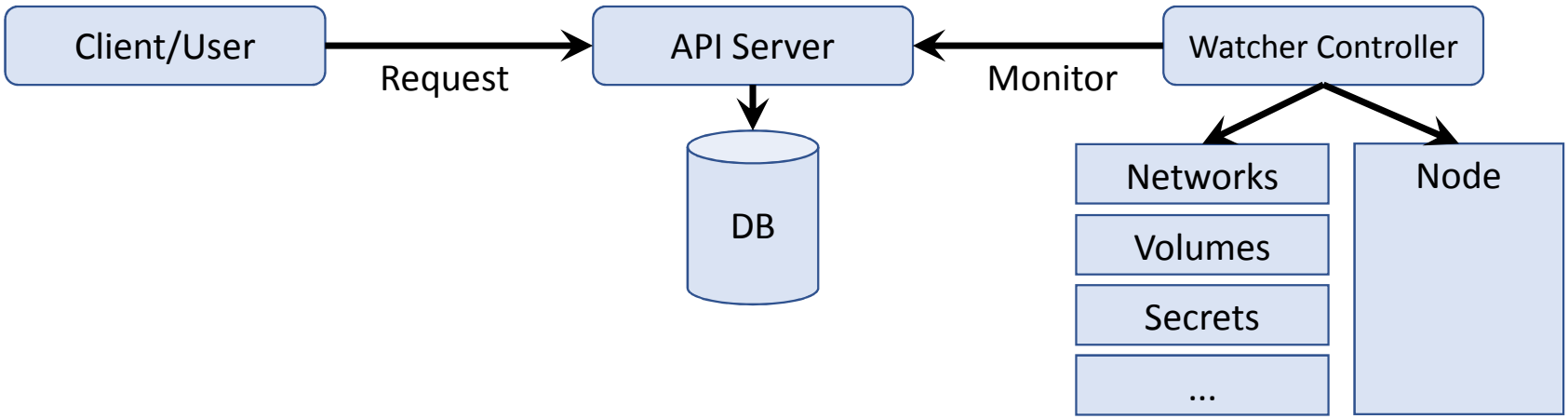
Kubernetes Architecture

heptio

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

Kubernetes Architecture



The API Server is the HTTP/REST frontend to the DB

(more on controllers later ...)

Kubernetes Architecture

heptio

- 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

heptio

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

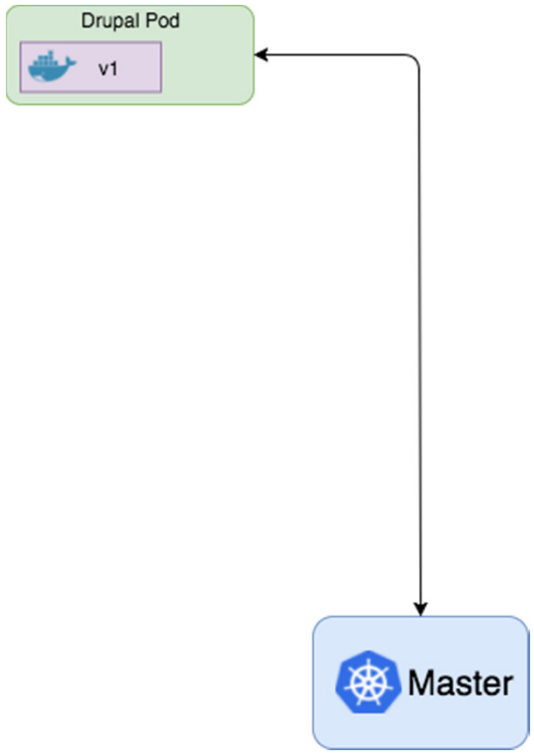
Kubernetes Objects

heptio

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
```

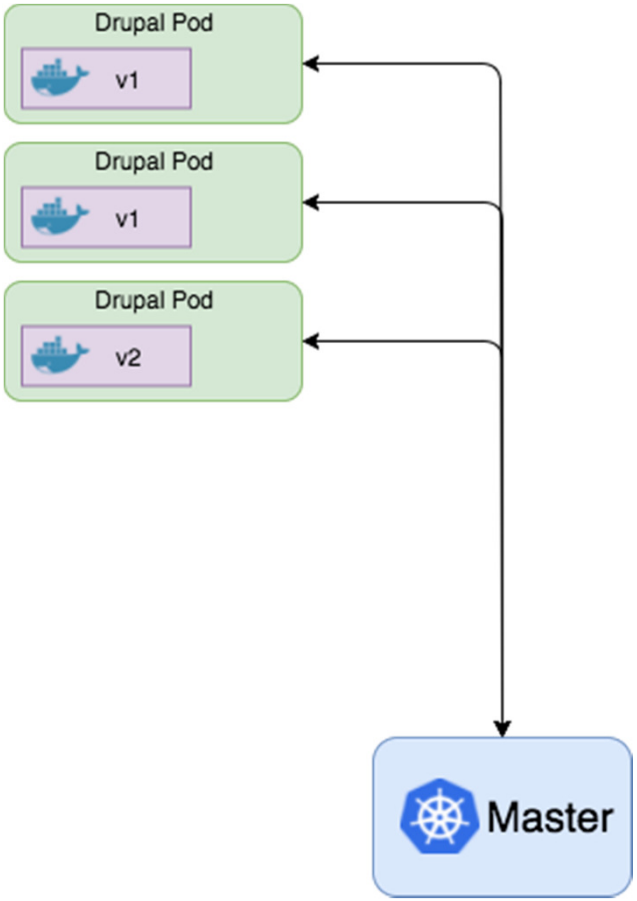
Kubernetes Objects: pods

heptio

- 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

Kubernetes Objects: 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

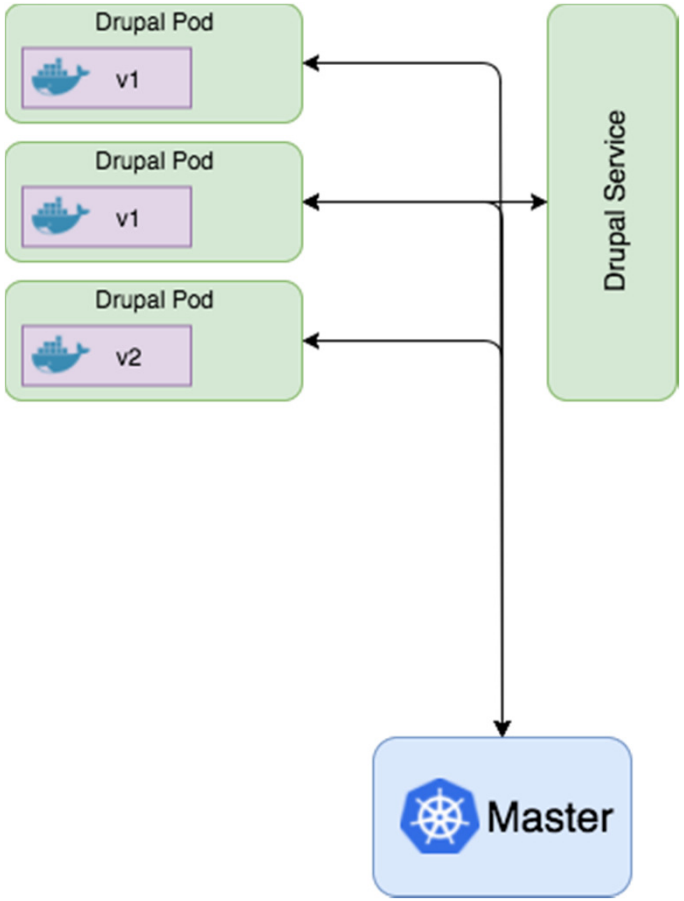
Kubernetes Objects: services

heptio

- 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

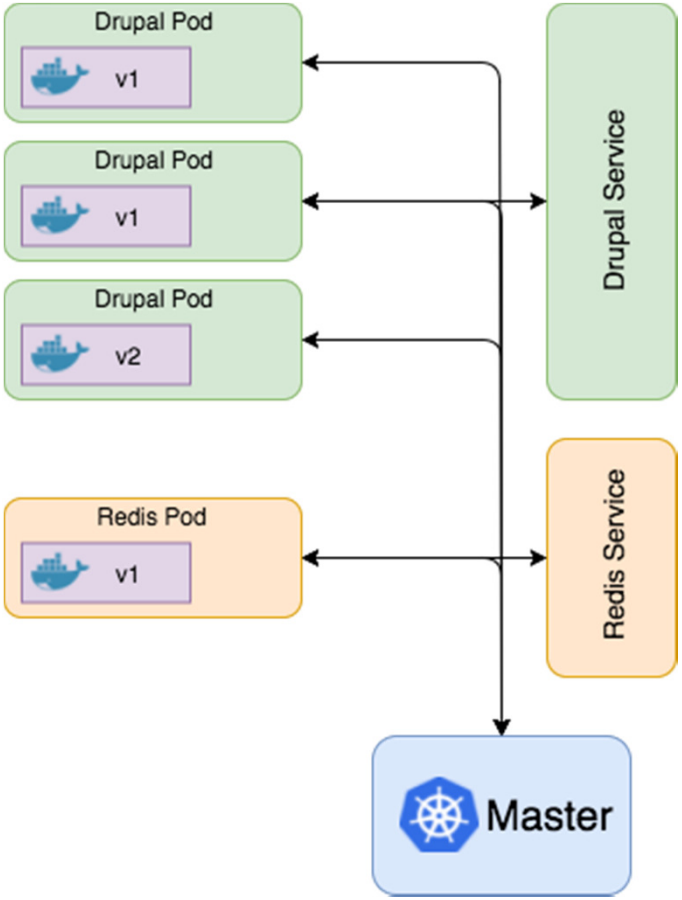
Kubernetes Objects: services



- 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

heptio

- 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

heptio

- 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

heptio

- 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

heptio

- 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

heptio

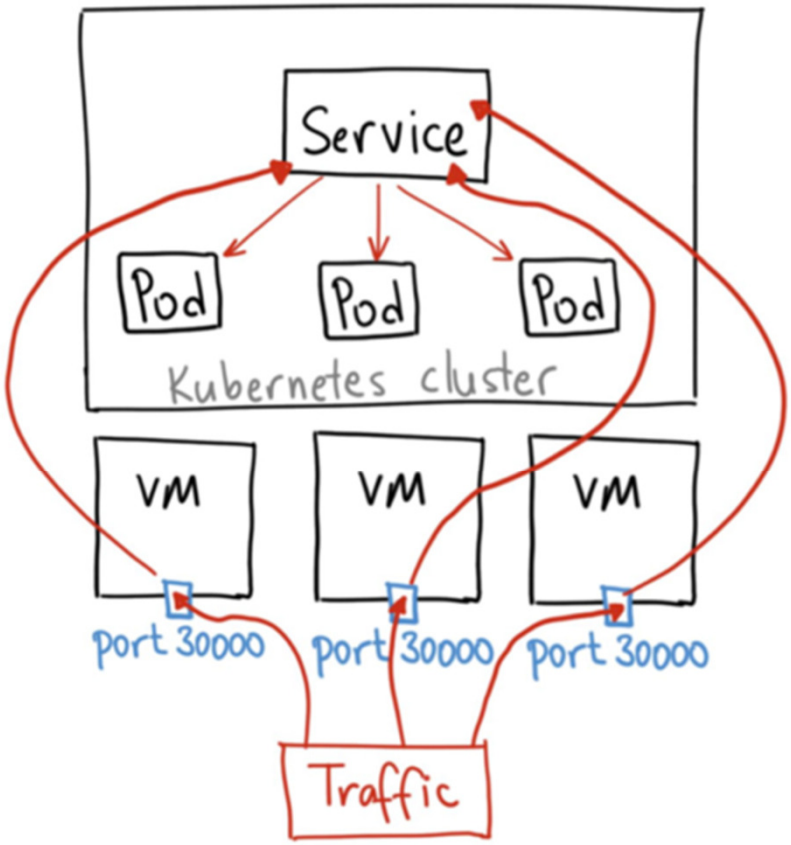
Type/Features	L2	L3	Overlay	Cloud
Summary	Pods communicate using Layer 2	Pod traffic is routed over underlay network	Pod traffic is encapsulated in an overlay network and uses the underlay network	Pod traffic is routed in a cloud virtual network
Underlying Technology	L2 ARP, Broadcast	Routing protocols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

Kubernetes Networking

heptio

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

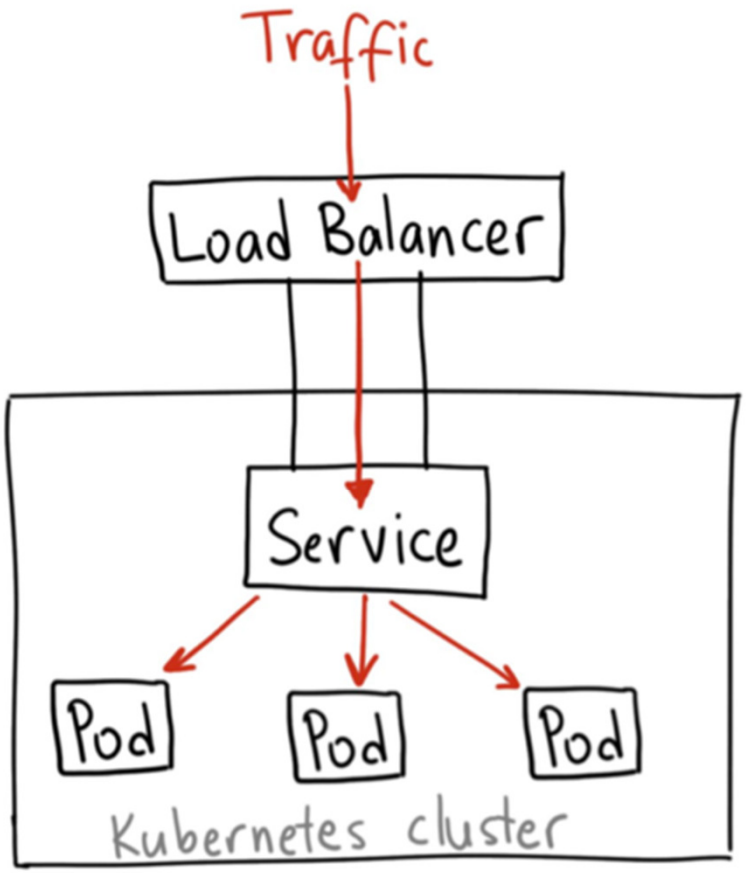
NodePort




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

heptio

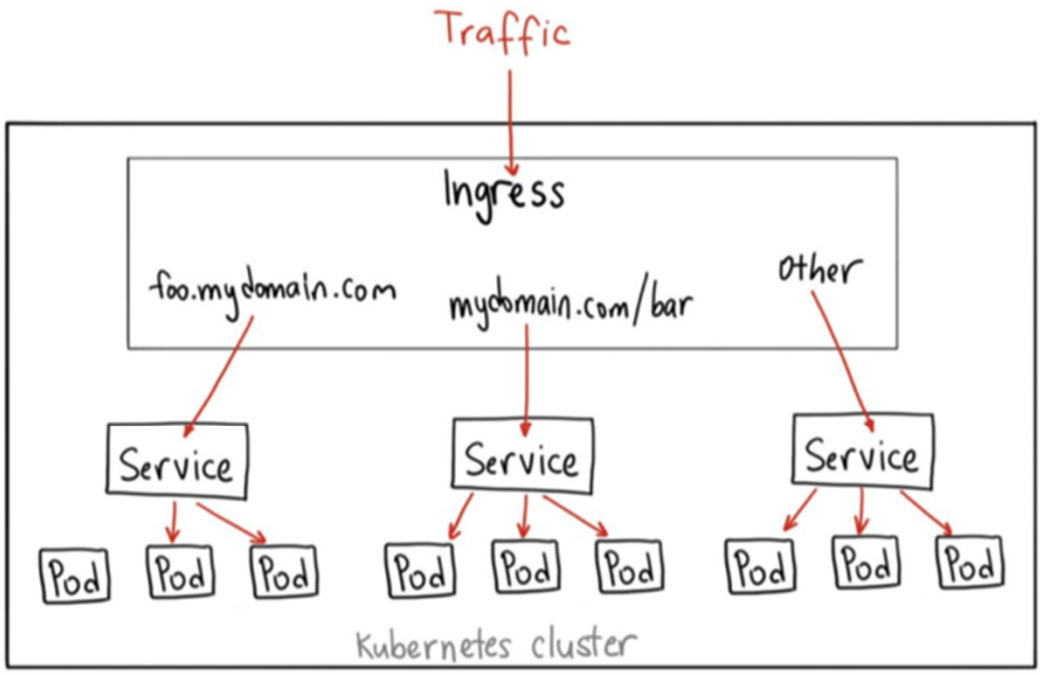
Load Balancer




```
apiVersion: v1
kind: Service
metadata:
  name: productpage
  labels:
    app: productpage
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 9080
selector:
  app: productpage
```

heptio

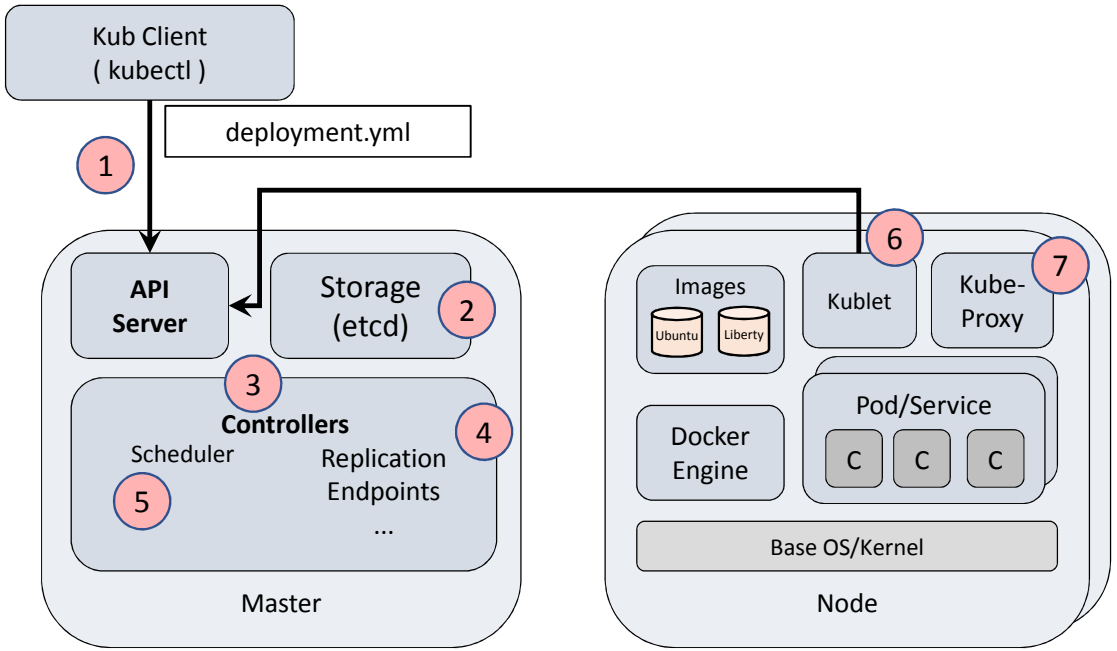
Load Balancer




```
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 "**kubectl**" 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



heptio

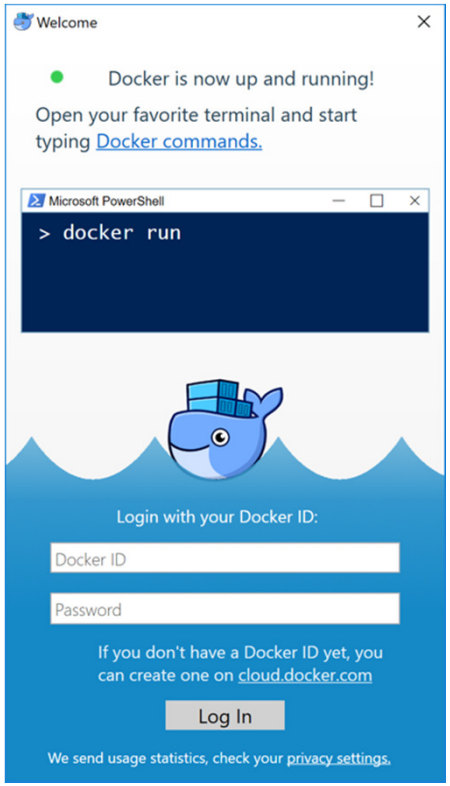
Kubernetes in Action!

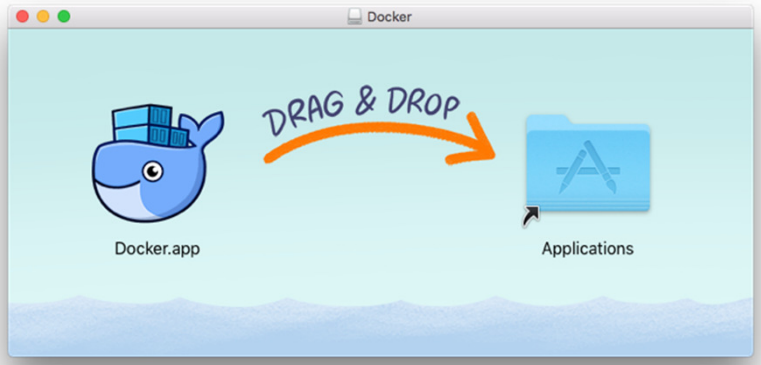
Where is Kubernetes?

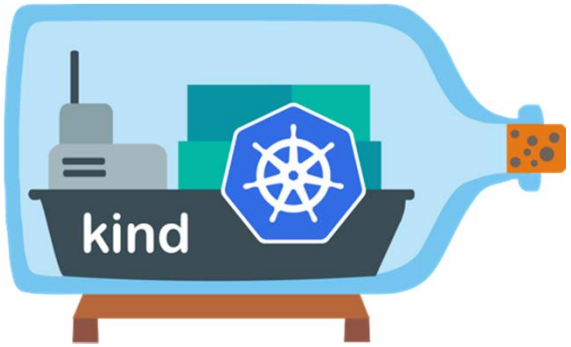
heptio

- Main Website - http://kubernetes.io
- Source Code - https://github.com/kubernetes
- YouTube - https://www.youtube.com/channel/UCZ2bu0qutTOM0tHYa_jkIwg

Getting Kubernetes





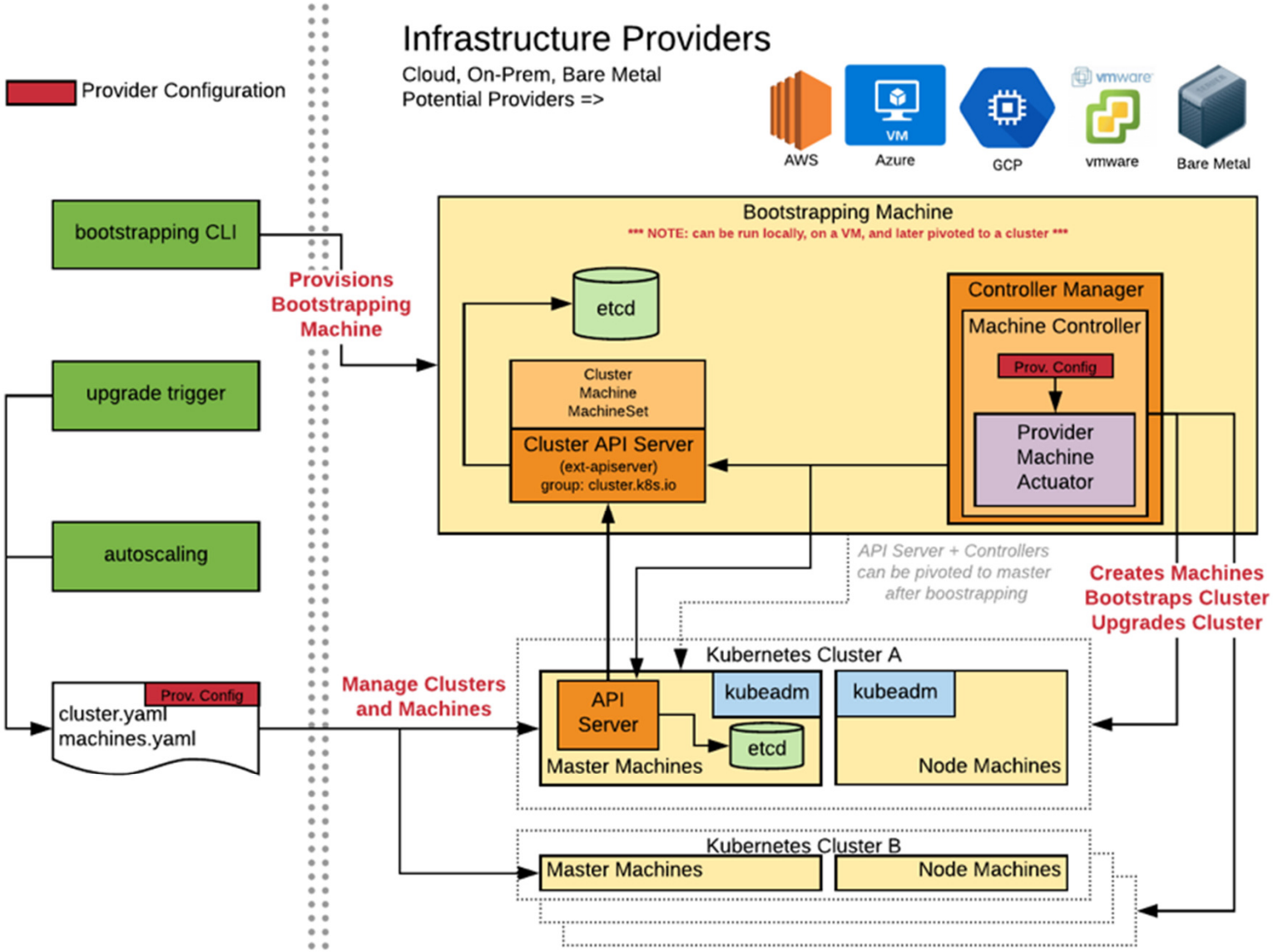


- Azure Kubernetes Service
- Amazon Elastic Container Service for Kubernetes
- Google Kubernetes Engine
- Digital Ocean Kubernetes

Cluster-API

heptio

- 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!


Infrastructure Providers
Cloud, On-Prem, Bare Metal
Potential Providers =>
Provider Configuration
AWS
Azure
VM
GCP
vmware
vmware
Bare Metal
bootstrapping CLI
upgrade trigger
autoscaling
Prov. Config
cluster.yaml
machines.yaml
Provisions Bootstrapping Machine
Manage Clusters and Machines
Bootstrapping Machine
*** NOTE: can be run locally, on a VM, and later pivoted to a cluster ***
etcd
Cluster
Machine
MachineSet
Cluster API Server
(ext-apiserver)
group: cluster.k8s.io
Controller Manager
Machine Controller
Prov. Config
Provider
Machine
Actuator
API Server + Controllers can be pivoted to master after boostrapping
Creates Machines
Bootstraps Cluster
Upgrades Cluster
Kubernetes Cluster A
API Server
kubeadm
etcd
Master Machines
kubeadm
Node Machines
Kubernetes Cluster B
Master Machines
Node Machines

Kubevirt

heptio

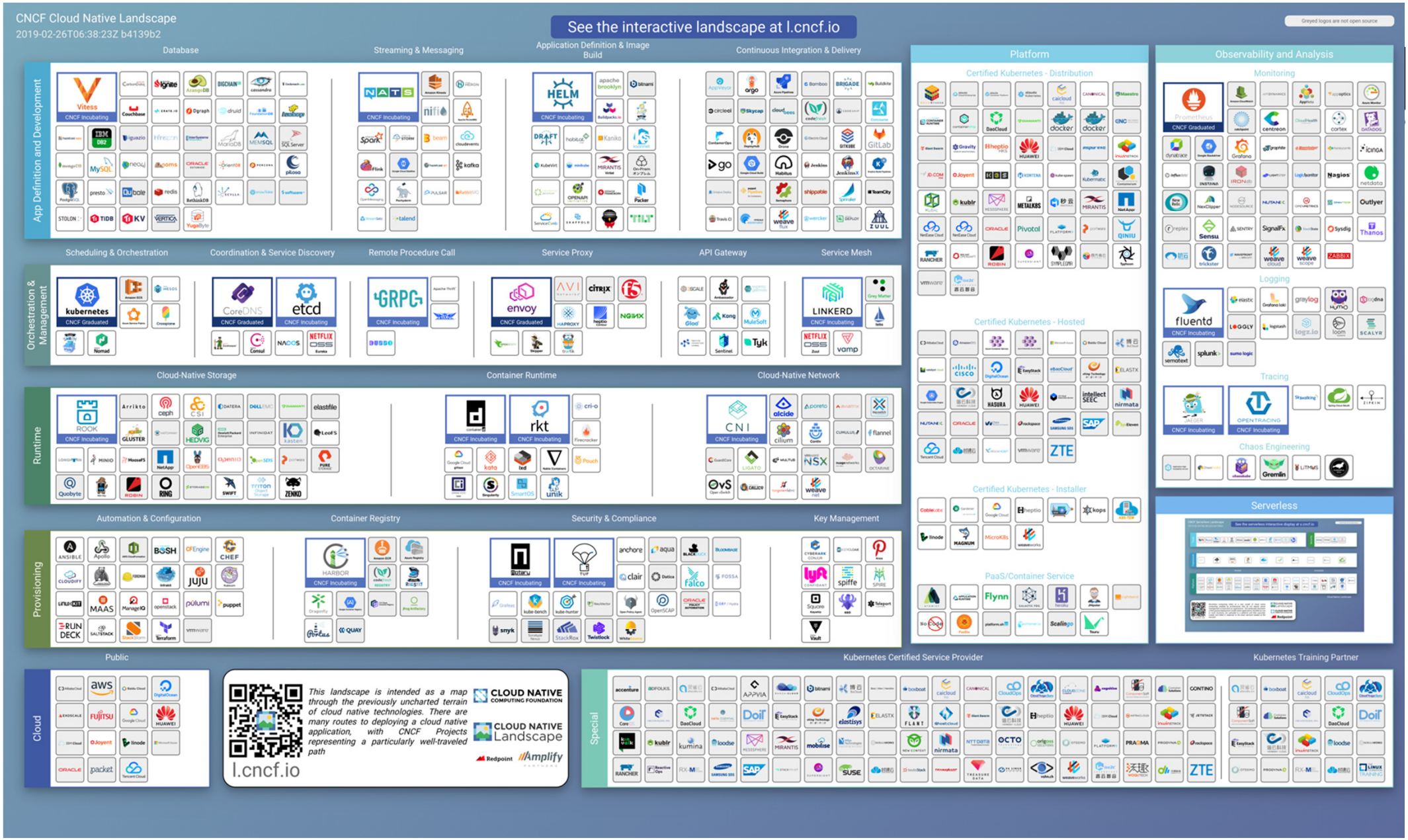
Leverage KubeVirt and Kubernetes to manage virtual machines for impractical-to-containerize apps.

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


CNCF Cloud Native Landscape
2019-02-26T06:38:23Z b4139b2
See the interactive landscape at l.cncf.io
Greyed logos are not open source
App Definition and Development
Database
Streaming & Messaging
Application Definition & Image Build
Continuous Integration & Delivery
Orchestration & Management
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
API Gateway
Service Mesh
Runtime
Cloud-Native Storage
Container Runtime
Cloud-Native Network
Provisioning
Automation & Configuration
Container Registry
Security & Compliance
Key Management
Public
Cloud
Platform
Certified Kubernetes - Distribution
Certified Kubernetes - Hosted
Certified Kubernetes - Installer
PaaS/Container Service
Kubernetes Certified Service Provider
Observability and Analysis
Monitoring
Logging
Tracing
Chaos Engineering
Serverless
Kubernetes Training Partner
Special
This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path
l.cncf.io
CLOUD NATIVE COMPUTING FOUNDATION
CLOUD NATIVE Landscape
Redpoint
Amplify

heptio

Questions?

Thankyou !