Scaling Java Applications using Docker

Vineet Reynolds · Red Hat

Who am I ?

What does this talk cover ?

What does this talk cover ?

- Docker

What does this talk cover ?

- Docker
- Scaling applications

What does this talk cover ?

- Docker
- Scaling applications
- Kubernetes

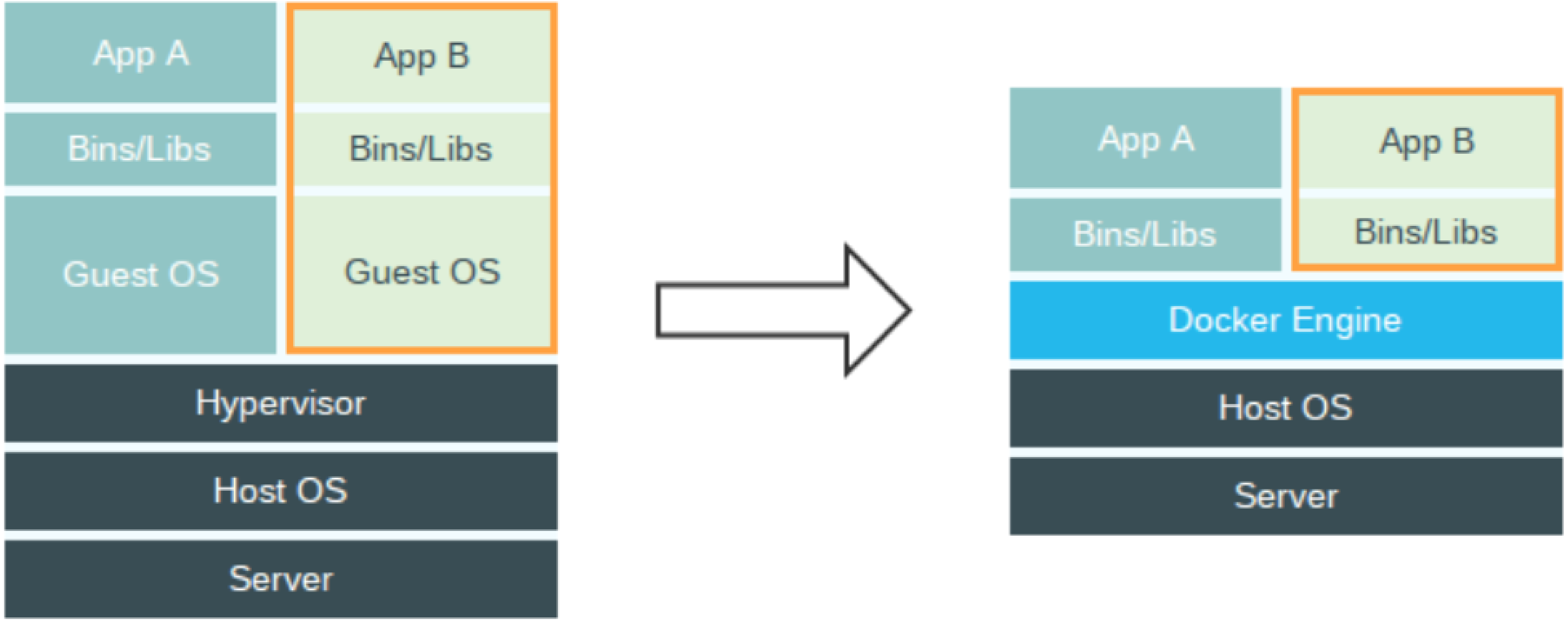
Docker

for the uninitiated

Containers

- Provide a light-weight virtualization solution
- Through the use of Linux kernel features -
    - `cgroups` to share/limit hardware resources
    - `namespaces` to provide isolation
- without requiring a guest OS

Containers

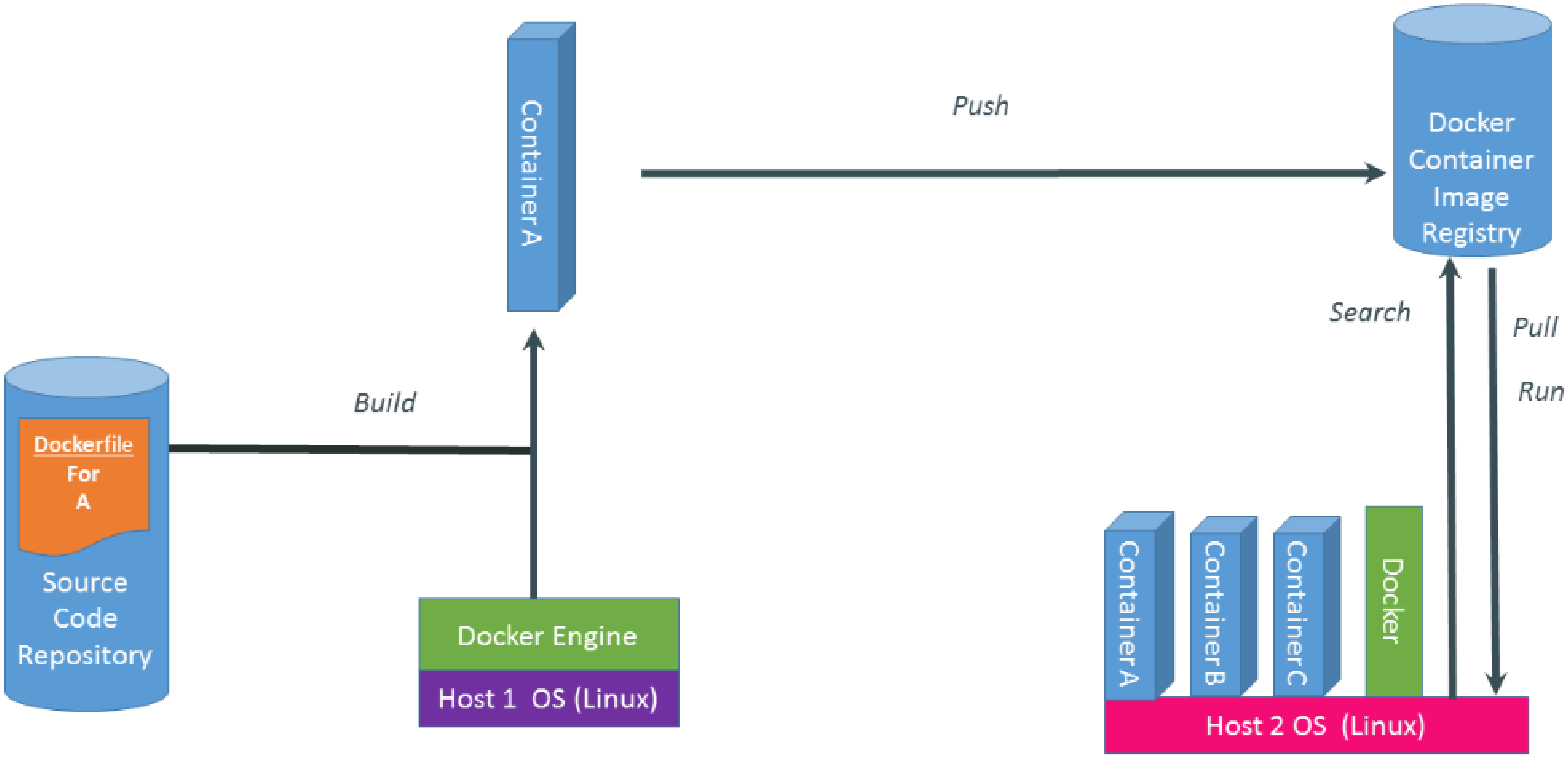

App A
App B
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Hypervisor
Host OS
Server
App A
App B
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Server

Docker

Docker is a shipping container system for code. Consists of -

- **A portable, lightweight runtime and packaging tool (Docker Engine)**
- **A cloud service for sharing applications and automating workflows (Docker Hub)**

Basics of the Docker system



Let’s run a container

- **Run a Java EE app in a Docker container**

```
docker run -d -p 8080:8080 vineetreynolds/badwildflycluster
```

Try to address HA

- **Run the same image in another Docker container**

```
docker run -d -p 9080:8080 vineetreynolds/badwildflycluster
```

Defining the cluster

Defining the cluster

- **First, node discovery**
  - JBoss EAP handles this with JGroups
  - JGroups ensures nodes are discovered
  - Solutions will be similar for other application servers

Replicating state

- Then, replicate state to handle failover
    - JBoss EAP handles this with Infinispan
    - Infinispan ensures atleast one replica for shared data

Running the app at scale

```
docker run -d -p 8080:8080 vineetreynolds/wildflycluster
docker run -d -p 9080:8080 vineetreynolds/wildflycluster
```

What about persistent data?

- Use database containers
  - Store the data on volumes mounted on the host
- Link database containers to application servers
  - Exposes database info to linked containers for usage

Example:

```
docker run --name mysqldb -e MYSQL_USER=mysql -e MYSQL_PASSWORD=mysql -e
MYSQL_DATABASE=sample -e MYSQL_ROOT_PASSWORD=supersecret -d mysql
```

```
docker run --name mywildfly --link mysqldb:db -p 8080:8080 -d
vineetreynolds/wildflycluster
```

This is now looking

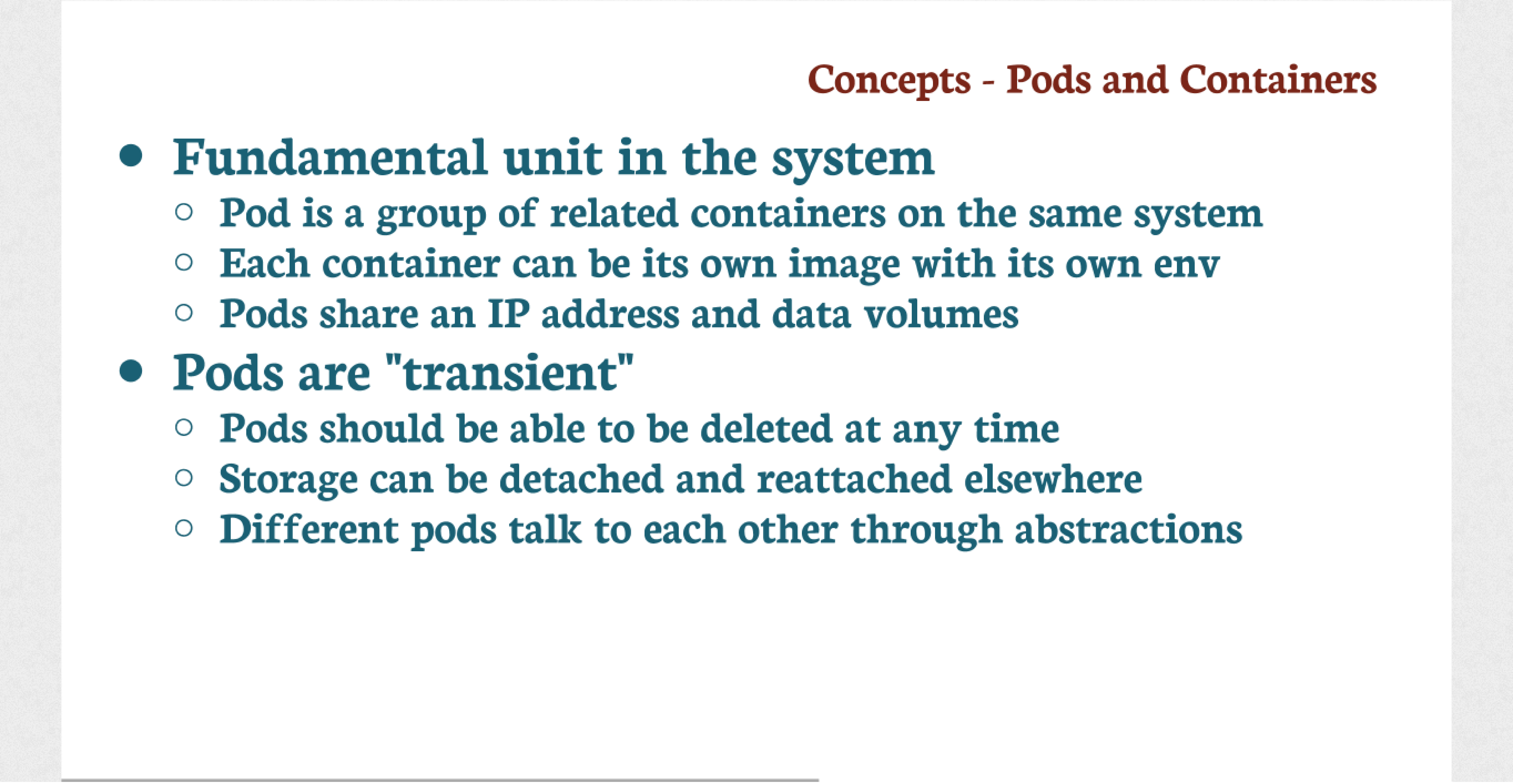
fragile !

Kubernetes

- Supports runtime and operational management of containers
- Describes the intended state of the cluster
    - Record links between containers - 'frontend' depends on 'backend'
    - Replicate containers onto the desired number of nodes; 'frontend' should always run on X nodes, 'backend' should run only on 1 node
- Provides self-healing capabilities to repair the cluster to intended state

Kubernetes

- Solves the **Cluster Container Management** problem
  - the substrate for running containers at scale
  - contains just the runtime and operational tools for containers
  - Composable system - only enough to enable other use cases



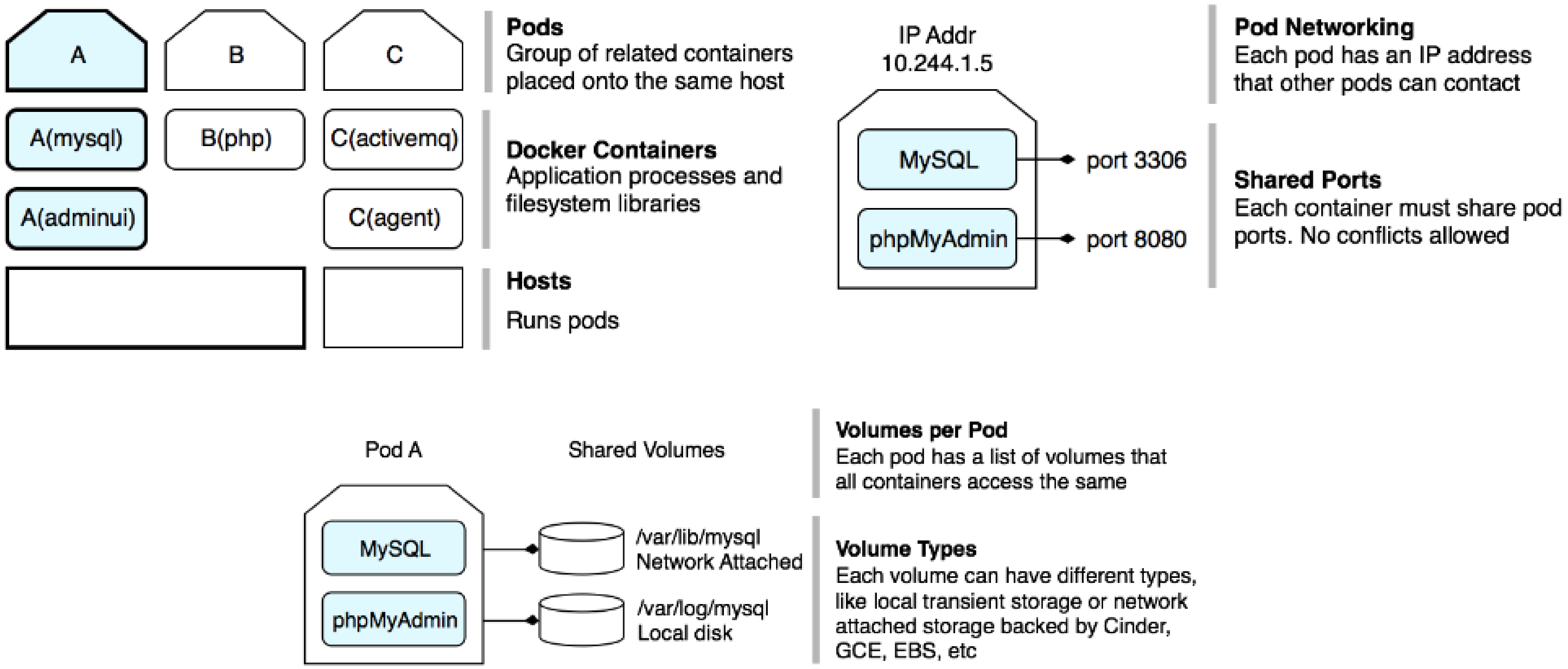
Concepts - Pods and Containers

- **Fundamental unit in the system**
  - Pod is a group of related containers on the same system
  - Each container can be its own image with its own env
  - Pods share an IP address and data volumes
- **Pods are "transient"**
  - Pods should be able to be deleted at any time
  - Storage can be detached and reattached elsewhere
  - Different pods talk to each other through abstractions

Concepts - Pods Examples

- Single container - JBoss, MySQL etc.
- Web server and log parser (one pod, two containers)
  - Web server container logs HTTP requests to disk
  - Log parser reads from disk and sends summary info elsewhere

Concepts - Pods (contd.)



Pods
Group of related containers placed onto the same host

Docker Containers
Application processes and filesystem libraries

Hosts
Runs pods

Pod Networking
Each pod has an IP address that other pods can contact

Shared Ports
Each container must share pod ports. No conflicts allowed

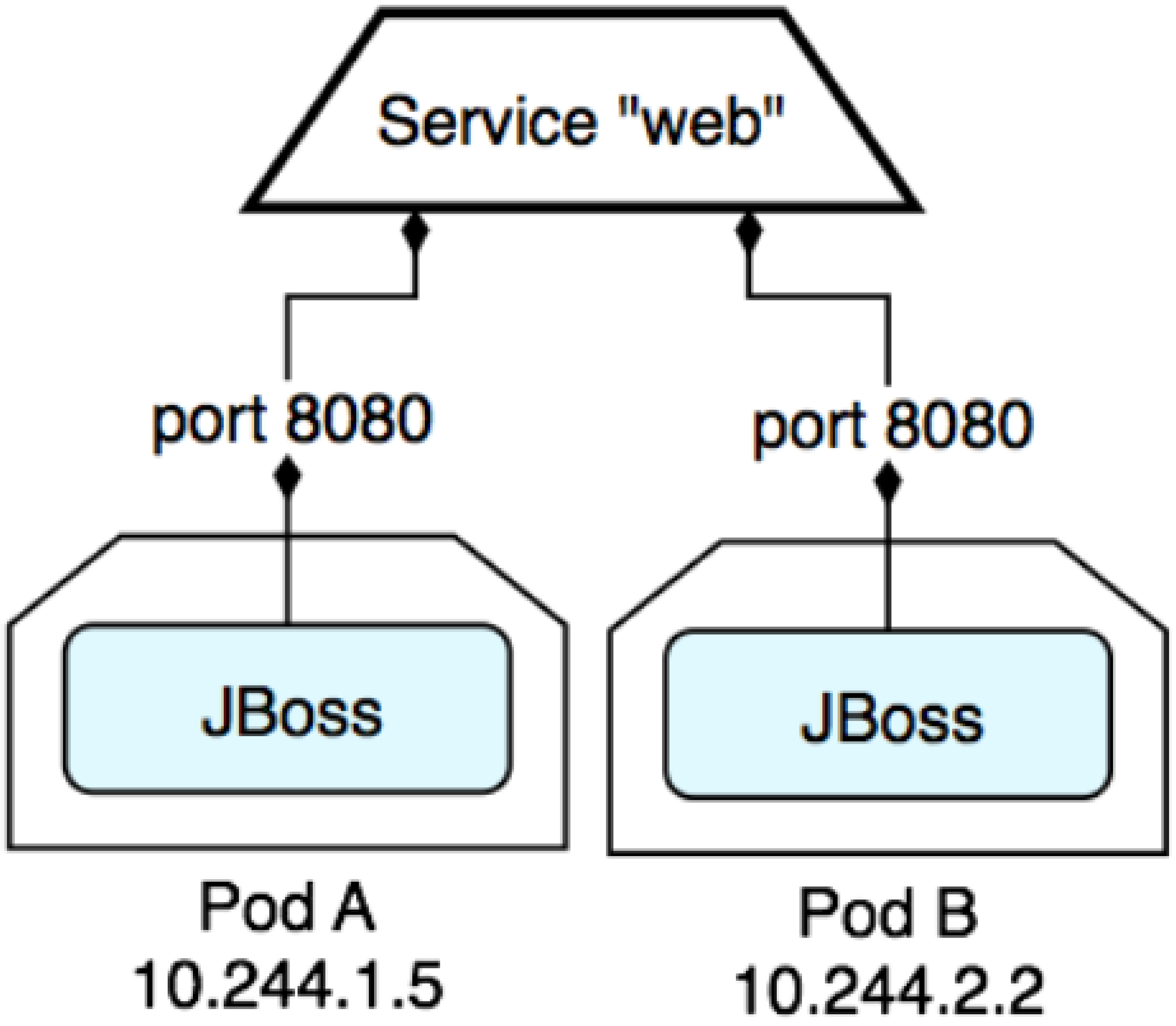
Volumes per Pod
Each pod has a list of volumes that all containers access the same

Volume Types
Each volume can have different types, like local transient storage or network attached storage backed by Cinder, GCE, EBS, etc

Services

- Abstract a set of pods as a single IP and port
  - Each host has a proxy that knows where other pods are
  - Simple TCP/UDP load balancing
  - No central load balancer (no SPOF)
- Creates environment variables in other pods
  - Like "docker link", but across hosts
  - Service named "mysql" gets `MYSQL_HOST` and `MYSQL_PORT`

Services(contd.)



Services abstract other pods
A service is a TCP port that may transparently load balance other ports

Replication controllers copy pods
A controller ensures there are a certain number of copies of a pod, so if a host is lost another pod gets created.

Scaling

applications in Kubernetes

Scaling with Replicas

- Replication controllers allow running multiple pods on multiple minions
- Define the number of pods in the intended state
- Kubernetes takes care of replicating the pods

The Kubernetes solution

clustering-controller.json

```
....
"podTemplate": {
    "desiredState": {
        "manifest": {
            "id": "wildfly",
            "version": "v1beta1",
            "containers": [
                {
                    "image": "vineetreynolds/wildflycluster",
                    "name": "wildfly-container",
....
```

How many replicas?

clustering-controller.json

```
....
"desiredState": {
    "replicas": 2,
    "replicaSelector": {
        "name": "wildfly"
    },
....
```

Creating the replica

```
kubectl create -f clustering-controller.json
```

Where’s
the catch?

Scaling with Replicas - Problems

- External access to the cluster
  - Pods are transient, and therefore …
  - Update external load balancers or edge routers with updated cluster state
  - What should be done -
    - when a container goes down ? Notify the load balancer
    - when a container is added ? Notify the load balancer
  - Cloud providers solve this out of the box - GCE/OpenShift
  - Refer the `createExternalLoadBalancer` flag for Kube services

Autoscaling

Autoscaling

- Applications experience peaks and valleys in usage
- Operators scale up resources on demand
- Currently, a feature in progress in Kubernetes
- Resource scaling will be driven by data from input sources
- Scope is horizontal scaling for now

Autoscaling

- Scaling based on traffic
- Scaling based on predictive analysis
- Scaling based on arbitrary data points - job execution time, number of sessions etc.

Questions?