redhat.

RED HAT® JBOSS®
MIDDLEWARE

Package your Java EE applications using Docker and Kubernetes

Arun Gupta, @arungupta
Red Hat

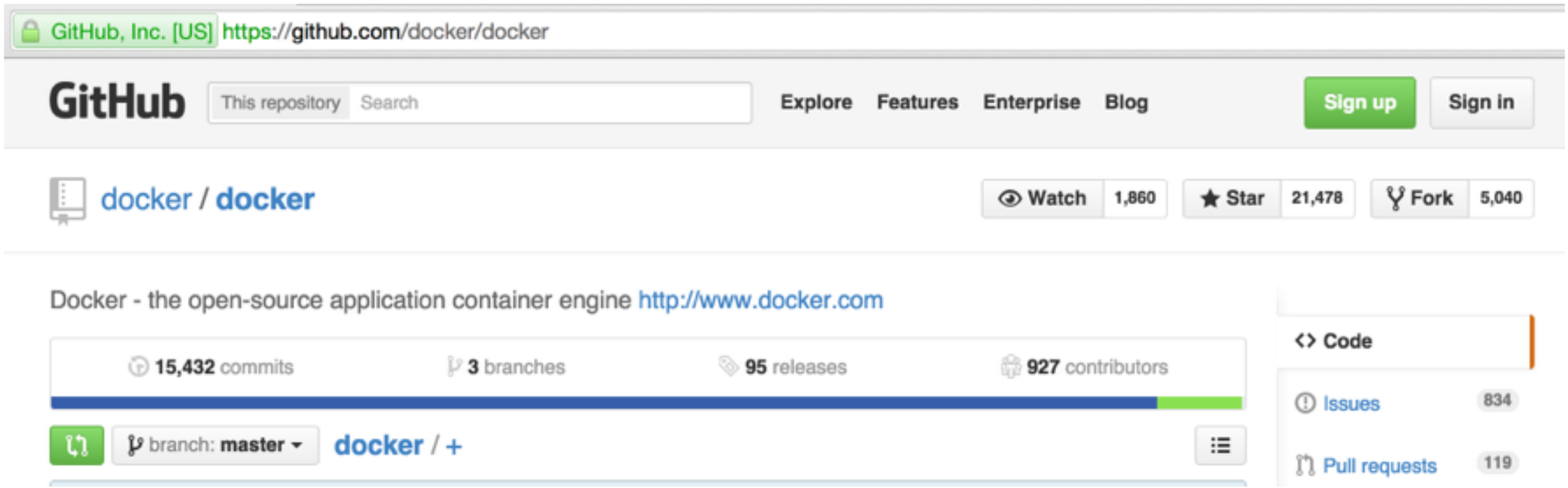
Arun Gupta

Director, Technical Marketing & Developer Advocacy

@arungupta
blog.arungupta.me
arungupta@redhat.com

What is Docker?

- Open source project and company

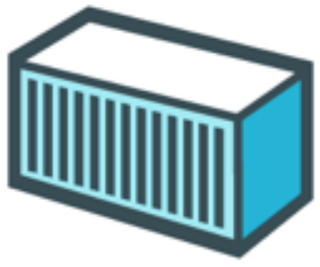


- Used to create containers for software applications

- Package Once Deploy Anywhere (PODA)

Build

Develop an app using Docker containers with any language and any toolchain.

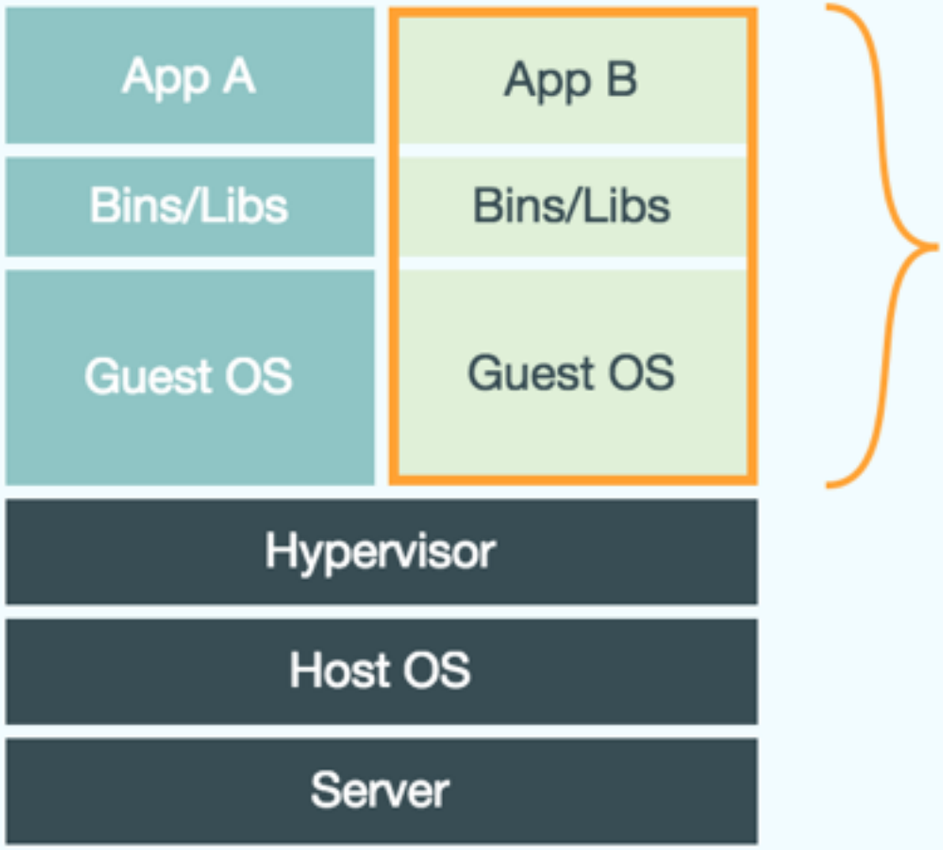


Ship

Ship the “Dockerized” app and dependencies anywhere - to QA, teammates, or the cloud - without breaking anything.

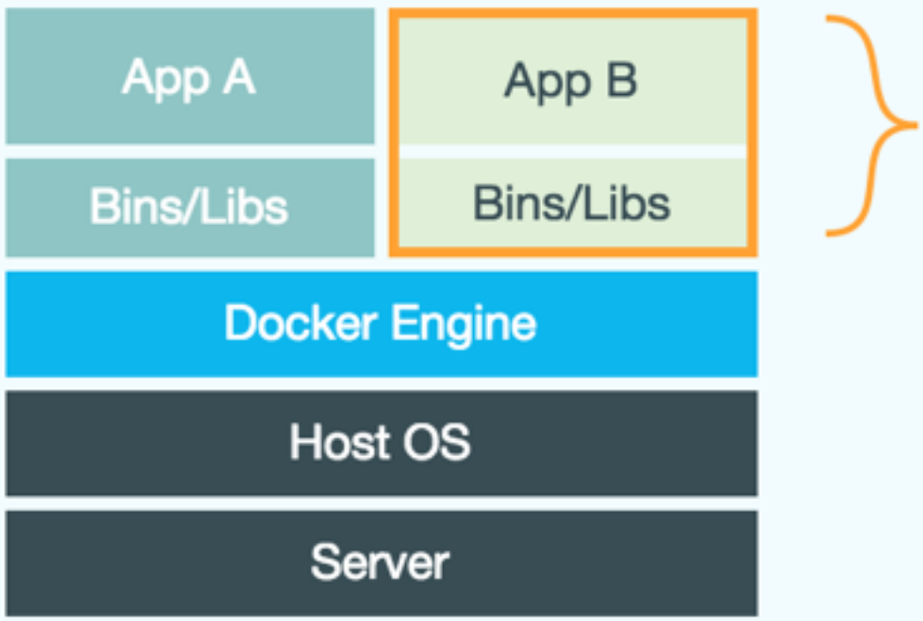
Run

Scale to 1000s of nodes, move between data centers and clouds, update with zero downtime and more.



Virtual Machines

Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.



Docker

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

Underlying Technology

- Written in Go

- Uses several Linux features
  - **Namespaces** to provide isolation
  - **Control groups** to share/limit hardware resources
  - **Union File System** makes it light and fast
  - **libcontainer** defines container format

Is it only Linux?

- Natively supported in Linux
- Can be installed on Mac or Windows using **`boot2docker`**
- Tiny Core Linux VM

Software Containers

- Decoupled apps and OS
- Repeatable
- Security sandbox
- Portability - “it works on my machine”
- Isolation
- Snapshotting
- Limit resource usage
- No external dependencies
- Sharing

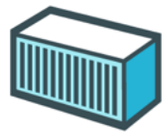
Build

Develop an app using Docker containers with any language and any toolchain.

- Image defined in text-based **Dockerfile**
- List of commands to build the image

```
FROM fedora:latest

CMD echo “Hello world”
```

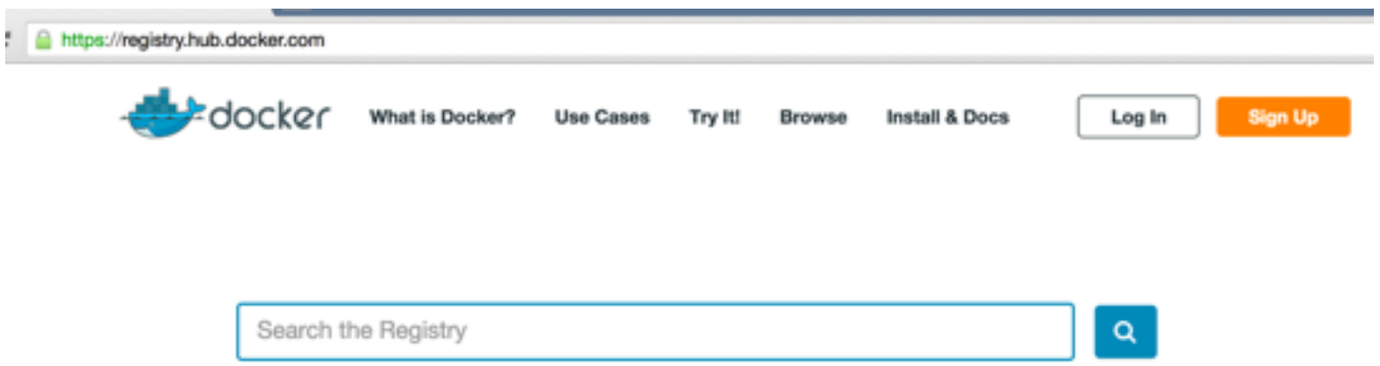
- `docker build` or `pull`



Ship

Ship the “Dockerized” app and dependencies anywhere - to QA, teammates, or the cloud - without breaking anything.

- Images shared using registry
- Docker Hub is public SaaS



- Private registries can be setup inside firewall
- `docker push` or `pull <IMAGE_ID>`

redhat

- Container built from the image

Run

Scale to 1000s of nodes, move between data centers and clouds, update with zero downtime and more.

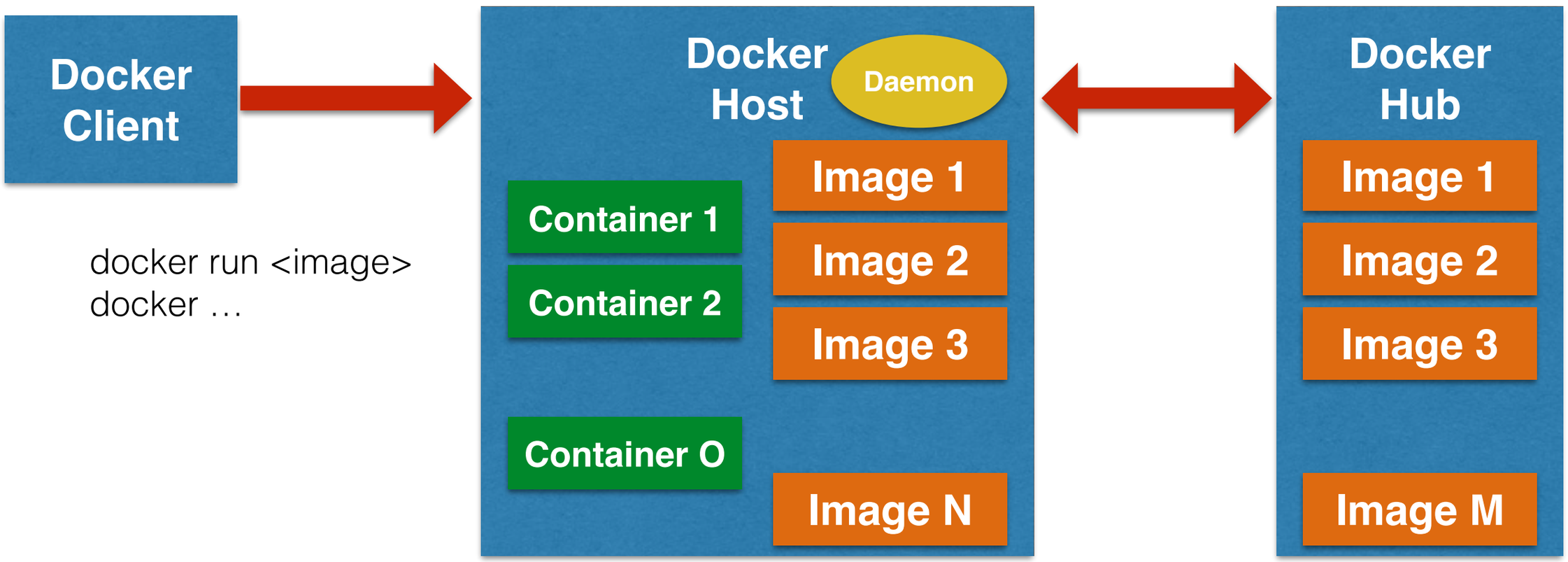
- Runtime representation of the image
- Self contained execution environment
- `docker run <IMAGE_ID>`

Docker commands

- **docker ps**: List running containers
- **docker stop**: Stop a running container
- **docker rm**: Remove a running container
- **docker rmi**: Remove an image
- …

https://docs.docker.com/reference/commandline/cli/

Docker Workflow



docker run <image>
docker …

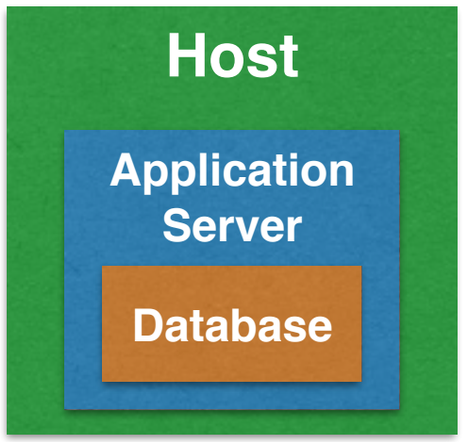
```
FROM centos
MAINTAINER Arun Gupta <arungupta@redhat.com>

# Execute system update
RUN yum -y update && yum clean all

# Install packages necessary to run EAP
RUN yum -y install xmlstarlet saxon augeas bsdtar unzip && yum clean all

# Create a user and group used to launch processes
# The user ID 1000 is the default for the first "regular" user on Fedora/RHEL,
# so there is a high chance that this ID will be equal to the current user
# making it easier to use volumes (no permission issues)
RUN groupadd -r jboss -g 1000 && useradd -u 1000 -r -g jboss -m -d /opt/jboss -s /sbin/nologin -c "JBoss user" jboss

# Set the working directory to jboss' user home directory
WORKDIR /opt/jboss

# User root user to install software
USER root

# Install necessary packages
RUN yum -y install java-1.7.0-openjdk-devel && yum clean all
#RUN yum -y install java-1.8.0-openjdk-devel && yum clean all

# Switch back to jboss user
USER jboss

# Set the JAVA_HOME variable to make it clear where Java is located
ENV JAVA_HOME /usr/lib/jvm/java

# Set the WILDFLY_VERSION env variable
ENV WILDFLY_VERSION 8.2.0.Final

# Add the WildFly distribution to /opt, and make wildfly the owner of the extracted tar content
# Make sure the distribution is available from a well-known place
RUN cd $HOME && curl -O http://download.jboss.org/wildfly/$WILDFLY_VERSION/wildfly-$WILDFLY_VERSION.zip && unzip wildfly-$WILDFLY

# Set the JBOSS_HOME env variable
ENV JBOSS_HOME /opt/jboss/wildfly

# Expose the ports we're interested in
EXPOSE 8080 9990

# Set the default command to run on boot
# This will boot WildFly in the standalone mode and bind to all interface
CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-c", "standalone-full.xml", "-b", "0.0.0.0"]
```

Recipe 1.1

- Docker Machine
  - Zero to Docker
    - Downloads boot2docker
    - Setup ssh keys and certificates

http://blog.arungupta.me/docker-machine-seutp-docker-host-techtip78

Recipe #1.2

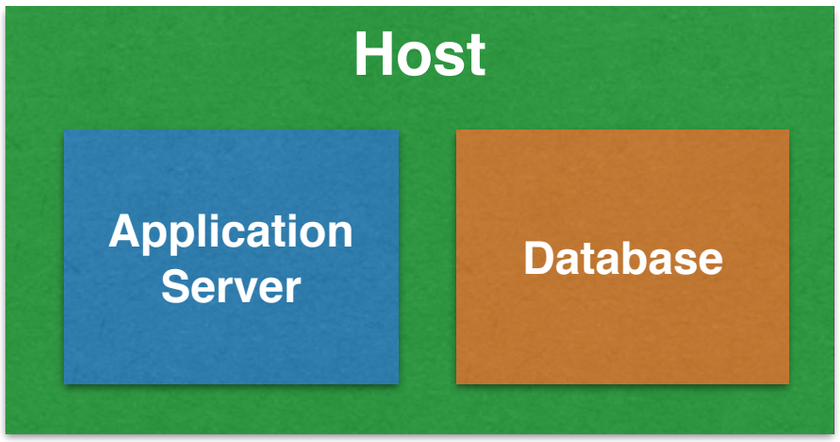



```
FROM jboss/wildfly

RUN curl -L https://github.com/javaee-samples/javaee7-hol/raw/master/solution/
movieplex7-1.0-SNAPSHOT.war -o /opt/jboss/wildfly/standalone/deployments/
movieplex7-1.0-SNAPSHOT.war
```

```
docker run -it -p 8080:8080 arungupta/javaee7-hol
```

http://blog.arungupta.me/javaee7-lab-wildfly-docker/

Recipe #1.3



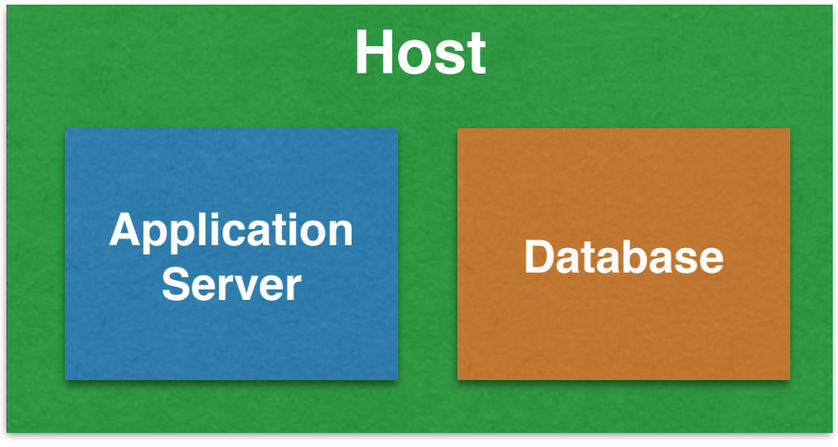

```
docker run --name mysqldb -e MYSQL_USER=mysql -e MYSQL_PASSWORD=mysql -e MYSQL_DATABASE=sample -e MYSQL_ROOT_PASSWORD=supersecret -d mysql
```

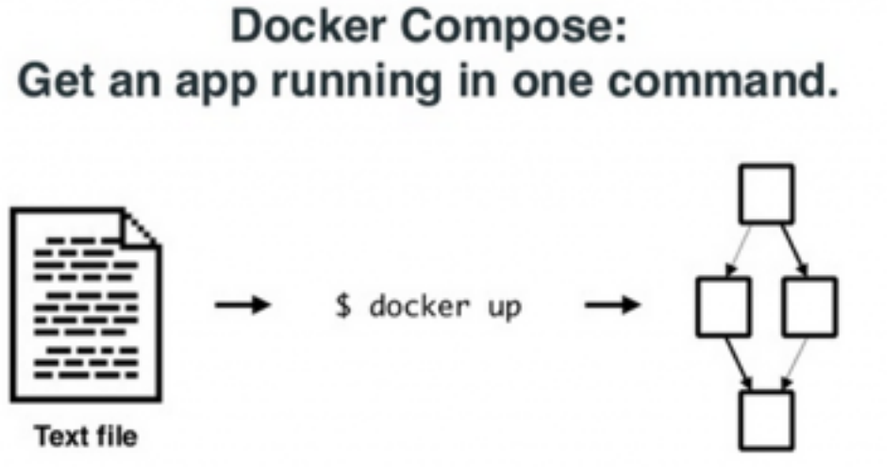
```
docker run --name mywildfly --link mysqldb:db -p 8080:8080 -d arungupta/wildfly-mysql-javaee7
```

```
data-source add --name=mysqlDS --driver-name=mysql --jndi-name=java:jboss/datasources/ExampleMySQLDS --connection-url=jdbc:mysql://$DB_PORT_3306_TCP_ADDR:$DB_PORT_3306_TCP_PORT/sample?useUnicode=true&amp;characterEncoding=UTF-8 --user-name=mysql --password=mysql --use-ccm=false --max-pool-size=25 --blocking-timeout-wait-millis=5000 --enabled=true
```

http://blog.arungupta.me/wildfly-javaee7-mysql-link-two-docker-container-techtip65/

Recipe #1.4



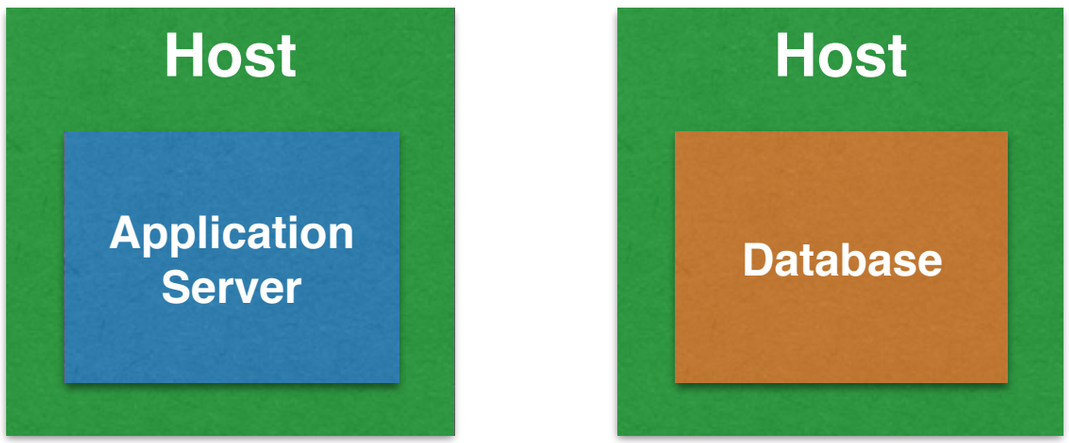



```
mysqldb:
  image: mysql:latest
  environment:
    MYSQL_DATABASE: sample
    MYSQL_USER: mysql
    MYSQL_PASSWORD: mysql
    MYSQL_ROOT_PASSWORD: supersecret
mywildfly:
  image: arungupta/wildfly-mysql-javaee7
  links:
    - mysqldb:db
  ports:
    - 8080:8080
```

```
docker-compose up -d
```

http://blog.arungupta.me/docker-compose-orchestrate-containers-techtip77/

Recipe #1.5




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

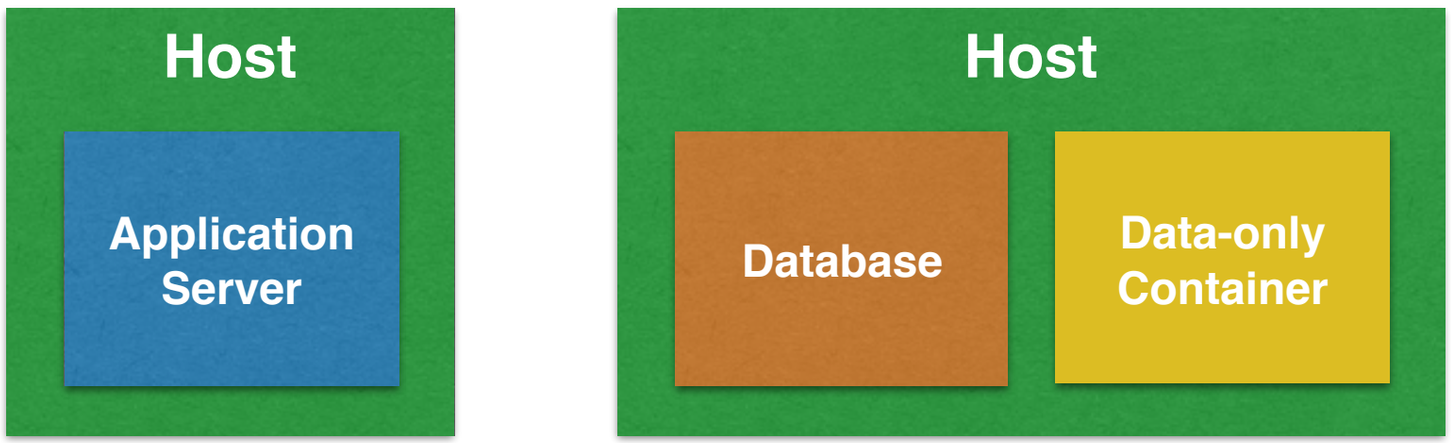
```
data-source add --name=mysqlDS --driver-name=mysql --jndi-name=java:jboss/datasources/ExampleMySQLDS --connection-url=jdbc:mysql://$MYSQL_HOST:$MYSQL_PORT/sampl
```

```
docker build -t arungupta/wildfly-mysql-javaee7 .
docker run --name mywildfly -e MYSQL_HOST=<IP_ADDRESS> -e MYSQL_PORT=5306 -p 8080:8080 -d arungupta/wildfly-mysql-javaee7
```

http://blog.arungupta.me/docker-container-linking-across-multiple-hosts-techtip69/

Recipe #1.6

- Data-only containers

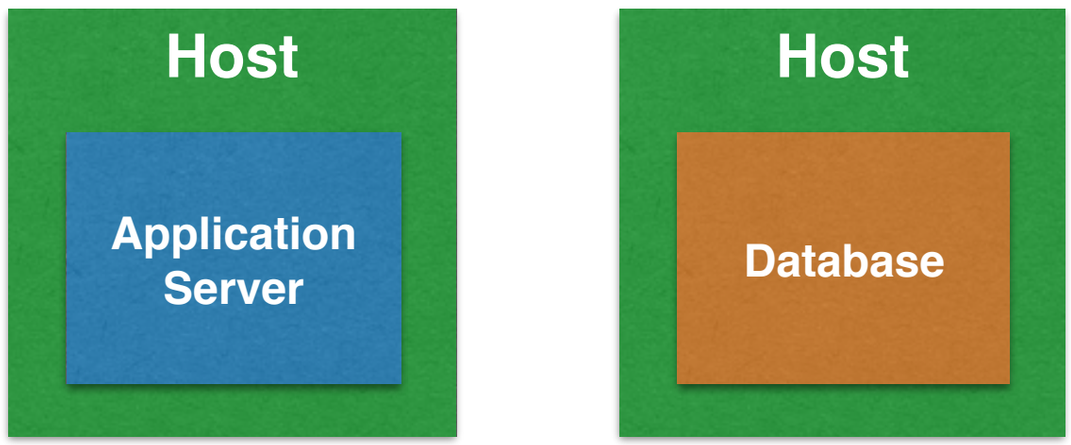



```
docker create --name mysql_data -v /var/lib/mysql mysql
```

```
docker run --name mysqldb --volumes-from mysql_data -v /var/lib/mysql:/var/lib/mysql -e MYSQL_USER
```

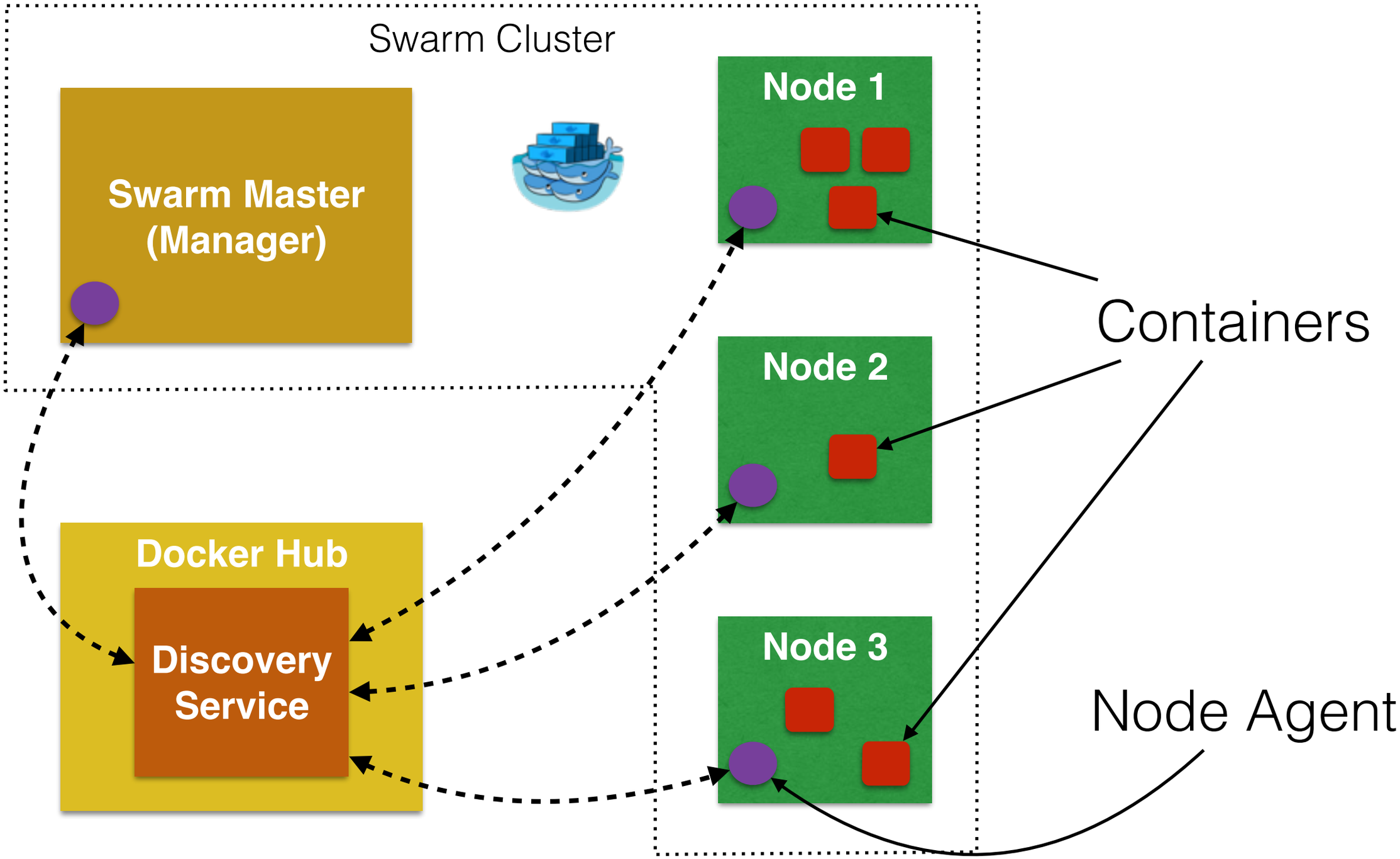
http://blog.arungupta.me/docker-mysql-persistence/

Recipe #1.7



- Docker Swarm: Native clustering for Docker
- Turns a pool of Docker hosts into a single, virtual host

Recipe #1.7



Recipe #1.8

- Running Java EE application on Docker Swarm
  - Connect to Docker Swarm
    `$(docker-machine env --swarm swarm-master)`
  - Start MySQL
    `docker run --name mysqldb . . . -p 3306:3306 -d mysql`
  - Get IP
    `docker inspect --format '{{ .Node.Ip }}' $(docker ps -q --filter 'name=*mysqldb*')`
  - Run WildFly
    `docker run --name mywildfly -e MYSQL_HOST=<IP> -e MYSQL_PORT=<PORT> -p 8080:8080 -d arungupta/wildfly-mysql-javaee7:host`

Recipe 1.9


```
docker run -it -p 8080:8080 -v /Users/arungupta/tmp/deployments:/opt/jboss/wildfly/standalone/deployments/:rw jboss/wildfly
```

Docker volumes + Local deployment

```
docker run -it -p 8080:8080 -p 9990:9990 arungupta/wildfly-management
```

Management API + Remote deployment

http://blog.arungupta.me/deploy-wildfly-docker-eclipse/

Recipe #1.10

- Arquillian Cube: Controls the lifecycle of Docker images as part of test cycle - automatically or manually

- Uses Docker REST API to talk to container

- Talk using WildFly remote adapter (in container)

```
git clone https://github.com/javaee-samples/javaee7-samples.git
git checkout docker
mvn test -f servlet/simple-servlet/pom.xml -Pwildfly-docker-arquillian
```

http://blog.arungupta.me/run-javaee-tests-wildfly-docker-arquillian-cube/

maven

Recipe 1.11

- **`docker-maven-plugin`**: Manage Docker images and containers from Maven
- Goals
  - Image: **`docker:build/push/remove`**
  - Container: **`docker:start/stop`**
  - Logs: **`docker:logs`**

redhat

maven

```
<plugin>
    <groupId>org.jolokia</groupId>
    <artifactId>docker-maven-plugin</artifactId>
    <version>0.11.5-M1</version>
    <configuration>
        <images>
            <image>
                <alias>user</alias>
                <name>arungupta/shoppingcart-user</name>
                <build>
                    <from>arungupta/wildfly:8.2</from>
                    <assembly>
                        <descriptor>assembly.xml</descriptor>
                        <basedir>/</basedir>
                    </assembly>
                    <ports>
                        <port>8080</port>
                    </ports>
                </build>
                <run>
                    <ports>
                        <port>user.port:8080</port>
                    </ports>
                </run>
            </image>
        </images>
    </configuration>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>build</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

redhat

maven

assembly.xml

```
<assembly . . .>
  <id>user</id>
  <dependencySets>
    <dependencySet>
      <includes>
        <include>org.javaee7.wildfly.samples:user</include>
      </includes>
      <outputDirectory>/opt/jboss/wildfly/standalone/deployments/</
outputDirectory>
      <outputFileNameMapping>user.war</outputFileNameMapping>
    </dependencySet>
  </dependencySets>
</assembly>
```

redhat

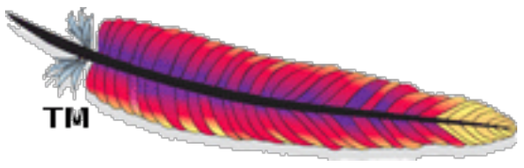
Recipe 1.xx

- Registry
- Save running container state
- List of exact version of the layers used
  - What if latest tag is updated?
  - How to have a predictable set of environment
- Load balancing containers

Docker: Pros and Cons

- PROS
  - Extreme application portability
  - Very easy to create and work with derivative
  - Fast boot on containers
- CONS
  - Host-centric solution
  - No higher-level provisioning
  - No usage tracking/reporting

Application Operating Environment



Kubernetes

- Open source orchestration system for Docker containers

- Provide declarative primitives for the “desired state”

- Self-healing

- Auto-restarting

- Schedule across hosts

- Replicating

Contributors | Commits | Code frequency | Punch card | Network | Members

Jun 1, 2014 – Feb 10, 2015

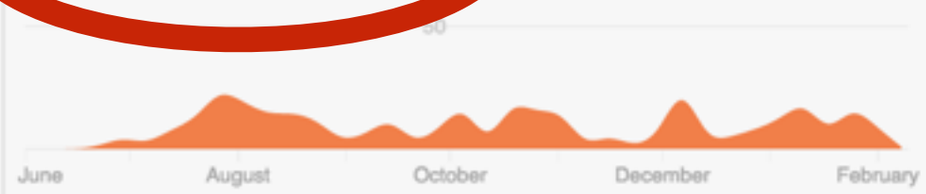
Contributions to master, excluding merge commits

Contributions: **Commits**

brendandburns #1
518 commits / 424,490 ++ / 11,507 --

lavalamp #2
419 commits / 73,245 ++ / 30,818 --

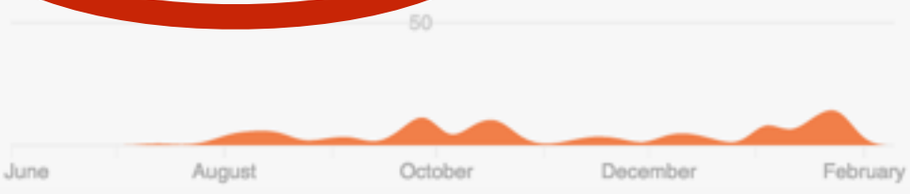
smarterclayton #3
339 commits / 42,405 ++ / 18,936 --



thockin #4
223 commits / 20,921 ++ / 7,816 --

jbeda #5
179 commits / 96,321 ++ / 45,499 --

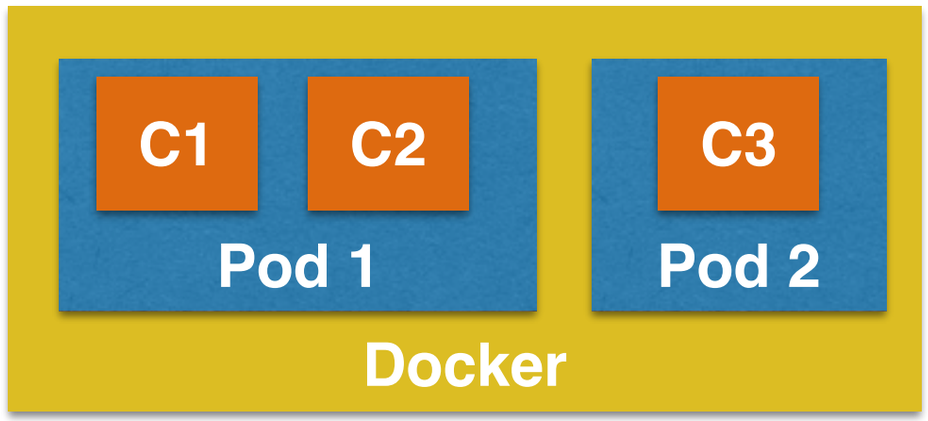
derekwaynecarr #6
138 commits / 16,353 ++ / 3,877 --



redhat

Concepts

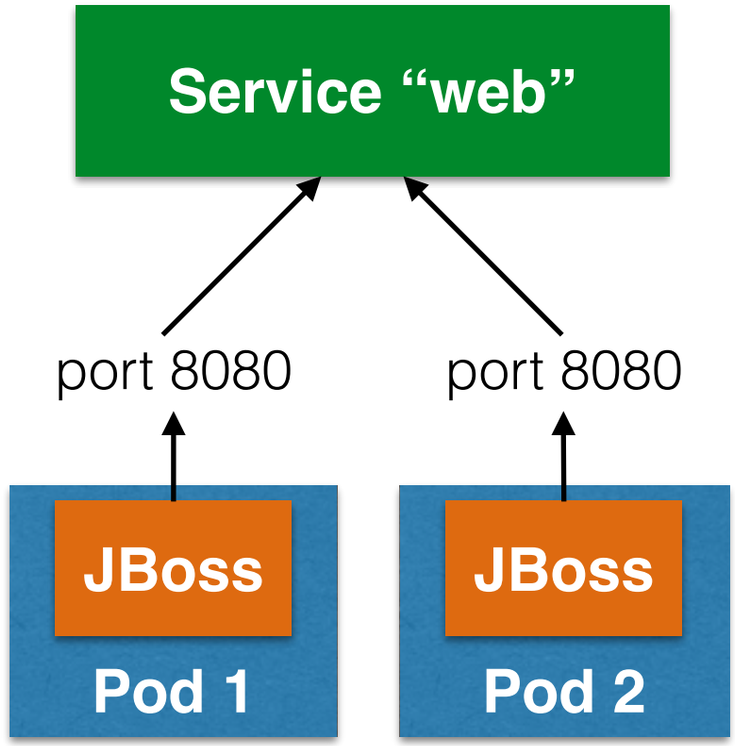
- **Pods**: collocated group of Docker containers that share an IP and storage volume



- **Service**: Single, stable name for a set of pods, also acts as LB

- **Replication Controller**: manages the lifecycle of pods and ensures specified number are running

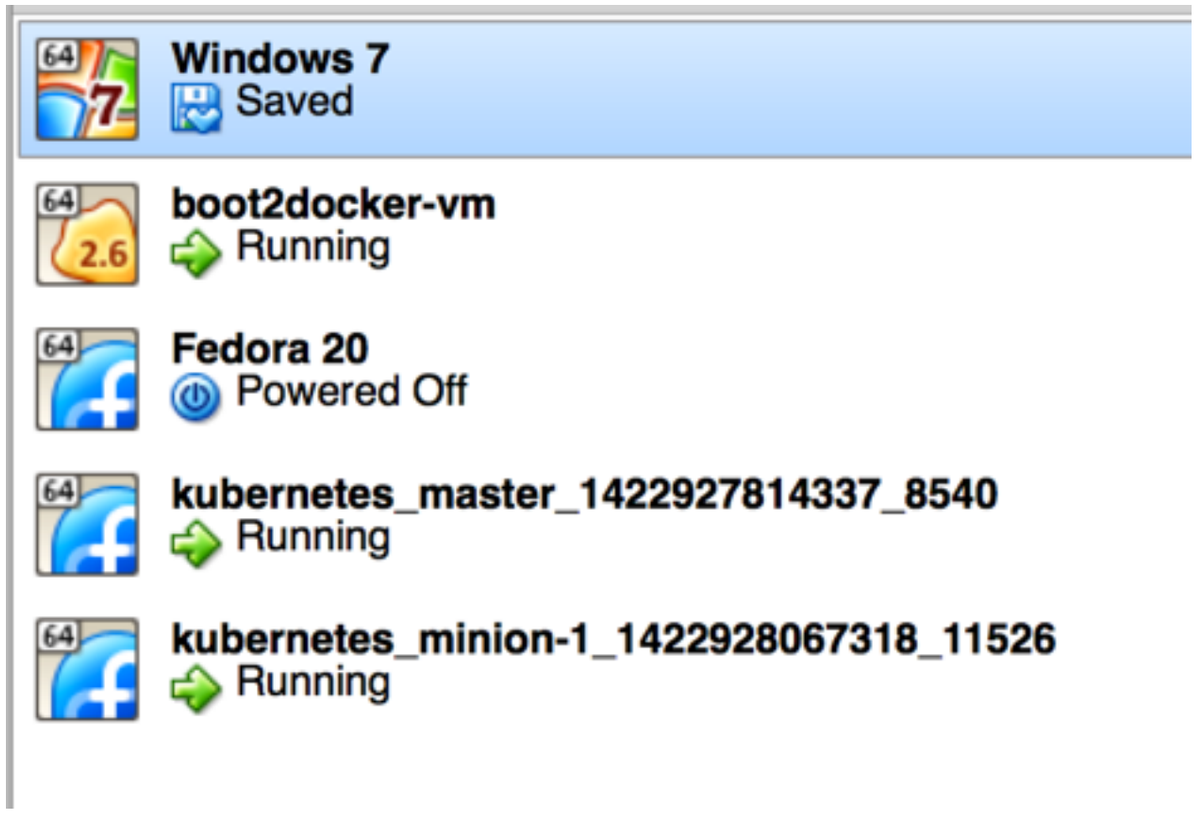
- **Label**: used to organize and select group of objects

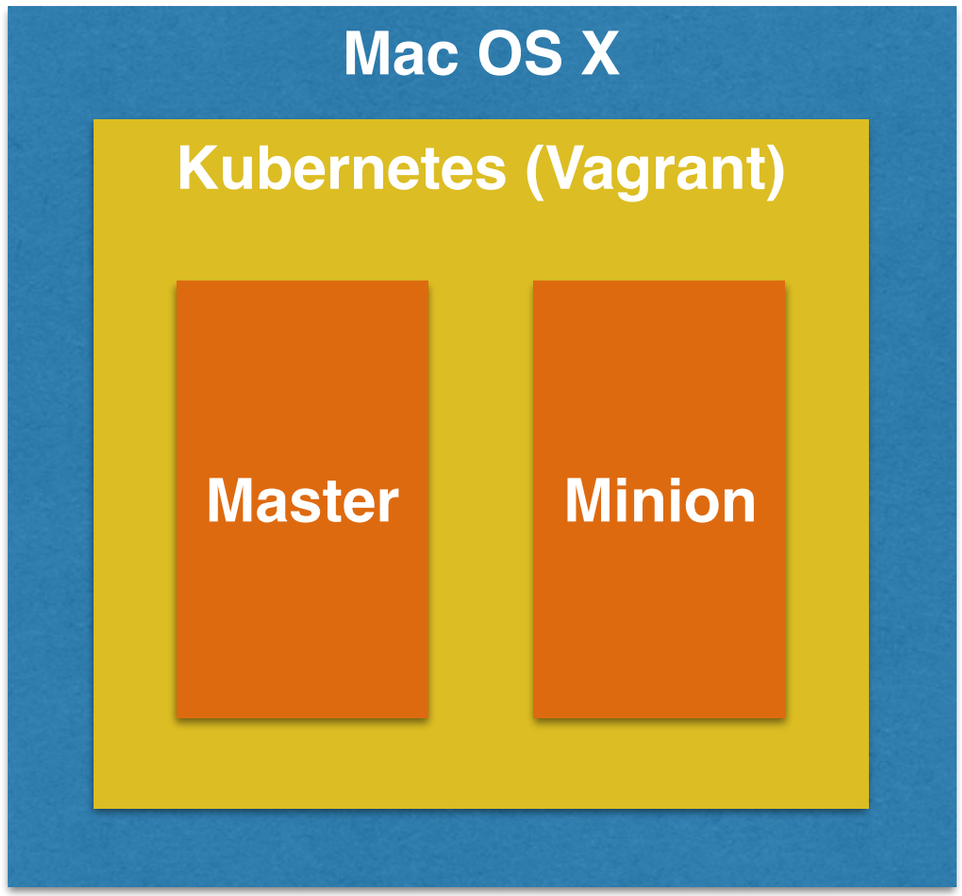


kubectl

- Controls the Kubernetes cluster manager
- `kubectl get pods` or `minions`
- `kubectl create -f <filename>`
- `kubectl update` or `delete`
- `kubectl resize –replicas=3 replicationcontrollers <name>`

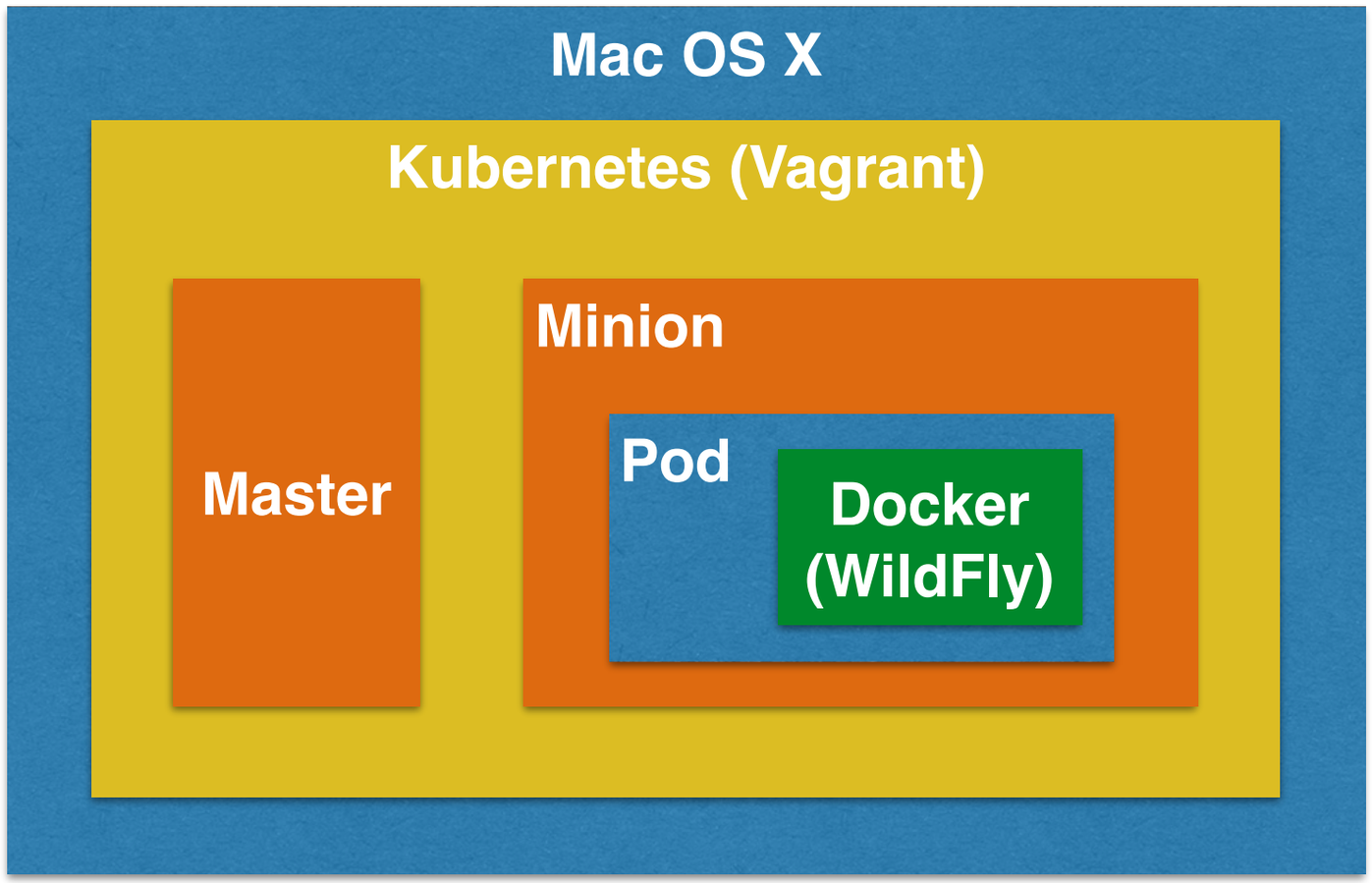
```
export KUBERNETES_PROVIDER=vagrant
./cluster/kube-up.sh
```





Recipe #2.1

```
{
  "id": "wildfly",
  "kind": "Pod",
  "apiVersion": "v1beta1",
  "desiredState": {
    "manifest": {
      "version": "v1beta1",
      "id": "wildfly",
      "containers": [{
        "name": "wildfly",
        "image": "arungupta/javaee7-hol",
        "cpu": 100,
        "ports": [{
          "containerPort": 8080,
          "hostPort": 8080
        },
        {
          "containerPort": 9090,
          "hostPort": 9090
        }]
      }]
    }
  },
  "labels": {
    "name": "wildfly"
  }
}
```



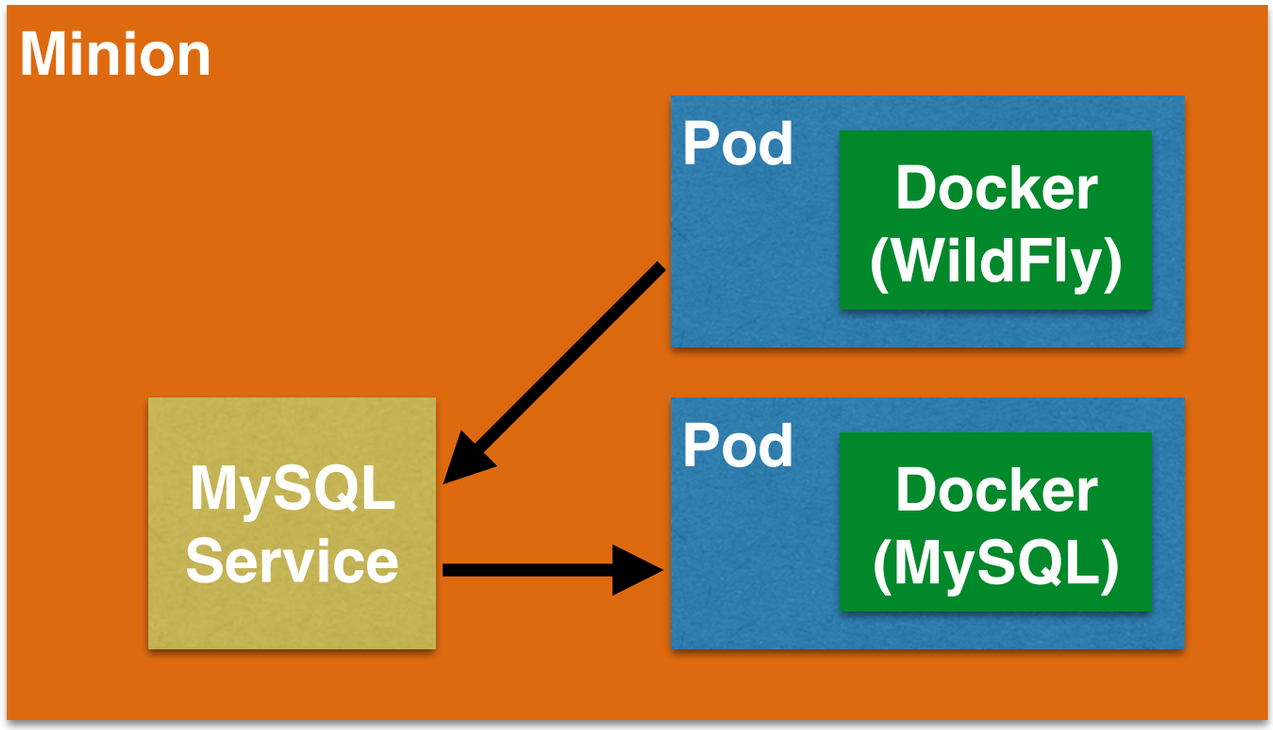
http://blog.arungupta.me/javaee7-wildfly-kubernetes-mac-vagrant/

Services

- Abstract a set of pods as a single IP and port
  - Simple TCP/UDP load balancing
- Creates environment variables in other pods
  - Like “Docker links” but across hosts
- Stable endpoint for pods to reference
  - Allows list of pods to change dynamically

Recipe #2.2

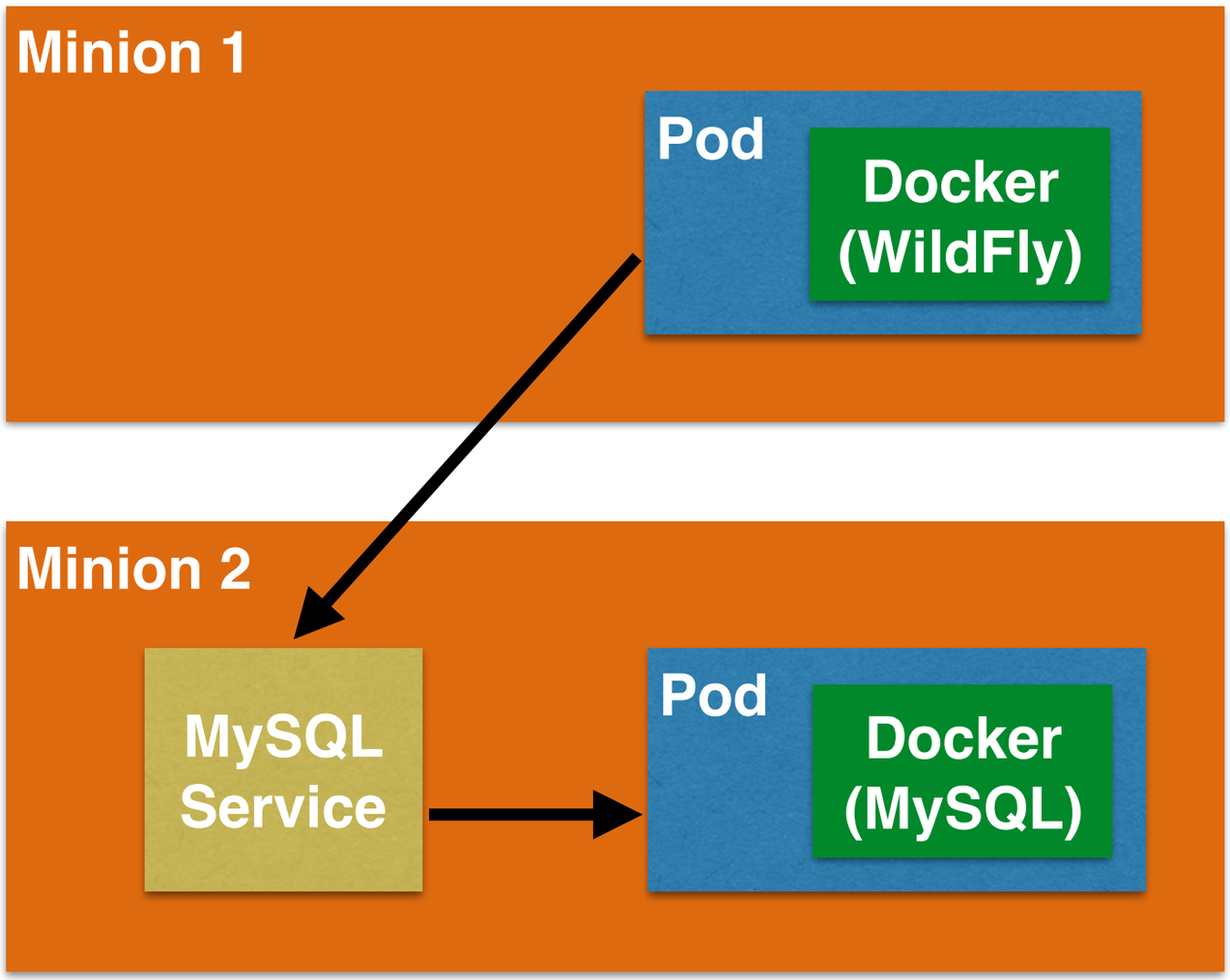
```
{
  "id": "mysql",
  "kind": "Pod",
  "apiVersion": "v1beta1",
  "desiredState": {
    "manifest": {
      "version": "v1beta1",
      "id": "mysql",
      "containers": [{
        "name": "mysql",
        "image": "mysql:latest",
        "cpu": 100,
        "env": [
              {
                  "name": "MYSQL_USER",
                  "value": "mysql"
              },
              {
                  "name": "MYSQL_PASSWORD",
                  "value": "mysql"
              },
              {
                  "name": "MYSQL_DATABASE",
                  "value": "sample"
              },
              {
                  "name": "MYSQL_ROOT_PASSWORD",
                  "value": "supersecret"
              }
          ],
        "ports": [{
          "containerPort": 3306,
          "hostPort": 3306
        }]
      }]
    }
  },
  "labels": {
    "name": "mysql"
  }
}
```




```
{
  "id": "mysql",
  "kind": "Service",
  "apiVersion": "v1beta1",
  "port": 3306,
  "containerPort": 3306,
  "selector": {
    "name": "mysql"
  },
  "labels": {
    "name": "mysql"
  }
}
```

http://blog.arungupta.me/mysql-kubernetes-service-access-wildfly-pod-techtip72/

Recipe #2.3


Minion 1
Pod
Docker
(WildFly)
Minion 2
MySQL
Service
Pod
Docker
(MySQL)

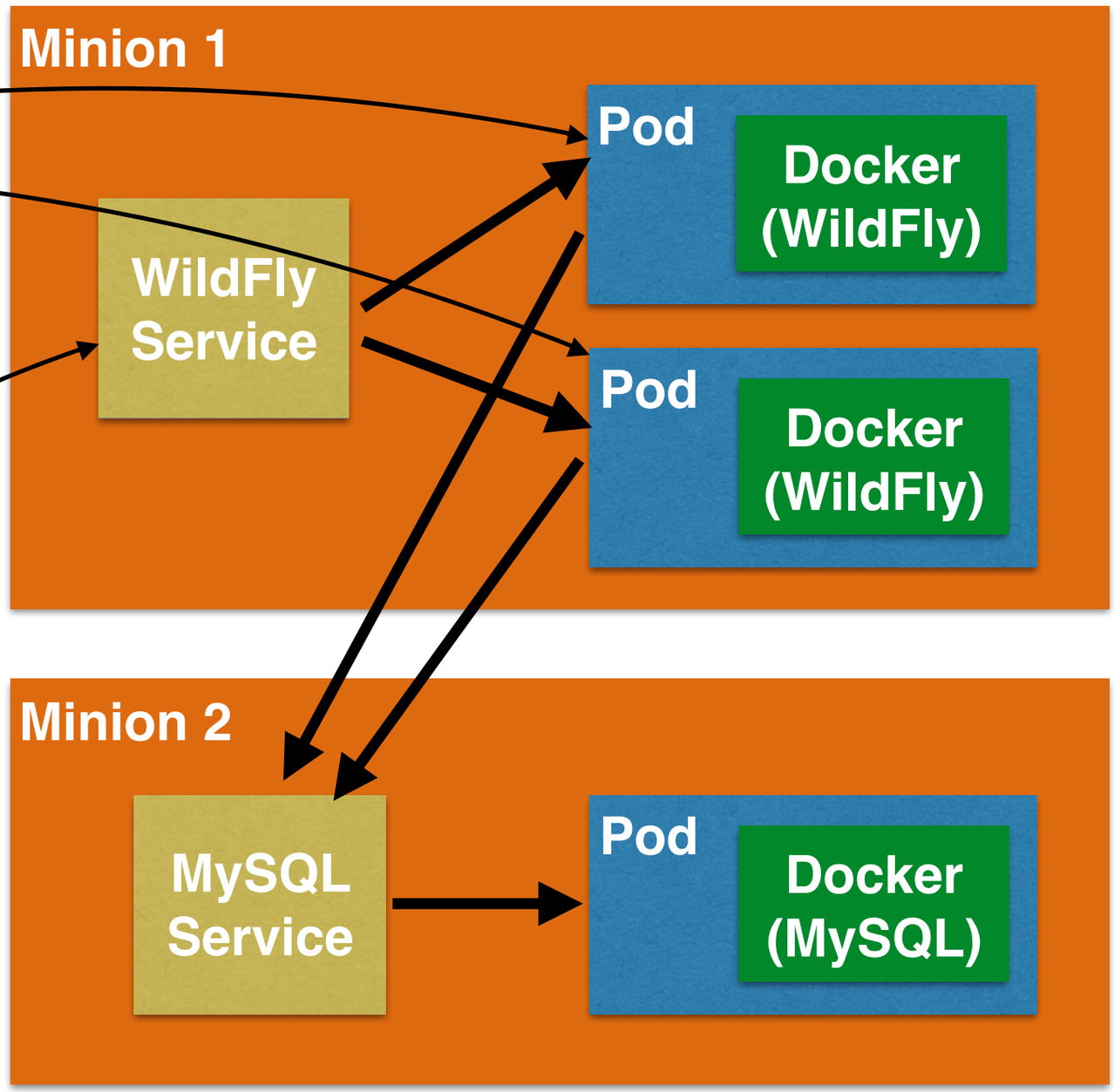
Replication Controller

- Ensures specified number of pod “replicas” are running
- Pod templates are cookie cutters
- Rescheduling
- Manual or auto-scale replicas
- Rolling updates

Recipe #2.4

```
id: wildfly-controller
kind: ReplicationController
apiVersion: v1beta1
desiredStat
  replica : 2
  replicaSe tor:
    name: wildfly
  podTemplate:
    desiredState:
      manifest:
        version: v1beta1
        id: wildfly
        containers:
          - name: javaee7-hol
            image: arungupta/javaee7-hol
    labels:
      name: wildfly
```

```
id: wildfly
kind: Service
apiVersion: v1beta1
port: 8080
containerPort: 8080
selector:
  name: wildfly
labels:
  name: wildfly
```



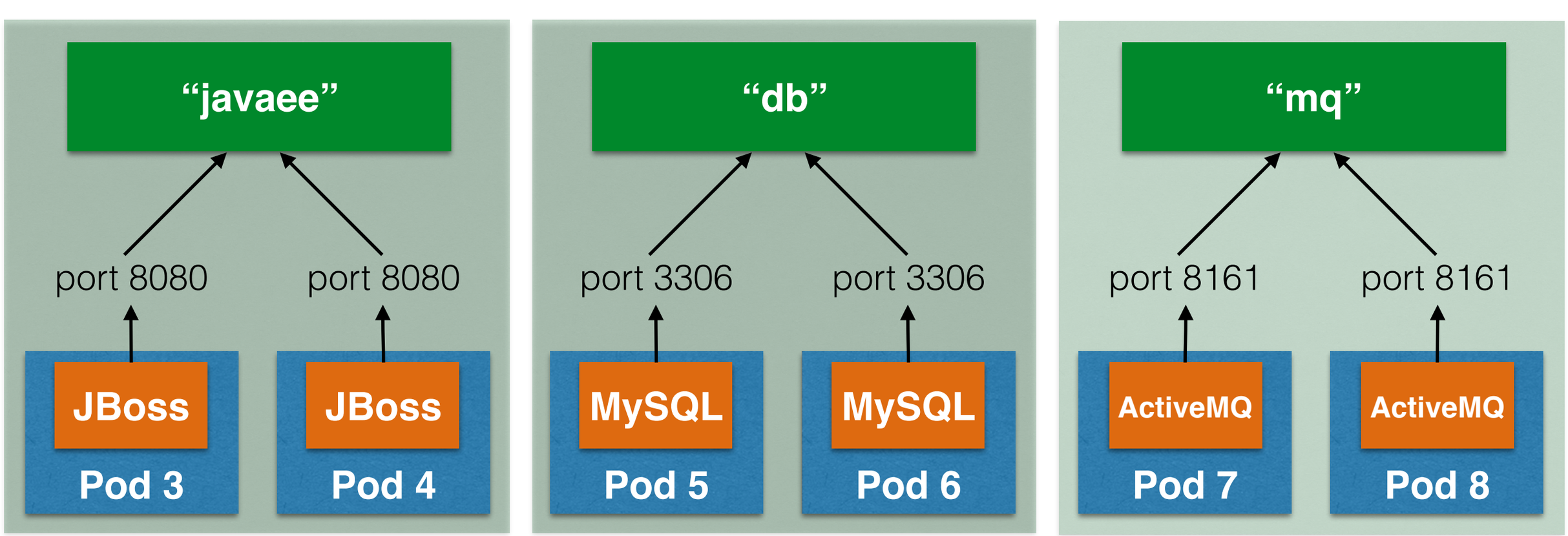
Kubernetes: Pros and Cons

- PROS
  - Manage related Docker containers as a unit
  - Container communication across hosts
  - Availability and scalability through automated deployment and monitoring of pods and their replicas, across hosts

Kubernetes: Pros and Cons

- CONS
  - Lifecycle of applications - build, deploy, manage, promote
  - Port existing source code to run in Kubernetes
  - DevOps: Dev -> Test -> Production
  - No multi-tenancy
  - On-premise (available on GCE)
    - Assumes inter-pod networking as part of infrastructure
    - Requires explicit load balancer

OPENSHIFT



redhat

User Experience

Containerized Services

Orchestration

Container

Container Host

OpenShift 3 Features

- Push to production - full DevOps
- Developer and operation tools for building web applications
- Centralized administration and management of application component libraries
- Team and user isolation of containers, builds, and network communication in an easy multi-tenancy system

Recipe #3.1

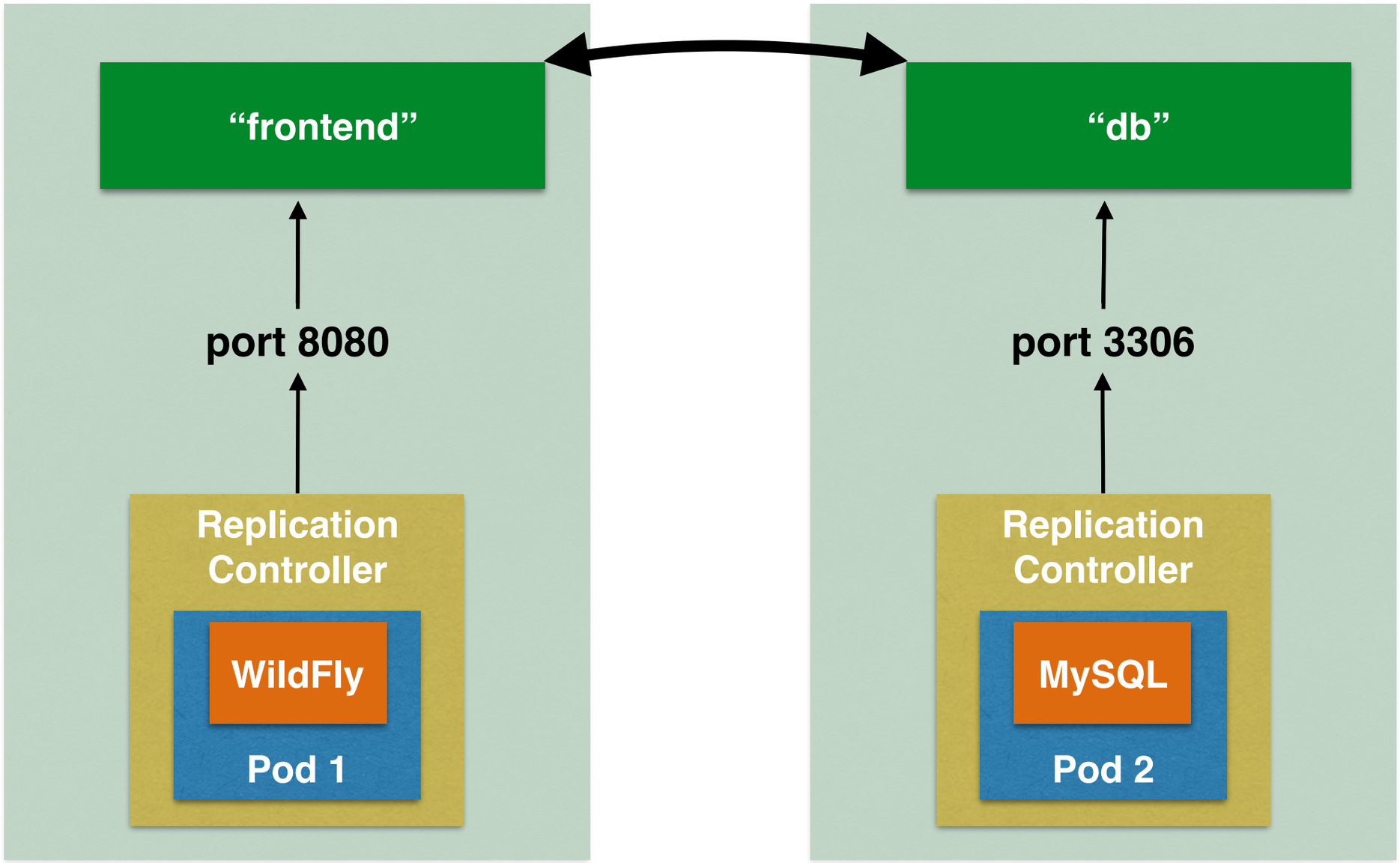
- Start OpenShift as Docker container

```
$ docker run -d --name "openshift-origin" --net=host --privileged \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v /tmp/openshift:/tmp/openshift \
    openshift/origin start
```

  - Or run natively

- Use **`osc`** (OpenShift Client) instead of **`kubectl`** with Kubernetes configuration file

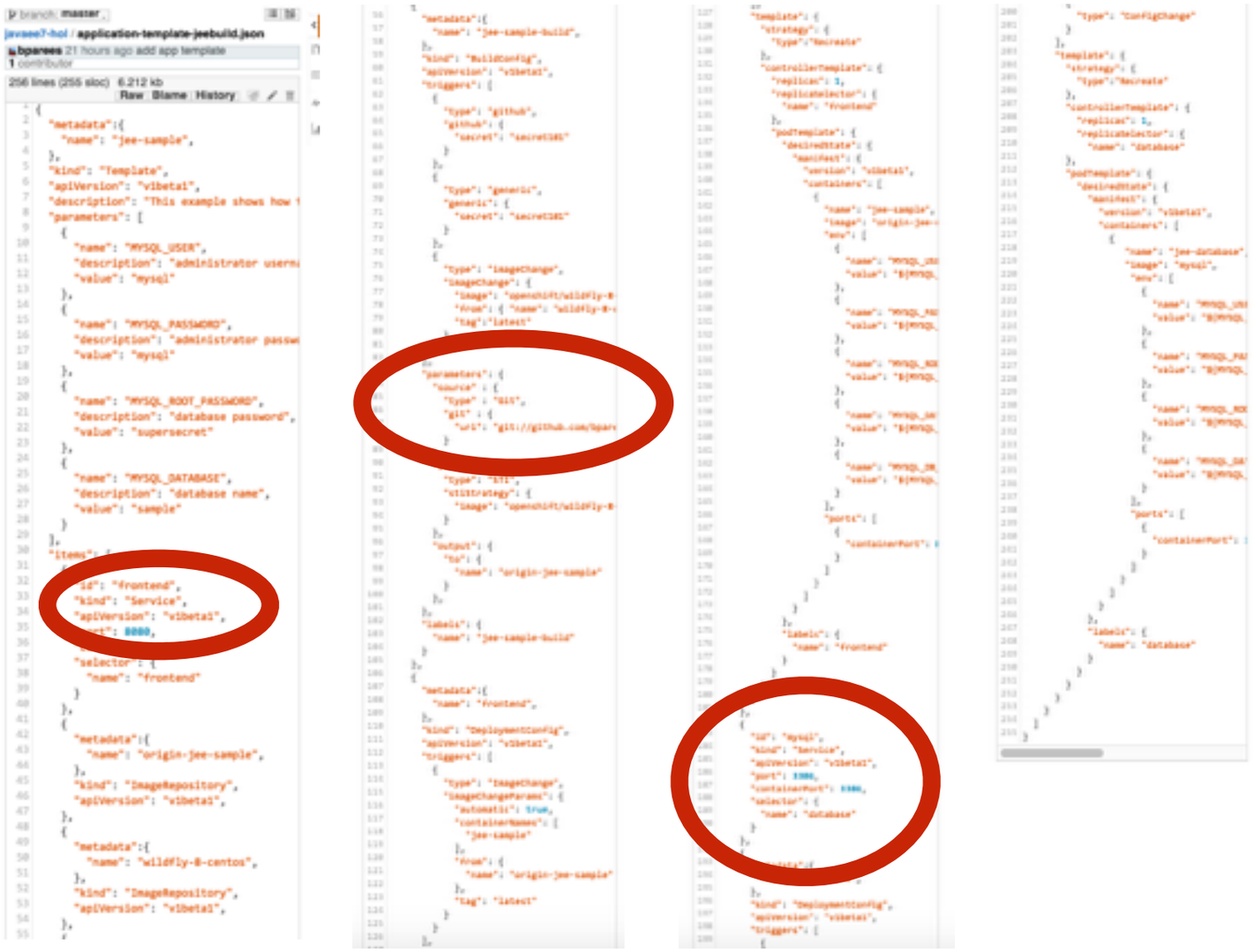
Recipe #3.2

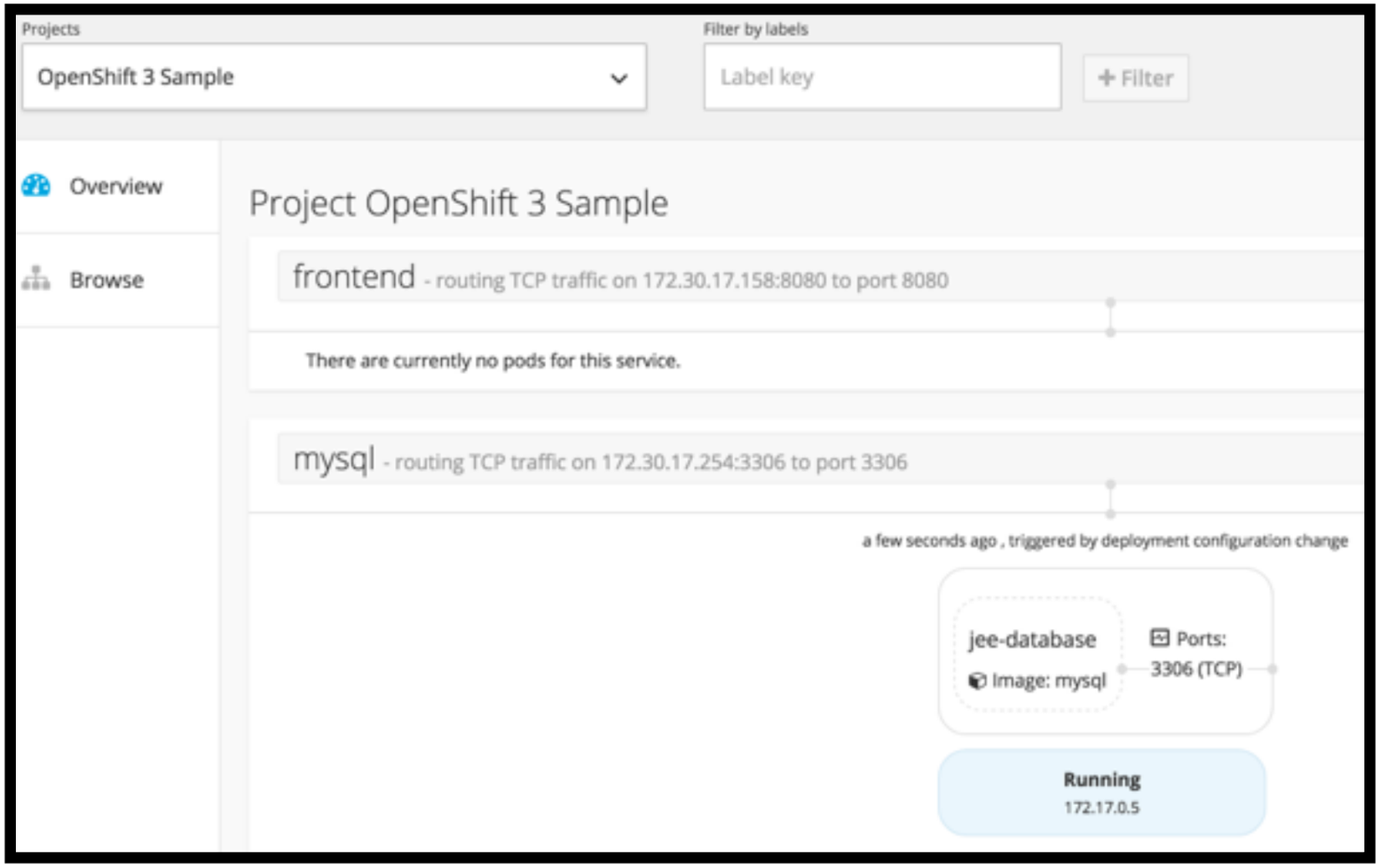


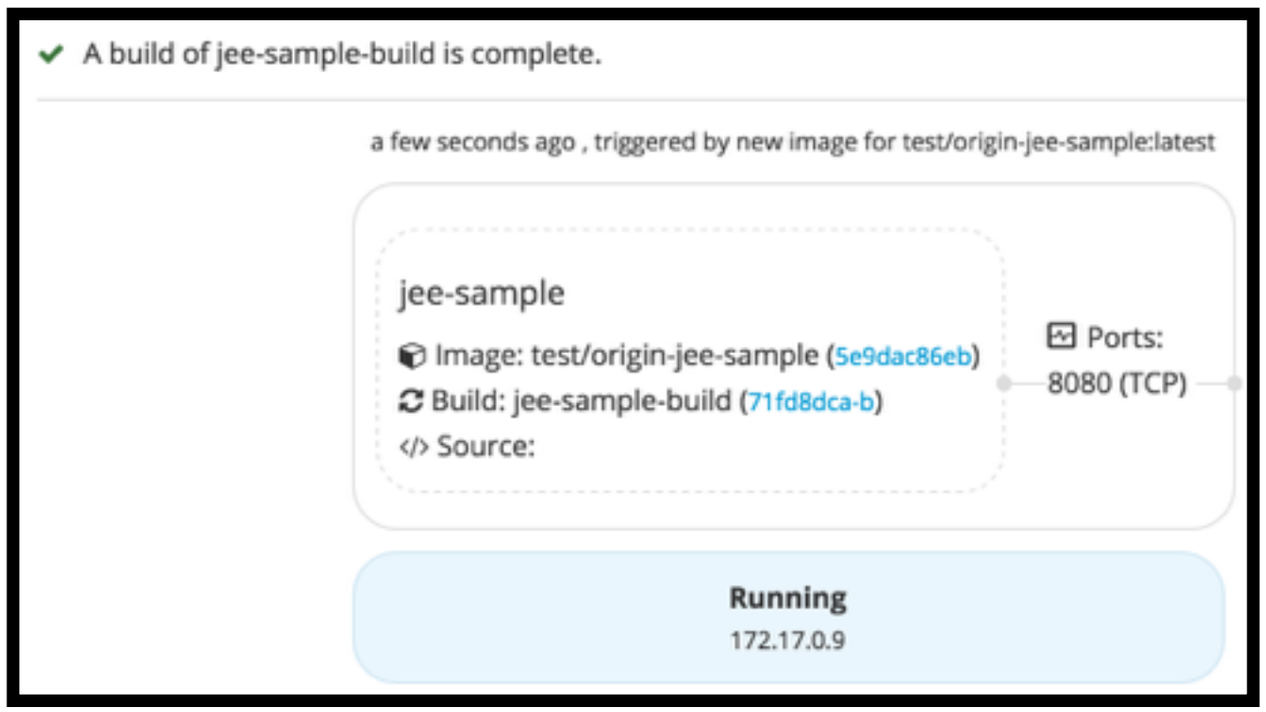
http://blog.arungupta.me/openshift-v3-getting-started-javaee7-wildfly-mysql/

Recipe #3.2

- (Alpha) tools generate project JSON configuration file that provide build/deployment




Projects
OpenShift 3 Sample
Filter by labels
Label key
+ Filter
Overview
Browse
Project OpenShift 3 Sample
frontend - routing TCP traffic on 172.30.17.158:8080 to port 8080
There are currently no pods for this service.
mysql - routing TCP traffic on 172.30.17.254:3306 to port 3306
a few seconds ago , triggered by deployment configuration change
jee-database
Image: mysql
Ports:
3306 (TCP)
Running
172.17.0.5


A build of jee-sample-build is complete.
a few seconds ago , triggered by new image for test/origin-jee-sample:latest
jee-sample
Image: test/origin-jee-sample (5e9dac86eb)
Build: jee-sample-build (71fd8dca-b)
</> Source:
Ports:
8080 (TCP)
Running
172.17.0.9

OPENSHIFT

Recipe #3.3

- Integration with JBoss Developer Studio (cooking)

JBoss Tools

redhat

Summary

- Container runtime and image distribution

- Roll your own solutions for everything
- Runtime and operational management of containers

- Lifecycle of applications - build, deploy, manage, promote
- Manage tens of thousands of applications with teams

References

- blog.arungupta.me/topics/containers/
- github.com/openshift/origin