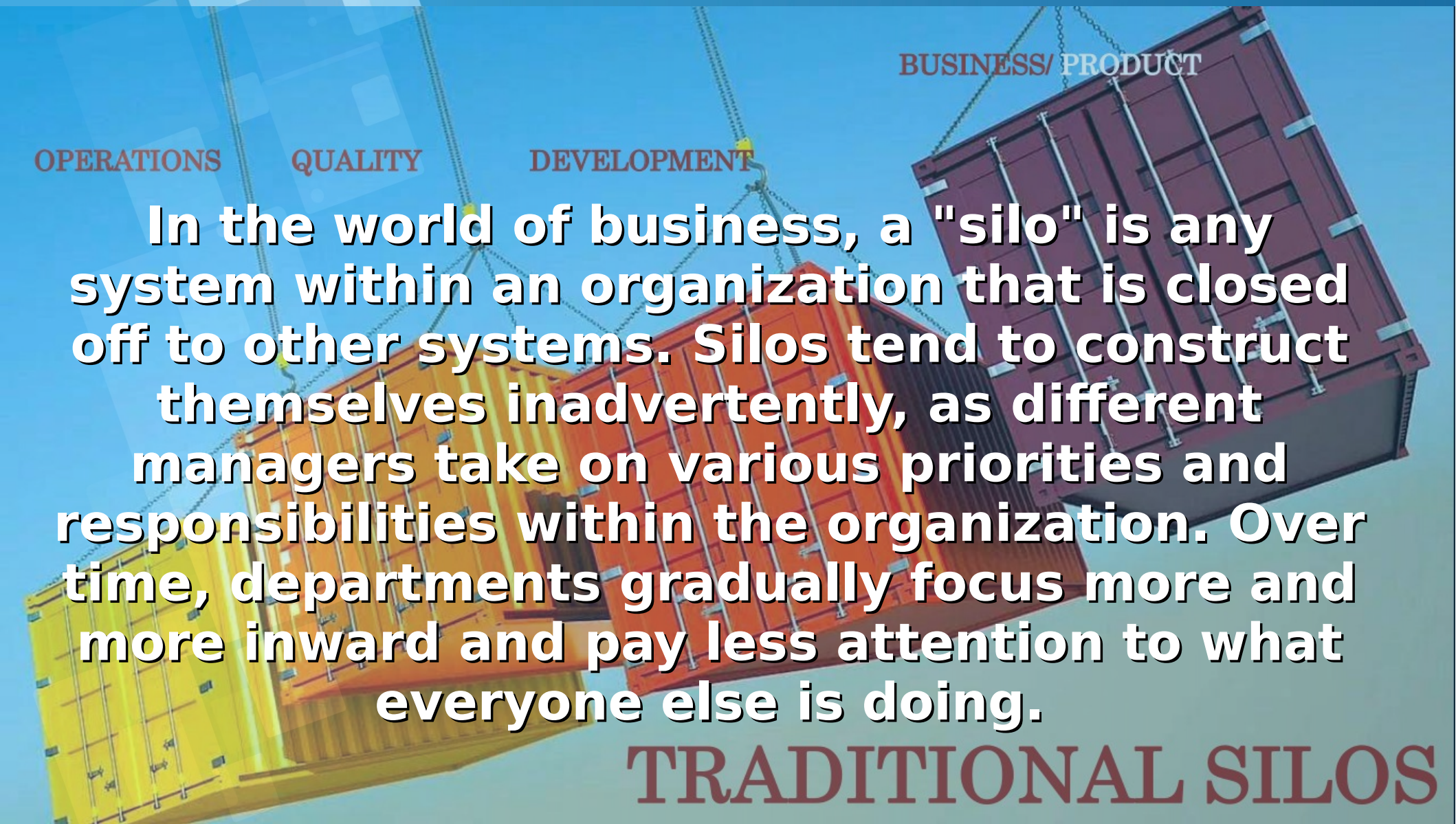
Journey to the Devops Automation with Docker, Kubernetes and OpenShift

Yusuf Hadiwinata Sutandar
LinuxGeek,OpenSourceEnthusiast,SecurityHobbies

Agenda

- Container & Docker Introduction
- Installing Docker Container & Management
- Managing Docker with Portainer
- Managing Docker with Openshift Origin
- Build Simple Docker Application
- Discussion

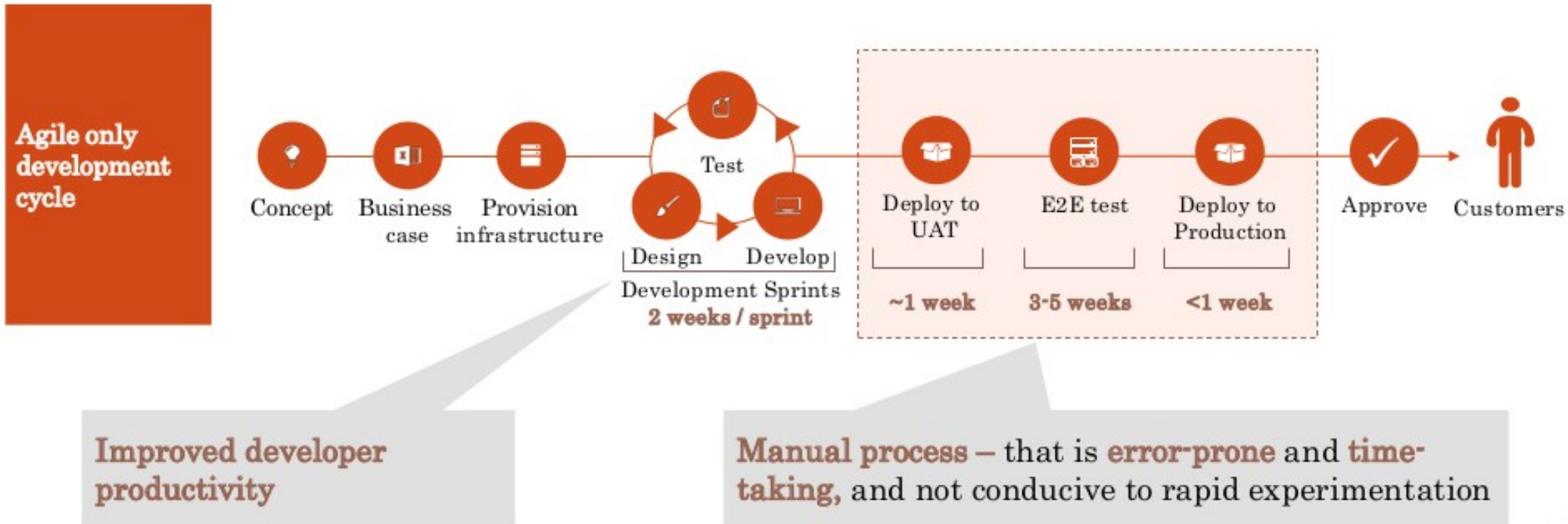
Traditional Development



In the world of business, a "silo" is any system within an organization that is closed off to other systems. Silos tend to construct themselves inadvertently, as different managers take on various priorities and responsibilities within the organization. Over time, departments gradually focus more and more inward and pay less attention to what everyone else is doing.

Traditional Problem

AGILE SOLVES THE PROBLEM OF BREAKING THE SILOS BETWEEN BUSINESS AND DEVELOPERS – IMPROVING DEVELOPER PRODUCTIVITY



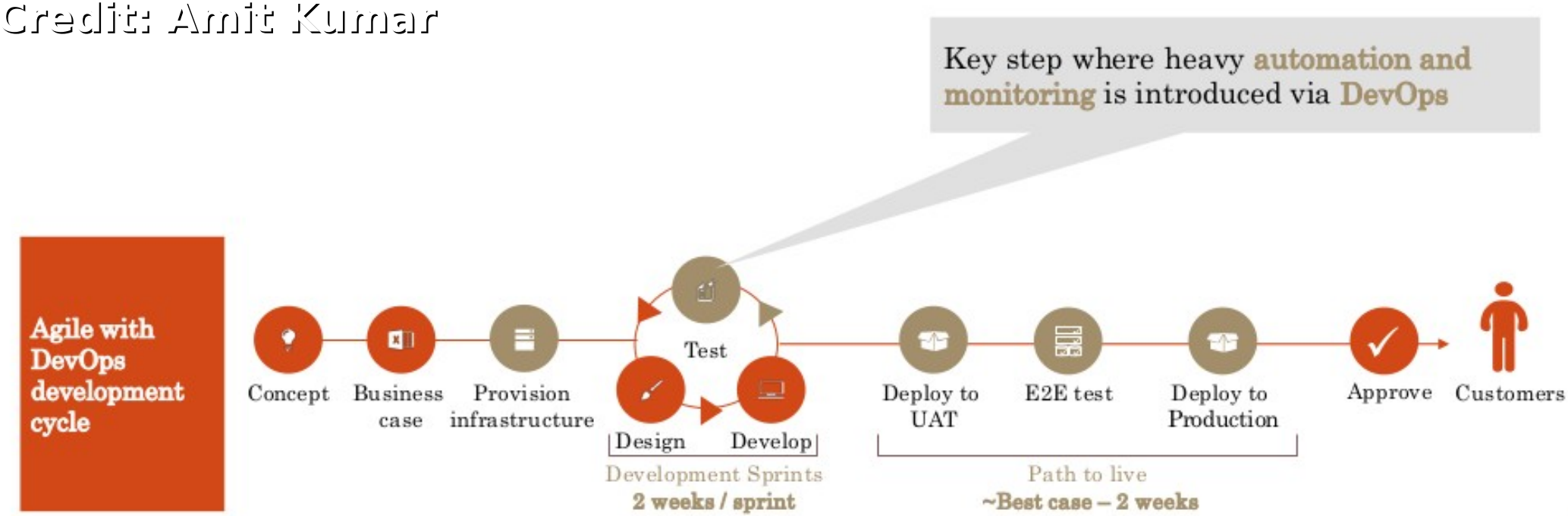
Credit: Amit Kumar

How DevOps Breaks Down Silos

DevOps Help Improve Collaboration Between Developers and IT Operations – Depoyment Automation & Self Services

On the other hand, DevOps culture is anti-silo by its very nature, while still retaining the subject matter experts that are so crucial to the software development process. DevOps requires developers, QA testers, operations engineers, and product managers to work closely together from the very beginning, which means that any existing silos will have to disappear quickly.

Credit: Amit Kumar



DevOps Tools & Software

Brief Intro to Container & Docker

History of Container
Docker Introduction

The Problem

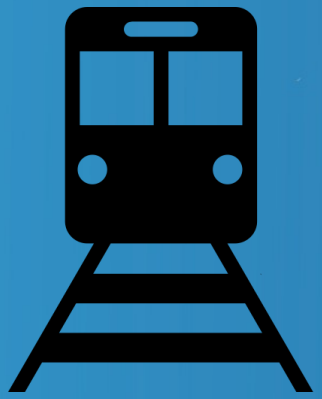
Cargo Transport Pre-1960

Solution?

Solution: Intermodal Shipping Container

Intermodal Shipping Container

Solution?

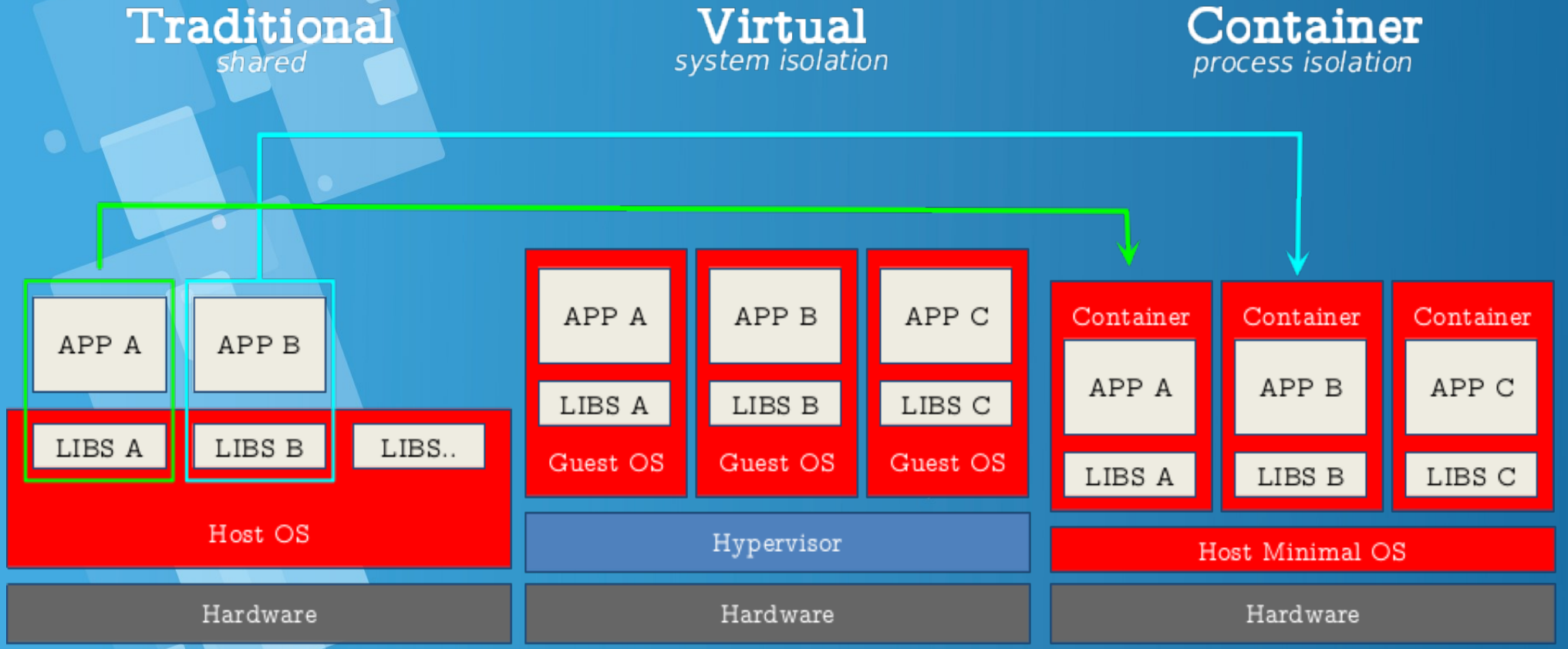


The Solution

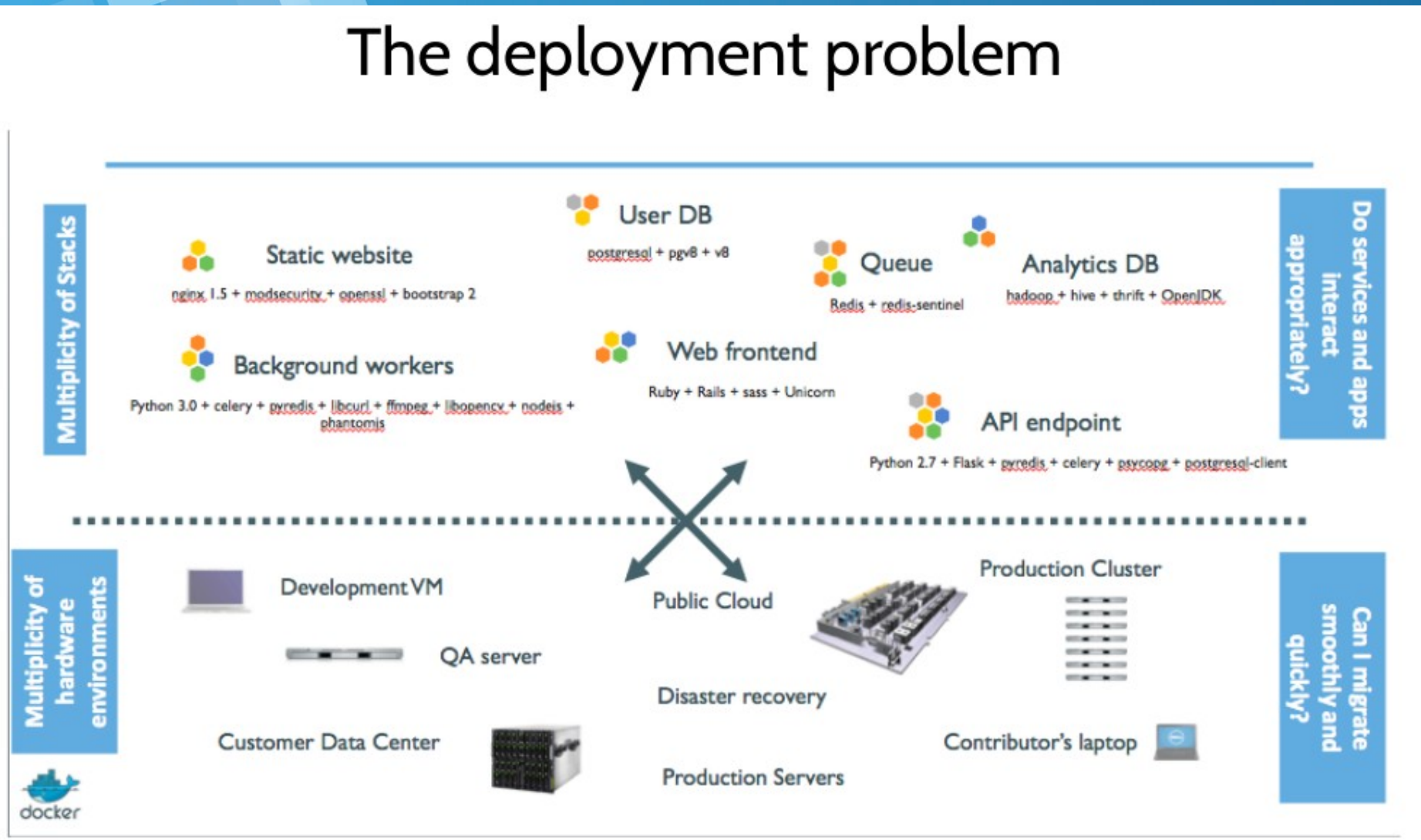
90% of all cargo now shipped in a standard container

Order of magnitude reduction in cost and time to load and unload ships, trains, trucks

The Evolution

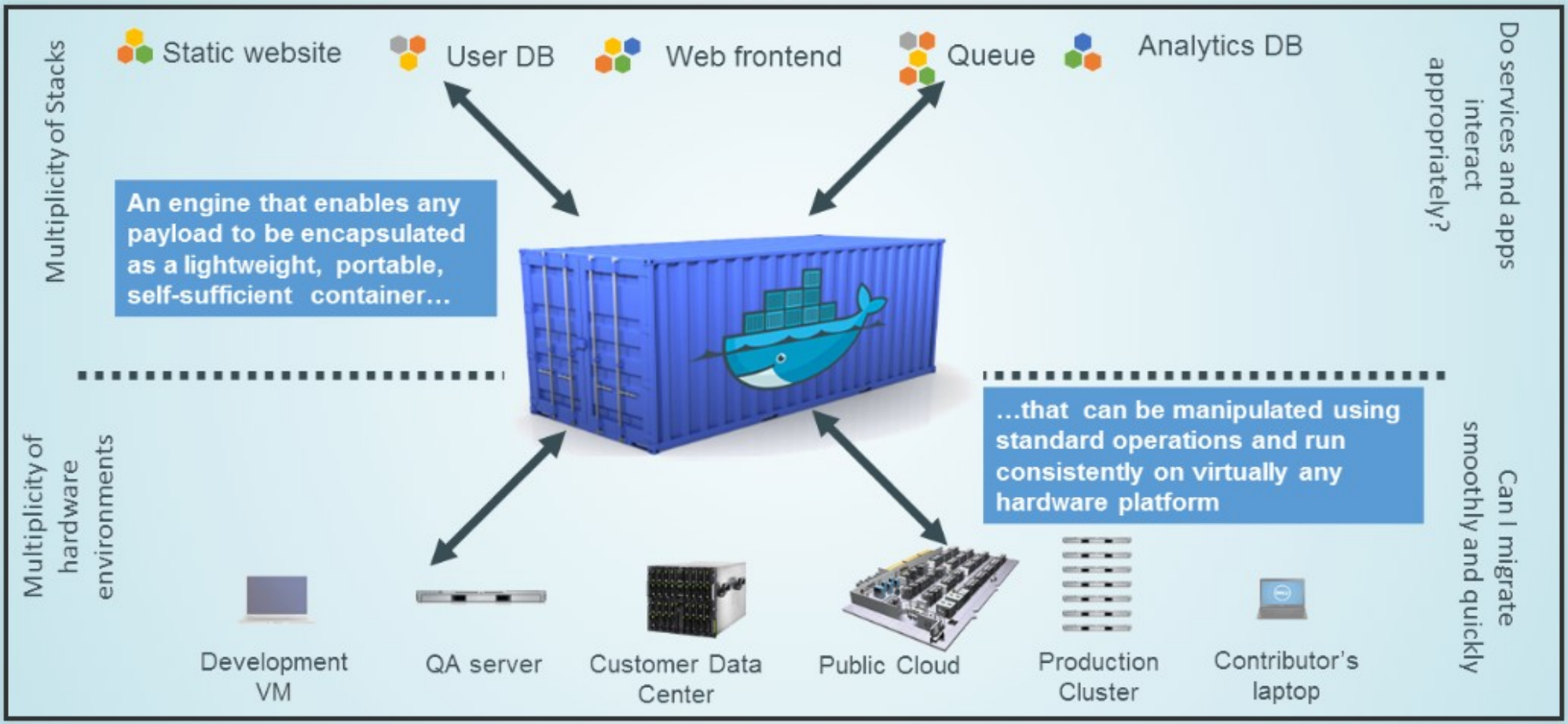


The App Deployment Problem



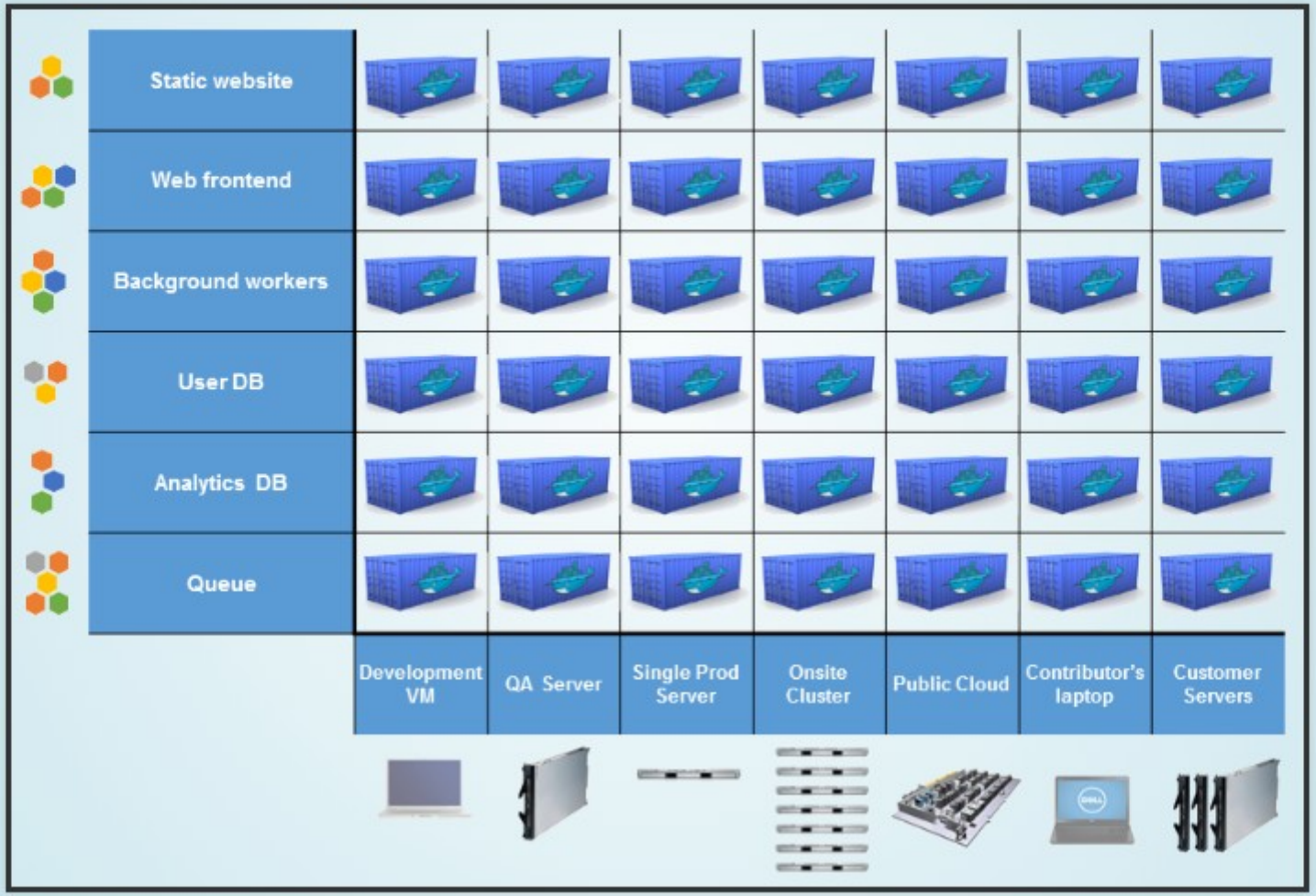
The App Deployment Solution

Docker is a Container System for Code



The App Deployment Solution

Docker Eliminates the Matrix from Hell



Container Technology

One way of looking at containers is as improved chroot jails. Containers allow an operating system (OS) process (or a process tree) to run isolated from other processes hosted by the same OS. Through the use of Linux kernel namespaces, it is possible to restrict a process view of:

- Other processes (including the pid number space)
- File systems
- User and group IDs
- IPC channels
- Devices
- Networking

Container Technology

Other Linux kernel features complement the process isolation provided by kernel namespaces:

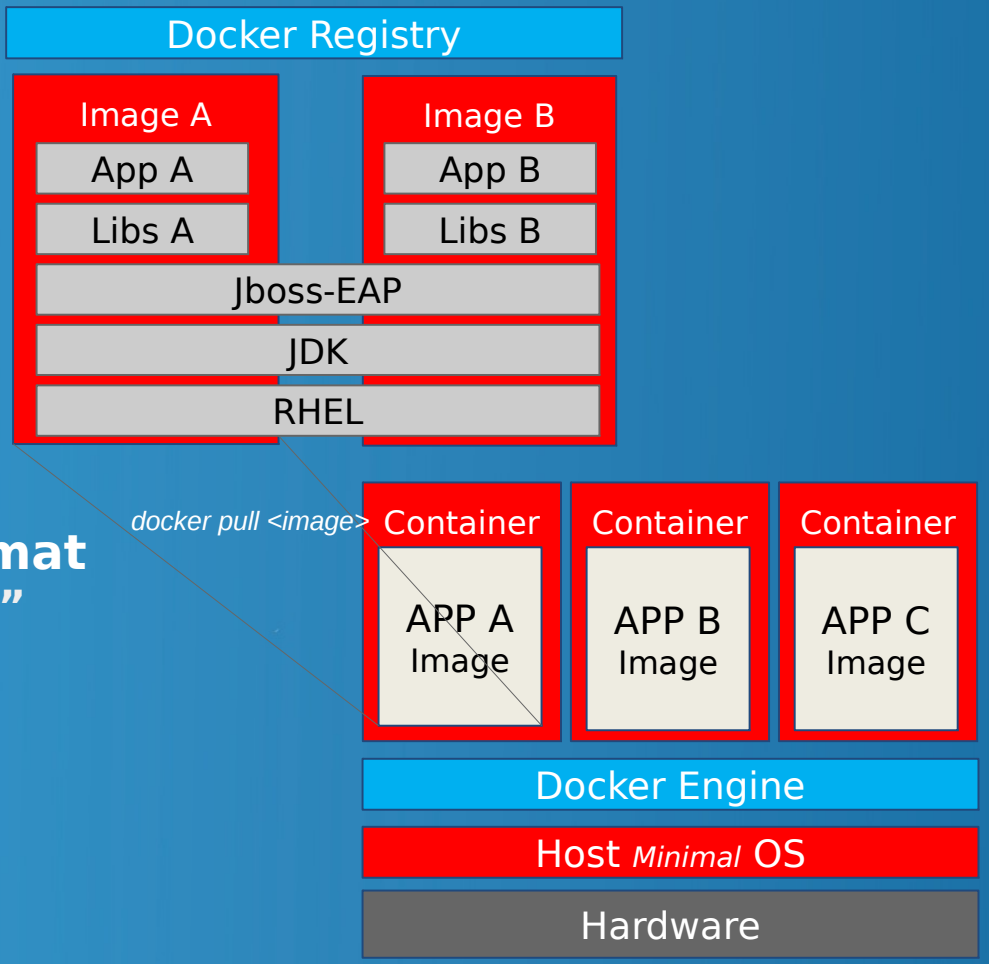
- Cgroups limit the use of CPU, RAM, virtual memory, and I/O bandwidth, among other hardware and kernel resources.
- Capabilities assign partial administrative faculties; for example, enabling a process to open a low

  network port (<1024) without allowing it to alter routing tables or change file ownership.
- SELinux enforces mandatory access policies even if the code inside the container finds a way to

  break its isolation

Container Technology

Images & Containers



- **Docker "Image"**
  - **Unified Packaging format**
    - **Like "war" or "tar.gz"**
    - **For any type of Application**
    - **Portable**

- **Docker "Container"**
  - **Runtime**
  - **Isolation**

Major Advantages of Containers

- **Low hardware footprint**
  - **Uses OS internal features to create an isolated environment where resources are managed using OS facilities such as namespaces and cgroups. This approach minimizes the amount of CPU and memory overhead compared to a virtual machine hypervisor. Running an application in a VM is a way to create isolation from the running environment, but it requires a heavy layer of services to support the same low hardware footprint isolation provided by containers.**
- **Environment isolation**
  - **Works in a closed environment where changes made to the host OS or other applications do not affect the container. Because the libraries needed by a container are self-contained, the application can run without disruption. For example, each application can exist in its own container with its own set of libraries. An update made to one container does not affect other containers, which might not work with the update.**

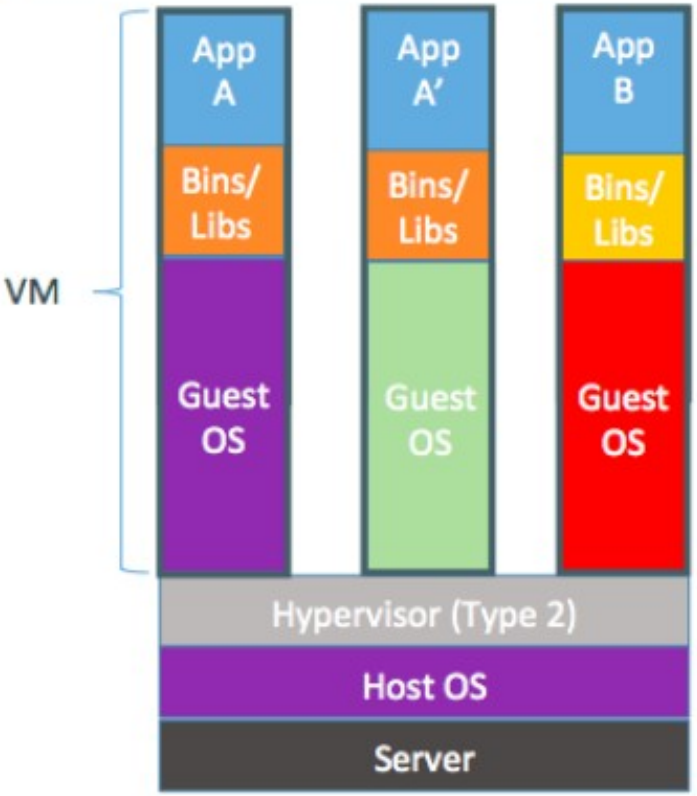
Major Advantages of Containers cont..

- **Quick deployment**
  - **Deploys any container quickly because there is no need for a full OS install or restart. Normally, to support the isolation, a new OS installation is required on a physical host or VM, and any simple update might require a full OS restart. A container only requires a restart without stopping any services on the host OS.**
- **Multiple environment deployment**
  - **In a traditional deployment scenario using a single host, any environment differences might potentially break the application. Using containers, however, the differences and incompatibilities are mitigated because the same container image is used.**
- **Reusability**
  - **The same container can be reused by multiple applications without the need to set up a full OS. A database container can be used to create a set of tables for a software application, and it can be quickly destroyed and recreated without the need to run a set of housekeeping tasks. Additionally, the same database container can be used by the production environment to deploy an application.**

Why are Containers Lightweight?

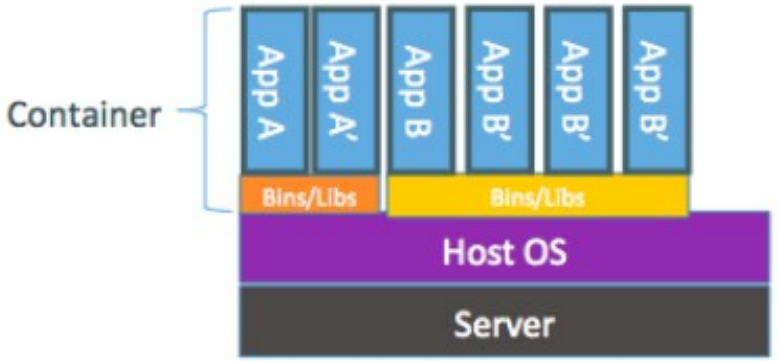
containers as **lightweight VMs**

Less overhead!



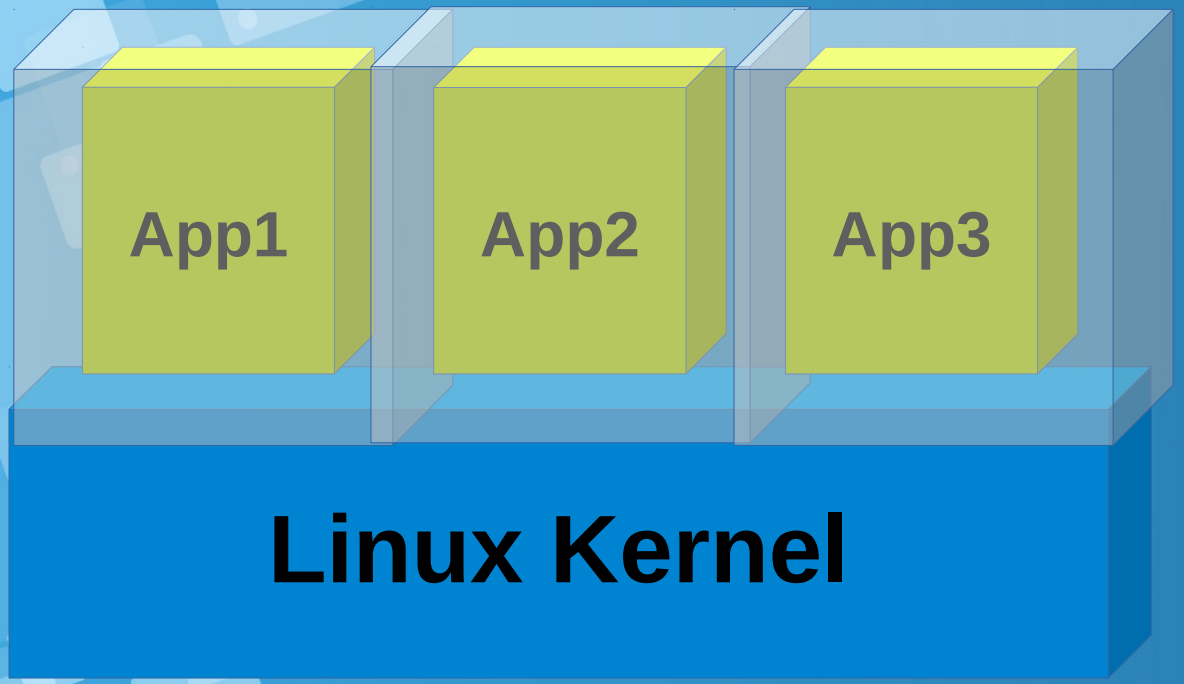
Containers are isolated, but share OS kernel and, where appropriate, bins/libraries

...result is significantly faster deployment, much less overhead, easier migration, faster restart



Is not Virtualizaiton :)

Isolation, not Virtualization



- Kernel Namespaces
  - Process
  - Network
  - IPC
  - Mount
  - User
- Resource Limits
  - Cgroups
- Security
  - SELinux

Container Solution

Virtual Machine and Container Complement each other

Virtual Machine

- Virtual machines include the application, the necessary binaries and libraries, and an entire guest operating system
- Each Guest OS has its own Kernel and user space

Containers

- Containers run as isolated processes in user space of host OS
- They share the kernel with other container (container-processes)
- Containers include the application and all of its dependencies
- Not tied to specific infrastructure

Container Problem

Containers before Docker

- No standardized exchange format.

(No, a rootfs tarball is not a format!)

- Containers are hard to use for developers.

(Where's the equivalent of docker run debian?)

- No re-usable components, APIs, tools.

(At best: VM abstractions, e.g. libvirt.)

Analogy:

- Shipping containers are not just steel boxes.
- They are steel boxes that are a standard size, with the same hooks and holes

Docker Solution

Containers after Docker

- Standardize the container format, because containers were not portable.
- Make containers easy to use for developers.
- Emphasis on re-usable components, APIs, ecosystem of standard tools.
- Improvement over ad-hoc, in-house, specific tools.

Docker Solution

Docker Architecture

Docker is one of the container implementations available for deployment and supported by companies such as Red Hat in their Red Hat Enterprise Linux Atomic Host platform. Docker Hub provides a large set of containers developed by the community.

Docker Solution

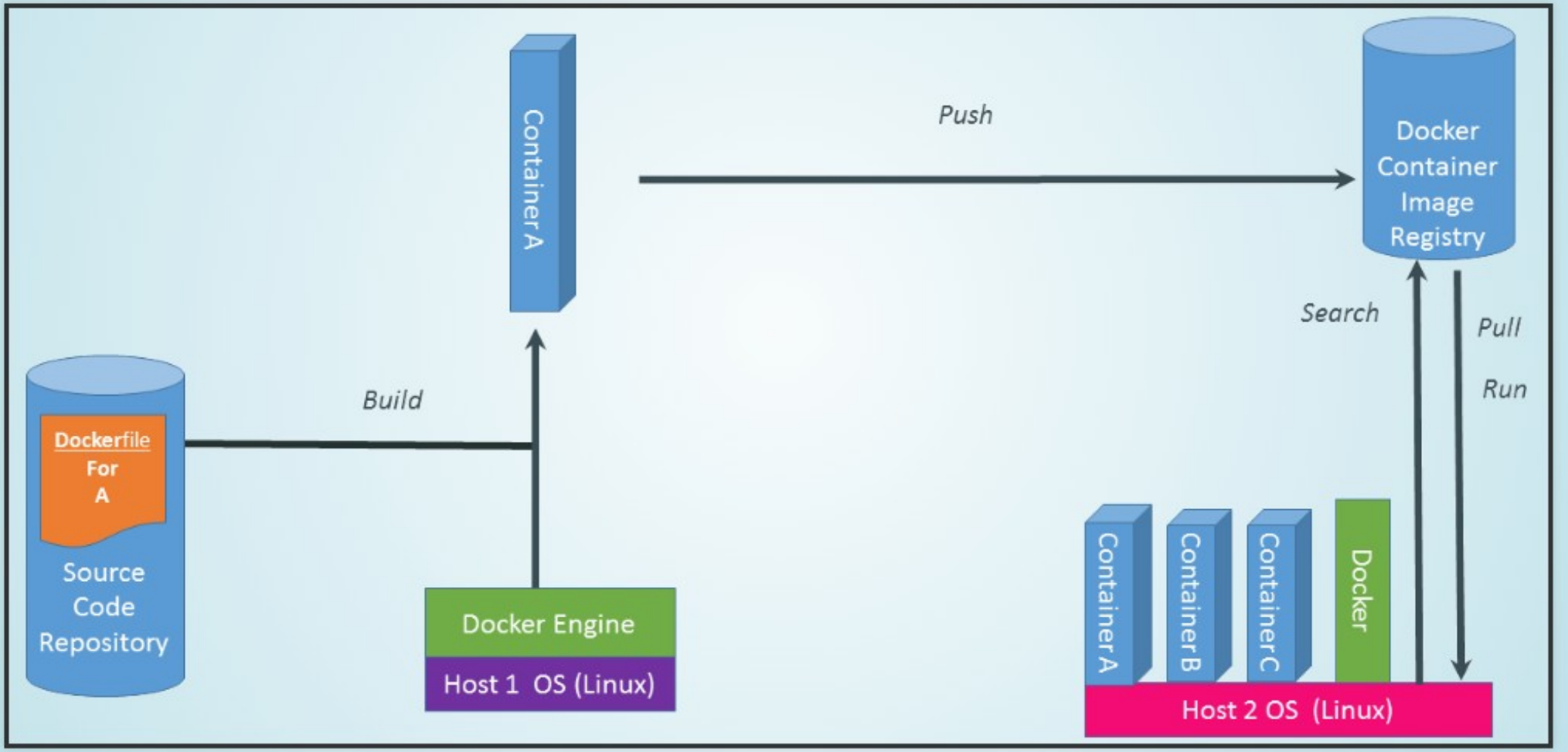
Docker Core Elements

- Images
  - Images are read-only templates that contain a runtime environment that includes application libraries and applications. Images are used to create containers. Images can be created, updated, or downloaded for immediate consumption.

- Registries
  - Registries store images for public or private use. The well-known public registry is Docker Hub, and it stores multiple images developed by the community, but private registries can be created to support internal image development under a company's discretion. This course runs on a private registry in a virtual machine where all the required images are stored for faster consumption.

- Containers
  - Containers are segregated user-space environments for running applications isolated from other applications sharing the same host OS.

Basics of a Docker System?



Ecosystem Support

- **DevOps Tools**
  - **Integrations with Chef, Puppet, Jenkins, Travis, Salt, Ansible +++**
- **Orchestration tools**
  - **Mesos, Heat, ++**
  - **Shipyard & others purpose built for Docker**
- **Applications**
  - **1000’s of Dockerized applications available at index.docker.io**

Ecosystem Support Continue..

- Operating systems
  - Virtually any distribution with a 2.6.32+ kernel
  - Red Hat/Docker collaboration to make work across RHEL 6.4+, Fedora, and other members of the family (2.6.32 +)
  - CoreOS—Small core OS purpose built with Docker
- OpenStack
  - Docker integration into NOVA (& compatibility with Glance, Horizon, etc.) accepted for Havana release
- Private PaaS
  - OpenShift, Solum (Rackspace, OpenStack), Other TBA
- Public PaaS
  - Deis, Voxoz, Cocaine (Yandex), Baidu PaaS
- Public IaaS
  - Native support in Rackspace, Digital Ocean,+++
  - AMI (or equivalent) available for AWS & other

What IT`s Said about Docker:

Developer Say:

Build Once, Run Anywhere

Operator: Configure Once, Run Anything

Why Developers Care

Developer Say:

Build Once, Run Anywhere

A clean, safe, hygienic, portable runtime environment for your app.

No worries about missing dependencies, packages and other pain points during subsequent deployments.

Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying.

Automate testing, integration, packaging...anything you can script.

Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.

Cheap, zero-penalty containers to deploy services. A VM without the overhead of a VM. Instant replay and reset of image snapshots.

Why Administrators Care

Operator: Configure Once, Run Anything

Make the entire lifecycle more efficient, consistent, and repeatable

Increase the quality of code produced by developers.

Eliminate inconsistencies between development, test, production, and customer environments.

Support segregation of duties.

Significantly improves the speed and reliability of continuous deployment and continuous integration systems.

Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VMs.

Managing Docker Containers

- Installing Docker
- create/start/stop/remove containers
- inspect containers
- interact, commit new images

Lab: Installing Docker - Requirment

- Requirment:
  - Centos 7.3 Minimal Install
  - Latest update with “yum -y update”
  - 16 GB OS Disk
  - 16 GB Unpartition Disk for Docker Storage
  - 2 GB Ram and 2 vCPU
  - Bridge Network for connecting to Internet and Accessing from Host (laptop)
  - Snapshoot VM

Lab: Installing Docker - PreSetup

- **Setting hostname at /etc/hosts file and server:**

```
[root@docker-host ~]# cat /etc/hosts | grep docker-host
192.168.0.6  docker-host

[root@docker-host ~]# hostnamectl set-hostname docker-host
[root@docker-host ~]# hostname
docker-host
```

- **Install needed packages and Latest Docker**

```
[root@docker-host ~]# yum install wget git net-tools bind-utils iptables-services
bridge-utils bash-completion
[root@docker-host ~]# curl -fsSL https://get.docker.com/ | sh
```

- **Edit /etc/sysconfig/docker file and add --insecure-registry 172.30.0.0/16 to the OPTIONS parameter (installing docker from repo only)**

```
[root@docker-host ~]# sed -i '/OPTIONS=.*/c\OPTIONS="--selinux-enabled --insecure-
registry 172.30.0.0/16"' /etc/sysconfig/docker
[root@docker-host ~]# systemctl is-active docker ; systemctl enable docker ; systemctl
start docker
```

- **Pulling your 1st container from internet**

```
[root@docker-host ~]# docker container run -ti ubuntu bash
```

Lab: Installing Docker - PreSetup

- **When you use Latest version docker, please configure this**

```
[root@docker-host ~]# vim /usr/lib/systemd/system/docker.service
Edit
ExecStart=/usr/bin/dockerd
to
ExecStart=/usr/bin/dockerd --insecure-registry 172.30.0.0/16 --insecure-registry
192.168.1.0/24

[root@docker-host ~]# systemctl daemon-reload ; systemctl restart docker
```

- **Optional Configuration for private registry**

```
[root@docker-host ~]# vim /etc/docker/daemon.json
Add
{
  "insecure-registries" : ["docker-registry:5000"]
}

[root@docker-host ~]# systemctl restart docker
```

Pulling your 1st container from private registry

```
[root@docker-host ~]# docker container run -ti docker-registry:5000/ubuntu bash
```

Installing Docker – Setting Docker Storage

- By default, docker-storage-setup looks for free space in the root volume group and creates an LVM thin pool. Hence you can leave free space during system installation in the root volume group and starting docker will automatically set up a thin pool and use it.

- Setting up a volume group and LVM thin pool on user specified block device for docker version 1.12

```
[root@docker-host ~]# echo DEVS=/dev/sdb >> /etc/sysconfig/docker-storage-setup
[root@docker-host ~]# systemctl restart docker
```

- LVM thin pool in a user specified volume group

```
[root@docker-host ~]# echo VG=docker-vg >> /etc/sysconfig/docker-storage-setup
[root@docker-host ~]# systemctl restart docker
```

- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/managing_containers/managing_storage_with_docker_formatted_containers

Lab: 1st time Playing w/ Docker

```
[root@docker-host ~]# docker ps
CONTAINER ID     IMAGE    COMMAND     CREATED    STATUS         PORTS         NAMES
```

```
[root@docker-host ~]# docker run -t centos bash
Unable to find image 'centos:latest' locally
Trying to pull repository docker.io/library/centos ...
sha256:aebf12af704307dfa0079b3babdca8d7e8ff6564696882bcb5d11f1d461f9ee9: Pulling from
docker.io/library/centos
d5e46245fe40: Pull complete
Digest: sha256:aebf12af704307dfa0079b3babdca8d7e8ff6564696882bcb5d11f1d461f9ee9
Status: Downloaded newer image for docker.io/centos:latest

[root@docker-host ~]# docker images --all
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
docker.io/centos    latest              3bee3060bfc8        46 hours ago        192.5 MB

[root@docker-host ~]# docker exec -it 60fec4b9a9bf bash
[root@60fec4b9a9bf /]# ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss+    0:00 bash
   29 ?        Ss     0:00 bash
   42 ?        R+     0:00 ps ax
```

Docker Management Commands

Command	Description
docker create image [command] docker run image [command]	create the container = create + start
docker start container. . . docker stop container. . . docker kill container. . . docker restart container. . .	start the container graceful 2 stop kill (SIGKILL) the container = stop + start
docker pause container. . . docker unpause container. . .	suspend the container resume the container
docker rm [-f 3] container. . .	destroy the container

docker run - Run a container

```
docker run [ options ] image [ arg0 arg1...]
```

- create a container and start it
  - the container filesystem is initialised from image image
  - arg0..argN is the command run inside the container (as PID 1)

```
[root@docker-host ~]# docker run centos /bin/hostname
f0d0720bd373
[root@docker-host ~]# docker run centos date +%H:%M:%S
17:10:13
[root@docker-host ~]# docker run centos true ; echo $?
0
[root@docker-host ~]# docker run centos false ; echo $?
1
```

docker run - Foreground mode vs. Detached mode

- Foreground mode is the default
  - stdout and stderr are redirected to the terminal
  - docker run propagates the exit code of the main process
- With -d, the container is run in detached mode:
  - displays the ID of the container
  - returns immediately

```
[root@docker-host ~]# docker run centos date
Wed Jun  7 15:35:48 UTC 2017
[root@docker-host ~]# docker run -d centos date
48b66ad5fc30c468ca0b28ff83dfec0d6e001a2f53e3d168bca754ea76d2bc04
[root@docker-host ~]# docker logs 48b66a
Tue Jan 20 17:32:16 UTC 2015
```

docker run - interactive mode

- By default containers are non-interactive
  - stdin is closed immediately
  - terminal signals are not forwarded 5

```
$ docker run -t debian bash
root@6fecc2e8ab22:/# date
^C
$
```

- With -i the container runs interactively
  - stdin is usable
  - terminal signals are forwarded to the container

```
$ docker run -t -i debian bash
root@78ff08f46cdb:/# date
Tue Jan 20 17:52:01 UTC 2015
root@78ff08f46cdb:/# ^C
root@78ff08f46cdb:/#
```

docker run - Set the container name

- **--name option, assigns a name for the container (by default a random name is generated → adjective name)**

```
[root@docker-host ~]# docker run -d -t debian
da005df0d3aca345323e373e1239216434c05d01699b048c5ff277dd691ad535
[root@docker-host ~]# docker run -d -t --name blahblah debian
0bd3cb464ff68eaf9fc43f0241911eb207fefd9c1341a0850e8804b7445ccd21
[root@docker-host ~]# docker ps
CONTAINER ID IMAGE        COMMAND      CREATED              .. NAMES
0bd3cb464ff6 debian:7.5   "/bin/bash"  6 seconds ago        blahblah
Da005df0d3ac debian:7.5   "/bin/bash"  About a minute ago   focused_raman
[root@docker-host ~]# docker stop blahblah focused_raman
```

- **Note: Names must be unique**

```
[root@docker-host ~]# docker run --name blahblah debian true
2015/01/20 19:31:21 Error response from daemon: Conflict, The name blahblah is already
assigned
to 0bd3cb464ff6. You have to delete (or rename) that container to be able to assign
blahblah to a
container again.
```

Inspecting the container

Command	Description
docker ps	list running containers
docker ps -a	list all containers
docker logs [-f 6] container	show the container output (stdout+stderr)
docker top container [ps options]	list the processes running inside the containers
docker diff container	show the differences with the image (modified files)
docker inspect container. .	show low-level infos (in json format)

Interacting with the container

Command	Description
docker attach container	attach to a running container (stdin/stdout/stderr)
docker cp container:path hostpath|- docker cp hostpath|- container:path	copy files from the container copy files into the container
docker export container	export the content of the container (tar archive)
docker exec container args. . .	run a command in an existing container (useful for debugging)
docker wait container	wait until the container terminates and return the exit code
docker commit container image	commit a new docker image (snapshot of the container)

Lab: Docker commit example

```
[root@docker-host ~]# docker run --name my-container -t -i debian
root@3b397d383faf:/# cat >> /etc/bash.bashrc <<EOF
> echo 'hello!'
> EOF
root@3b397d383faf:/# exit
[root@docker-host ~]# docker start --attach my-container
my-container
hello!
root@3b397d383faf:/# exit
[root@docker-host ~]# docker diff my-container
C /etc
C /etc/bash.bashrc
A /.bash_history
C /tmp
[root@docker-host ~]# docker commit my-container hello
a57e91bc3b0f5f72641f19cab85a7f3f860a1e5e9629439007c39fd76f37c5dd
[root@docker-host ~]# docker stop my-container; docker rm my-container
my-container
[root@docker-host ~]# docker run --rm -t -i hello
hello!
root@386ed3934b44:/# exit

[root@docker-host ~]# docker images --all
REPOSITORY       TAG                 IMAGE ID            CREATED                  SIZE
debian           latest              a25c1eed1c6f        Less than a second ago   123MB
hello            latest              52442a43a78b        59 seconds ago           123MB
centos           latest              3bee3060bfc8        46 hours ago             193MB
ubuntu           latest              7b9b13f7b9c0        4 days ago               118MB
```

Inputs/Outputs

- External volumes (persistent data)
- Devices & Links
- Publishing ports (NAT)

docker run - Mount external volumes

```
docker run -v /hostpath:/containerpath[:ro] ...
```

- -v mounts the location /hostpath from the host filesystem at the
  - location /containerpath inside the container
  - With the ":ro" suffix, the mount is read-only
- Purposes:
  - store persistent data outside the container
  - provide inputs: data, config files, . . . (read-only mode)
  - inter-process communicattion (unix sockets, named pipes)

Lab: Mount examples

- **Persistent data**

```
[root@docker-host ~]# docker run --rm -t -i -v /tmp/persistent:/persistent debian
root@0aeedfeb7bf9:/# echo "blahblah" >/persistent/foo
root@0aeedfeb7bf9:/# exit
[root@docker-host ~]# cat /tmp/persistent/foo
blahblah
[root@docker-host ~]# docker run --rm -t -i -v /tmp/persistent:/persistent debian
root@6c8ed008c041:/# cat /persistent/foo
blahblah
```

- **Inputs (read-only volume)**

```
[root@docker-host ~]# mkdir /tmp/inputs
[root@docker-host ~]# echo hello > /tmp/inputs/bar
[root@docker-host ~]# docker run --rm -t -i -v /tmp/inputs:/inputs:ro debian
root@05168a0eb322:/# cat /inputs/bar
hello
root@05168a0eb322:/# touch /inputs/foo
touch: cannot touch `/inputs/foo': Read-only file system
```

Lab: Mount examples continue ...

- **Named pipe**

```
[root@docker-host ~]# mkfifo /tmp/fifo
[root@docker-host ~]# docker run -d -v /tmp/fifo:/fifo debian sh -c 'echo blah blah>
/fifo'
ff0e44c25e10d516ce947eae9168060ee25c2a906f62d63d9c26a154b6415939
[root@docker-host ~]# cat /tmp/fifo
blah blah
```

- **Unix socket**

```
[root@docker-host ~]# docker run --rm -t -i -v /dev/log:/dev/log debian
root@56ec518d3d4e:/# logger blah blah blah
root@56ec518d3d4e:/# exit
[root@docker-host ~]# cat /var/log/messages | grep blah
Oct 17 15:39:39 docker-host root: blah blah blah
```

docker run-inter-container links (legacy links)

- Containers cannot be assigned a static IP address (by design)

  → service discovery is a must

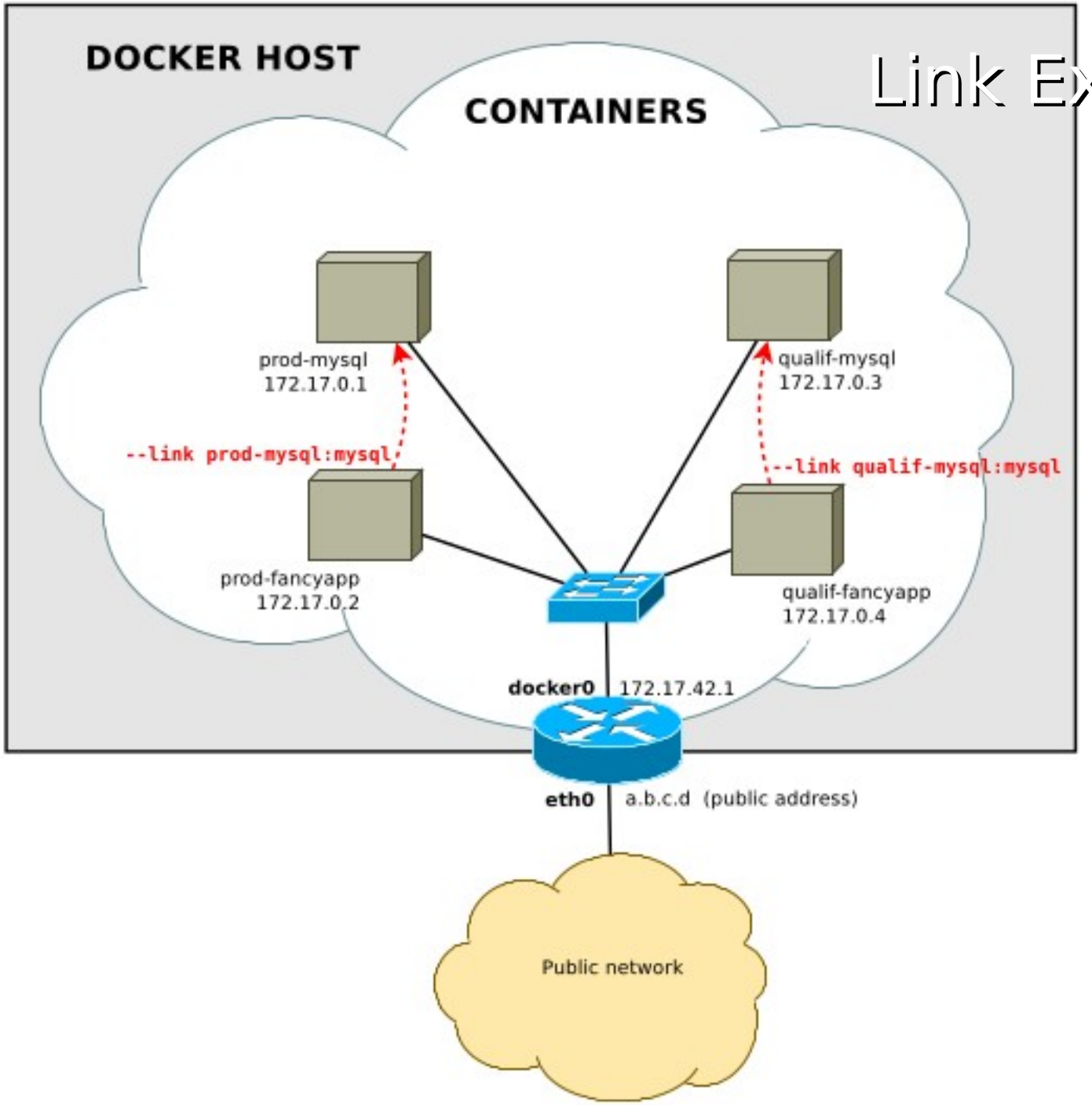
  Docker “links” are the most basic way to discover a service

```
docker run --link ctr:alias . . .
```

- → container ctr will be known as alias inside the new container

```
[root@docker-host ~]# docker run --name my-server debian sh -c 'hostname -i && sleep
500' &
172.17.0.4
[root@docker-host ~]# docker run --rm -t -i --link my-server:srv debian
root@d752180421cc:/# ping srv
PING srv (172.17.0.4): 56 data bytes
64 bytes from 172.17.0.4: icmp_seq=0 ttl=64 time=0.195 ms
```

Link Example



User-defined networks (since v1.9.0)

- by default new containers are connected to the main network (named “bridge”, 172.17.0.0/16)

- the user can create additional networks:

```
docker network create NETWORK
```

- newly created containers are connected to one network:

```
docker run -t --name test-network --net=NETWORK debian
```

- container may be dynamically attached/detached to any
- network:

```
docker inspect test-network | grep -i NETWORK
docker network list
docker network connect NETWORK test-network
docker network connect bridge test-network
docker network disconnect NETWORK test-network
```

- networks are isolated from each other, communications is possible by attaching a container to multiple networks

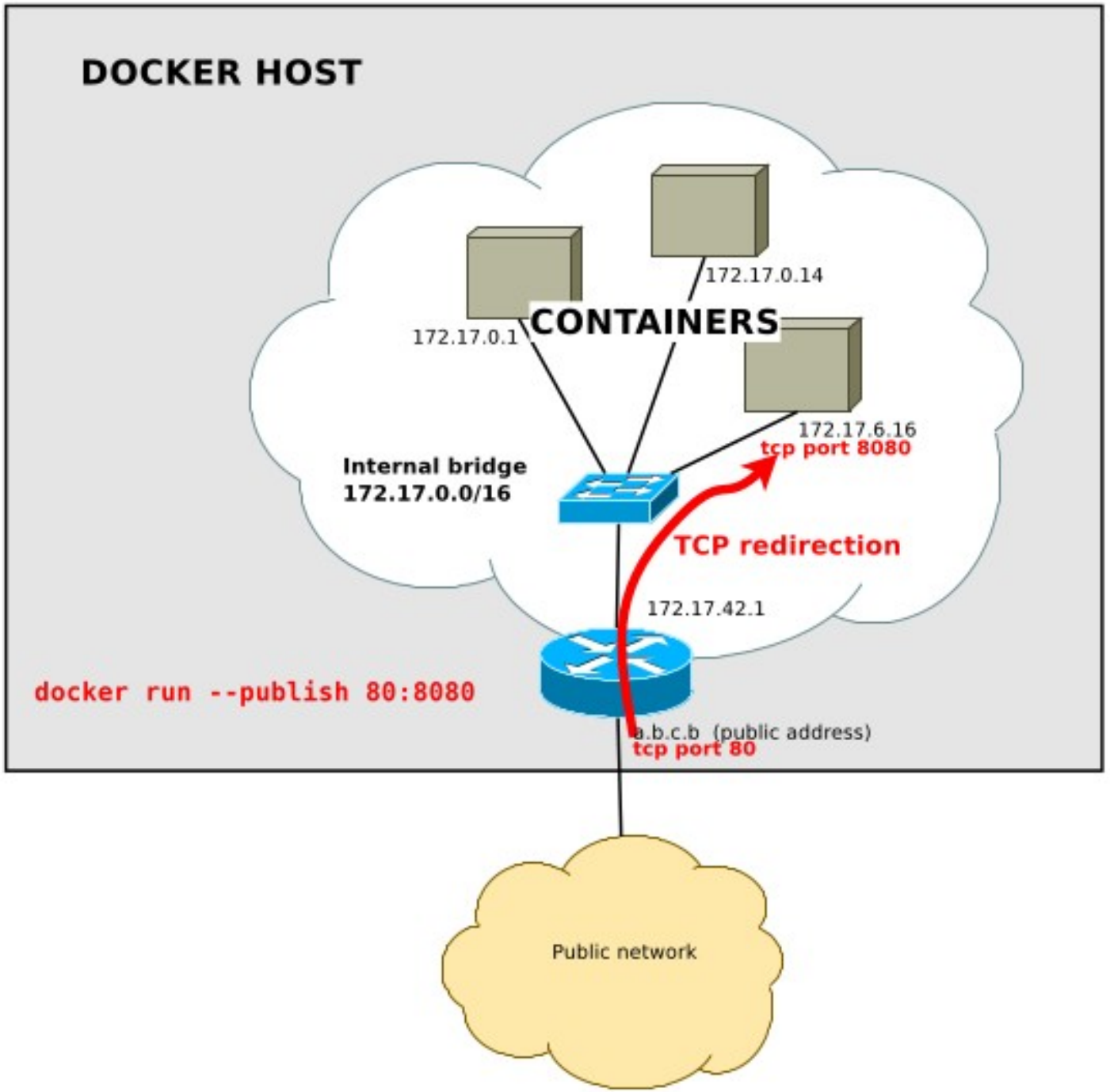
docker run - Publish a TCP port

- Containers are deployed in a private network, they are not
- reachable from the outside (unless a redirection is set up)

```
docker run -p [ipaddr:]hostport:containerport
docker run -t  -p 9000:9000 debian
```

- → redirect incoming connections to the TCP port hostport of the host to the TCP port containerport of the container
- The listening socket binds to 0.0.0.0 (all interfaces) by default or to ipaddr if given

publish example



Publish example

- **bind to all host addresses**

```
[root@docker-host ~]# docker run -d -p 80:80 nginx
52c9105e1520980d49ed00ecf5f0ca694d177d77ac9d003b9c0b840db9a70d62
[root@docker-host ~]# docker inspect 6174f6951f19 | grep IPAddress
[root@docker-host ~]# wget -nv http://localhost/
2016-01-12 18:32:52 URL:http://localhost/ [612/612] -> "index.html" [1]
[root@docker-host ~]# wget -nv http://172.17.0.2/
2016-01-12 18:33:14 URL:http://172.17.0.2/ [612/612] -> "index.html" [1]
```

- **bind to 127.0.0.1**

```
[root@docker-host ~]# docker run -d -p 127.0.0.1:80:80 nginx
4541b43313b51d50c4dc2722e741df6364c5ff50ab81b828456ca55c829e732c

[root@docker-host ~]# wget -nv http://localhost/
2016-01-12 18:37:10 URL:http://localhost/ [612/612] -> "index.html.1" [1]

[root@docker-host ~]# wget http://172.17.0.2/
--2016-01-12 18:38:32-- http://172.17.0.2/
Connecting to 172.17.42.1:80... failed: Connection refused.
```

Managing docker images

- Docker images
- Image management commands
- Example: images & containers

Docker images

A docker image is a snapshot of the filesystem + some metadata

- immutable
- copy-on-write storage
  - for instantiating containers
  - for creating new versions of the image (multiple layers)
- identified by a unique hex ID (hashed from the image content)
- may be tagged 8 with a human-friendly name

  eg: debian:wheezy debian:jessie debian:latest

Image management commands

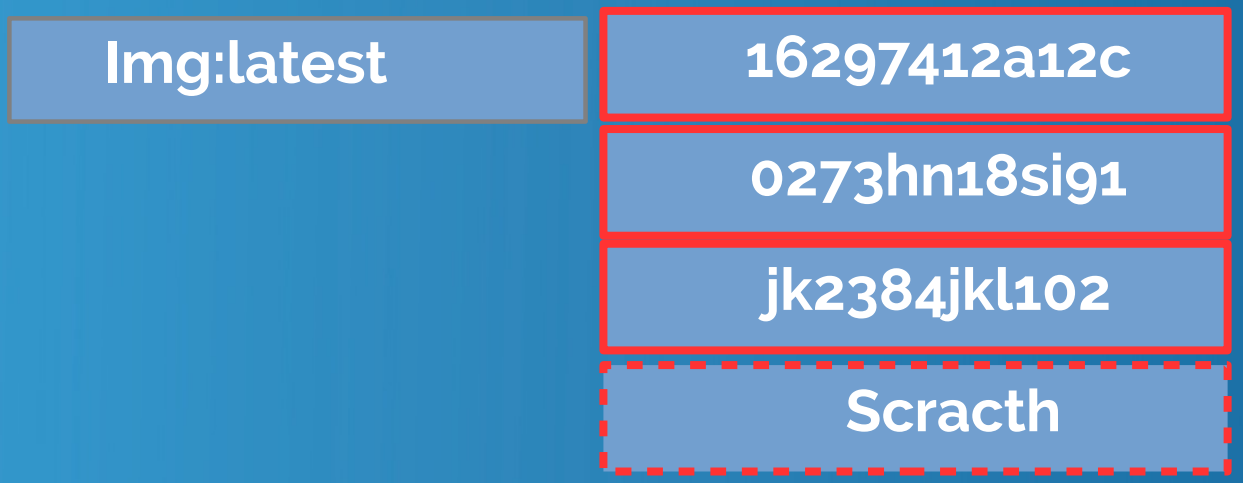
Command	Description
docker images	list all local images
docker history image	show the image history (list of ancestors)
docker inspect image. . .	show low-level infos (in json format)
docker tag image tag	tag an image
docker commit container image	create an image (from a container)
docker import url|- [tag]	create an image (from a tarball)
docker rmi image. . .	delete images

Example: images & containers

Scracth

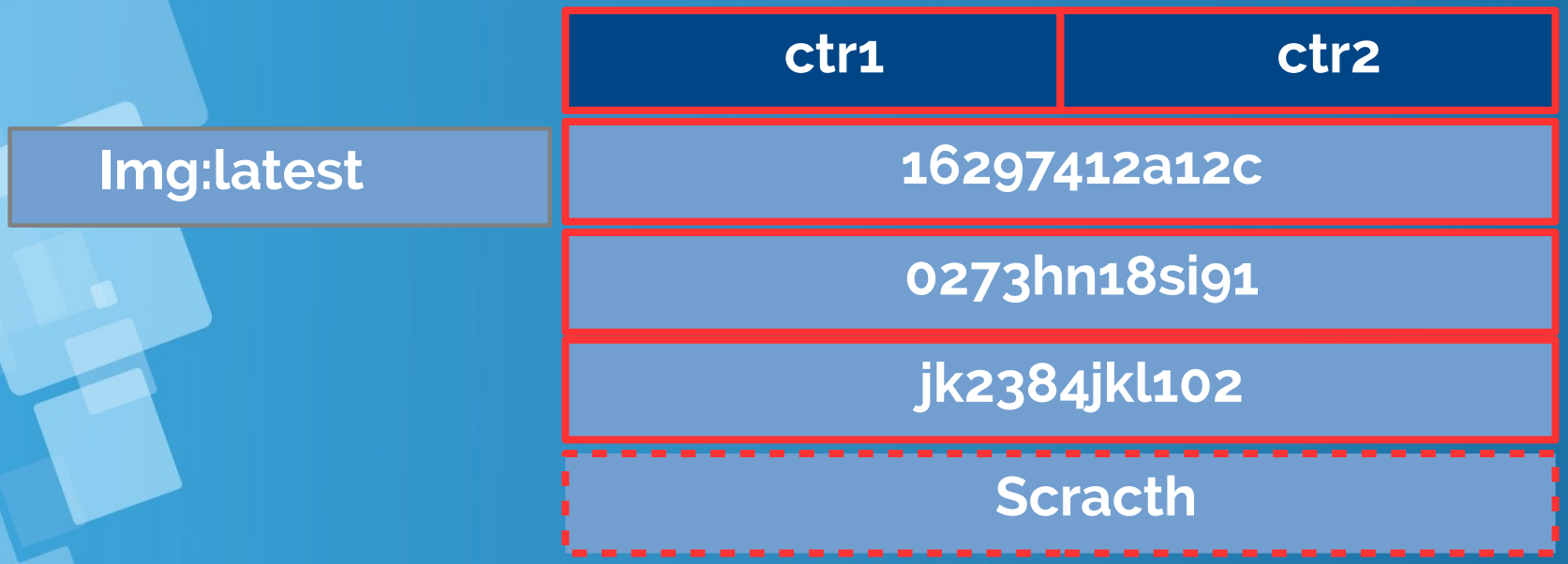
Example: images & containers

```
docker pull image
```



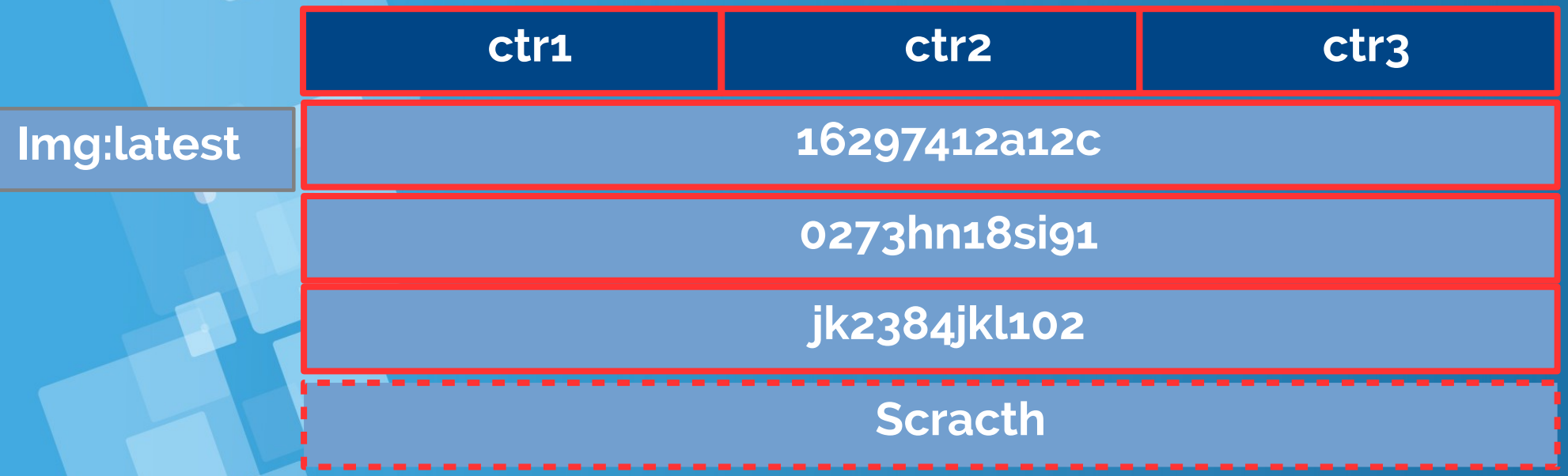
Example: images & containers

```
docker run –-name ctr2 img
```



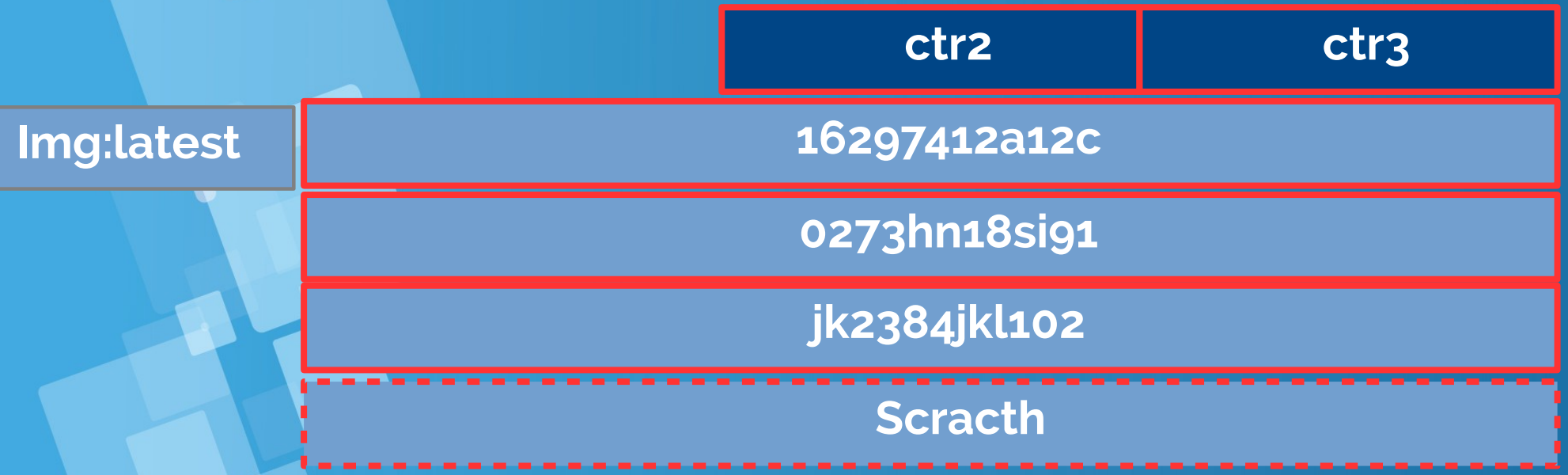
Example: images & containers

```
docker run –-name ctr3 img
```



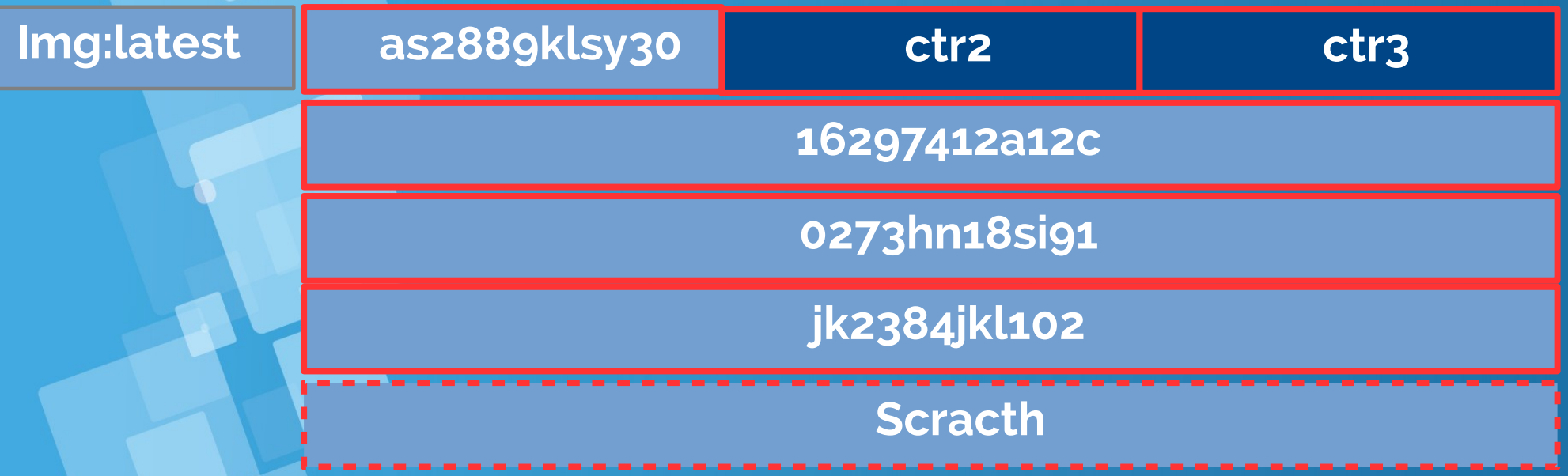
Example: images & containers

```
docker rm ctr1
```



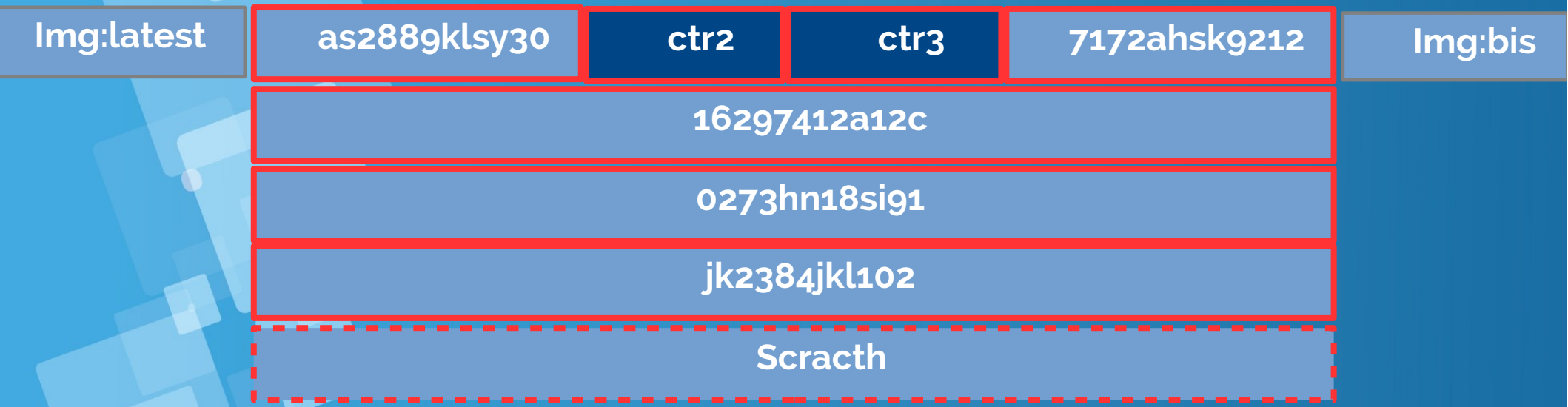
Example: images & containers

```
docker commit ctr2
```



Example: images & containers

```
docker commit ctr2 img:bis
```



Example: images & containers

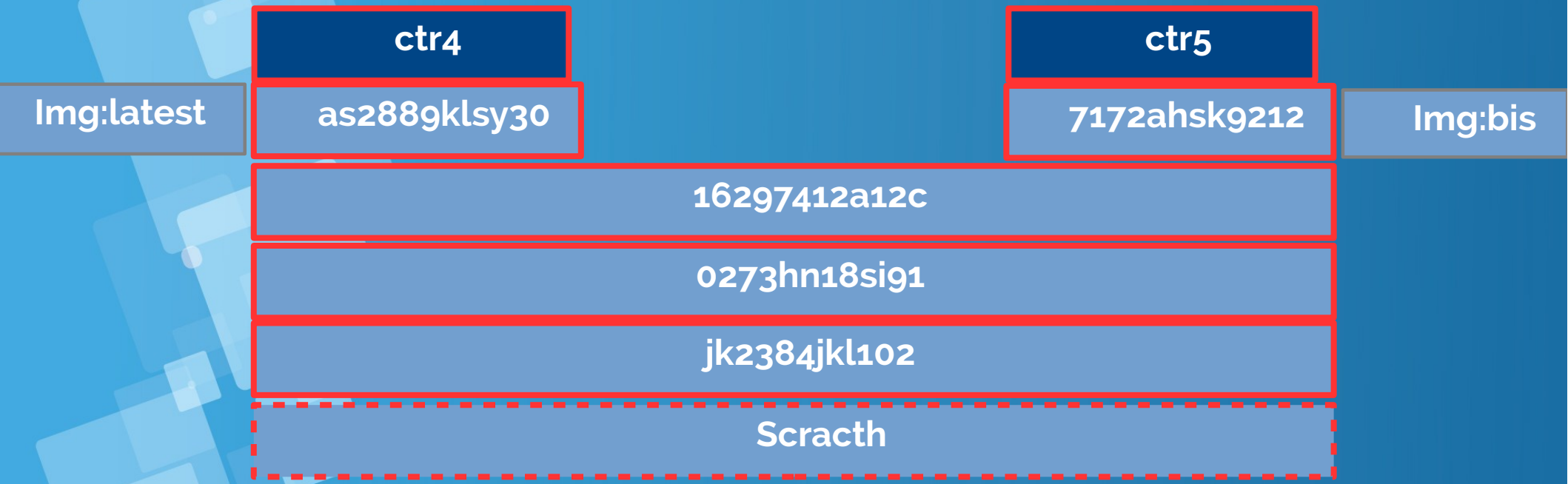
```
docker run ctr4 img
```

Example: images & containers

```
docker run –-name ctr5 img:bis
```

Example: images & containers

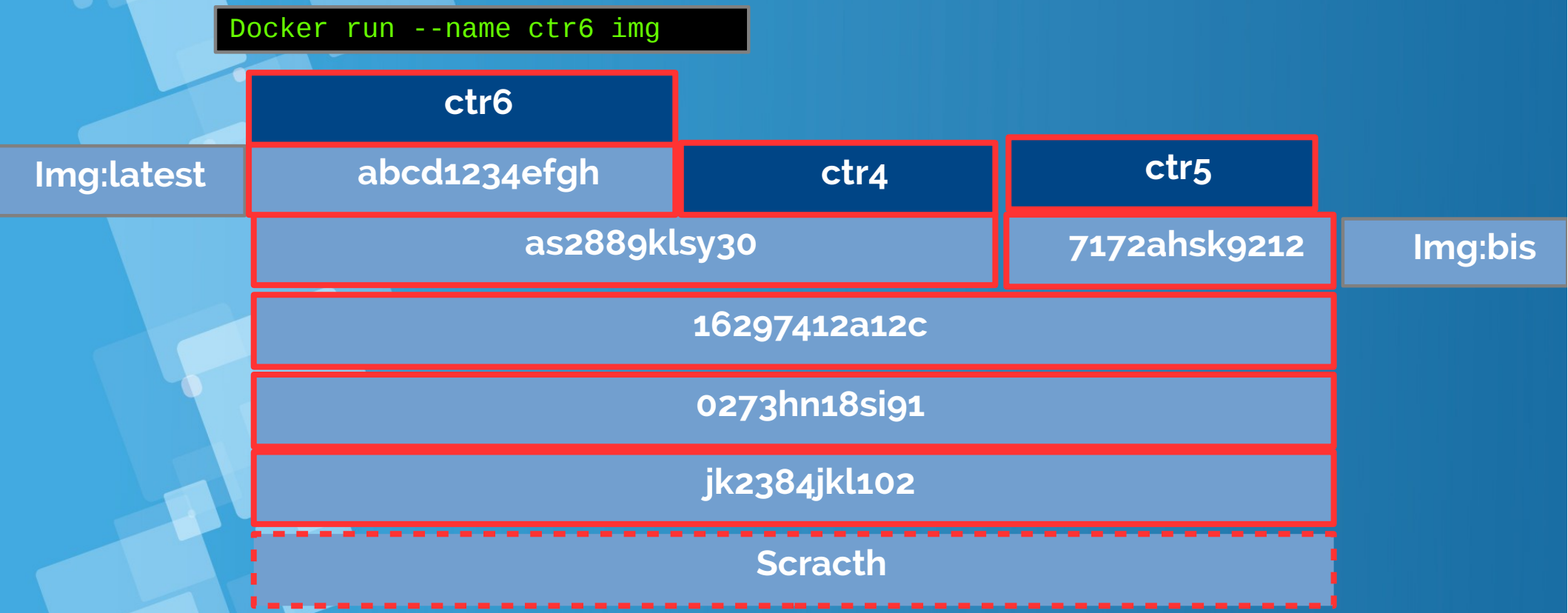
```
Docker rm ctr2 ctr3
```



Example: images & containers

```
Docker commit ctr4 img
```

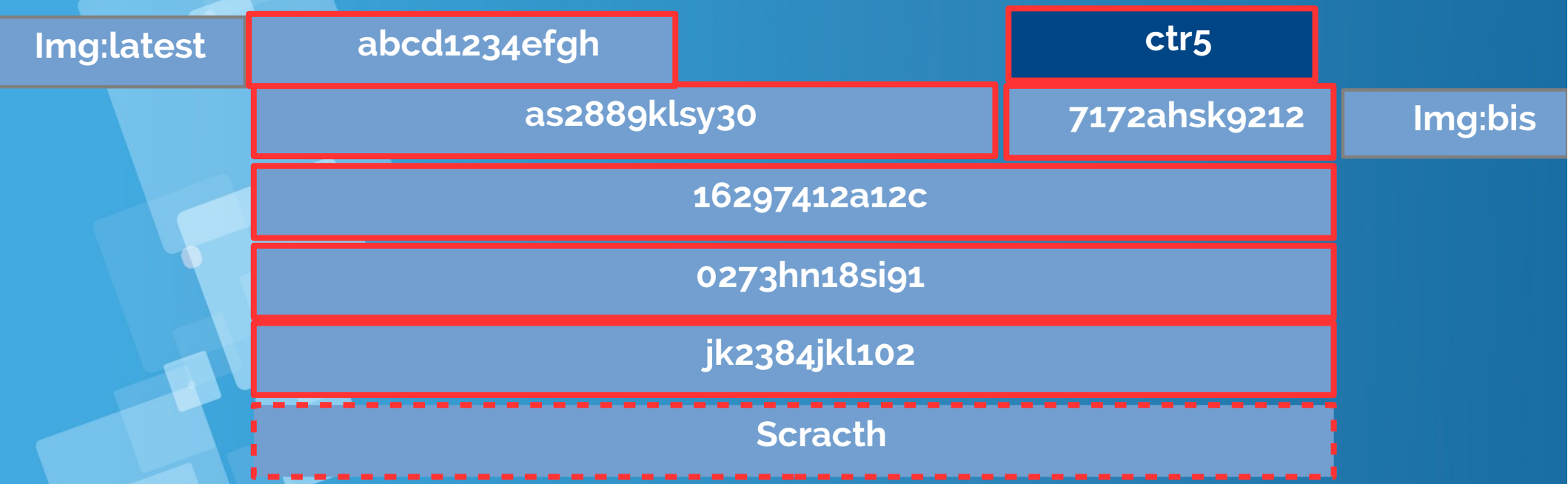
Example: images & containers



Example: images & containers

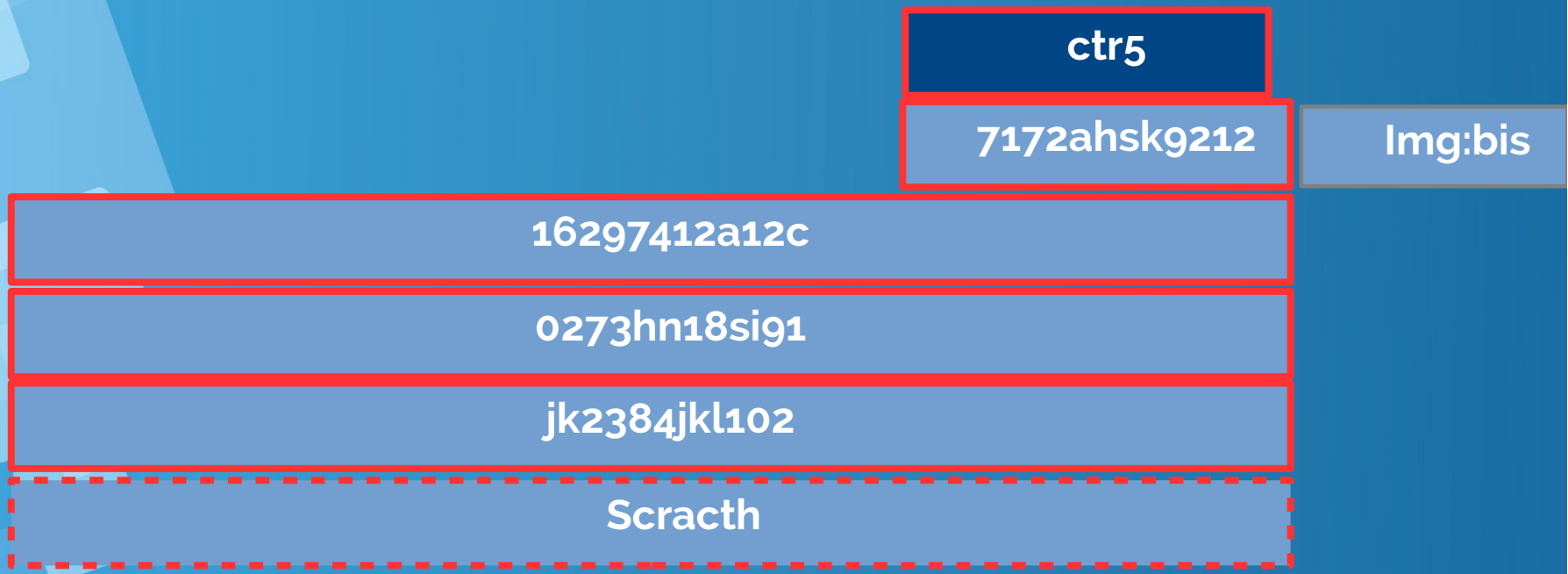
Example: images & containers

```
Docker rm ctr6
```



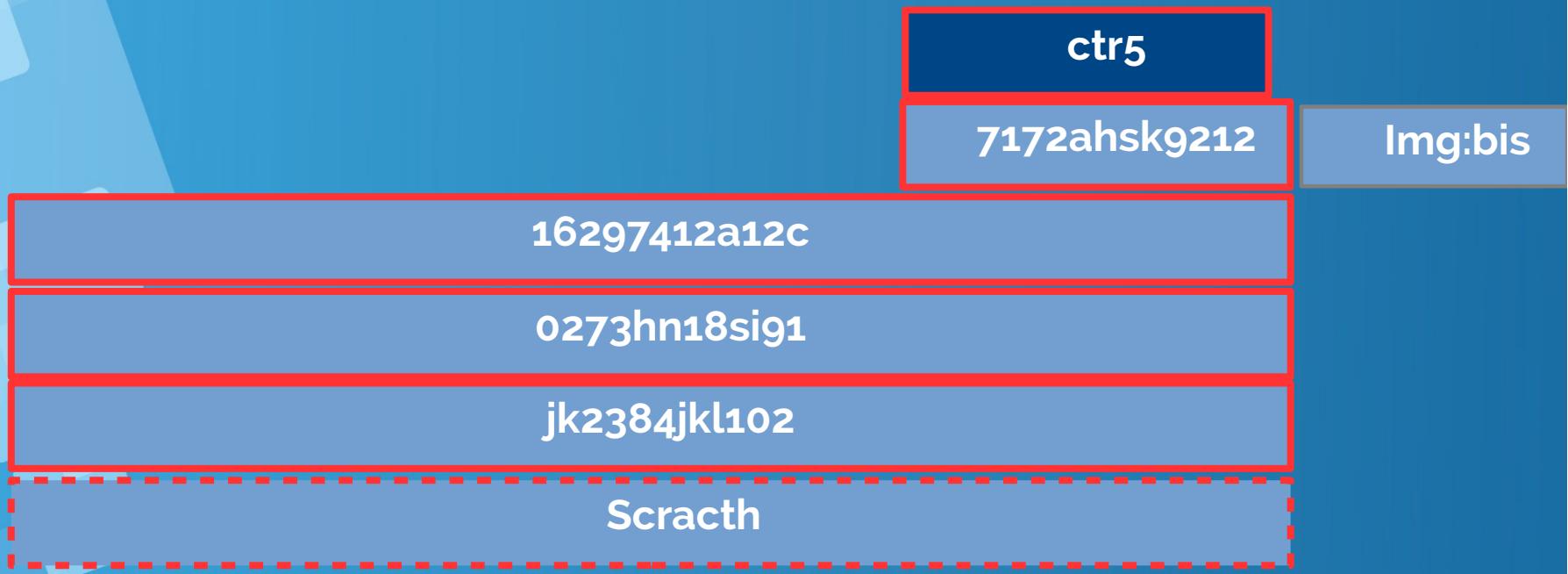
Example: images & containers

```
Docker rmi img
```



Example: images & containers

```
Docker rmi img:bis
Error: img:bis is reference by ctr5
```



Example: images & containers

```
Docker rmi -f img:bis
```

Example: images & containers

```
Docker rm ctr5
```

Images tags

- A docker tag is made of two parts: “REPOSITORY:TAG”
- The TAG part identifies the version of the image. If not provided, the default is “:latest”

```
[root@docker-host ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED                  SIZE
debian              latest              a25c1eed1c6f        Less than a second ago   123MB
hello               latest              52442a43a78b        13 minutes ago           123MB
centos              latest              3bee3060bfc8        46 hours ago             193MB
ubuntu              latest              7b9b13f7b9c0        5 days ago               118MB
nginx               latest              958a7ae9e569        7 days ago               109MB
```

Image transfer commands

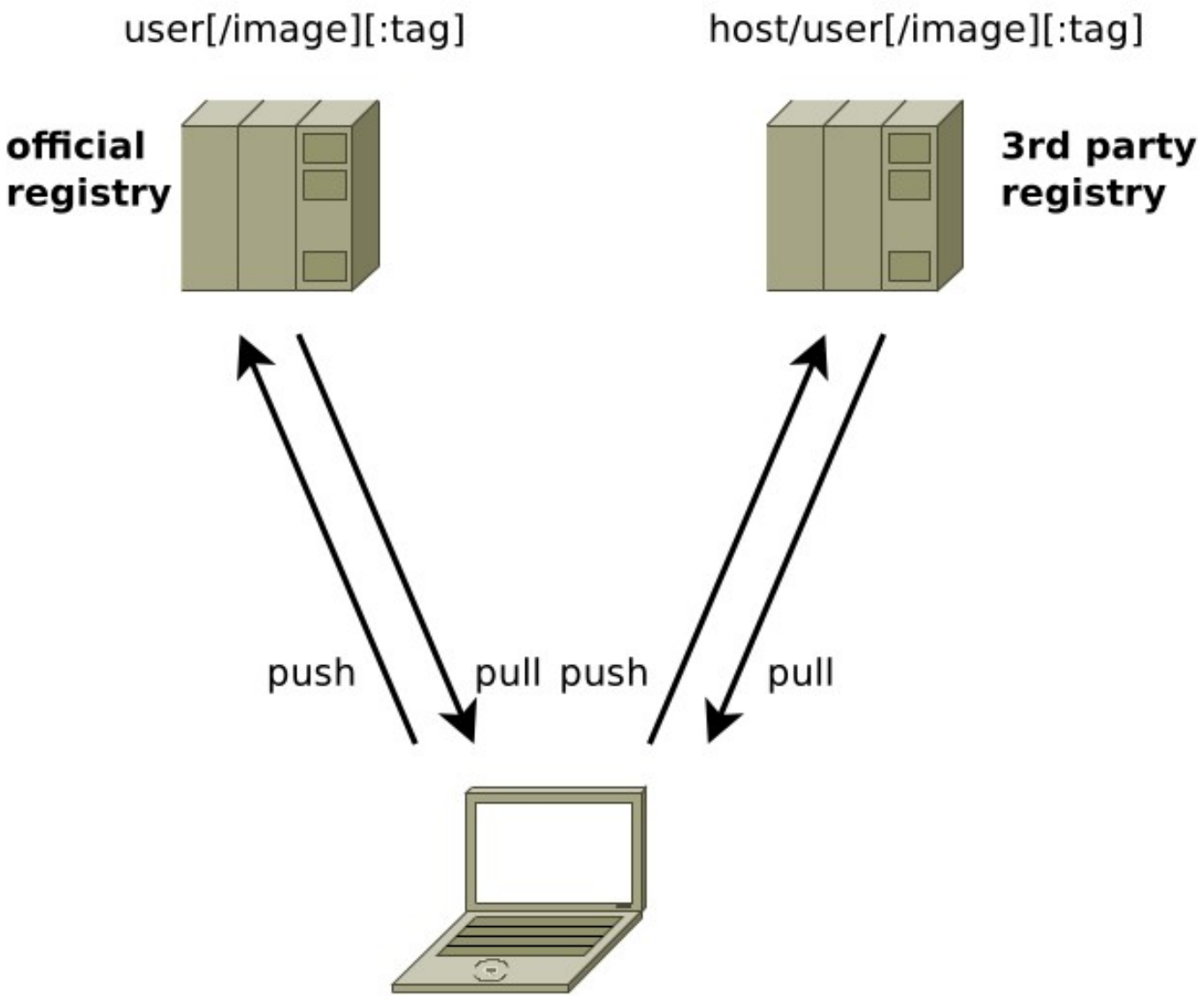
Using the registry API

docker pull repo[:tag]. . .	pull an image/repo from a registry
docker push repo[:tag]. . .	push an image/repo from a registry
docker search text	search an image on the official registry
docker login . . .	login to a registry
docker logout . . .	logout from a registry

Manual transfer

docker save repo[:tag]. . .	export an image/repo as a tarbal
docker load	load images from a tarball
docker-ssh . . .	proposed script to transfer images between two daemons over ssh

Transferring images



Lab: Image creation from a container

Let's start by running an interactive shell in a ubuntu container.

```
[root@docker-host]# curl -fsSL https://get.docker.com/ | sh
[root@docker-host]# systemctl start docker
[root@docker-host]# systemctl status docker
[root@docker-host]# systemctl enable docker

[root@docker-host ~]# docker run -ti ubuntu bash
```

Install the figlet package in this container

```
root@880998ce4c0f:/# apt-get update -y ; apt-get install figlet
root@880998ce4c0f:/# exit
```

Get the ID of this container using the ls command (do not forget the -a option as the non running container are not returned by the ls command).

```
[root@docker-host]# docker ps -a
```

Run the following command, using the ID retreived, in order to commit the container and create an image out of it.

```
[root@docker-host ~]# docker commit 880998ce4c0f
sha256:1a769da2b98b04876844f96594a92bd708ca27ee5a8868d43c0aeb5985671161
```

Lab: Image creation from a container

Once it has been commited, we can see the newly created image in the list of available images.

```
[root@docker-host ~]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              1a769da2b98b        59 seconds ago      158MB
ubuntu              latest              7b9b13f7b9c0        6 days ago          118MB
```

From the previous command, get the ID of the newly created image and tag it so it's named intra-tag.

```
[root@docker-host ~]# docker image tag 1a769da2b98b tag-intra
[root@docker-host ~]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
tag-intra           latest              1a769da2b98b        3 minutes ago       158MB
ubuntu              latest              7b9b13f7b9c0        6 days ago          118MB
```

As figlet is present in our tag-intra image, the command ran returns the following output.

```
[root@docker-host ~]# docker container run tag-intra figlet hello
 _              _ _
| |__   ___  | | | ___
| '_ \ / _ \ | | |/ _ \
| | | |  __/ | | | (_) |
|_| |_|\___| |_|_|\___/
```

Docker builder

- What is the Docker builder
- DockerFile
- Docker-Compose
- Introduction to Kompose

Docker images

- Docker's builder relies on
  - a DSL describing how to build an image
  - a cache for storing previous builds and have quick iterations

- The builder input is a context, i.e. a directory containing:
  - a file named Dockerfile which describe how to build the container
  - possibly other files to be used during the build

Dockerfile format

- comments start with “#”
- commands fit on a single line (possibly continuated with \)
- first command must be a FROM (indicates the parent image or scratch to start from scratch)

Build an image

- build an image from the context located at path and optionally tag it as tag

```
docker build [ -t tag ] path
```

- The command:
  1. makes a tarball from the content 10 of path
  2. uploads the tarball to the docker daemon which will:
     - 2.1 execute the content of Dockerfile, committing an intermediate image after each command
     - 2.2 (if requested) tag the final image as tag

Dockerfile Description

- Each Dockerfile is a script, composed of various commands (instructions) and arguments listed successively to automatically perform actions on a base image in order to create (or form) a new one. They are used for organizing things and greatly help with deployments by simplifying the process start-to-finish.

- Dockerfiles begin with defining an image FROM which the build process starts. Followed by various other methods, commands and arguments (or conditions), in return, provide a new image which is to be used for creating docker containers.

- They can be used by providing a Dockerfile's content - in various ways - to the docker daemon to build an image (as explained in the "How To Use" section).

Dockerfile Commands (Instructions)

- ADD
  - The ADD command gets two arguments: a source and a destination. It basically copies the files from the source on the host into the container's own filesystem at the set destination. If, however, the source is a URL (e.g. http://github.com/user/file/), then the contents of the URL are downloaded and placed at the destination.
  - Example

```
# Usage: ADD [source directory or URL] [destination directory]

ADD /my_app_folder /my_app_folder
```

Dockerfile Commands (Instructions)

- CMD
  - The command CMD, similarly to RUN, can be used for executing a specific command. However, unlike RUN it is not executed during build, but when a container is instantiated using the image being built. Therefore, it should be considered as an initial, default command that gets executed (i.e. run) with the creation of containers based on the image.
  - To clarify: an example for CMD would be running an application upon creation of a container which is already installed using RUN (e.g. RUN apt-get install …) inside the image. This default application execution command that is set with CMD becomes the default and replaces any command which is passed during the creation.
  - Example

```
# Usage 1: CMD application "argument", "argument", ..

CMD "echo" "Hello docker!"
```

Dockerfile Commands (Instructions)

- **ENTRYPOINT**
  - ENTRYPOINT argument sets the concrete default application that is used every time a container is created using the image. For example, if you have installed a specific application inside an image and you will use this image to only run that application, you can state it with ENTRYPOINT and whenever a container is created from that image, your application will be the target.
  - If you couple ENTRYPOINT with CMD, you can remove "application" from CMD and just leave "arguments" which will be passed to the ENTRYPOINT.
  - Example:

```
# Usage: ENTRYPOINT application "argument", "argument", ..
# Remember: arguments are optional. They can be provided by CMD
#           or during the creation of a container.
ENTRYPOINT echo

# Usage example with CMD:
# Arguments set with CMD can be overridden during *run*
CMD "Hello docker!"
ENTRYPOINT echo
```

Dockerfile Commands (Instructions)

- ENV
  - The ENV command is used to set the environment variables (one or more). These variables consist of "key = value" pairs which can be accessed within the container by scripts and applications alike. This functionality of docker offers an enormous amount of flexibility for running programs.
  - Example:

```
# Usage: ENV key value
ENV SERVER_WORKS 4
```

- EXPOSE
  - The EXPOSE command is used to associate a specified port to enable networking between the running process inside the container and the outside world (i.e. the host).
  - Example:

```
# Usage: EXPOSE [port]
EXPOSE 8080
```

Dockerfile Commands (Instructions)

- FROM
  - FROM directive is probably the most crucial amongst all others for Dockerfiles. It defines the base image to use to start the build process. It can be any image, including the ones you have created previously. If a FROM image is not found on the host, docker will try to find it (and download) from the docker image index. It needs to be the first command declared inside a Dockerfile.
  - Example:

```
# Usage: FROM [image name]
FROM ubuntu
```

Dockerfile Commands (Instructions)

- MAINTAINER
  - One of the commands that can be set anywhere in the file - although it would be better if it was declared on top - is MAINTAINER. This non-executing command declares the author, hence setting the author field of the images. It should come nonetheless after FROM.
  - Example:

```
# Usage: MAINTAINER [name]
MAINTAINER authors_name
```

- RUN
  - The RUN command is the central executing directive for Dockerfiles. It takes a command as its argument and runs it to form the image. Unlike CMD, it actually is used to build the image (forming another layer on top of the previous one which is committed).
  - Example

```
# Usage: RUN [command]
RUN aptitude install -y riak
```

Dockerfile Commands (Instructions)

- **USER**
  - The USER directive is used to set the UID (or username) which is to run the container based on the image being built.
  - Example:

```
# Usage: USER [UID]
USER 751
```

- **VOLUME**
  - The VOLUME command is used to enable access from your container to a directory on the host machine (i.e. mounting it).
  - Example:

```
# Usage: VOLUME ["/dir_1", "/dir_2" ..]
VOLUME ["/my_files"]
```

- **WORKDIR**
  - The WORKDIR directive is used to set where the command defined with CMD is to be executed.
  - Example:

```
# Usage: WORKDIR /path
WORKDIR ~/
```

Summary Builder main commands

Command	Description
FROM image|scratch	base image for the build
MAINTAINER email	name of the mainainer (metadata)
COPY path dst	copy path from the context into the container at location dst
ADD src dst	same as COPY but untar archives and accepts http urls
RUN args. . .	run an arbitrary command inside the container
USER name	set the default username
WORKDIR path	set the default working directory
CMD args. . .	set the default command
ENV name value	set an environment variable

Dockerfile example

- How to Use Dockerfiles
  - Using the Dockerfiles is as simple as having the docker daemon run one. The output after executing the script will be the ID of the new docker image.
  - Usage:

```
# Build an image using the Dockerfile at current location
# Example: sudo docker build -t [name] .
[root@docker-host ~]# docker build -t nginx_yusuf .
```

Lab: Dockerfile example

- **build an image from the context located at path and optionally tag it as tag**

```
##################################################################
# Dockerfile to build nginx container images
# Based on debian latest version
##################################################################

# base image: last debian release
FROM debian:latest

# name of the maintainer of this image
MAINTAINER yusuf.hadiwinata@gmail.com

# install the latest upgrades
RUN apt-get update && apt-get -y dist-upgrade && echo yusuf-test > /tmp/test

# install nginx
RUN apt-get -y install nginx

# set the default container command
# -> run nginx in the foreground
CMD ["nginx", "-g", "daemon off;"]

# Tell the docker engine that there will be somenthing listening on the tcp port 80
EXPOSE 80
```

Lab: Build & Run Dockerfile example

```
[root@docker-host nginx_yusuf]# docker build -t nginx_yusuf .
Sending build context to Docker daemon   2.56kB
Step 1/6 : FROM debian:latest
 ---> a25c1eed1c6f
Step 2/6 : MAINTAINER yusuf.hadiwinata@gmail.com
 ---> Running in 94409ebe59ac
 ---> eaefc54975b7
Removing intermediate container 94409ebe59ac
Step 3/6 : RUN apt-get update && apt-get -y dist-upgrade
 ---> Running in 425285dbf037
Get:1 http://security.debian.org jessie/updates InRelease [63.1 kB]
Ign http://deb.debian.org jessie InRelease
Get:2 http://deb.debian.org jessie-updates InRelease [145 kB]
Get:3 http://deb.debian.org jessie Release.gpg [2373 B]
Get:4 http://deb.debian.org jessie-updates/main amd64 Packages [17.6 kB]
Get:5 http://security.debian.org jessie/updates/main amd64 Packages [521 kB]
Get:6 http://deb.debian.org jessie Release [148 kB]
Get:7 http://deb.debian.org jessie/main amd64 Packages [9065 kB]

------------------- DIPOTONG -------------------------

Processing triggers for sgml-base (1.26+nmu4) ...
 ---> 88795938427f
Removing intermediate container 431ae6bc8e0a
Step 5/6 : CMD nginx -g daemon off;
 ---> Running in 374ff461f187
 ---> 08e1433ccd68
Removing intermediate container 374ff461f187
Step 6/6 : EXPOSE 80
 ---> Running in bac435c454a8
 ---> fa8de9e81136
Removing intermediate container bac435c454a8
Successfully built fa8de9e81136
Successfully tagged nginx_yusuf:latest
```

Lab: Dockerfile example

- Using the image we have build, we can now proceed to the final step: creating a container running a nginx instance inside, using a name of our choice (if desired with -name [name]).
- Note: If a name is not set, we will need to deal with complex, alphanumeric IDs which can be obtained by listing all the containers using sudo docker ps -l.

```
[root@docker-host nginx_yusuf]# docker run --name my_first_nginx_instance -i -t
nginx_yusuf bash
root@4b90e5d6dda8:/#
root@4b90e5d6dda8:/#
root@4b90e5d6dda8:/# cat /tmp/test
yusuf-test
```

Docker Compose

Manage a collection of containers

group.yml

```
name: counter

containers:
  web:
    build: .
    command: python app.py
    ports:
      - "5000:5000"
    volumes:
      - .:/code
    links:
      - redis
  redis:
    image: redis:latest
```

Dockerfile vs Docker Compose which is better?

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration. To learn more about all the features of Compose see the list of features.

Compose is great for development, testing, and staging environments, as well as CI workflows. You can learn more about each case in Common Use Cases.

- Using Compose is basically a three-step process.
  - Define your app's environment with a Dockerfile so it can be reproduced anywhere.
  - Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
  - Lastly, run docker-compose up and Compose will start and run your entire app.

Lab: Docker Compose Ubuntu, php7-fpm, Nginx and MariaDB Example

This lab will setup PHP application using Docker and docker-compose. This will setup a developement environment with PHP7-fpm, MariaDB and Nginx.

- Clone or download the sample project from Github.

```
[root@docker-host ~]# git clone https://github.com/isnuryusuf/docker-php7.git
Cloning into 'docker-php7'...
remote: Counting objects: 62, done.
remote: Total 62 (delta 0), reused 0 (delta 0), pack-reused 62
Unpacking objects: 100% (62/62), done.
      960 B

[root@docker-host ~]# cd docker-php7
[root@docker-host ~]# yum -y install epel-release
[root@docker-host ~]# yum install -y python-pip
[root@docker-host ~]# pip install docker-compose
[root@docker-host docker-php7]# docker-compose up
```

Docker Compose Ubuntu, php7-fpm, Nginx and MariaDB Example

- **Inside docker-php7 have a directory structure like this.**

```
├── app
│   └── public
│       └── index.php
├── database
├── docker-compose.yml
├── fpm
│   ├── Dockerfile
│   └── supervisord.conf
├── nginx
│   ├── Dockerfile
│   └── default.conf
```

- **app - Our application will be kept in this directory.**
- **database is where MariaDB will store all the database files.**
- **fpm folder contains the Dockerfile for php7-fpm container and the Supervisord config**
- **nginx folder contains the Dockerfile for nginx and the default nginx config which will be copied to the container.**
- **docker-compose.yml - Our docker-compose configuration. In this file, we define the containers and services that we want to start, along with associated volumes, ports, etc. When we run docker-compose up, it reads this file and builds the images.**

Docker Compose Ubuntu, php7-fpm, Nginx and MariaDB Example

- **docker-compose.yml**

```
version: "2"
services:
  nginx:
      build:
          context: ./nginx
      ports:
          - "8080:80"
      volumes:
          - ./app:/var/app
  fpm:
      build:
          context: ./fpm
      volumes:
          - ./app:/var/app
      expose:
          - "9000"
      environment:
          - "DB_HOST=db"
          - "DB_DATABASE=laravel"
  db:
      image: docker-registry:5000/mariadb:latest
      environment:
          - MYSQL_ROOT_PASSWORD=root
          - MYSQL_DATABASE=laravel
      volumes:
          - ./database:/var/lib/mysql
```

Docker Compose Ubuntu, php7-fpm, Nginx and MariaDB Example

- **Docker compose up output command**

```
[root@docker-host docker-php7]# docker-compose up
Creating network "dockerphp7_default" with the default driver
Building fpm
Step 1 : FROM ubuntu:latest
Trying to pull repository docker.io/library/ubuntu ...
sha256:ea1d854d38be82f54d39efe2c67000bed1b03348bcc2f3dc094f260855dff368: Pulling from
docker.io/library/ubuntu
bd97b43c27e3: Pull complete
6960dc1aba18: Pull complete
2b61829b0db5: Pull complete
1f88dc826b14: Pull complete
73b3859b1e43: Pull complete
Digest: sha256:ea1d854d38be82f54d39efe2c67000bed1b03348bcc2f3dc094f260855dff368
Status: Downloaded newer image for docker.io/ubuntu:latest
 ---> 7b9b13f7b9c0
Step 2 : RUN apt-get update && apt-get install -y software-properties-common language-
pack-en-base     && LC_ALL=en_US.UTF-8 add-apt-repository -y ppa:ondrej/php     && apt-
get update     && apt-get install -y php7.0 php7.0-fpm php7.0-mysql mcrypt php7.0-gd
curl        php7.0-curl php-redis php7.0-mbstring sendmail supervisor     && mkdir
/run/php     && apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
 ---> Running in 812423cbbeac
```

Docker Compose Ubuntu, php7-fpm, Nginx and MariaDB Example

```
db_1    | Warning: Unable to load '/usr/share/zoneinfo/leap-seconds.list' as time zone. Skipping it.
db_1    | 2017-06-09  4:49:13 140253349907200 [Warning] 'proxies_priv' entry '@% root@f972a4b24306' ignored in --skip-name-resolve mode.
db_1    |
db_1    | 2017-06-09  4:49:13 140253349604096 [Note] mysqld: Normal shutdown
db_1    |
db_1    | 2017-06-09  4:49:13 140253349604096 [Note] Event Scheduler: Purging the queue. 0 events
db_1    | 2017-06-09  4:49:13 140252534535936 [Note] InnoDB: FTS optimize thread exiting.
db_1    | 2017-06-09  4:49:13 140253349604096 [Note] InnoDB: Starting shutdown...
db_1    | 2017-06-09  4:49:14 140253349604096 [Note] InnoDB: Waiting for page_cleaner to finish flushing of buffer pool
db_1    | 2017-06-09  4:49:15 140253349604096 [Note] InnoDB: Shutdown completed; log sequence number 1616829
db_1    | 2017-06-09  4:49:15 140253349604096 [Note] mysqld: Shutdown complete
db_1    |
db_1    |
db_1    | MySQL init process done. Ready for start up.
db_1    |
db_1    | 2017-06-09  4:49:16 140568343840704 [Note] mysqld (mysqld 10.1.24-MariaDB-1~jessie) starting as process 1 ...
db_1    | 2017-06-09  4:49:16 140568343840704 [Note] InnoDB: Using mutexes to ref count buffer pool pages
db_1    | 2017-06-09  4:49:16 140568343840704 [Note] InnoDB: The InnoDB memory heap is disabled
db_1    | 2017-06-09  4:49:16 140568343840704 [Note] InnoDB: Mutexes and rw_locks use GCC atomic builtins
db_1    | 2017-06-09  4:49:16 140568343840704 [Note] InnoDB: GCC builtin __atomic_thread_fence() is used for memory barrier
db_1    | 2017-06-09  4:49:16 140568343840704 [Note] InnoDB: Compressed tables use zlib 1.2.8
db_1    | 2017-06-09  4:49:16 140568343840704 [Note] InnoDB: Using Linux native AIO
db_1    | 2017-06-09  4:49:16 140568343840704 [Note] InnoDB: Using SSE crc32 instructions
db_1    | 2017-06-09  4:49:16 140568343840704 [Note] InnoDB: Initializing buffer pool, size = 256.0M
db_1    | 2017-06-09  4:49:16 140568343840704 [Note] InnoDB: Completed initialization of buffer pool
db_1    | 2017-06-09  4:49:16 140568343840704 [Note] InnoDB: Highest supported file format is Barracuda.
db_1    | 2017-06-09  4:49:17 140568343840704 [Note] InnoDB: 128 rollback segment(s) are active.
db_1    | 2017-06-09  4:49:17 140568343840704 [Note] InnoDB: Waiting for purge to start
db_1    | 2017-06-09  4:49:17 140568343840704 [Note] InnoDB:  Percona XtraDB (http://www.percona.com) 5.6.36-82.0 started; log sequence number 161682
9
db_1    | 2017-06-09  4:49:17 140567547737856 [Note] InnoDB: Dumping buffer pool(s) not yet started
db_1    | 2017-06-09  4:49:17 140568343840704 [Note] Plugin 'FEEDBACK' is disabled.
db_1    | 2017-06-09  4:49:17 140568343840704 [Note] Server socket created on IP: '::'.
db_1    | 2017-06-09  4:49:17 140568343840704 [Warning] 'proxies_priv' entry '@% root@f972a4b24306' ignored in --skip-name-resolve mode.
db_1    | 2017-06-09  4:49:17 140568343840704 [Note] mysqld: ready for connections.
db_1    | Version: '10.1.24-MariaDB-1~jessie'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306  mariadb.org binary distribution
```

Introduction to Kompose

Why developers like Docker Compose?

- Simple (to learn and adopt)
  - Very easy to run containerised applications
  - One line command
- Local development
- Declarative
- Great UX
- Developer friendly

Introduction to Kompose

Devs say this about Kubernetes/OpenShift

- Many new concepts to learn
  - Pods, Deployment, RC, RS, Job, DaemonSets, Routes/Ingress, Volumes ... phew!!!
  - Complex / complicated
  - Difficult
- Difficult/Complicated UX (especially when getting started)
- Setting up local development requires work
- Not developer friendly

Introduction to Kompose

How do we bridge this gap?

- We saw an opportunity here!
- Can we reduce the learning curve?
- Can we make adopting Kubernetes/OpenShift simpler?
  - What can we do make it simple?
  - Which bits need to be made simple?
  - How can we make it simple?

Introduction to Kompose

Enter Kompose!
(Kompose with a "K")

Docker Compose to OpenShift in *one* command
More info: http://kompose.io/

Docker Management

- Portainer.io

The Easiest Way to Manage Docker

Portainer is OpenSource lightweigh Management UI which allows you to easly manage your docker Host or Swarm Cluster

Available on Linux, Windows & OSX

Installing Portainer.io

- Portainer runs as a lightweight Docker container (the Docker image weights less than 4MB) on a Docker engine or Swarm cluster. Therefore, you are one command away from running container on any machine using Docker.
- Use the following Docker command to run Portainer:

```
[root@docker-host ~]# docker volume create portainer_data
[root@docker-host ~]# docker run -d -p 9000:9000 -v
/var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data docker-
registry:5000/portainer/portainer:latest
```

- you can now access Portainer by pointing your web browser at http://DOCKER_HOST:9000

  Ensure you replace DOCKER_HOST with address of your Docker host where Portainer is running.

Manage a new endpoint

- After your first authentication, Portainer will ask you information about the Docker endpoint you want to manage.

  You'll have the following choices:

- Not available for Windows Containers (Windows Server 2016) - Manage the local engine where Portainer is running (you'll need to bind mount the Docker socket via -v /var/run/docker.sock:/var/run/docker.sock on the Docker CLI when running Portainer)
- Manage a remote Docker engine, you'll just have to specify the url to your Docker endpoint, give it a name and TLS info if needed

Declare initial endpoint via CLI

- You can specify the initial endpoint you want Portainer to manage via the CLI, use the -H flag and the tcp:// protocol to connect to a remote Docker endpoint:

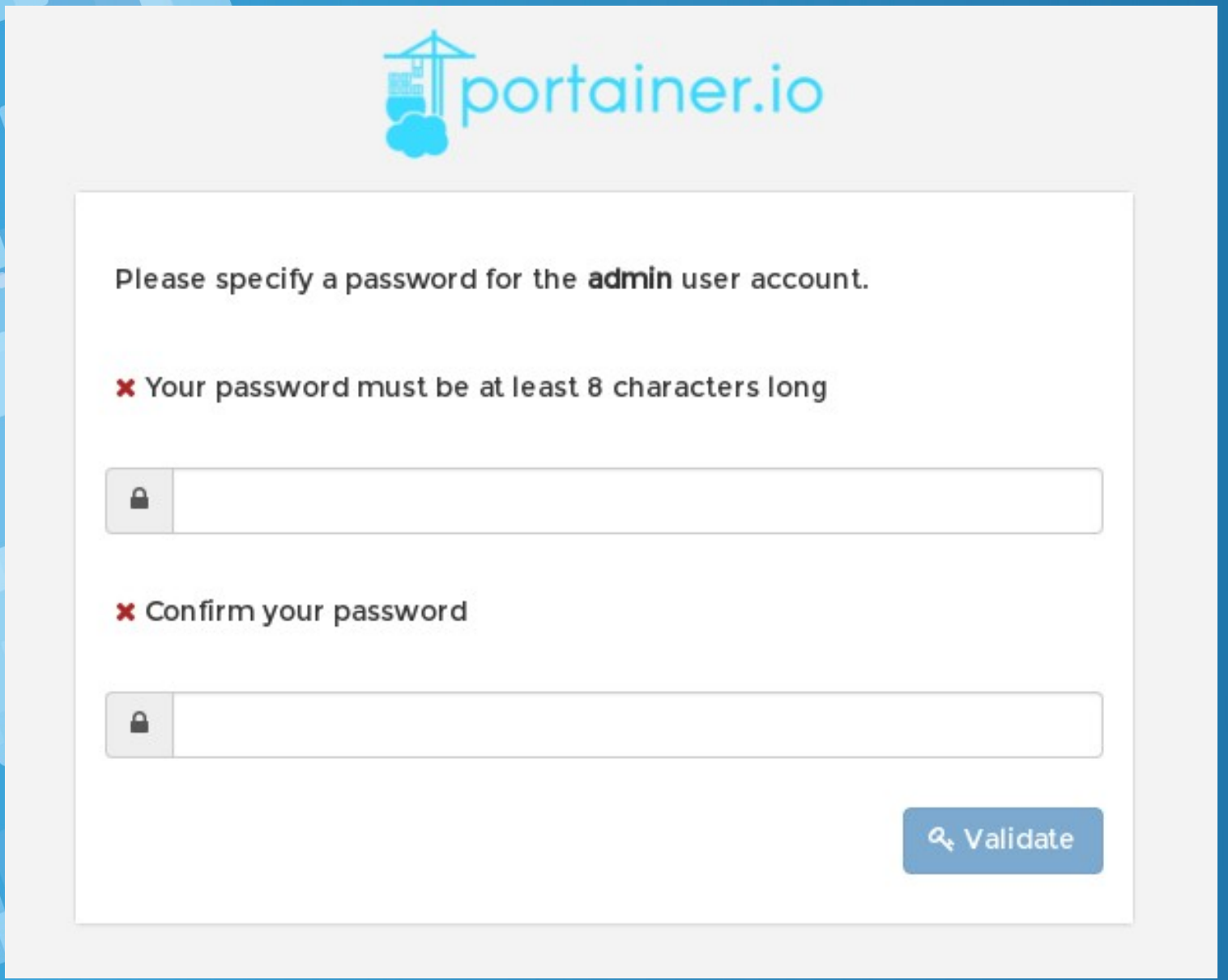
```
[root@docker-host ~]# docker run -d -p 9000:9000 portainer/portainer -H
tcp://<REMOTE_HOST>:<REMOTE_PORT>
```

- Ensure you replace REMOTE_HOST and REMOTE_PORT with the address/port of the Docker engine you want to manage.
- You can also bind mount the Docker socket to manage a local Docker engine (not available for Windows Containers (Windows Server 2016)):

```
[root@docker-host ~]# docker run -d -p 9000:9000 -v
/var/run/docker.sock:/var/run/docker.sock portainer/portainer
```

- Note: If your host is using SELinux, you'll need to pass the --privileged flag to the Docker run command:

Portainer Web Console - Initialization



Portainer 1st Access

Connect Portainer to a Docker engine or Swarm cluster endpoint

- (•) Manage the Docker instance where Portainer is running
- ( ) Manage a remote Docker instance

⚠ This feature is not yet available for native Docker Windows containers.
On Linux and when using Docker for Mac or Docker for Windows or Docker Toolbox, ensure that you have started Portainer container with the following Docker flag `-v "/var/run/docker.sock:/var/run/docker.sock"`

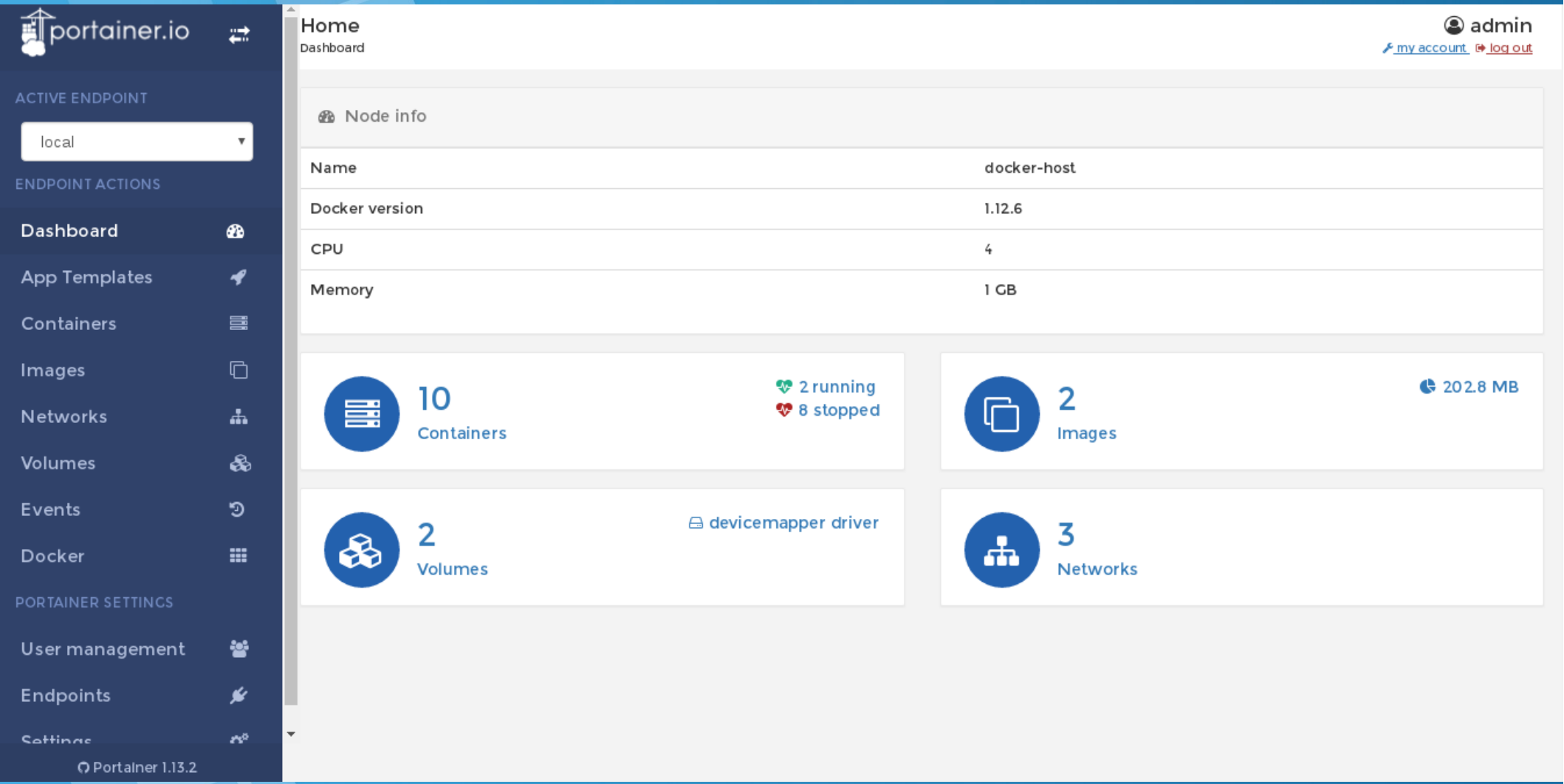
Connect

Portainer.io Manage Locally

- **Ensure that you have started Portainer container with the following Docker flag -v "/var/run/docker.sock:/var/run/docker.sock"**

```
[root@docker-host ~]# docker ps
CONTAINER ID        IMAGE                  COMMAND             CREATED             STATUS
            PORTS                    NAMES
afd62e28aee5        portainer/portainer    "/portainer"        5 minutes ago       Up 5
minutes        0.0.0.0:9000->9000/tcp   adoring_brown
60fec4b9a9bf        centos                 "bash"              21 minutes ago      Up 21
minutes                                 high_kare
[root@docker-host ~]# docker stop afd62e28aee5
afd62e28aee5
[root@docker-host ~]# docker run -v "/var/run/docker.sock:/var/run/docker.sock" -d -p
9000:9000 portainer/portainer
db232db974fa6c5a232f0c2ddfc0404dfac6bd34c087934c4c51b0208ececf0f
```

Portainer.io Dashboard



Portainer Documentation URL

https://portainer.readthedocs.io/

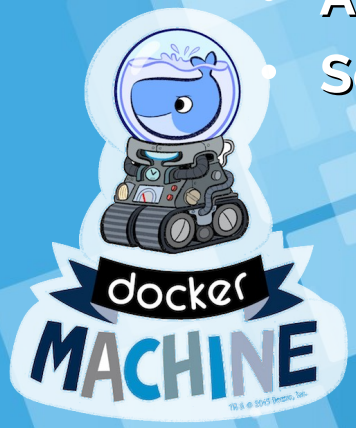
Docker Orchestration

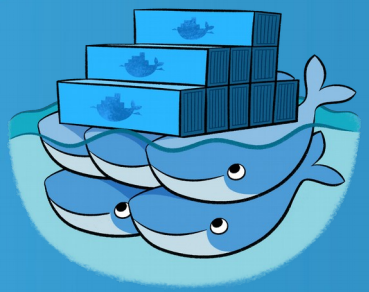
- Docker Machine
- Docker Swarm
- Docker Compose
- Kubernetes & Openshift

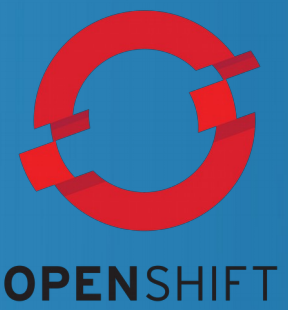
Docker / Container Problems

We need more than just packing and isolation

- Scheduling : Where should my containers run?
- Lifecycle and health : Keep my containers running despite failures
- Discovery : Where are my containers now?
- Monitoring : What's happening with my containers?
- Auth{n,z} : Control who can do things to my containers
- Aggregates : Compose sets of containers into jobs
- Scaling : Making jobs bigger or smaller

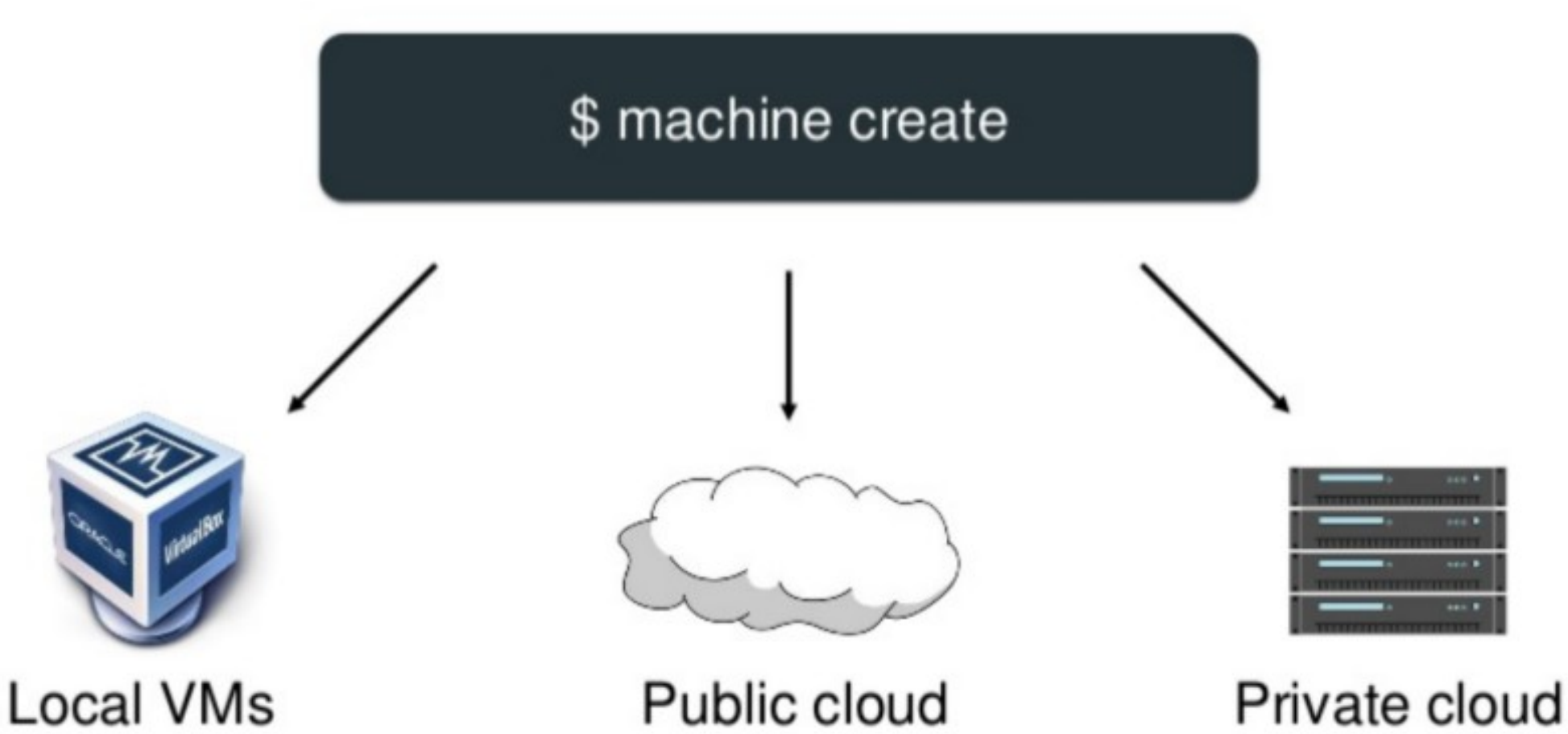






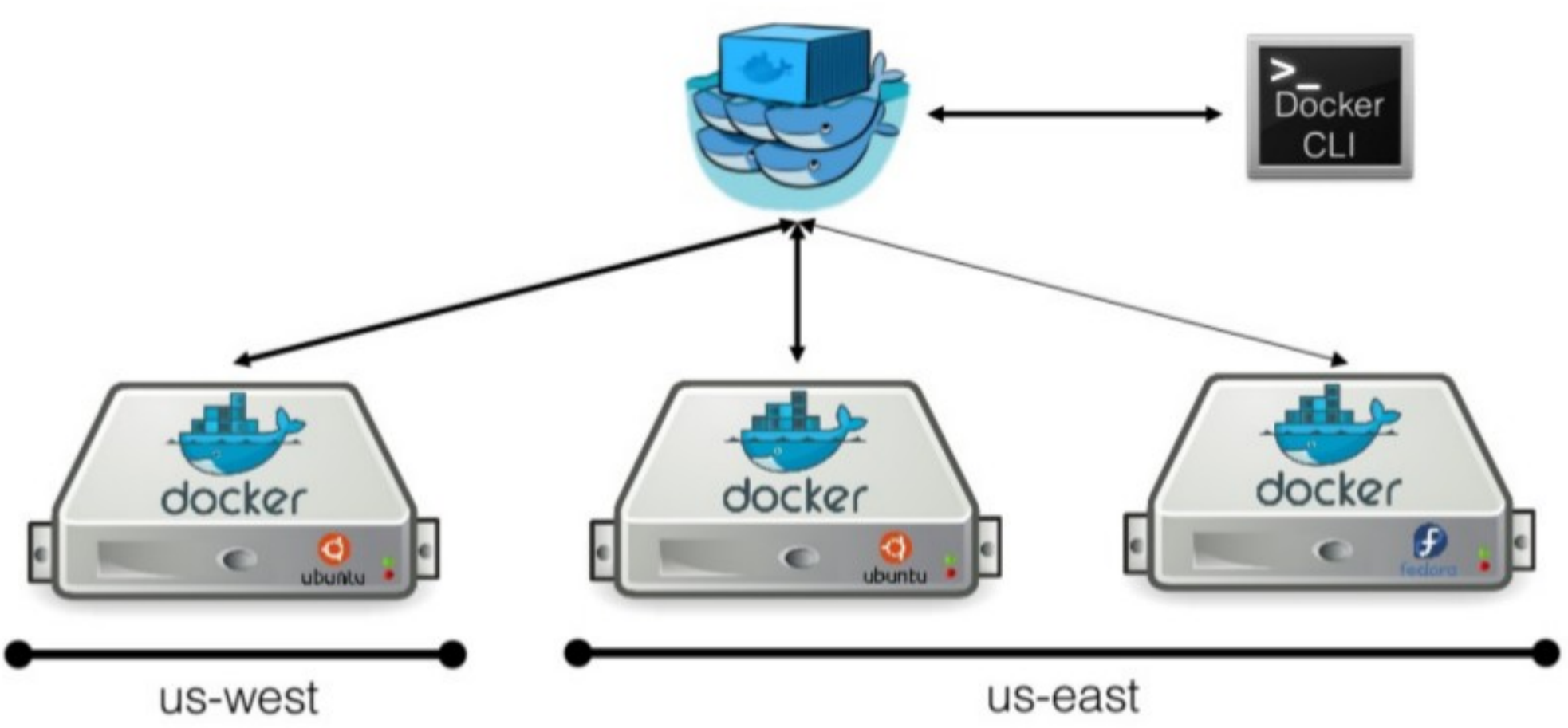
Docker Machine

Abstraction for provisionning and using docker hosts



Docker Swarm

Manage a cluster of docker hosts



Swarm mode overview

- Feature highlights
  - Cluster management integrated with Docker Engine: Use the Docker Engine CLI to create a swarm of Docker Engines where you can deploy application services. You don't need additional orchestration software to create or manage a swarm.
  - Decentralized design: Instead of handling differentiation between node roles at deployment time, the Docker Engine handles any specialization at runtime. You can deploy both kinds of nodes, managers and workers, using the Docker Engine. This means you can build an entire swarm from a single disk image.
  - Declarative service model: Docker Engine uses a declarative approach to let you define the desired state of the various services in your application stack. For example, you might describe an application comprised of a web front end service with message queueing services and a database backend.

Swarm mode overview

- Feature highlights
  - Scaling: For each service, you can declare the number of tasks you want to run. When you scale up or down, the swarm manager automatically adapts by adding or removing tasks to maintain the desired state.
  - Desired state reconciliation: The swarm manager node constantly monitors the cluster state and reconciles any differences between the actual state and your expressed desired state. For example, if you set up a service to run 10 replicas of a container, and a worker machine hosting two of those replicas crashes, the manager will create two new replicas to replace the replicas that crashed. The swarm manager assigns the new replicas to workers that are running and available.
  - Multi-host networking: You can specify an overlay network for your services. The swarm manager automatically assigns addresses to the containers on the overlay network when it initializes or updates the application.

Swarm mode overview

- Feature highlights
  - Service discovery: Swarm manager nodes assign each service in the swarm a unique DNS name and load balances running containers. You can query every container running in the swarm through a DNS server embedded in the swarm.
  - Load balancing: You can expose the ports for services to an external load balancer. Internally, the swarm lets you specify how to distribute service containers between nodes.
  - Secure by default: Each node in the swarm enforces TLS mutual authentication and encryption to secure communications between itself and all other nodes. You have the option to use self-signed root certificates or certificates from a custom root CA.
  - Rolling updates: At rollout time you can apply service updates to nodes incrementally. The swarm manager lets you control the delay between service deployment to different sets of nodes. If anything goes wrong, you can roll-back a task to a previous version of the service.

Docker Swarm Lab - Init your swarm

- **Create a Docker Swarm first**

```
[root@docker-host]# curl -fsSL https://get.docker.com/ | sh
[root@docker-host]# systemctl start docker
[root@docker-host]# systemctl status docker
[root@docker-host]# systemctl enable docker


[root@docker-host]# docker swarm init
Swarm initialized: current node (73yn8s77g2xz3277f137hye41) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-0xg56f2v9tvy0lg9d4j7xbf7cf1mg8ylm7d19f39gqvc41d1yk-
0trhxa6skixvif1o6pultvcp3 \
    10.7.60.26:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
instructions.
```

- **Show members of swarm**
  From the first terminal, check the number of nodes in the swarm (running this command from the second terminal worker will fail as swarm related commands need to be issued against a swarm manager).

```
[root@docker-host ~]# docker node ls
ID                            HOSTNAME     STATUS  AVAILABILITY  MANAGER STATUS
73yn8s77g2xz3277f137hye41 *   docker-host  Ready   Active        Leader
```

Docker Swarm - Clone the voting-app

- Let's retrieve the voting app code from Github and go into the application folder.
- Ensure you are in the first terminal and do the below:

```
[root@docker-host ~]# git clone https://github.com/docker/example-voting-app
Cloning into 'example-voting-app'...

remote: Counting objects: 374, done.
remote: Total 374 (delta 0), reused 0 (delta 0), pack-reused 374
Receiving objects: 100% (374/374), 204.32 KiB | 156.00 KiB/s, done.
Resolving deltas: 100% (131/131), done.

[root@docker-host ~]# cd example-voting-app
```

Docker Swarm - Deploy a stack

- A stack is a group of service that are deployed together. The docker-stack.yml in the current folder will be used to deploy the voting app as a stack.
- Ensure you are in the first terminal and do the below:

```
[root@docker-host]# curl -fsSL https://get.docker.com/ | sh
[root@docker-host]# systemctl start docker
[root@docker-host]# systemctl status docker
[root@docker-host]# systemctl enable docker

[root@docker-host]# docker stack deploy --compose-file=docker-stack.yml voting_stack
```

- Check the stack deployed from the first terminal

```
[root@docker-host ~]# docker stack ls
NAME          SERVICES
voting_stack  6
```

Docker Swarm - Deploy a stack

- **Check the service within the stack**

```
[root@docker-host ~]# docker stack services voting_stack
ID             NAME                      MODE         REPLICAS  IMAGE
10rt1wczotze  voting_stack_visualizer  replicated  1/1       dockersample
s/visualizer:stable
8lqj31k3q5ek  voting_stack_redis       replicated  2/2       redis:alpine
nhb4igkkyg4y  voting_stack_result      replicated  2/2       dockersample
s/examplevotingapp_result:before
nv8d2z2qhlx4  voting_stack_db          replicated  1/1       postgres:9.4
ou47zdyf6cd0  voting_stack_vote        replicated  2/2       dockersample
s/examplevotingapp_vote:before
rpnxwmoipagq  voting_stack_worker      replicated  1/1       dockersample
s/examplevotingapp_worker:latest
```

- **Check the service within the stack**

```
[root@docker-host ~]# docker service ps voting_stack_vote
ID             NAME                    IMAGE
      NODE   DESIRED STATE  CURRENT STATE           ERROR  PORTS
my7jqgze7pgg  voting_stack_vote.1  dockersamples/examplevotingapp_vote:be
fore  node1  Running        Running 56 seconds ago
3jzgk39dyr6d  voting_stack_vote.2  dockersamples/examplevotingapp_vote:be
fore  node2  Running        Running 58 seconds ago
```

Docker Swarm - Creating services

- The next step is to create a service and list out the services. This creates a single service called web that runs the latest nginx, type the below commands in the first terminal:

```
[root@docker-host ~]# docker service create -p 80:80 --name web nginx:latest

[root@docker-host example-voting-app]# docker service ls | grep nginx
24jakxhfl06l        web                 replicated          1/1
nginx:latest                            *:80->80/tcp
```

- **Scaling Up Application**

```
[root@docker-host ~]# docker service inspect web

[root@docker-host ~]# docker service scale web=15
web scaled to 15

[root@docker-host ~]# docker service ls | grep nginx
24jakxhfl06l        web                 replicated          15/15
nginx:latest                            *:80->80/tcp
```

Docker Swarm - Creating services

- **Scaling Down Application**

```
[root@docker-host ~]# docker service scale web=10
web scaled to 10

[root@docker-host ~]# docker service ps web
ID                  NAME                IMAGE               NODE                DESIRED
STATE               CURRENT STATE               ERROR               PORTS
jrgkmkvm4idf        web.2               nginx:latest        docker-host         Running
                    Running about a minute ago
dmreadcm745k        web.4               nginx:latest        docker-host         Running
                    Running about a minute ago
5iik87rbsfc2        web.6               nginx:latest        docker-host         Running
                    Running about a minute ago
7cuzpz79q2hp        web.7               nginx:latest        docker-host         Running
                    Running about a minute ago
ql7g7k3dlbqw        web.8               nginx:latest        docker-host         Running
                    Running about a minute ago
k0bzk7m51cln        web.9               nginx:latest        docker-host         Running
                    Running about a minute ago
0teod07eihns        web.10              nginx:latest        docker-host         Running
                    Running about a minute ago
sqxfaqlnkpab        web.11              nginx:latest        docker-host         Running
                    Running about a minute ago
mkrsmwgti606        web.12              nginx:latest        docker-host         Running
                    Running about a minute ago
ucomtg454jlk        web.15              nginx:latest        docker-host         Running
                    Running about a minute ago
```

Kubernetes is a Solution?

Kubernetes – Container Orchestration at Scale

Greek for "Helmsman"; also the root of the word "Governor" and "cybernetic"

- Container Cluster Manager
  - Inspired by the technology that runs Google
- Runs anywhere
  - Public cloud
  - Private cloud
  - Bare metal
- Strong ecosystem
  - Partners: Red Hat, VMware, CoreOS..
  - Community: clients, integration

Kubernetes Resource Types

- Pods
  - Represent a collection of containers that share resources, such as IP addresses and persistent storage volumes. It is the basic unit of work for Kubernetes.
- Services
  - Define a single IP/port combination that provides access to a pool of pods. By default, services connect clients to pods in a round-robin fashion.

Kubernetes Resource Types

- Replication Controllers
  - A framework for defining pods that are meant to be horizontally scaled. A replication controller includes a pod definition that is to be replicated, and the pods created from it can be scheduled to different nodes.
- Persistent Volumes (PV)
  - Provision persistent networked storage to pods that can be mounted inside a container to store data.
- Persistent Volume Claims (PVC)
  - Represent a request for storage by a pod to Kubernetes

OpenShift Resource Types

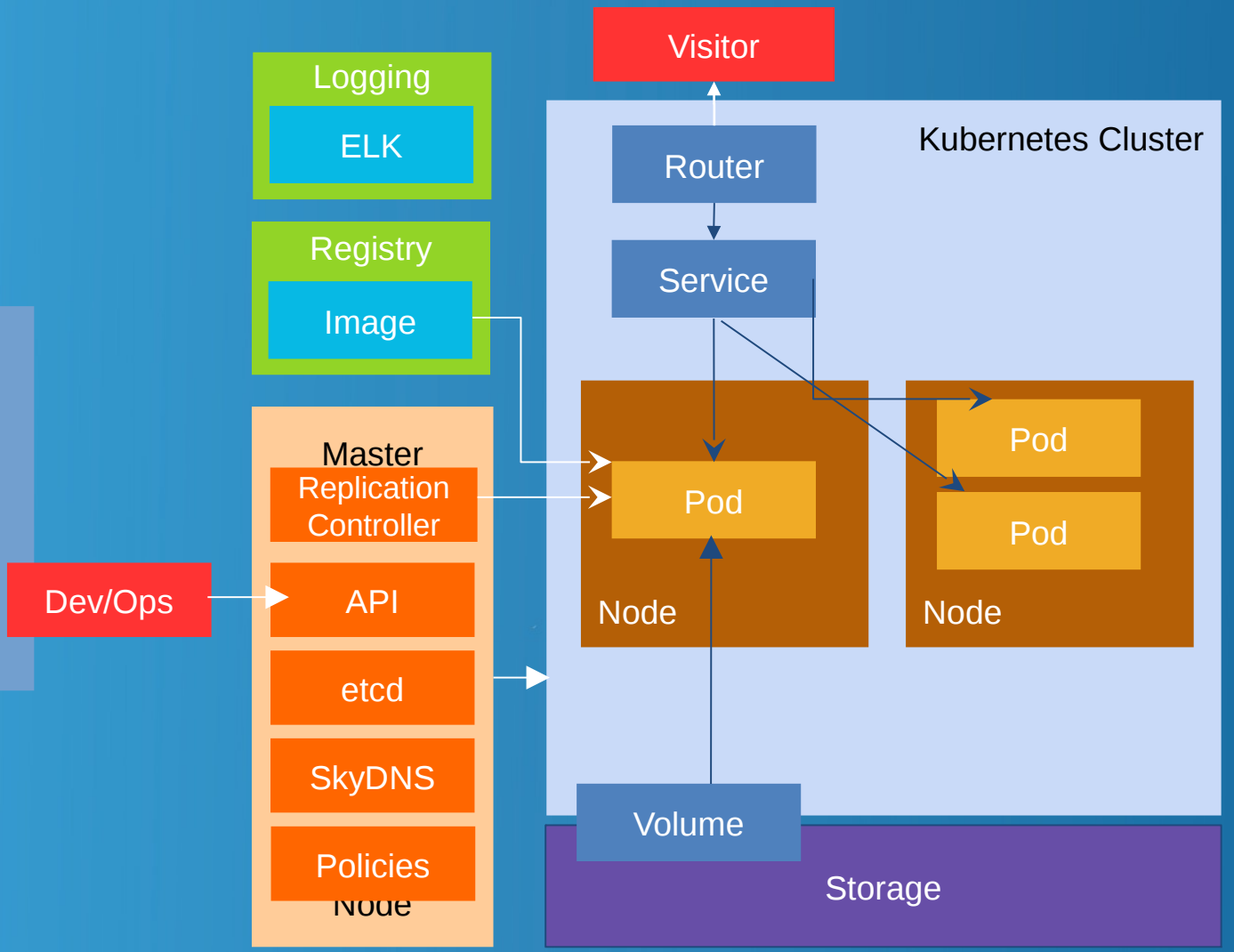
- Deployment Configurations (dc)
  - Represent a set of pods created from the same container image, managing workflows such as rolling updates. A dc also provides a basic but extensible Continuous Delivery workflow.
- Build Configurations (bc)
  - Used by the OpenShift Source-to-Image (S2I) feature to build a container image from application source code stored in a Git server. A bc works together with a dc to provide a basic but extensible Continuous Integration/Continuous Delivery workflow.
- Routes
  - Represent a DNS host name recognized by the OpenShift router as an ingress point for applications and microservices.

Kubernetes Solution Detail

Core Concepts

Pod

- Labels & Selectors
- ReplicationController
- Service
- Persistent Volumes



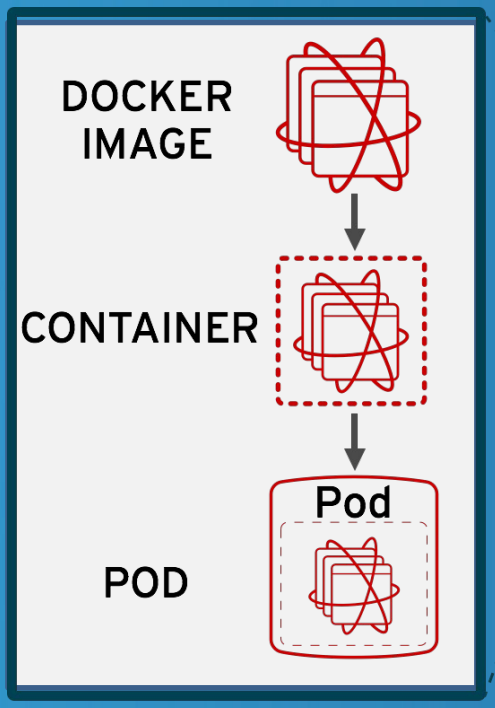
Kubernetes: The Pods

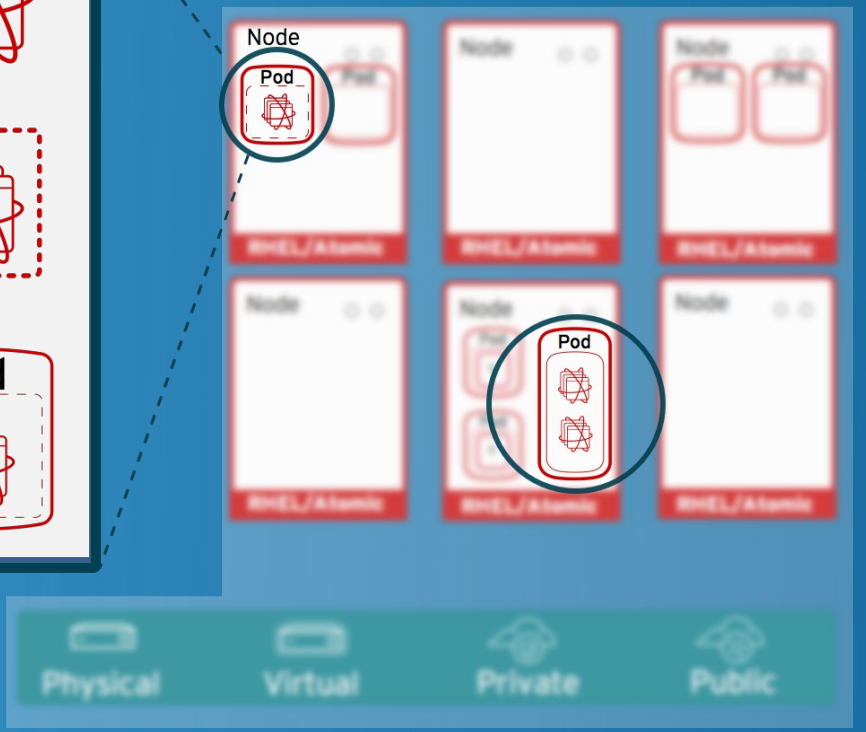
POD Definition:

- Group of Containers
- Related to each other
- Same namespace
- Emphemeral

Examples:

- Wordpress
- MySQL
- ~~Wordpress + MySQL~~
- ELK
- Nginx+Logstash
- Auth-Proxy+PHP
- App + data-load





Kubernetes: Building Pod

```
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name": "hello-openshift"
  },
  "spec": {
    "containers": [
      {
        "name": "hello-openshift",
        "image": "openshift/hello-openshift",
        "ports": [
          {
            "containerPort": 8080
          }
        ]
      }
    ]
  }
}
```

- OpenShift/Kubernetes runs containers inside Kubernetes pods, and to create a pod from a container image, Kubernetes needs a pod resource definition. This can be provided either as a **JSON** or **YAML** text file, or can be generated from defaults by oc new-app or the web console.
- This JSON object is a pod resource definition because it has attribute "**kind**" with value "**Pod**". It contains a single "**container**" whose name is "**hello-openshift**" and that references the "**image**" named "**openshift/hello-openshift**". The container also contains a single "**ports**", which listens to TCP port **8080**.

```
# kubectl create –f hello-openshift.yaml
```

```
# oc create –f hello-openshift.yaml
```

Kubernetes: List Pod

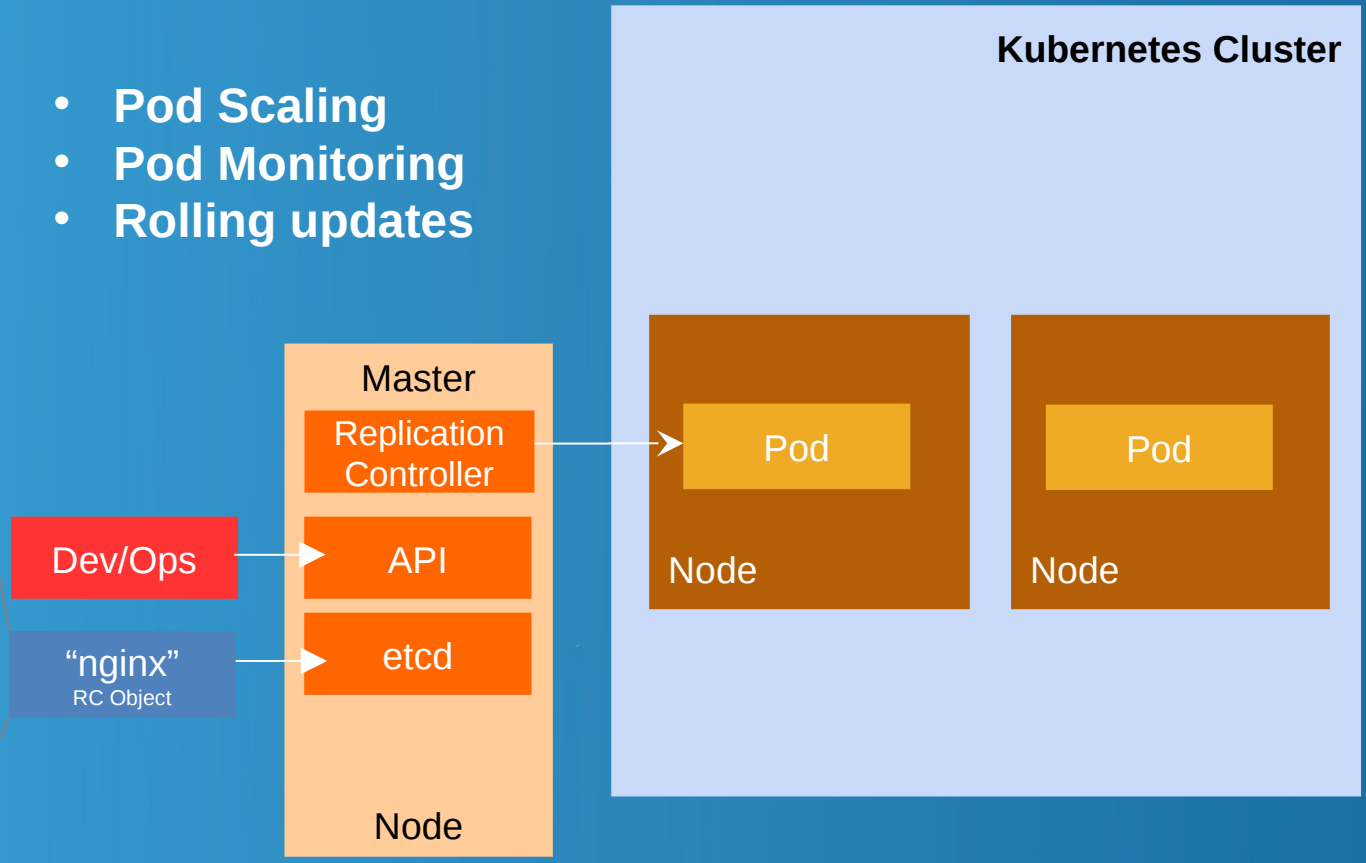
```
[root@centos-16gb-sgp1-01 ~]# oc get pod
NAME                          READY     STATUS      RESTARTS   AGE
bgdemo-1-build                0/1       Completed   0          16d
bgdemo-1-x0wlq                1/1       Running     0          16d
dc-gitlab-runner-service-3-wgn8q   1/1       Running     0          8d
dc-minio-service-1-n0614      1/1       Running     5          23d
frontend-1-build              0/1       Completed   0          24d
frontend-prod-1-gmcrw         1/1       Running     2          23d
gitlab-ce-7-kq0jp             1/1       Running     2          24d
hello-openshift               1/1       Running     2          24d
jenkins-3-8grrq               1/1       Running     12         21d
os-example-aspnet-2-build     0/1       Completed   0          22d
os-example-aspnet-3-6qncw     1/1       Running     0          21d
os-sample-java-web-1-build    0/1       Completed   0          22d
os-sample-java-web-2-build    0/1       Completed   0          22d
os-sample-java-web-3-build    0/1       Completed   0          22d
os-sample-java-web-3-sqf41    1/1       Running     0          22d
os-sample-python-1-build      0/1       Completed   0          22d
os-sample-python-1-p5b73      1/1       Running     0          22d
```

Kubernetes: Replication Controller

- **Pod Scaling**
- **Pod Monitoring**
- **Rolling updates**

```
kind: ReplicationController
metadata:
   name: nginx
spec:
   replicas: 2
   selector:
      app: nginx
template:
   metadata:
      name: nginx
      labels:
         app: nginx
   spec:
      containers:
       - name: nginx
         image: nginx:v2.2
         ports:
          - containerPort: 80
```

```
# kubectl create –f nginx-rc.yaml
```



Kubernetes: Service

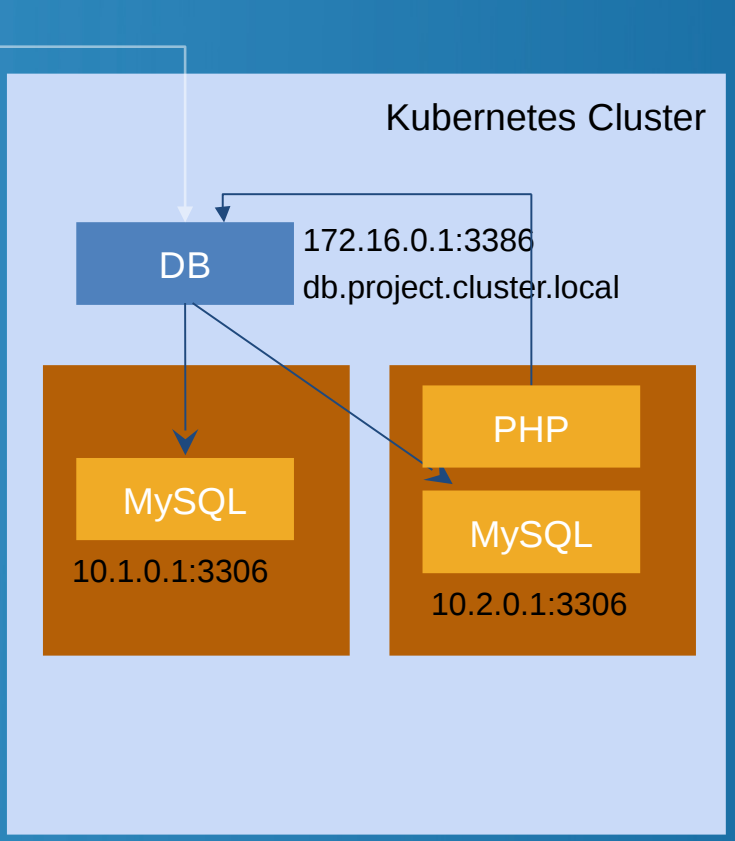
Service Definition:

- Load-Balanced Virtual-IP *(layer 4)*
- Abstraction layer for your App
- Enables Service Discovery
  - DNS
  - ENV

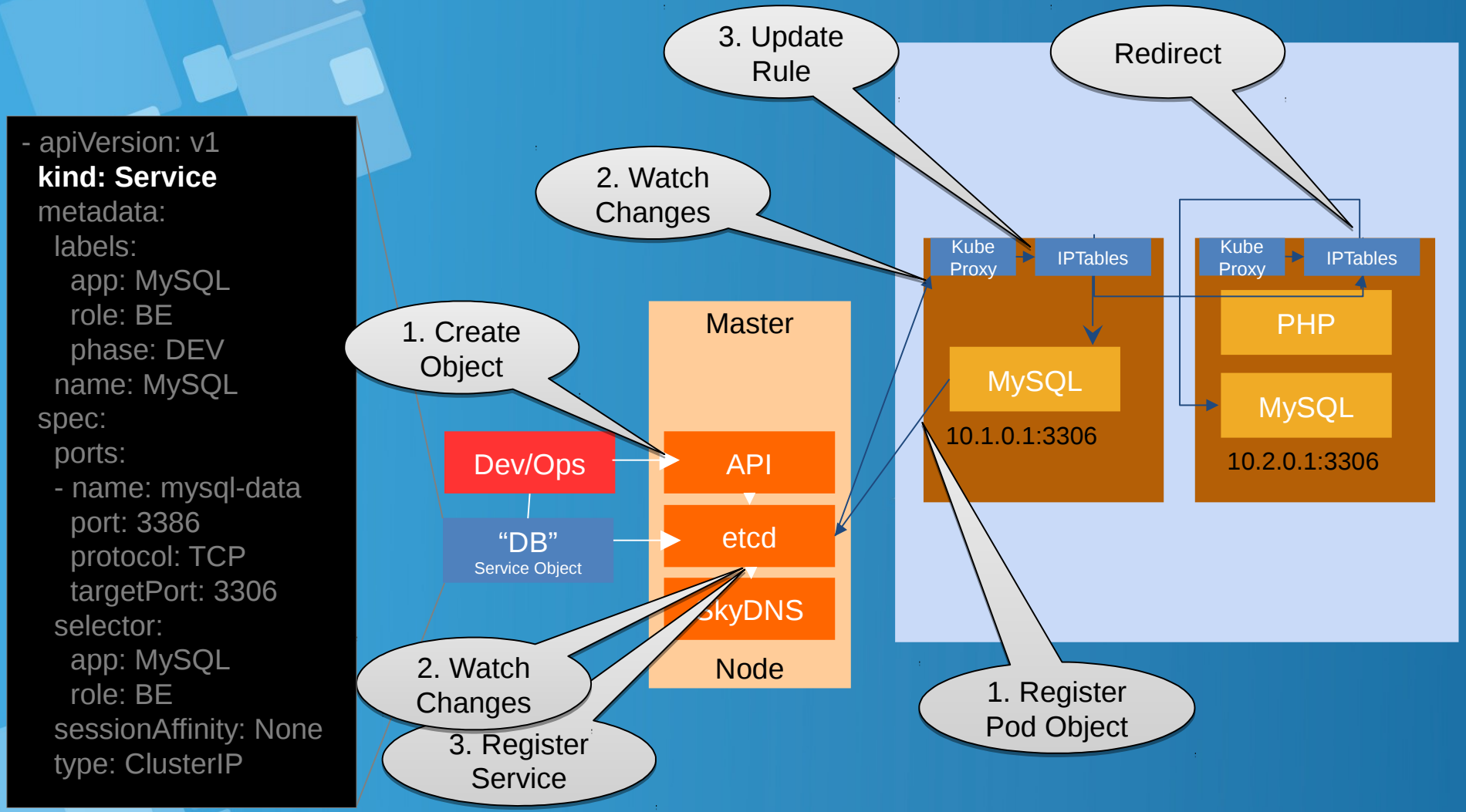
Examples:

- frontend
- database
- api

```
<?php
   mysql_connect(getenv("db_host"))
   mysql_connect("db:3306")
?>
```



Kubernetes: Service Continue..

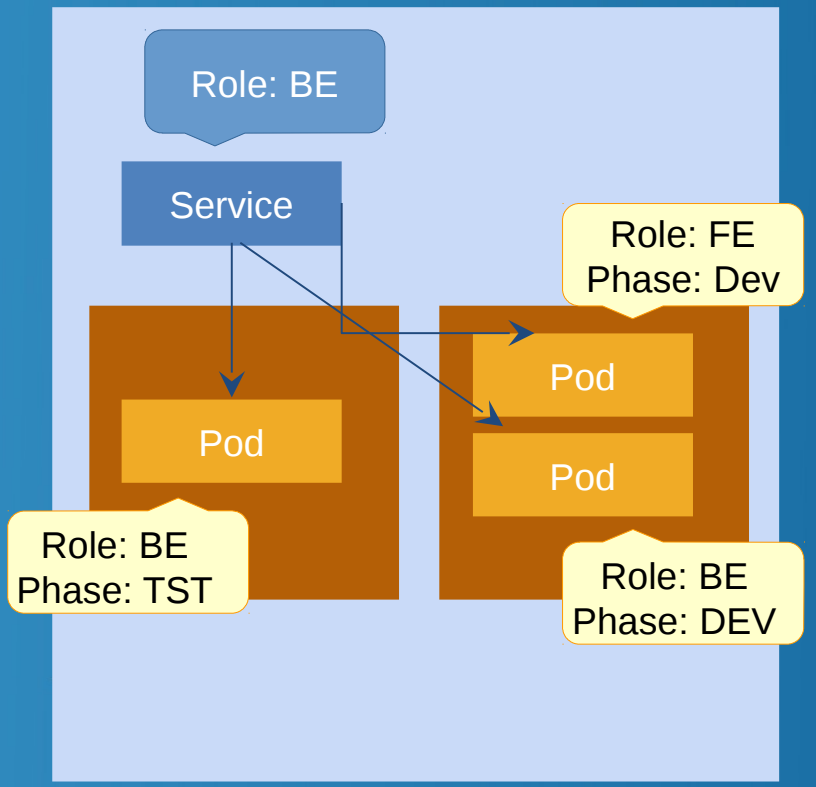


Kubernetes: Labels & Selectors

think SQL 'select ... where ...'

```
- apiVersion: v1
  kind: Service
  metadata:
    labels:
      app: MyApp
      role: BE
      phase: DEV
    name: MyApp
  spec:
    ports:
    - name: 80-tcp
      port: 80
      protocol: TCP
      targetPort: 8080
    selector:
      app: MyApp
      role: BE
    sessionAffinity: None
    type: ClusterIP
```

```
- apiVersion: v1
  kind: Pod
  metadata:
    labels:
      app: MyApp
      role: BE
      phase: DEV
    name: MyApp
```



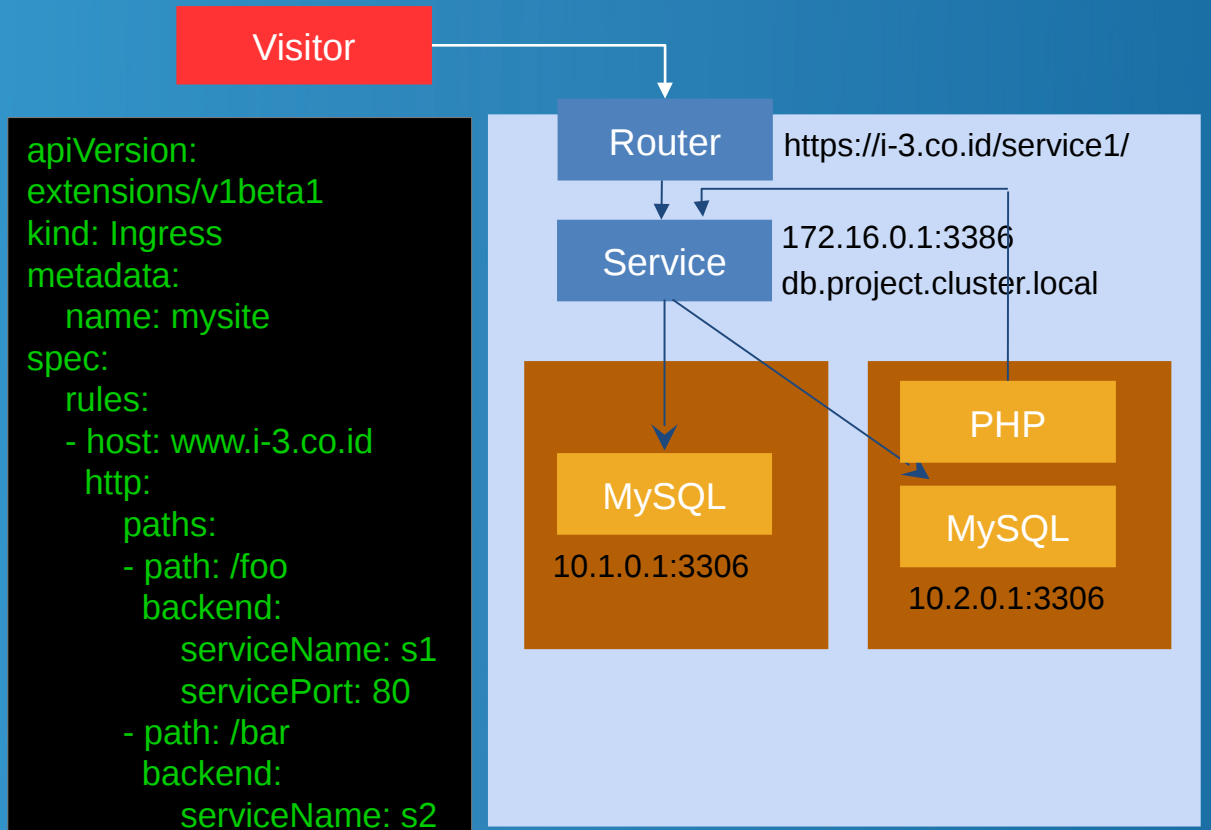
Kubernetes: Ingress / Router

- Router Definition:
  - Layer 7 Load-Balancer / Reverse Proxy
  - SSL/TLS Termination
  - Name based Virtual Hosting
  - Context Path based Routing
  - Customizable (image)
    - HA-Proxy
    - F5 Big-IP

Examples:

- https://www.i-3.co.id/myapp1/
- http://www.i-3.co.id/myapp2/

```
apiVersion:
extensions/v1beta1
kind: Ingress
metadata:
   name: mysite
spec:
   rules:
   - host: www.i-3.co.id
     http:
        paths:
        - path: /foo
          backend:
             serviceName: s1
             servicePort: 80
        - path: /bar
          backend:
             serviceName: s2
             servicePort: 80
```



Kubernetes: Router Detail

```
[root@centos-16gb-sgp1-01 ~]# oc env pod router-1-b97bv --list
# pods router-1-b97bv, container router
DEFAULT_CERTIFICATE_DIR=/etc/pki/tls/private
ROUTER_EXTERNAL_HOST_HOSTNAME=
ROUTER_EXTERNAL_HOST_HTTPS_VSERVER=
ROUTER_EXTERNAL_HOST_HTTP_VSERVER=
ROUTER_EXTERNAL_HOST_INSECURE=false
ROUTER_EXTERNAL_HOST_INTERNAL_ADDRESS=
ROUTER_EXTERNAL_HOST_PARTITION_PATH=
ROUTER_EXTERNAL_HOST_PASSWORD=
ROUTER_EXTERNAL_HOST_PRIVKEY=/etc/secret-volume/router.pem
ROUTER_EXTERNAL_HOST_USERNAME=
ROUTER_EXTERNAL_HOST_VXLAN_GW_CIDR=
ROUTER_SERVICE_HTTPS_PORT=443
ROUTER_SERVICE_HTTP_PORT=80
ROUTER_SERVICE_NAME=router
ROUTER_SERVICE_NAMESPACE=default
ROUTER_SUBDOMAIN=
STATS_PASSWORD=XXXXXX
STATS_PORT=1936
STATS_USERNAME=admin
```

- Check the router environment variables to find connection parameters for the HAProxy process running inside the pod

Kubernetes: Router-HAProxy

HAProxy

Statistics Report for pid 3301

> General process information

pid = 3301 (process #1, nbproc = 1)
uptime = 1d 14h17m37s
system limits: memmax = unlimited; ulimit-n = 40057
maxsock = 40057; maxconn = 20000; maxpipes = 0
current conns = 2; current pipes = 0/0; conn rate = 0/sec
Running tasks: 1/53; idle = 100 %

- active UP
- active UP, going down
- active DOWN, going up
- active or backup DOWN
- active or backup DOWN for maintenance (MAINT)
- active or backup SOFT STOPPED for maintenance
- backup UP
- backup UP, going down
- backup DOWN, going up
- not checked

Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:

- Scope :
- Hide 'DOWN' servers
- Refresh now
- CSV export

External resources:

- Primary site
- Updates (v1.5)
- Online manual

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	2	-	1	2	20 000	27 577			1 668 964	4 591 520	0	0	13 788					OPEN								
Backend	0	0		0	0		0	0	2 000	0	0	0s	1 668 964	4 591 520	0	0		0	0	0	0	1d14h UP		0	0	0		0		

public

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	3	-	1	2	20 000	33			37 357 332	27 189 019	0	0	6					OPEN								

public_ssl

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	18	-	0	7	20 000	1 175			2 245 759	19 381 416	0	0	0					OPEN								

be_sni

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
fe_sni	0	0	-	0	18		0	6	-	1 015	1 015	2m35s	2 059 121	6 125 257		0		0	0	0	0			1	Y	-				-
Backend	0	0		0	18		0	6	2 000	1 015	1 015	2m35s	2 059 121	6 125 257	0	0		0	0	0	0	1d14h UP		1	1	0		0	0s	

fe_sni

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	18	-	0	6	20 000	1 015			1 591 702	3 659 588	0			60				OPEN								

be_no_sni

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
fe_no_sni	0	0	-	0	1		0	1	-	6	6	14m46s	1 956	14 511		0									Y					-

HAPROXY
Powering Your Uptime

Kubernetes: Persistent Storage

For Ops:

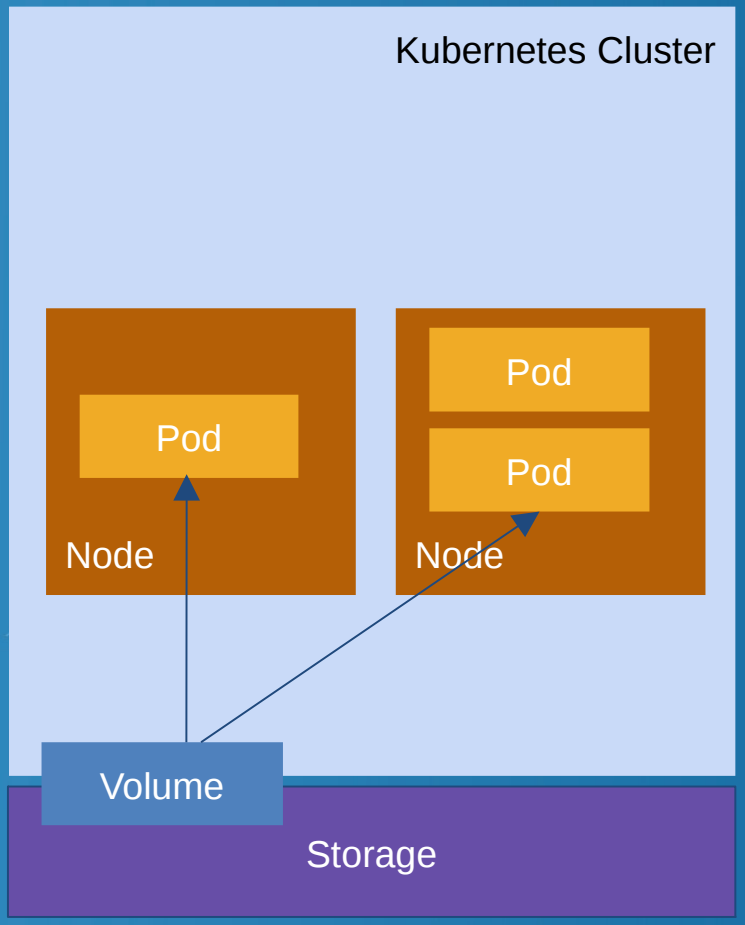
- Google
- AWS EBS
- OpenStack's Cinder
- Ceph
- GlusterFS
- NFS
- iSCSI
- FibreChannel
- EmptyDir

for Dev:

- "Claim"

```
kind: PersistentVolume
metadata:
   name: pv0003
spec:
   capacity:
      storage: 8Gi
   accessModes:
    - ReadWriteOnce
   nfs:
      path: /tmp
      server: 172.17.0.2
```

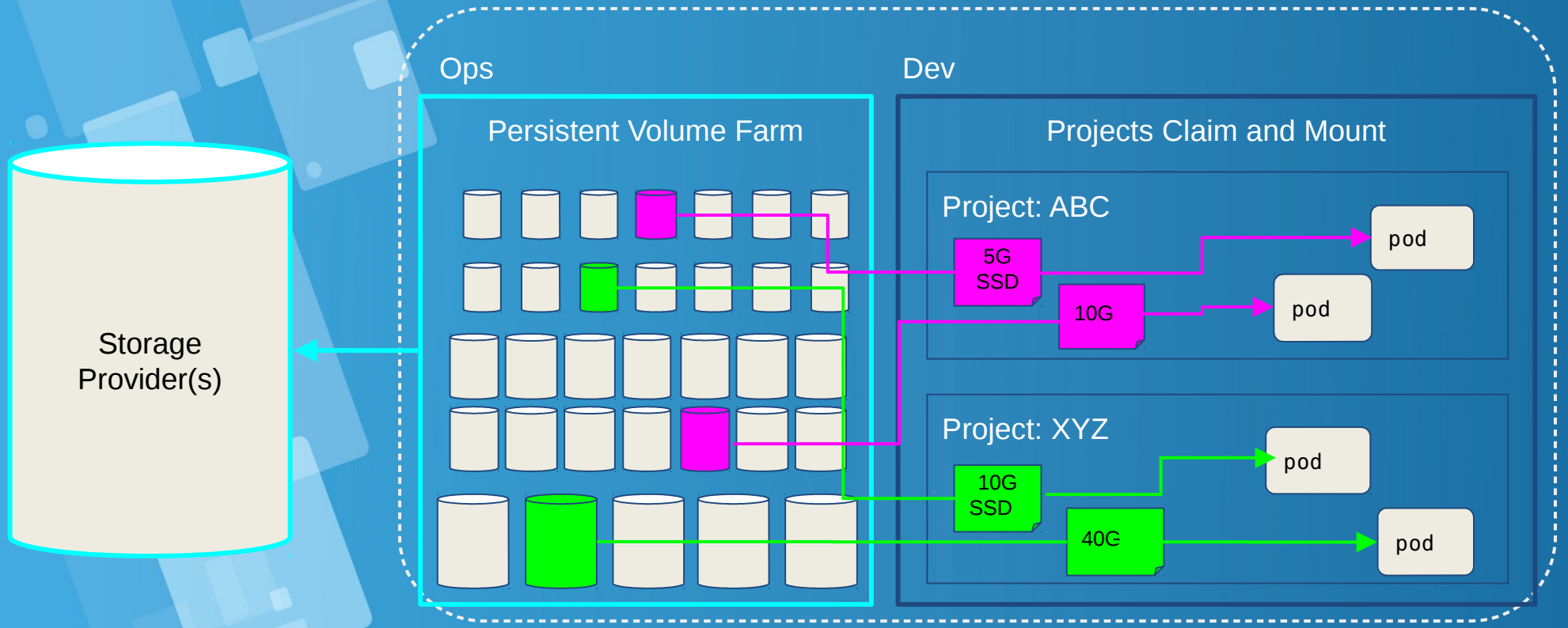
```
kind: PersistentVolumeClaim
metadata:
   name: myclaim
spec:
   accessModes:
    - ReadWriteOnce
   resources:
      requests:
         storage: 8Gi
```



Kubernetes: Persistent Storage

- Kubernetes provides a framework for managing external persistent storage for containers. Kubernetes recognizes a PersistentVolume resource, which defines local or network storage. A pod resource can reference a PersistentVolumeClaim resource in order to access a certain storage size from a PersistentVolume.
- Kubernetes also specifies if a PersistentVolume resource can be shared between pods or if each pod needs its own PersistentVolume with exclusive access. When a pod moves to another node, it keeps connected to the same PersistentVolumeClaim and PersistentVolume instances. So a pod's persistent storage data follows it, regardless of the node where it is scheduled to run.

Kubernetes: Persisten Volume Claim



Kubernetes: Networking

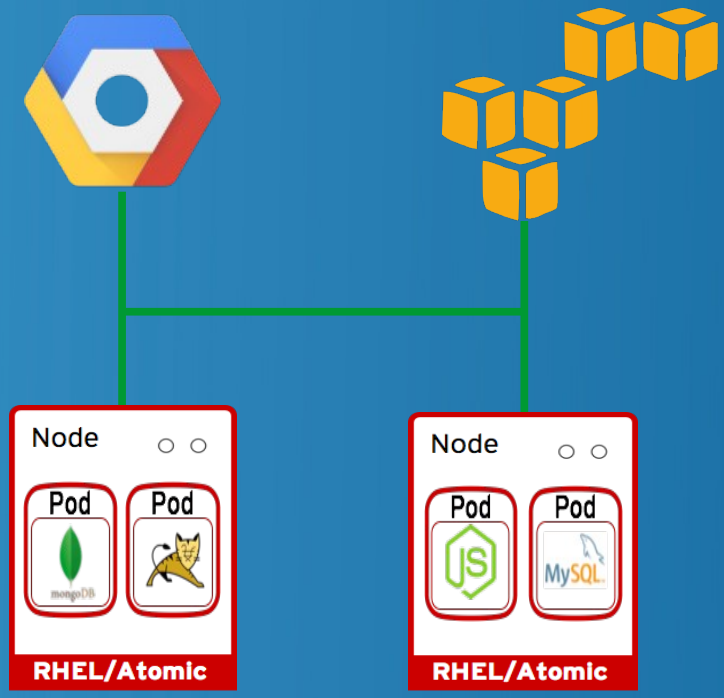
- Each Host = 256 IPs
- Each POD = 1 IP

Programmable Infra:

- GCE / GKE
- AWS
- OpenStack
- Nuage

Overlay Networks:

- Flannel
- Weave
- OpenShift-SDN
- Open vSwitch

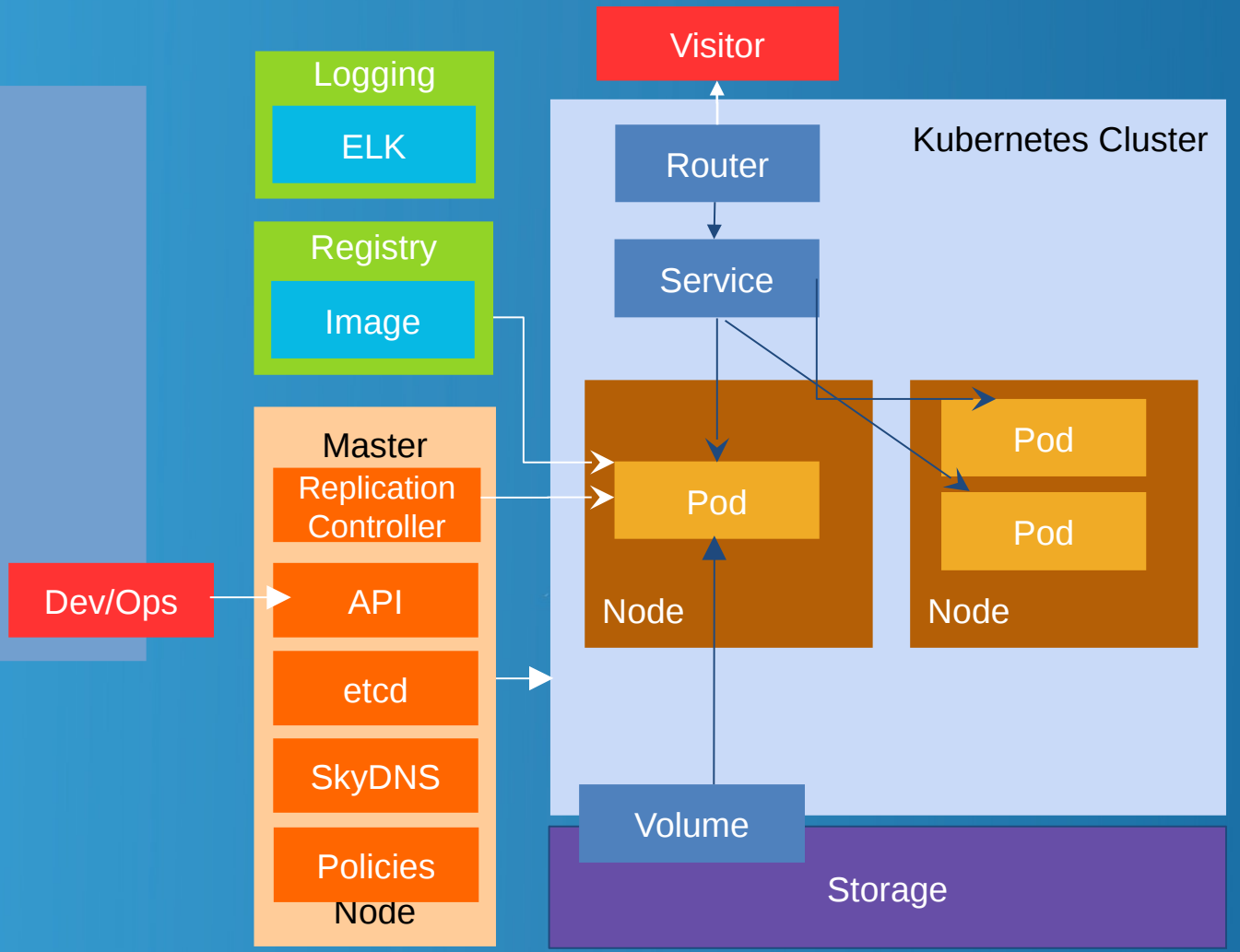


Kubernetes: Networking

- Docker networking is very simple. Docker creates a virtual kernel bridge and connects each container network interface to it. Docker itself does not provide a way to allow a pod from one host to connect to a pod from another host. Docker also does not provide a way to assign a public fixed IP address to an application so external users can access it.
- Kubernetes provides service and route resources to manage network visibility between pods and from the external world to them. A service load-balances received network requests among its pods, while providing a single internal IP address for all clients of the service (which usually are other pods). Containers and pods do not need to know where other pods are, they just connect to the service. A route provides an external IP to a service, making it externally visible.

Kubernetes: Hosting Platform

- Scheduling
- Lifecycle and health
- Discovery
- Monitoring
- Auth{n,z}
- Scaling



Kubernetes: High Avaibility

- High Availability (HA) on an Kubernetes/OpenShift Container Platform cluster has two distinct aspects: HA for the OCP infrastructure itself, that is, the masters, and HA for the applications running inside the OCP cluster.
- For applications, or "pods", OCP handles this by default. If a pod is lost, for any reason, Kubernetes schedules another copy, connects it to the service layer and to the persistent storage. If an entire Node is lost, Kubernetes schedules replacements for all its pods, and eventually all applications will be available again. The applications inside the pods are responsible for their own state, so they need to be HA by themselves, if they are stateful, employing proven techniques such as HTTP session replication or database replication.

Authentication Methods

- Authentication is based on OAuth , which provides a standard HTTP-based API for authenticating both interactive and non-interactive clients.
  - HTTP Basic, to delegate to external Single Sign-On (SSO) systems
  - GitHub and GitLab, to use GitHub and GitLab accounts
  - OpenID Connect, to use OpenID-compatible SSO and Google Accounts
  - OpenStack Keystone v3 server
  - LDAP v3 server

Kubernetes: Authorization policies

- There are two levels of authorization policies:
  - Cluster policy: Controls who has various access levels to Kubernetes / OpenShift Container Platform and all projects. Roles that exist in the cluster policy are considered cluster roles.
  - Local policy: Controls which users have access to their projects. Roles that exist in a local policy are considered local roles.

- Authorization is managed using the following:
  - Rules: Sets of permitted verbs on a set of resources; for example, whether someone can delete projects.
  - Roles: Collections of rules. Users and groups can be bound to multiple roles at the same time.
  - Binding: Associations between users and/or groups with a role.

OpenShift as a Development Platform

Project spaces
Build tools
Integration with your IDE

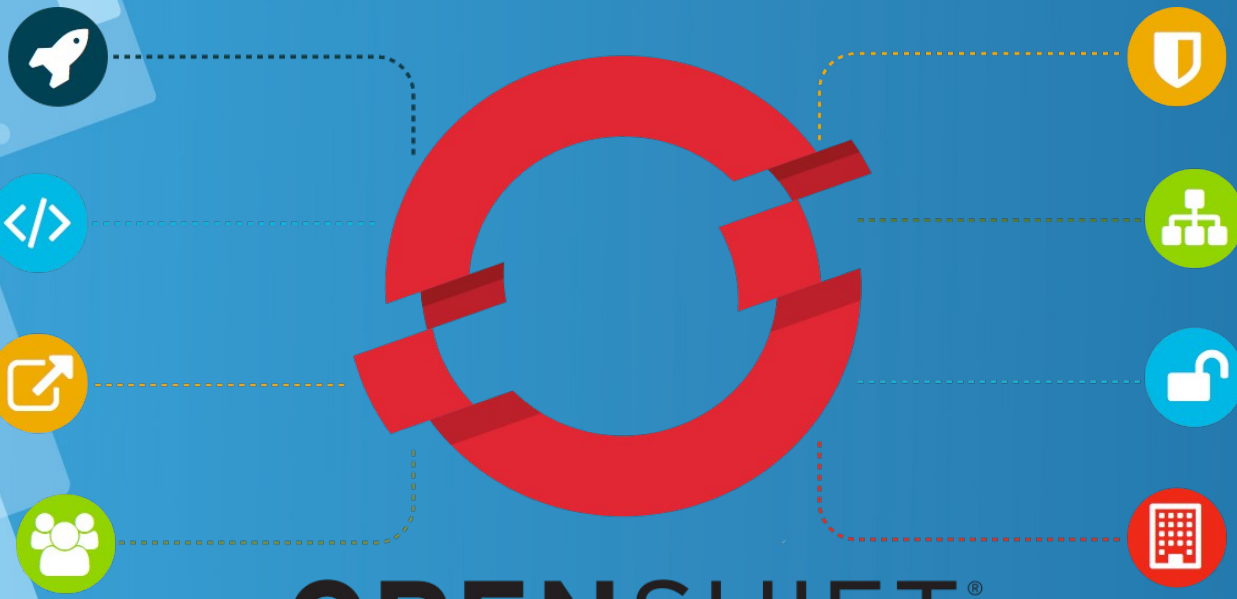
We Need more than just Orchestration

Self Service
-Templates
- Web Console

Multi-Language

Automation
- Deploy
- Build

DevOps
Collaboration



Secure
- Namespaced
- RBAC

Scalable
- Integrated LB

Open Source

Enterprise
- Authentication
- Web Console
- Central Logging

This past week at KubeCon 2016, Red Hat CTO Chris Wright (@kernelcdub) gave a keynote entitled OpenShift is Enterprise-Ready Kubernetes. There it was for the 1200 people in attendance: OpenShift is 100% Kubernetes, plus all the things that you'll need to run it in production environments. - https://blog.openshift.com/enterprise-ready-kubernetes/

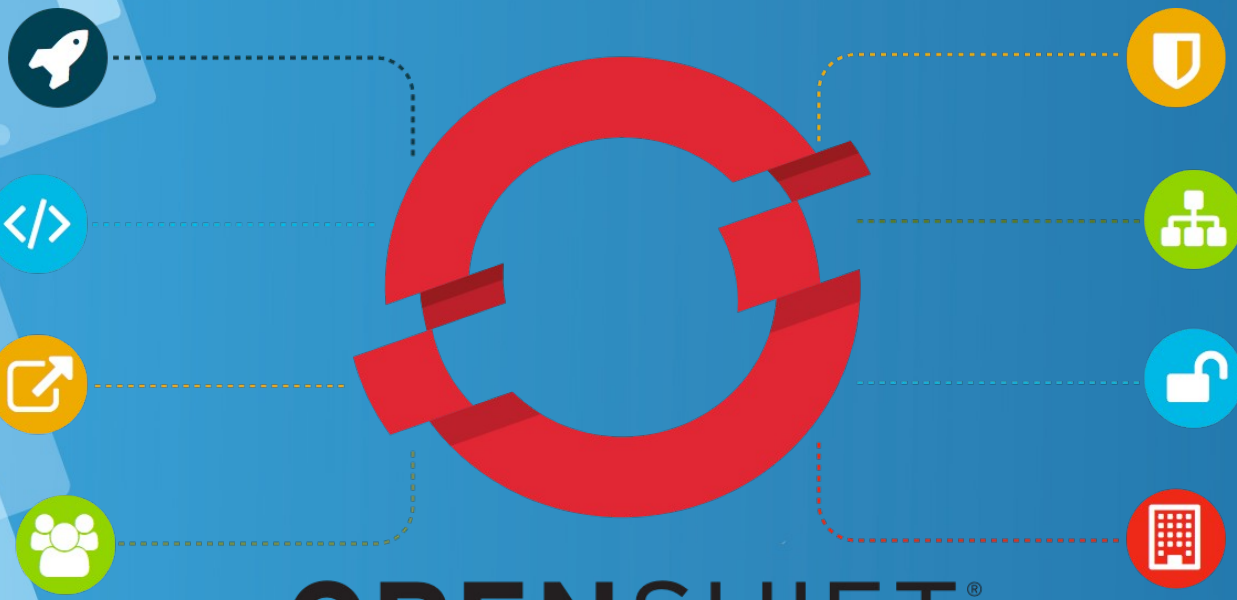
OpenShift is Red Hat Container Application Platform (PaaS)

Self Service
-Templates
- Web Console

Multi-Language

Automation
- Deploy
- Build

DevOps
Collaboration



Secure
- Namespaced
- RBAC

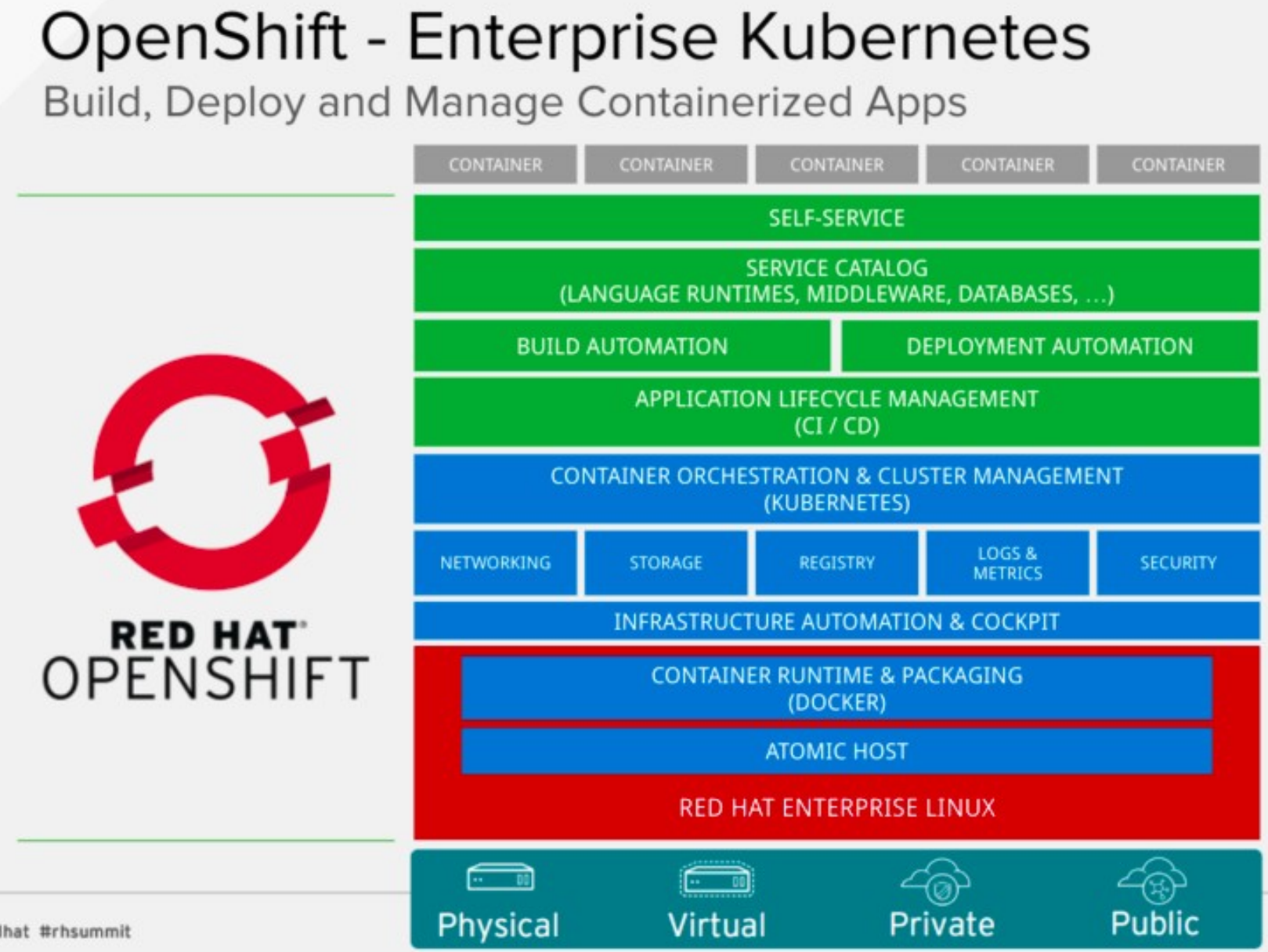
Scalable
- Integrated LB

Open Source

Enterprise
- Authentication
- Web Console
- Central Logging

https://blog.openshift.com/red-hat-chose-kubernetes-openshift/
https://blog.openshift.com/chose-not-join-cloud-foundry-foundation-recommendations-2015/

OpenShift=Enterprise K8s



OpenShift Software Stack

DevOps Tools and User Experience
Web Console, CLI, REST API, SCM integration

Containerized Services
Auth, Networking, Image Registry

Runtimes and xPaaS
Java, Ruby, Node.js and more

Kubernetes
Container orchestration and management

Etcd
Cluster state and configs

OCP-kubernetes Extensions

Docker
Container API and packaging format

RHEL
Container optimized OS

OpenShift Technology

Basic container infrastructure is shown, integrated and enhanced by Red Hat

- The base OS is RHEL/CentOS/Fedora.
- Docker provides the basic container management API and the container image file format.
- Kubernetes is an open source project aimed at managing a cluster of hosts (physical or virtual) running containers. It works with templates that describe multicontainer applications composed of multiple resources, and how they interconnect. If Docker is the "core" of OCP, Kubernetes is the "heart" that keeps it moving.
- Etcd is a distributed key-value store, used by Kubernetes to store configuration and state information about the containers and other resources inside the OCP cluster.

Kubernetes Embedded

https://master:8443/api = Kubernetes API
/oapi = OpenShift API
/console = OpenShift WebConsole

OpenShift:

- 1 Binary for Master
- 1 Binary for Node
- 1 Binary for Client

- Docker-image
- Vagrant-image

Kubernetes:

- ApiServer, Controller, Scheduler, Etcd
- KubeProxy, Kubelet
- Kubectl

Project Namespace

Project

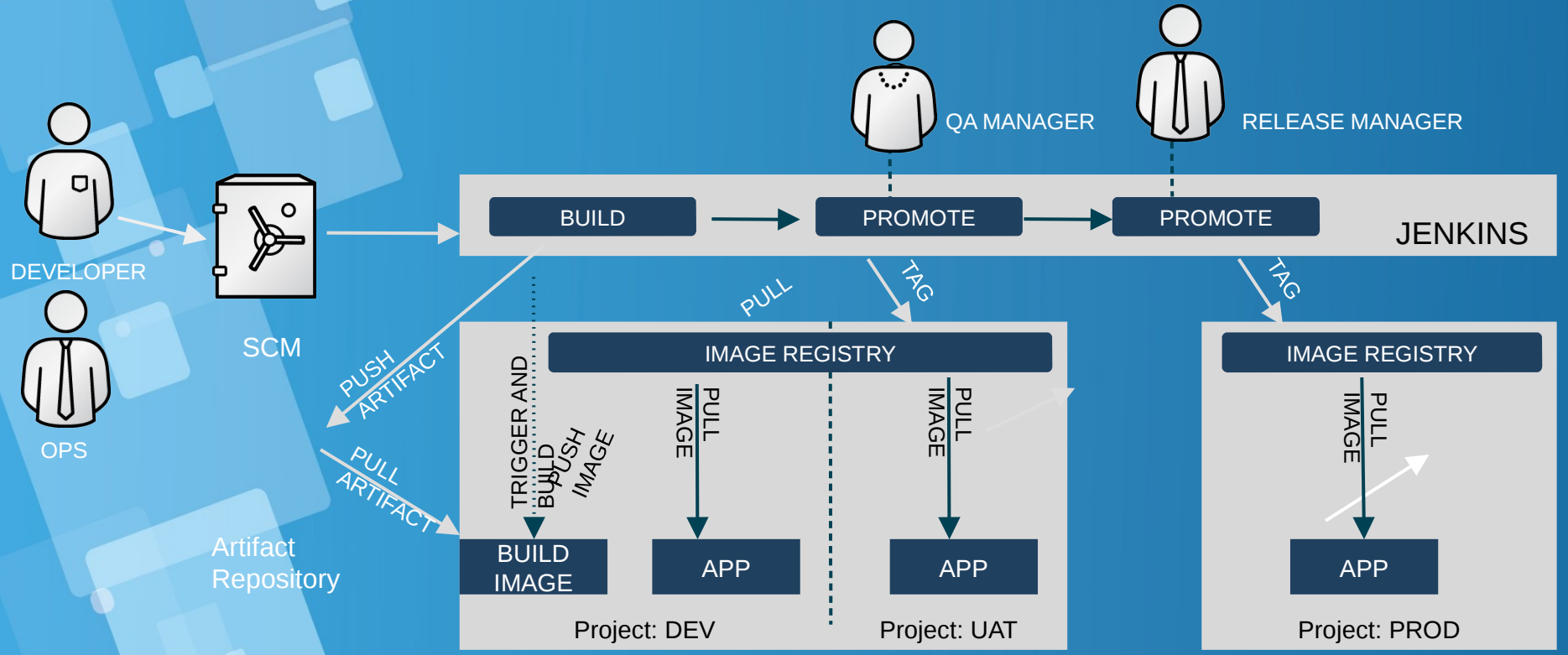
- Sandboxed Environment
- Network VXLan
- Authorization Policies
- Resource Quotas
- *Ops in Control, Dev Freedom*

App

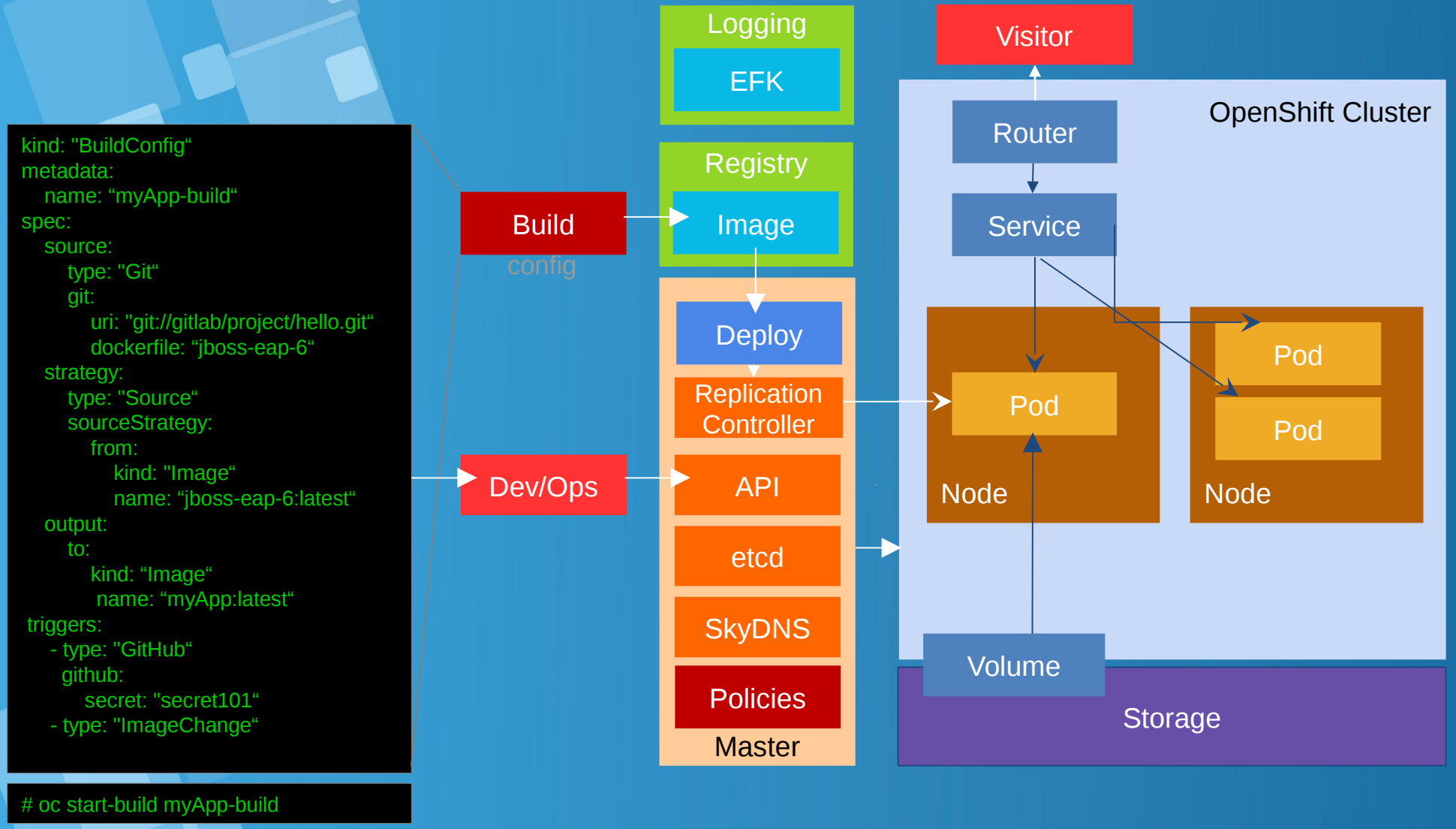
- Images run in Containers
- Grouped together as a Service
- Defined as Template


```
oc new-project  Project-Dev
oc policy add-role-to-user admin scientist1
oc new-app
  --source=https://gitlab/MyJavaApp
  --docker-image=jboss-eap
```

CI/CD Flow


DEVELOPER
OPS
SCM
QA MANAGER
RELEASE MANAGER
BUILD
PROMOTE
PROMOTE
JENKINS
PUSH ARTIFACT
PULL ARTIFACT
Artifact Repository
PULL
TAG
TAG
IMAGE REGISTRY
TRIGGER AND BUILD
PUSH IMAGE
PULL IMAGE
PULL IMAGE
BUILD IMAGE
APP
APP
Project: DEV
Project: UAT
IMAGE REGISTRY
PULL IMAGE
APP
Project: PROD

OpenShift Build & Deploy Architecture



Building Images

- OpenShift/Kubernetes can build a pod from three different sources
  - A container image: The first source leverages the Docker container ecosystem. Many vendors package their applications as container images, and a pod can be created to run those application images inside OpenShift
  - A Dockerfile: The second source also leverages the Docker container ecosystem. A Dockerfile is the Docker community standard way of specifying a script to build a container image from Linux OS distribution tools.
  - Application source code (Source-to-Image or S2I): The third source, S2I, empowers a developer to build container images for an application without dealing with or knowing about Docker internals, image registries, and Dockerfiles

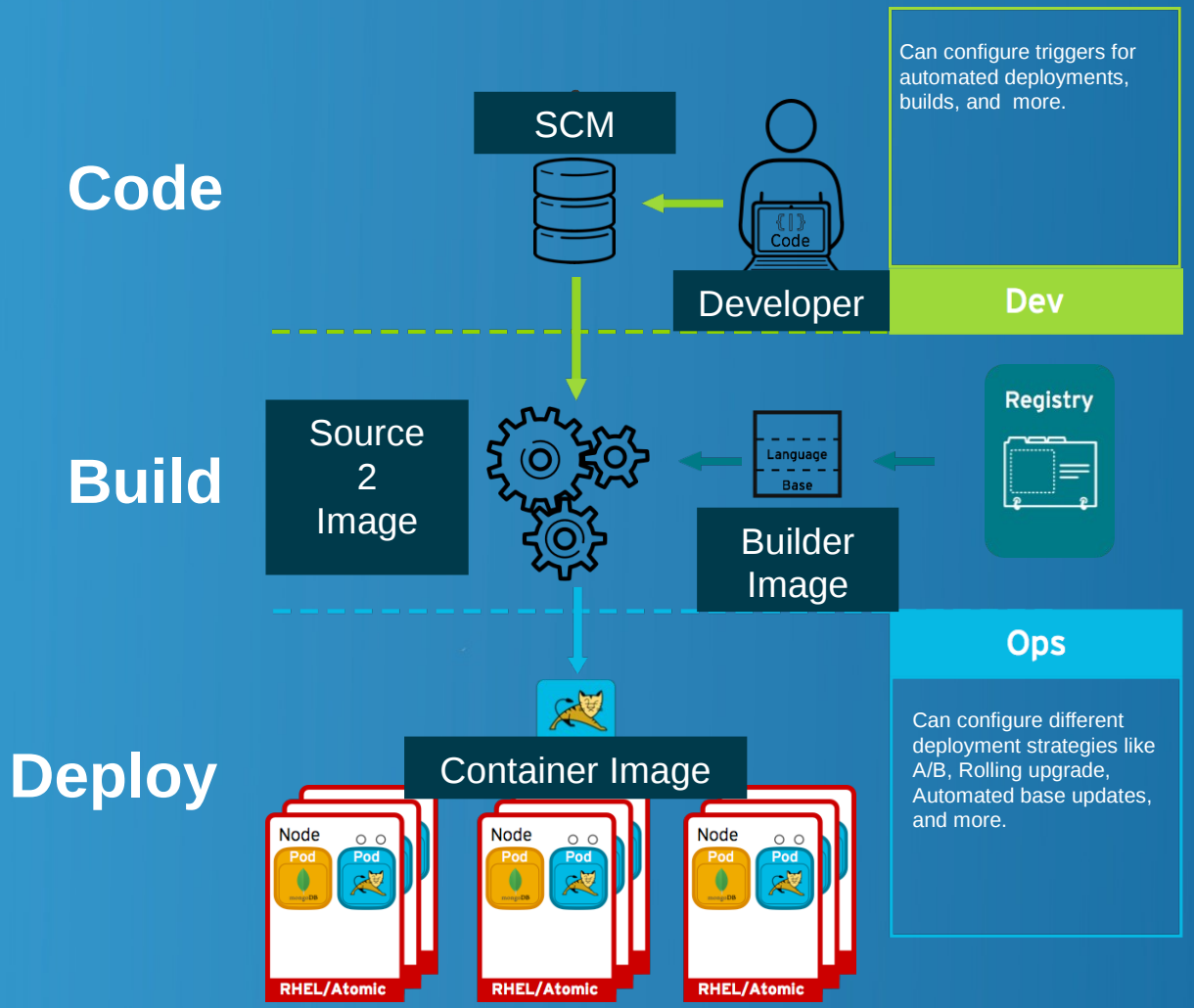
Build & Deploy an Image

Builder Images

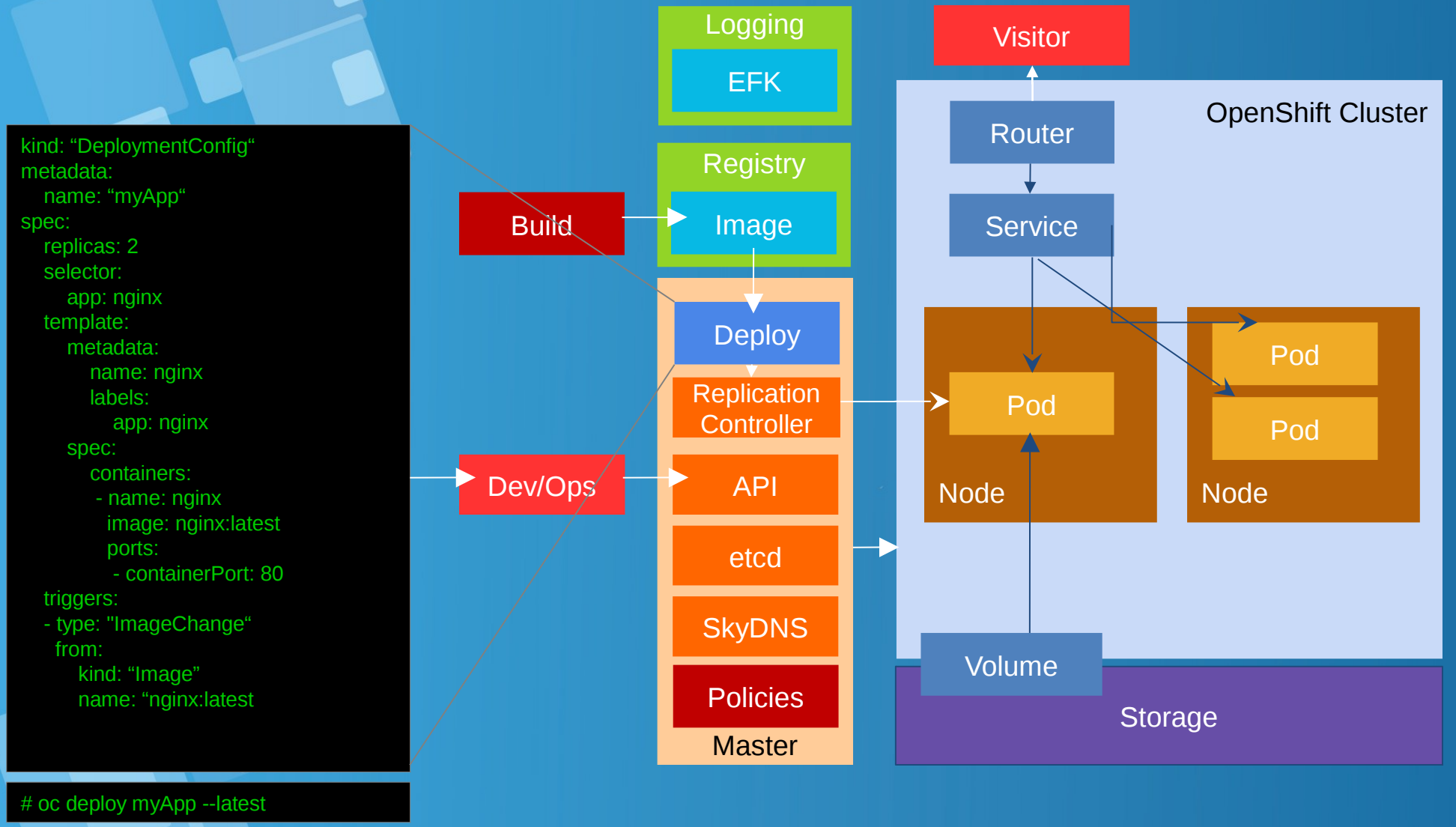
- Jboss-EAP
- PHP
- Python
- Ruby
- Jenkins
- Customer
  - C++ / Go
  - S2I (bash) scripts

Triggers

- Image Change (tagging)
- Code Change (webhook)
- Config Change



OpenShit Build & Deploy Architecture



Pop Quiz

- What is a valid source for building a pod in OpenShift or Kubernetes (Choose three)?

  A) Java, Node.js, PHP, and Ruby source code

  B) RPM packages

  C) Container images in Docker format

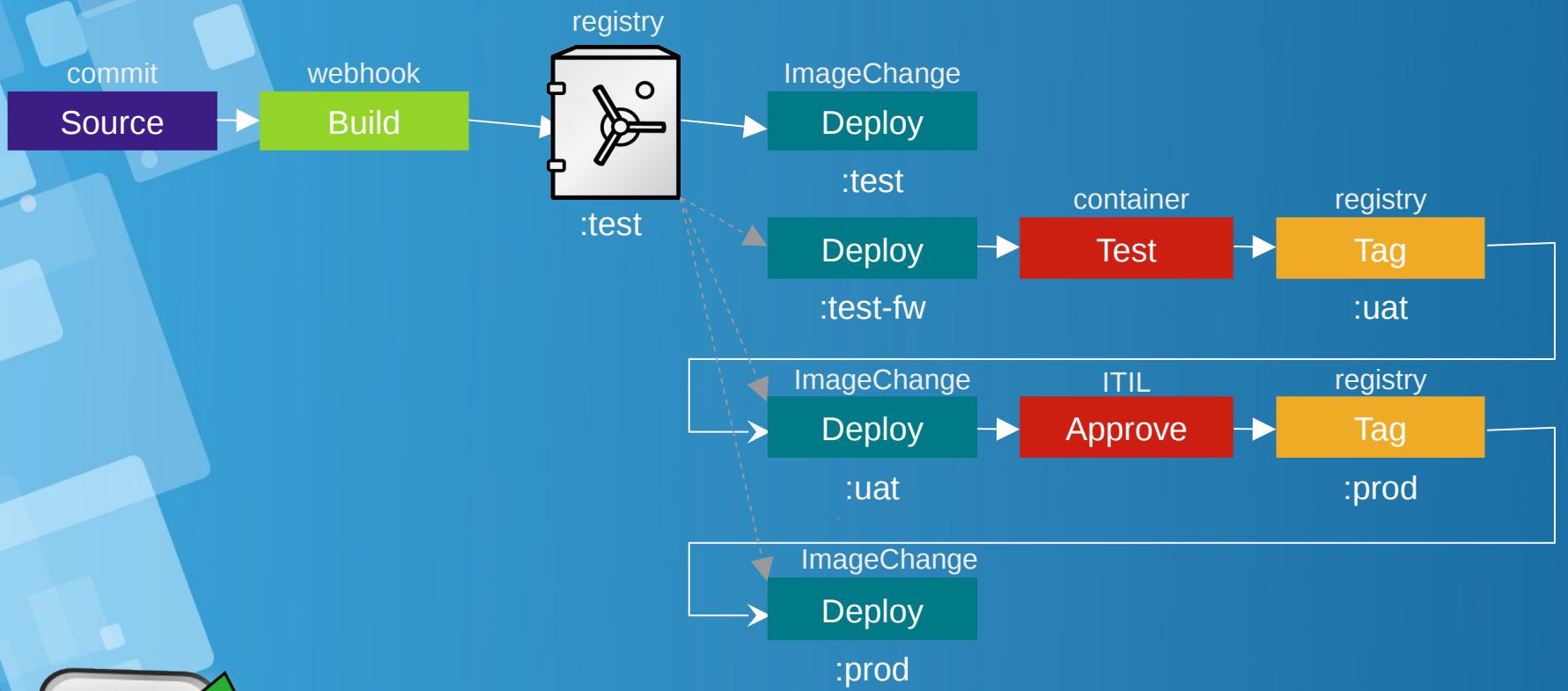
  D) XML files describing the pod metadata

  E) Makefiles describing how to build an application

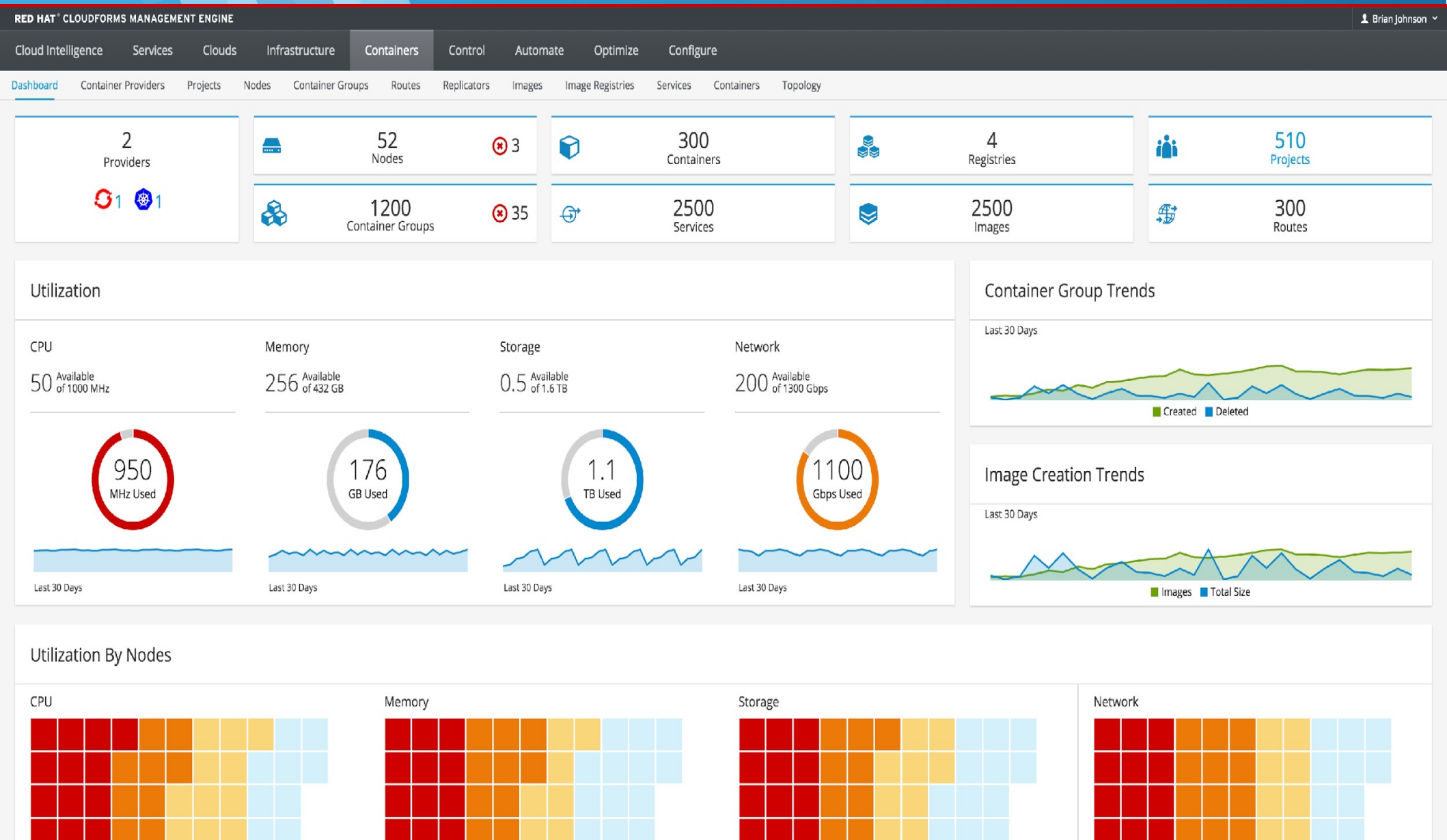
  F) Dockerfiles

Answers the question and Win Merchandize

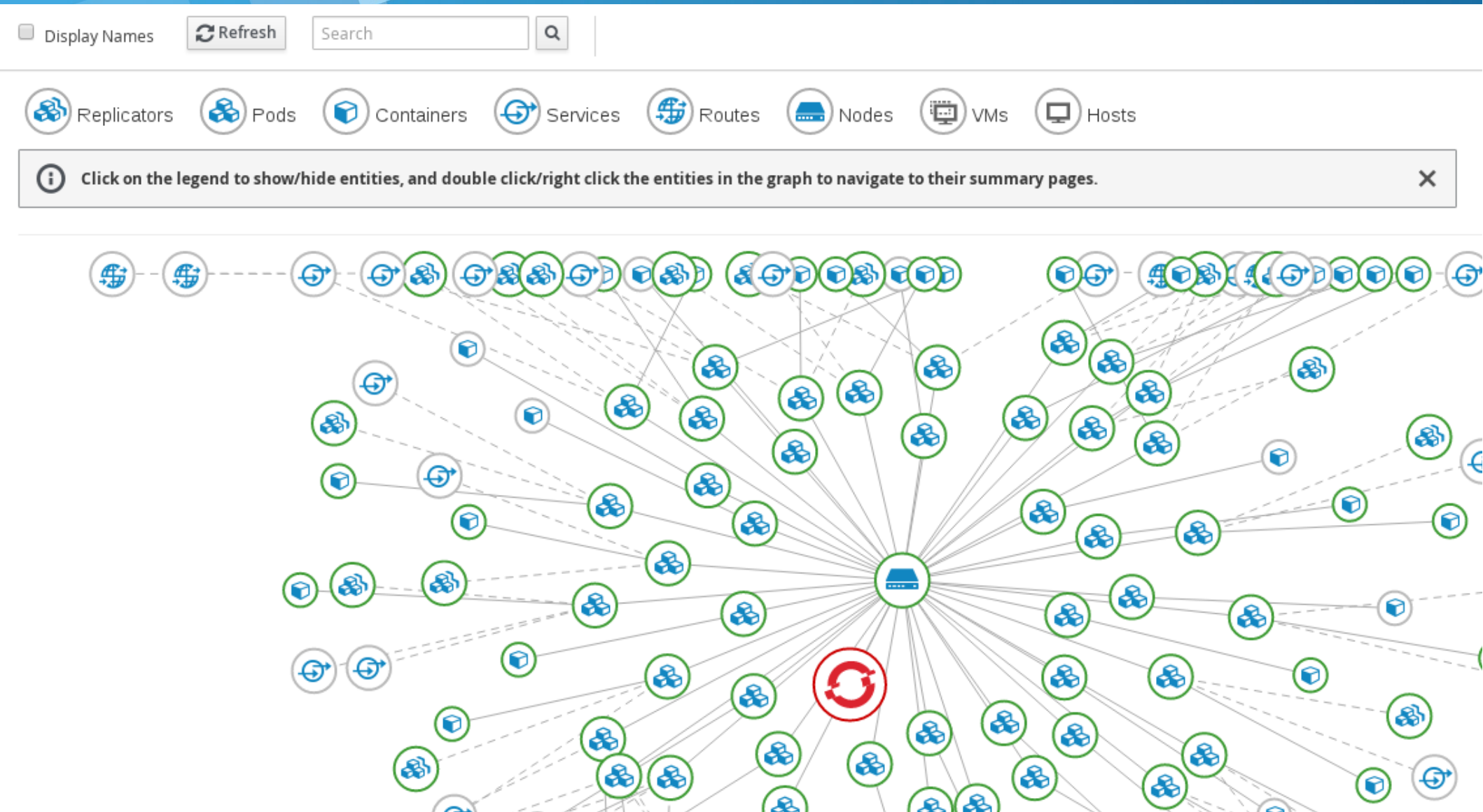
Continous Integration Pipeline Example



Monitoring & Inventory: CloudForm



CloudForm Management



Search

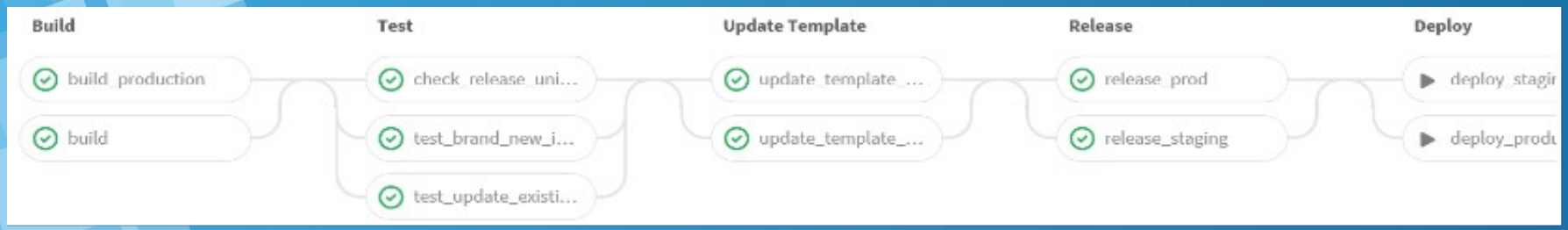
Pods

		Name	Provider	Project Name	Ready	Containers	Phase	Restart Policy	DNS Policy
☐		cakephp-mysql-persistent-2-qwgh7	Openshift-devops-yhs	default	False	1/1	Running	Always	ClusterFirst
☐		cloudforms-1-3b65h	Openshift-devops-yhs	openshift-infra	True	1/1	Running	Always	ClusterFirst
☐		database-1-phqrz	Openshift-devops-yhs	test	True	1/1	Running	Always	ClusterFirst
☐		dc-gitlab-runner-service-1-2xqrm	Openshift-devops-yhs	devops	True	1/1	Running	Always	ClusterFirst
☐		dc-minio-service-1-n0614	Openshift-devops-yhs	devops	True	1/1	Running	Always	ClusterFirst
☐		docker-registry-1-8gz8k	Openshift-devops-yhs	default	True	1/1	Running	Always	ClusterFirst
☐		frontend-1-build	Openshift-devops-yhs	devops	False	0/1	Succeeded	Never	ClusterFirst
☐		frontend-1-vvxx1	Openshift-devops-yhs	test	True	1/1	Running	Always	ClusterFirst
☐		frontend-1-w48hp	Openshift-devops-yhs	test	True	1/1	Running	Always	ClusterFirst

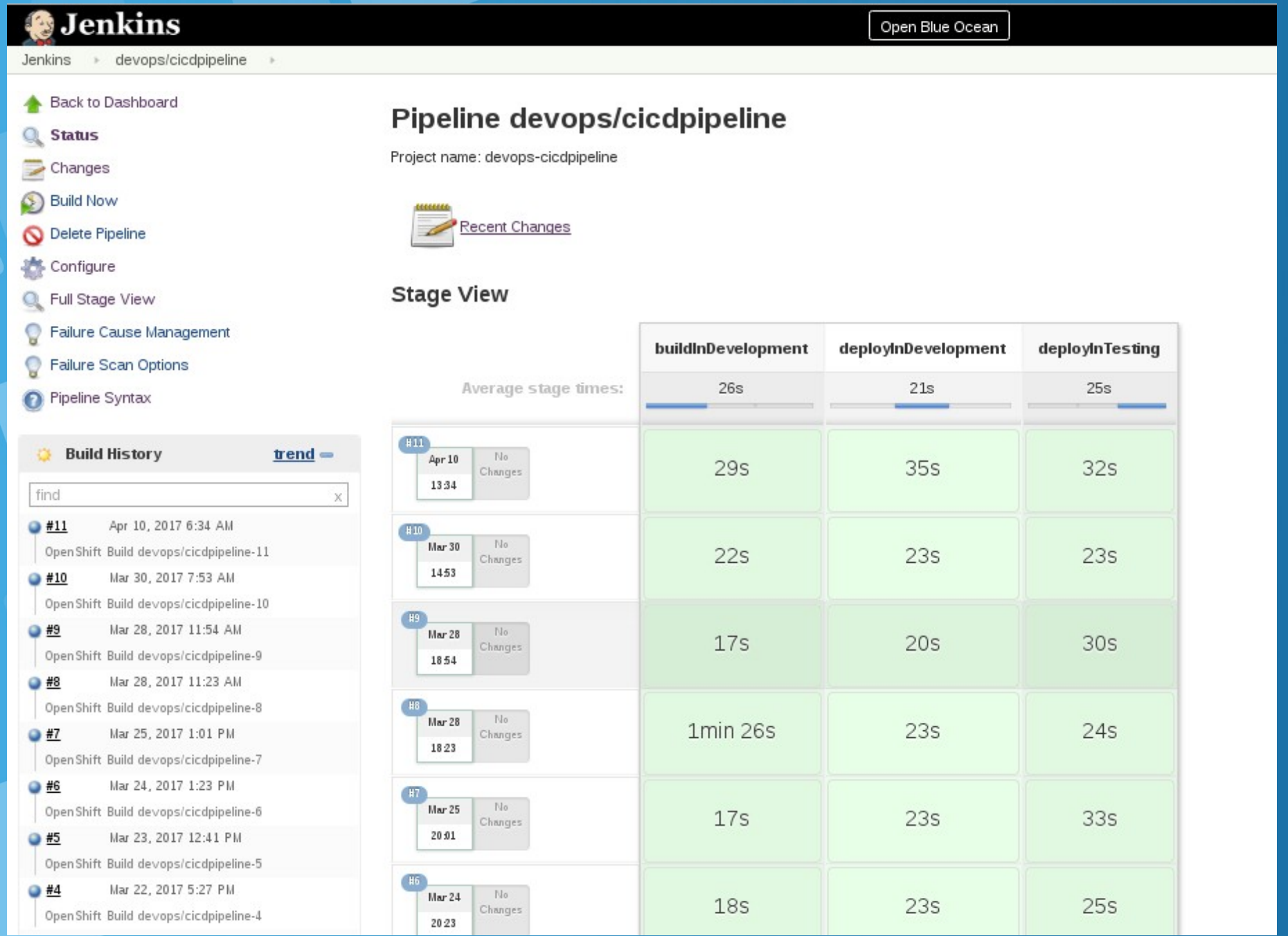
(Check All) Name 20 items 1-20 of 44

Openshift as a tool for developers

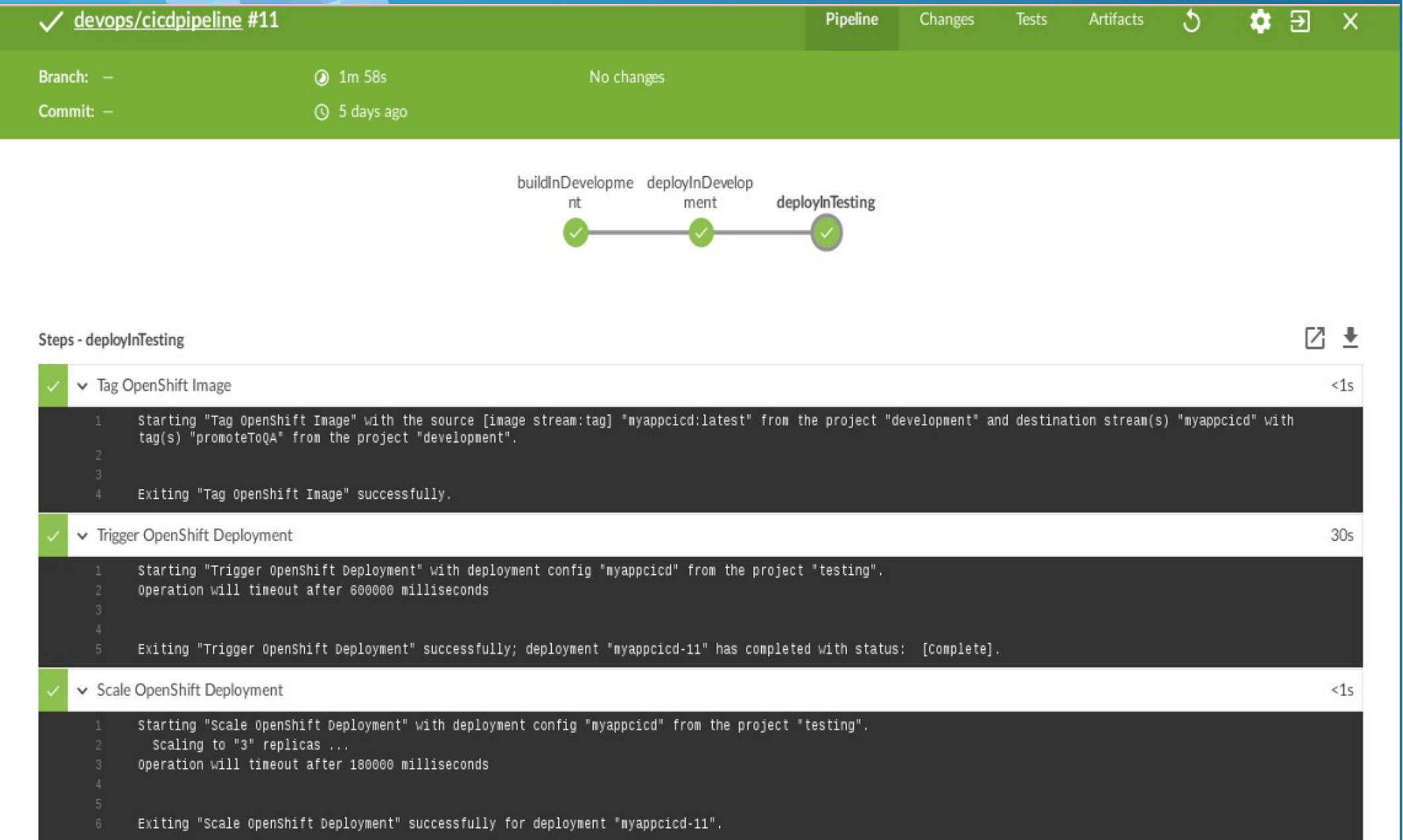
- Facilitate deployment and operation of web applications:
- Getting started with a web application/prototype
- Automate application deployment, rollback changes
- No need to maintain a VM and its OS
- Switch hosting platform (container portability)
- Good integration with code hosting (GitLab)
- CI/CD pipelines (GitLab/Jenkins)
- GitLab Review apps



Openshift: Jenkins CI example



BlueOcean...


devops/cicdpipeline #11
Pipeline
Changes
Tests
Artifacts
Branch: –
Commit: –
1m 58s
5 days ago
No changes
buildInDevelopment
deployInDevelopment
deployInTesting
Steps - deployInTesting
Tag OpenShift Image
<1s
Starting "Tag OpenShift Image" with the source [image stream:tag] "myappcicd:latest" from the project "development" and destination stream(s) "myappcicd" with tag(s) "promoteToQA" from the project "development".
Exiting "Tag OpenShift Image" successfully.
Trigger OpenShift Deployment
30s
Starting "Trigger OpenShift Deployment" with deployment config "myappcicd" from the project "testing".
Operation will timeout after 600000 milliseconds
Exiting "Trigger OpenShift Deployment" successfully; deployment "myappcicd-11" has completed with status: [Complete].
Scale OpenShift Deployment
<1s
Starting "Scale OpenShift Deployment" with deployment config "myappcicd" from the project "testing".
Scaling to "3" replicas ...
Operation will timeout after 180000 milliseconds
Exiting "Scale OpenShift Deployment" successfully for deployment "myappcicd-11".

Q & A

Any Question?

Lets continue to Openshift Lab..

Installing OpenShift Origin

Preparing OS
All-In-One OpenShift
Post-Installation

Installing Openshift Origin

OpenShift: Make sure Docker installed and configured
Skip this step, if all requirment already match

- **Command installing docker from Centos Distribution (not latest)**

```
[root@docker-host ~]# yum install docker
[root@docker-host ~]# sed -i '/OPTIONS=.*/c\OPTIONS="--selinux-enabled --insecure-registry 172.30.0.0/16"' /etc/sysconfig/docker
[root@docker-host ~]# systemctl is-active docker
[root@docker-host ~]# systemctl enable docker
[root@docker-host ~]# systemctl start docker
```

- **When you use Latest version docker, please configure this**

```
[root@docker-host ~]# vim /usr/lib/systemd/system/docker.service
Edit
ExecStart=/usr/bin/dockerd
to
ExecStart=/usr/bin/dockerd --insecure-registry 172.30.0.0/16 --insecure-registry 192.168.1.0/24

[root@docker-host ~]# systemctl daemon-reload ; systemctl restart docker
```

Installing Openshift Origin

- **Setting hostname at /etc/hosts file, for example:**
  ip-address domain-name.tld

```
[root@docker-host ~]# cat /etc/hosts | grep docker
10.7.60.26   docker-host
```

- **Enable Centos Openshift origin repo**

```
[root@docker-host ~]# yum install centos-release-openshift-origin
```

- **Installing Openshift Origin and Origin client**

```
[root@docker-host ~]# yum install wget git net-tools bind-utils iptables-services
bridge-utils bash-completion origin-clients origin
```

Skip all step above, if all requirment already match

Installing Openshift Origin

OpenShift: Setting Up

- Pick One, don't do all four
  - oc cluster up
  - Running in a Docker Container
  - Running from a rpm
  - Installer Installation Steps
- Refer to github.com/isnuryusuf/openshift-install/
  - File: openshift-origin-quickstart.md

Installing OpenShift – oc cluster up

```
[root@docker-host ~]# oc cluster up --public-hostname=192.168.1.178
-- Checking OpenShift client ... OK
-- Checking Docker client ... OK
-- Checking Docker version ... OK
-- Checking for existing OpenShift container ... OK
-- Checking for openshift/origin:v1.5.1 image ...
   Pulling image openshift/origin:v1.5.1
   Pulled 0/3 layers, 3% complete
   Pulled 0/3 layers, 19% complete
   Pulled 0/3 layers, 35% complete
   Pulled 0/3 layers, 52% complete
   Pulled 1/3 layers, 60% complete
   Pulled 1/3 layers, 75% complete
   Pulled 1/3 layers, 90% complete
   Pulled 2/3 layers, 97% complete
   Pulled 3/3 layers, 100% complete
   Extracting
   Image pull complete
-- Checking Docker daemon configuration ... OK
-- Checking for available ports ... OK
-- Checking type of volume mount ...
   Using nsenter mounter for OpenShift volumes
-- Creating host directories ... OK
-- Finding server IP ...
   Using 10.7.60.26 as the server IP
-- Starting OpenShift container ...
   Creating initial OpenShift configuration
   Starting OpenShift using container 'origin'
   Waiting for API server to start listening
   OpenShift server started
-- Adding default OAuthClient redirect URIs ... OK
-- Installing registry ... OK
-- Installing router ... OK
-- Importing image streams ... OK
-- Importing templates ... OK
-- Login to server ... OK
-- Creating initial project "myproject" ... OK
-- Removing temporary directory ... OK
```

Installing OpenShift – firewall

- **If you got error when running oc cluster up**

```
" ... OK
-- Removing temporary directory ... OK-- Checking container networking ... FAIL
   Error: containers cannot communicate with the OpenShift master
   Details:
     The cluster was started. However, the container networking test failed.
   Solution:
     Ensure that access to ports tcp/8443 and udp/53 is allowed on 10.7.60.26.
     You may need to open these ports on your machine's firewall.
   Caused By:
     Error: Docker run error rc=1
     Details:
       Image: openshift/origin:v1.5.1
       Entrypoint: [/bin/bash]
       Command: [-c echo 'Testing connectivity to master API' && curl -s -S -k
https://10.7.60.26:8443 && echo 'Testing connectivity to master DNS server' && for i in
{1..10}; do    if curl -s -S -k https://kubernetes.default.svc.cluster.local; then
exit 0;   fi;    sleep 1; done; exit 1]
       Output:
         Testing connectivity to master API
       Error Output:
         curl: (7) Failed connect to 10.7.60.26:8443; No route to host
```

- **Running following command**

```
[root@docker-host ~]# oc cluster down
[root@docker-host ~]# iptables -I INPUT 1  -p tcp --dport 8443 -j ACCEPT
[root@docker-host ~]# iptables -I INPUT 1  -p udp --dport 53 -j ACCEPT
[root@docker-host ~]# iptables -I INPUT 1  -p tcp --dport 53 -j ACCEPT
[root@docker-host ~]# iptables -I INPUT 1  -p tcp --dport 443 -j ACCEPT
[root@docker-host ~]# iptables -I INPUT 1 -p tcp --dport 80 -j ACCEPT
[root@docker-host ~]# oc cluster up
```

Installing OpenShift Origin

- **Installation is success, take note on Server URL**

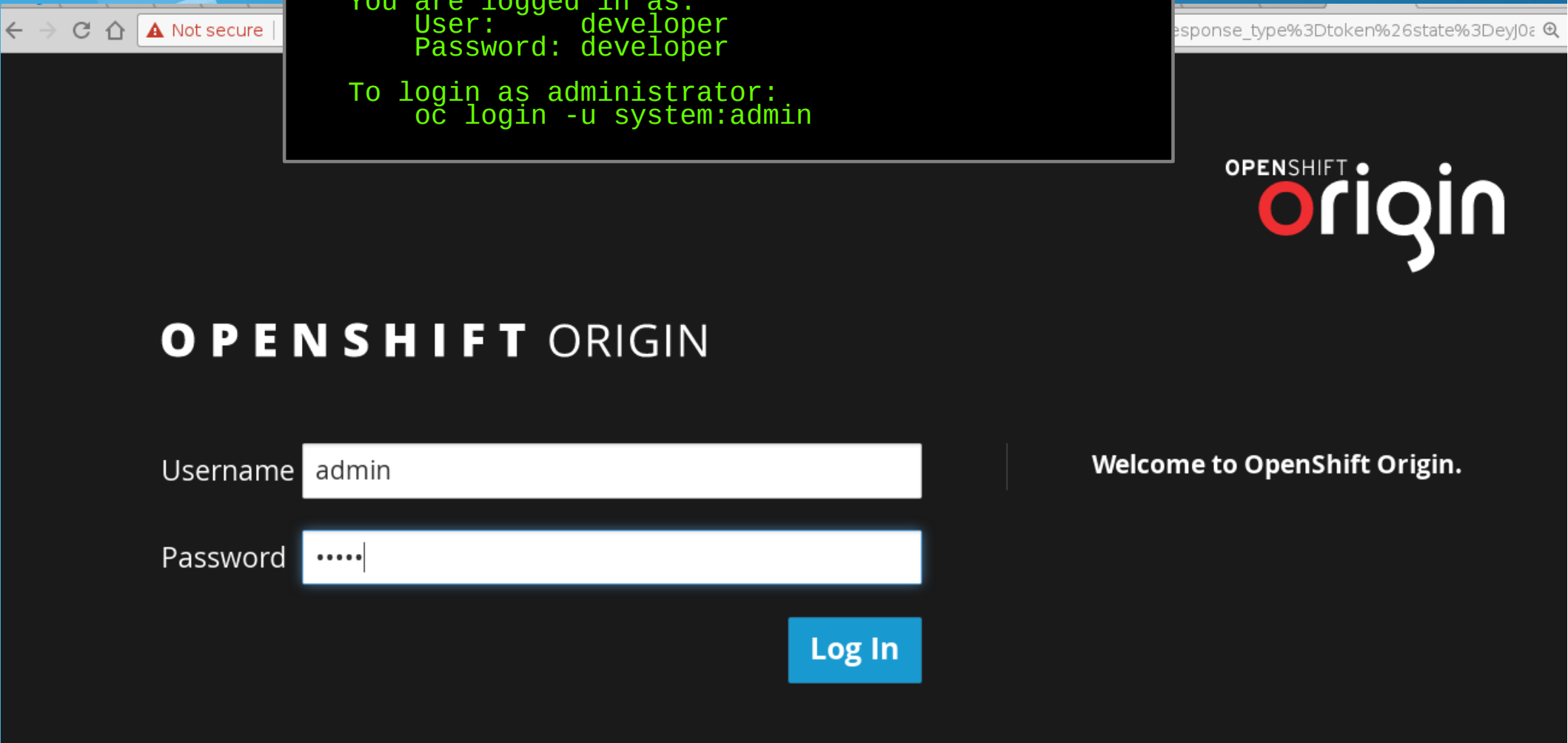
```
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://10.7.60.26:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```

- **Test login from Command line using oc command**

```
[root@docker-host ~]# oc login -u system:admin
Logged into "https://10.7.60.26:8443" as "system:admin" using existing credentials.

You have access to the following projects and can switch between them with 'oc project
<projectname>':

    default
    kube-system
  * myproject
    openshift
    openshift-infra

Using project "myproject".
```

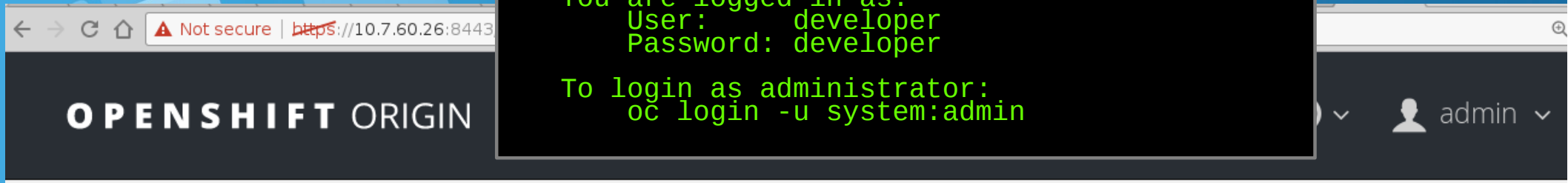
Installing OpenShift Origin

```
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://10.7.60.26:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```



Installing OpenShift Origin

```
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://10.7.60.26:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```



Welcome to OpenShift.

OpenShift helps you quickly develop, host, and scale applications.
Create a project for your application.

New Project

To learn more, visit the OpenShift documentation.

Creating project

OpenShift: Test create new user and project

```
[root@docker-host ~]# oc login
Authentication required for https://10.7.60.26:8443 (openshift)
Username: test
Password: test
Login successful.

You don't have any projects. You can try to create a new project, by running

    oc new-project <projectname>

[root@docker-host ~]# oc new-project test-project
Now using project "test-project" on server "https://10.7.60.26:8443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git

to build a new example application in Ruby.
```

Origin 1st App Deployment

OpenShift: Test create new app deployment

```
[root@docker-host ~]# oc new-app openshift/deployment-example
--> Found Docker image 1c839d8 (23 months old) from Docker Hub for
"openshift/deployment-example"

    * An image stream will be created as "deployment-example:latest" that will track
this image
    * This image will be deployed in deployment config "deployment-example"
    * Port 8080/tcp will be load balanced by service "deployment-example"
      * Other containers can access this service through the hostname "deployment-
example"
    * WARNING: Image "openshift/deployment-example" runs as the 'root' user which may
not be permitted by your cluster administrator

--> Creating resources ...
    imagestream "deployment-example" created
    deploymentconfig "deployment-example" created
    service "deployment-example" created
--> Success
    Run 'oc status' to view your app.
```

OpenShift: Monitor you deployment

```
[root@docker-host ~]# oc status
In project test-project on server https://10.7.60.26:8443

svc/deployment-example - 172.30.96.17:8080
  dc/deployment-example deploys istag/deployment-example:latest
    deployment #1 deployed about a minute ago - 1 pod

View details with 'oc describe <resource>/<name>' or list everything with 'oc get all'.
```

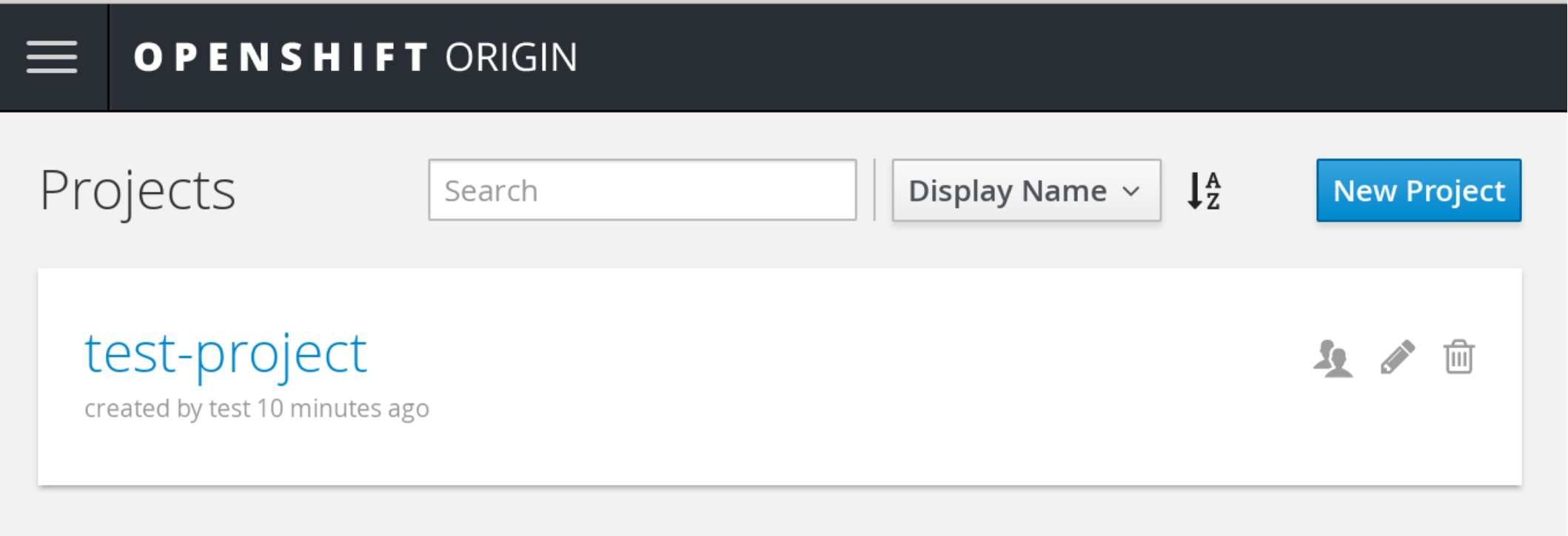
Origin 1st App Deployment

OpenShift: List all resources from 1st deployment

```
[root@docker-host ~]# oc get all
NAME                     DOCKER REPO                                                TAGS
UPDATED
is/deployment-example    172.30.1.1:5000/test-project/deployment-example    latest    2
minutes ago

NAME                     REVISION    DESIRED    CURRENT    TRIGGERED BY
dc/deployment-example    1           1          1          config,image(deployment-
example:latest)

NAME                       DESIRED    CURRENT    READY      AGE
rc/deployment-example-1    1          1          1          2m

NAME                      CLUSTER-IP      EXTERNAL-IP    PORT(S)    AGE
svc/deployment-example    172.30.96.17    <none>         8080/TCP   2m

NAME                             READY      STATUS    RESTARTS   AGE
po/deployment-example-1-jxctr    1/1        Running   0          2m

[root@docker-host ~]# oc get pod
NAME                        READY     STATUS    RESTARTS   AGE
deployment-example-1-jxctr    1/1       Running   0          3m

[root@docker-host ~]# oc get svc
NAME                 CLUSTER-IP      EXTERNAL-IP    PORT(S)    AGE
deployment-example   172.30.96.17    <none>         8080/TCP   3m
```

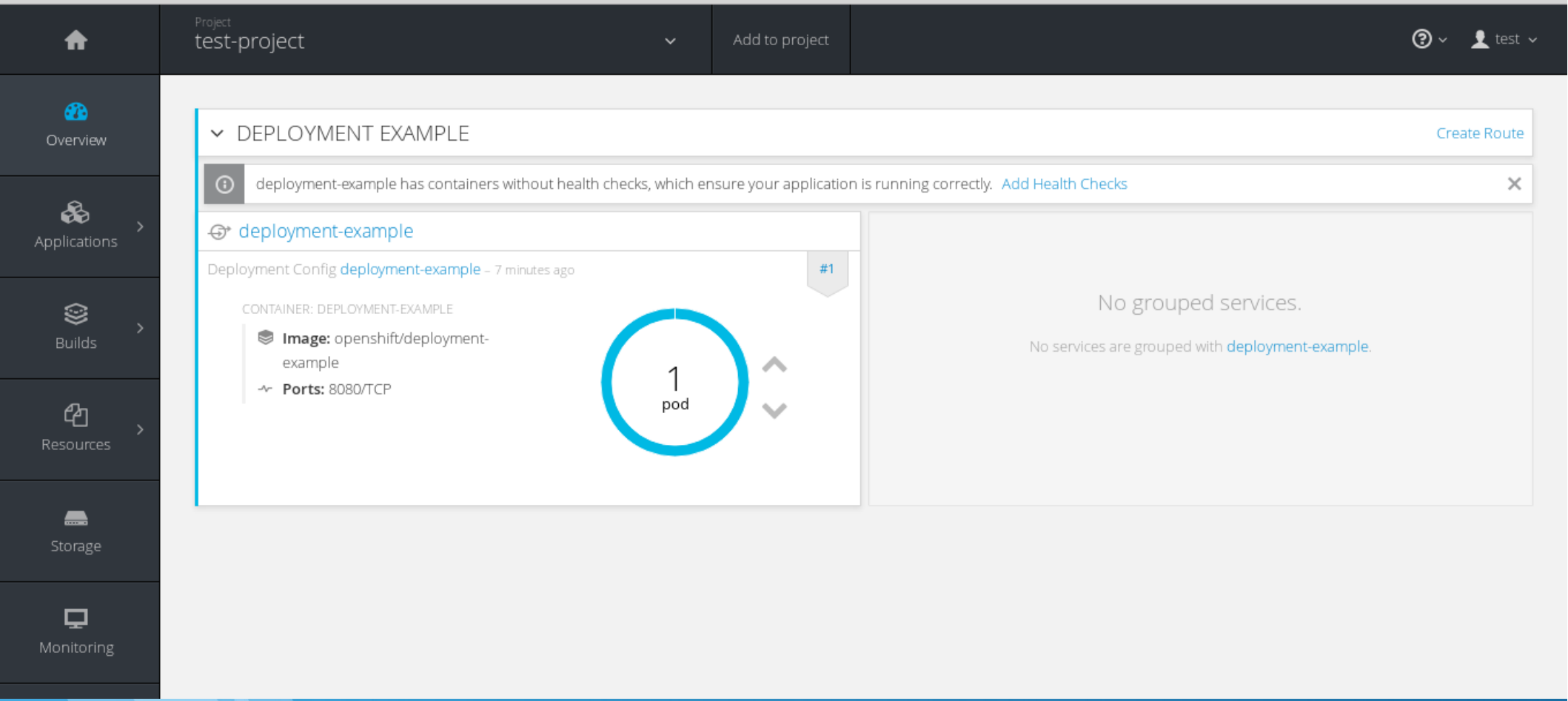
Origin 1st App Deployment

OpenShift: Accessing 1st deployment from Web Gui
Login to Openshift Origin Webconsole using user/pass test/test
Choose and click test-project



Origin 1st App Deployment

OpenShift: Web Console show 1st deployment app



Origin 1st App Deployment

OpenShift: from docker ps output, you can see the new docker container is running

```
[root@docker-host ~]# docker ps | grep deployment-example
be02326a13be        openshift/deployment-example@sha256:ea913--dipotong--ecf421f99
"/deployment v1"         15 minutes ago      Up 15 minutes
k8s_deployment-example.92c6c479_deployment-example-1-jxctr_test-project_d2549bbf-4c6d-
11e7-9946-080027b2e552_6eb2de05
9989834d0c74        openshift/origin-pod:v1.5.1
                                "/pod"                   15 minutes ago      Up 15
minutes                         k8s_POD.bc05fe90_deployment-example-1-jxctr_test-
project_d2549bbf-4c6d-11e7-9946-080027b2e552_55ba483b
```

Origin 1st App Deployment

OpenShift: Test your 1st App using curl and from inside container host

```
[root@docker-host ~]# oc status
In project test-project on server https://10.7.60.26:8443

svc/deployment-example - 172.30.96.17:8080
  dc/deployment-example deploys istag/deployment-example:latest
    deployment #1 deployed 23 minutes ago - 1 pod

View details with 'oc describe <resource>/<name>' or list everything with 'oc get all'.

[root@docker-host ~]# curl 172.30.96.17:8080
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Deployment Demonstration</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
    HTML{height:100%;}
    BODY{font-family:Helvetica,Arial;display:flex;display:-webkit-flex;align-
items:center;justify-content:center;-webkit-align-items:center;-webkit-box-
align:center;-webkit-justify-content:center;height:100%;}
    .box{background:#006e9c;color:white;text-align:center;border-
radius:10px;display:inline-block;}
    H1{font-size:10em;line-height:1.5em;margin:0 0.5em;}
    H2{margin-top:0;}
  </style>
</head>
<body>
<div class="box"><h1>v1</h1><h2></h2></div>
</body>
```

Origin 2nd App Deployment

Lets continue to 2nd App deployment, but for now, we using Web Console

1) Klik “Add to Project” on Web Console
2) Choose “Import YAML / JSON”
3) Run following command:

```
[root@docker-host ~]# oc new-app https://github.com/openshift/ruby-hello-world -o yaml > myapp.yaml

[root@docker-host ~]# cat myapp.yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: ImageStream
  metadata:
    annotations:
      openshift.io/generated-by: OpenShiftNewApp
    creationTimestamp: null
    labels:
      app: ruby-hello-world
    name: ruby-22-centos7

-------DIPOTONG------------
```

Origin 2nd App Deployment

1) Copy and Paste all output from “cat myapp.yaml” to Web Console and click “Create” buton

test-project » Add to Project

Browse Catalog | Deploy Image | Import YAML / JSON

Create or replace resources from their YAML or JSON definitions. If adding a template, you'll have the option to process the template.

Browse...

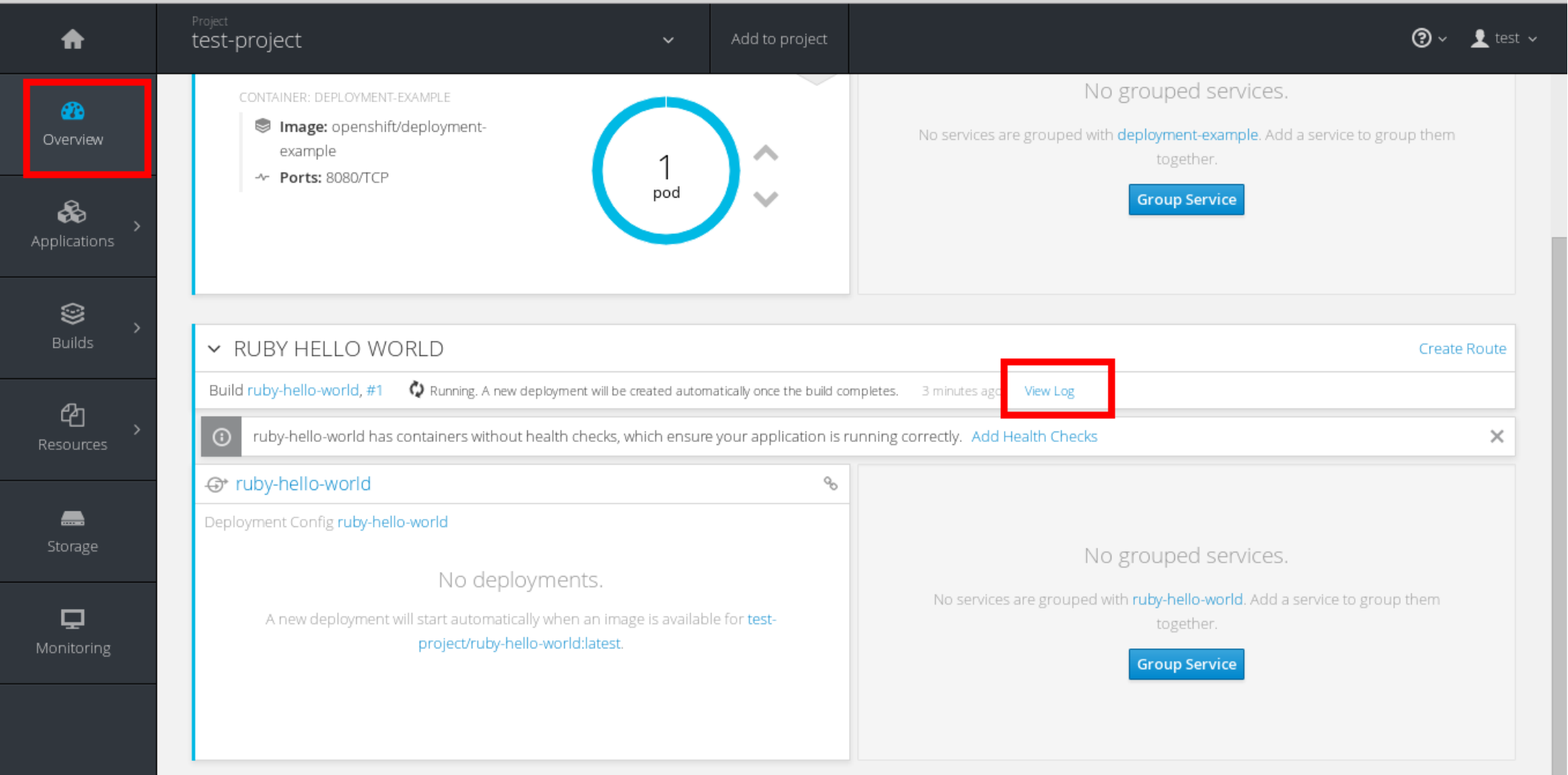
Upload file by dragging & dropping, selecting it, or pasting from the clipboard.

Clear Value

```
apiVersion: v1
items:
- apiVersion: v1
  kind: ImageStream
  metadata:
    annotations:
      openshift.io/generated-by: OpenShiftNewApp
    creationTimestamp: null
    labels:
      app: ruby-hello-world
    name: ruby-22-centos7
  spec:
    tags:
    - annotations:
        openshift.io/imported-from: centos/ruby-22-centos7
      from:
        kind: DockerImage
        name: centos/ruby-22-centos7
      generation: null
      importPolicy: {}
      name: latest
      referencePolicy:
        type: ""
  status:
    dockerImageRepository: ""
```

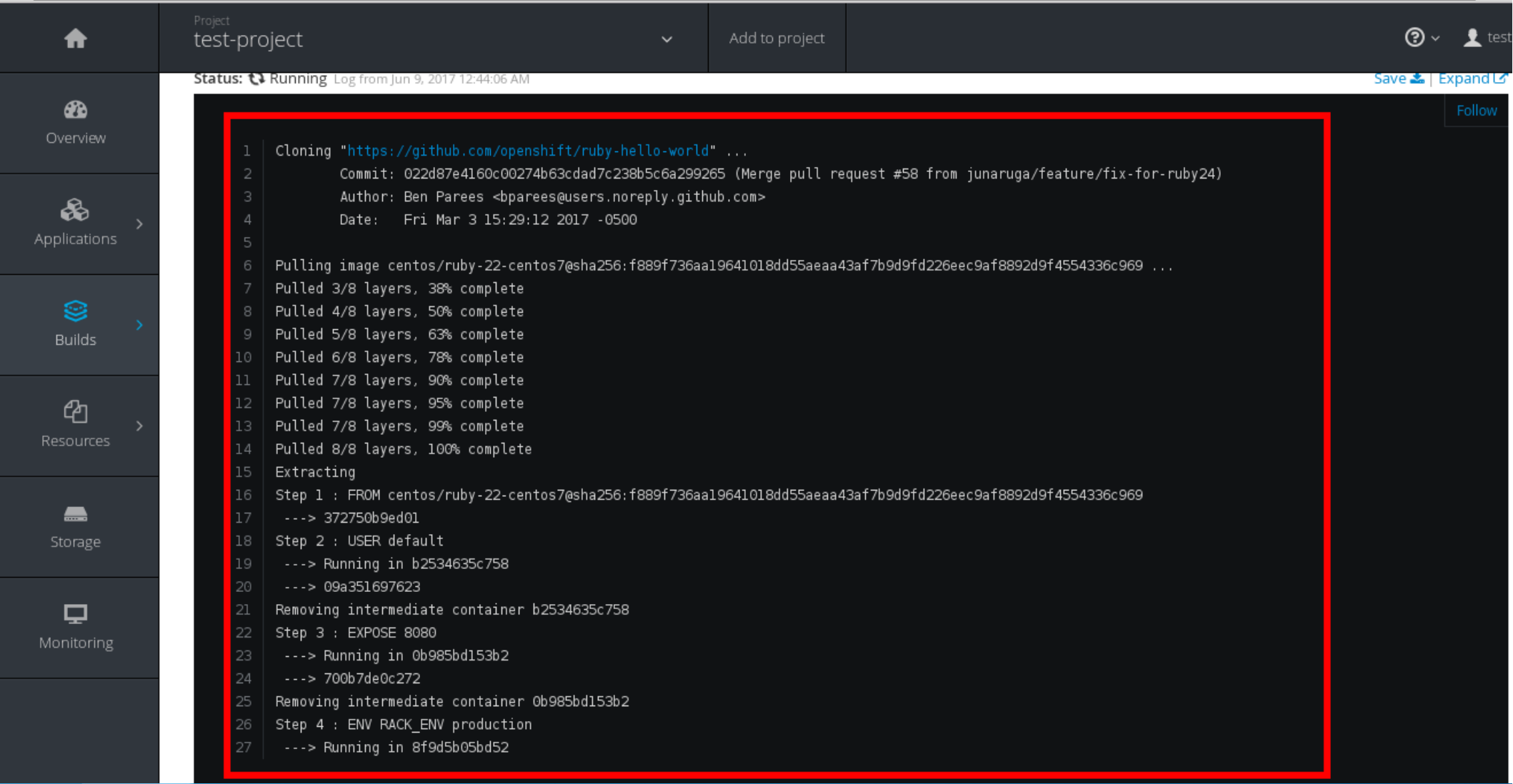
Origin 2nd App Deployment

Check your deployment process from Web Console



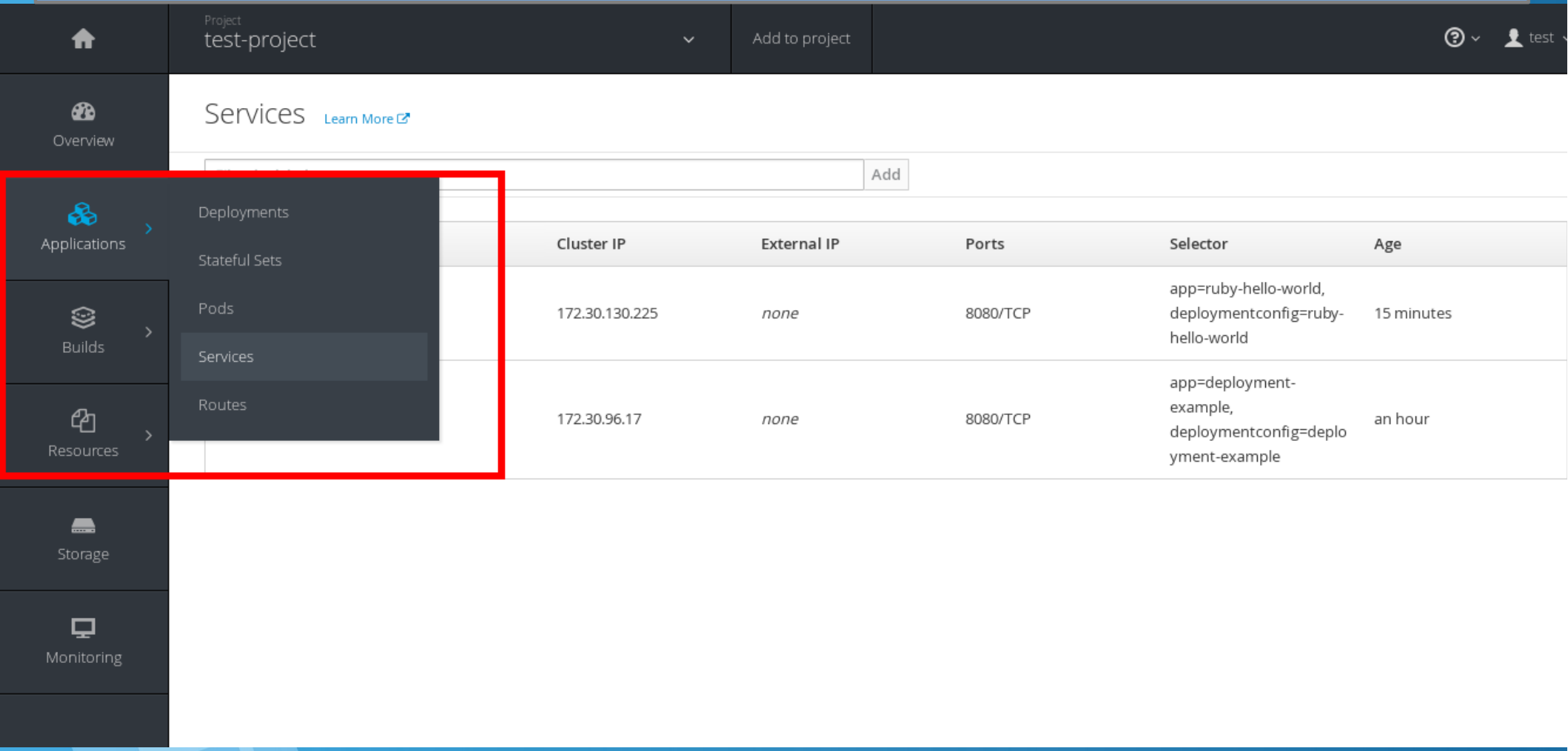
Origin 2nd App Deployment

Check your deployment process from Web Console



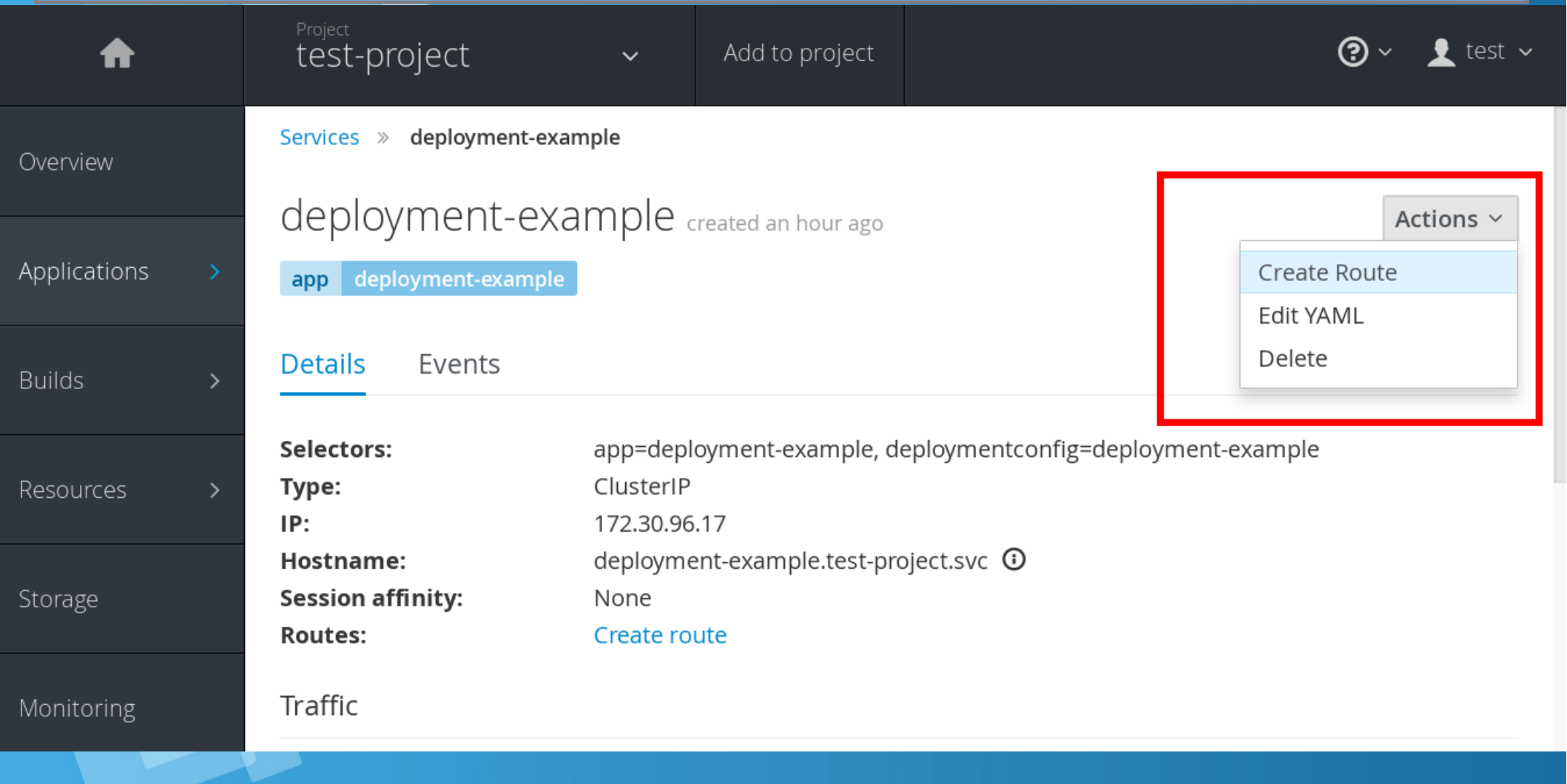
Expose your Apps

Expose your deployment using domain name



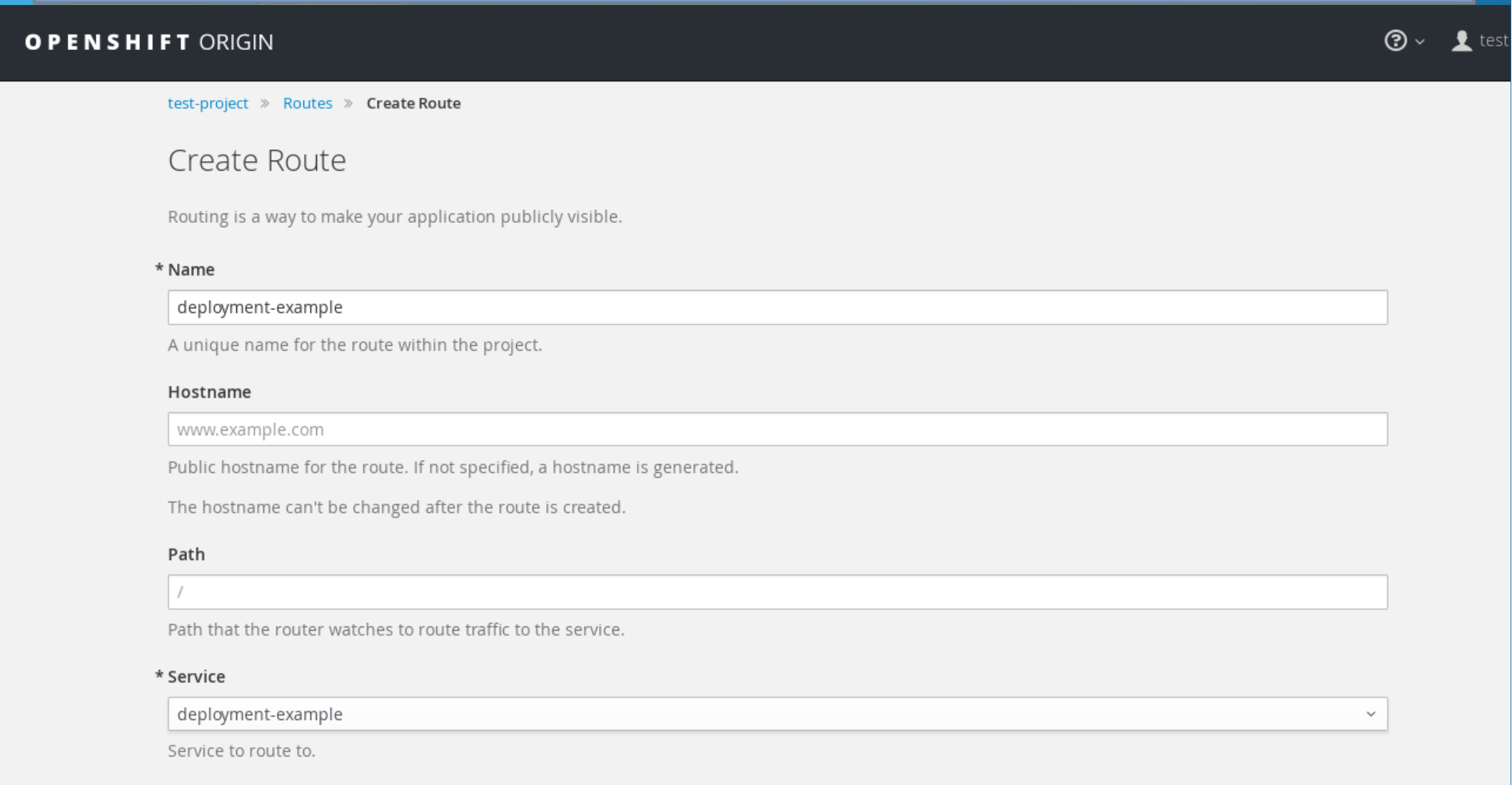
Expose your Apps

Expose your deployment using domain name



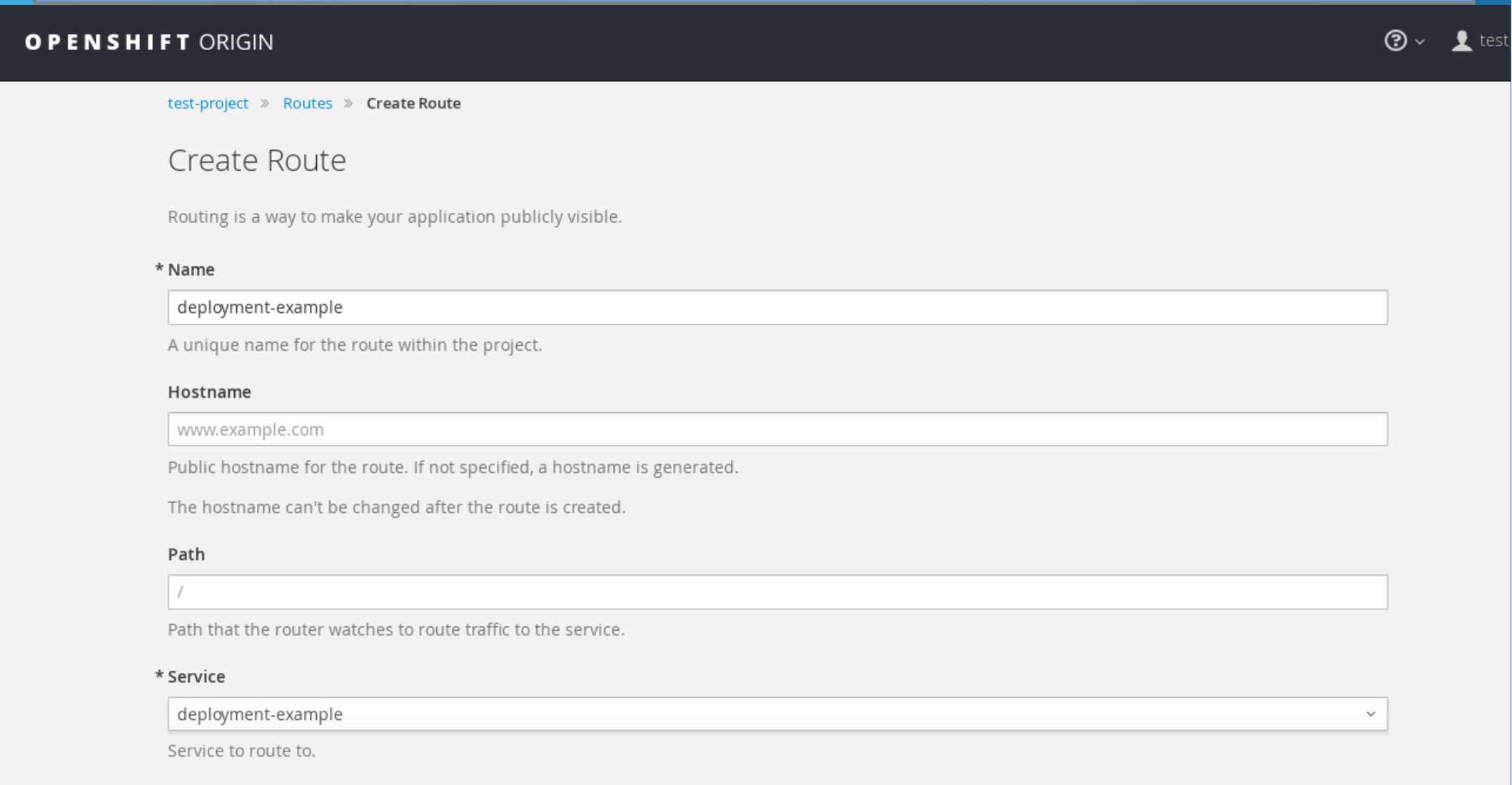
Expose your Apps

Expose your deployment using domain name



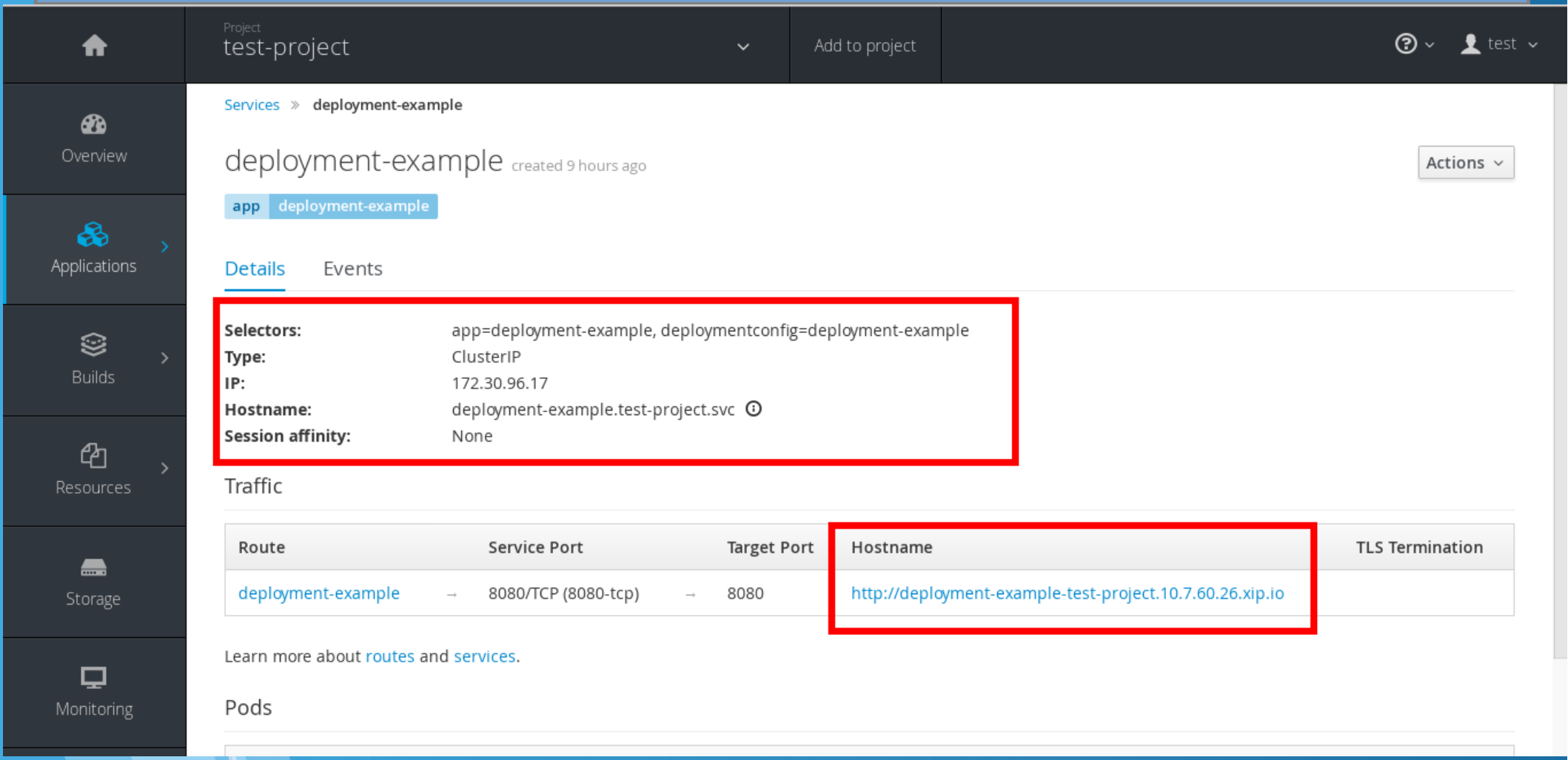
Expose your Apps

Expose your deployment using domain name



Expose your Apps

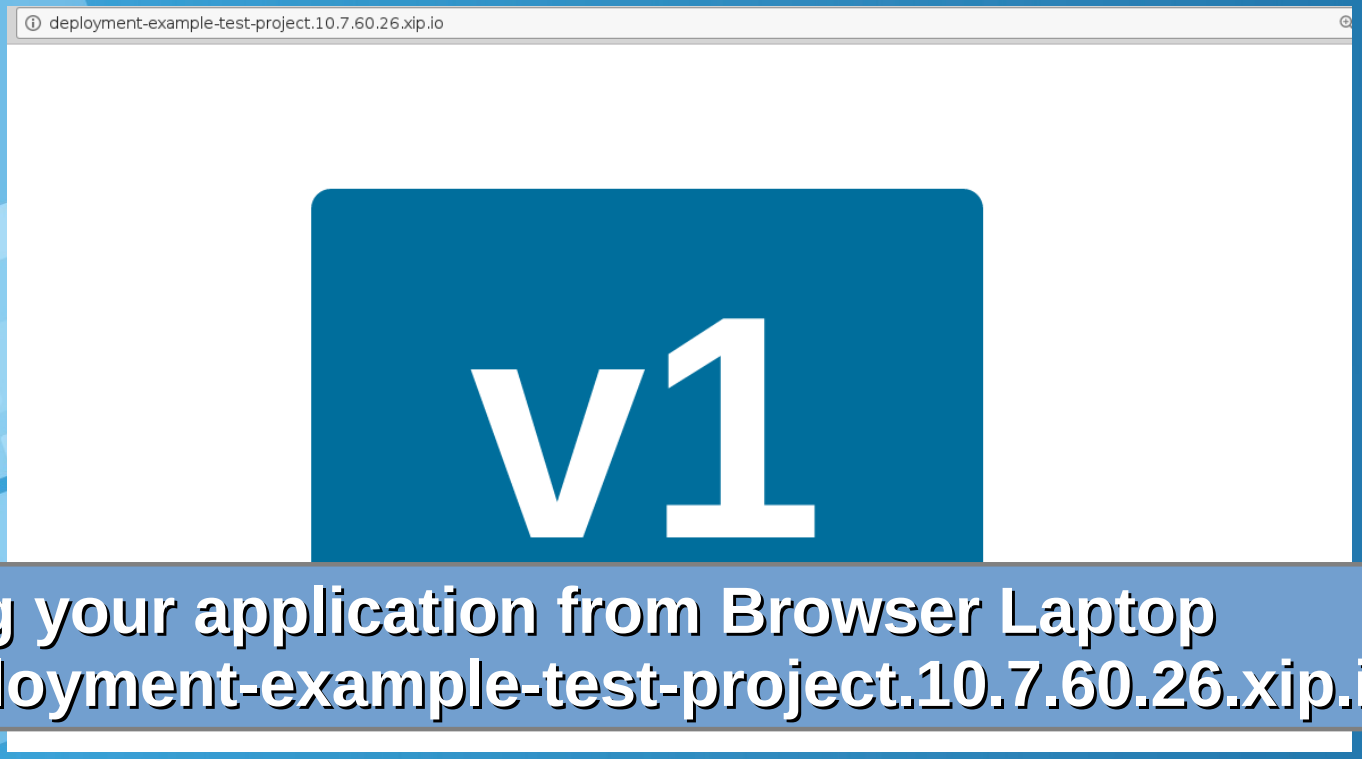
Expose your deployment using domain name



Expose your Apps

Configure static DNS file /etc/hosts on you Laptop and allow port 80, 443 from docker-host firewall

```
root@nobody:/media/yusuf/OS/KSEI# cat /etc/hosts | grep xip.io
10.7.60.26   deployment-example-test-project.10.7.60.26.xip.io

[root@docker-host ~]# iptables -I INPUT 1  -p tcp --dport 80 -j ACCEPT
[root@docker-host ~]# iptables -I INPUT 1  -p tcp --dport 443 -j ACCEPT
```

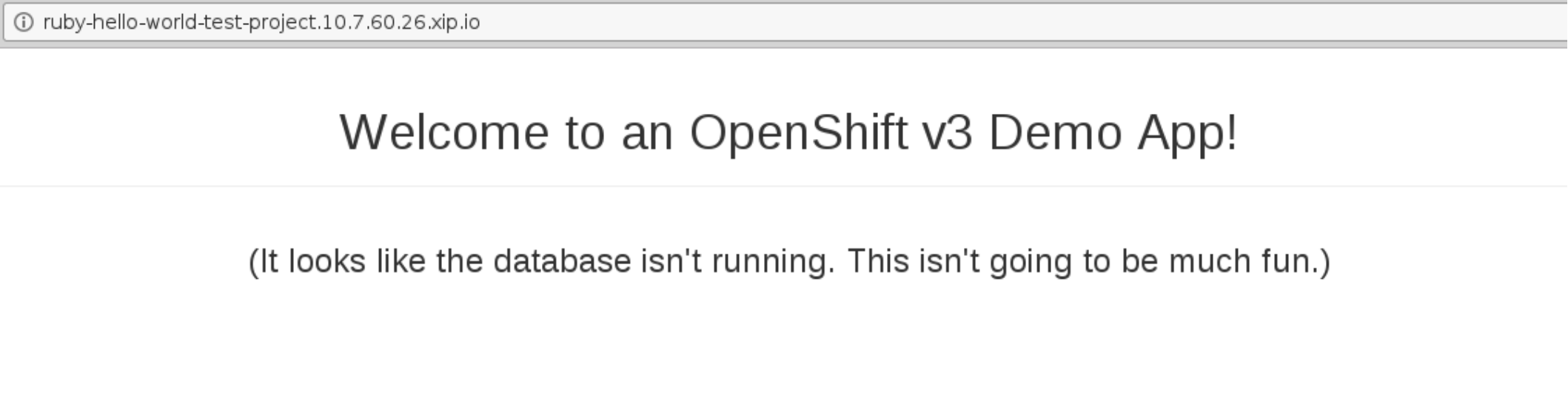


Accessing your application from Browser Laptop http://deployment-example-test-project.10.7.60.26.xip.io/

Expose your Apps

Expose your second Apps using command line

```
[root@docker-host ~]# oc expose service ruby-hello-world
route "ruby-hello-world" exposed

[root@docker-host ~]# oc get routes | grep ruby
ruby-hello-world    ruby-hello-world-test-project.10.7.60.26.xip.io                    ruby-
hello-world     8080-tcp                  None
```



Origin 3rd App Deployment

The next way to build openshift apps is using source-to-image (s2i) here is the step

1) **Clik “Add to project”**
2) **Choose the Laguage, for example: PHP**
3) **Use latest PHP Version: 7.0-latest and click select**
4) **Input you App Name, for example: my-php-app**
5) **On GIT Repo URL, input: https://github.com/OpenShiftDemos/os-sample-php**
6) **And click Create to start deployment**

Origin 3rd App Deployment

OPENSHIFT ORIGIN

test-project » Add to Project » PHP » Next Steps

Application created. Continue to overview.

Manage your app

The web console is convenient, but if you need deeper control you may want to try our command line tools.

Command line tools

Download and install the `oc` command line tool. After that, you can start by logging in, switching to this particular project, and displaying an overview of it, by doing:

```
oc login https://10.7.60.26:8443
oc project test-project
oc status
```

For more information about the command line tools, check the CLI Reference and Basic CLI Operations.

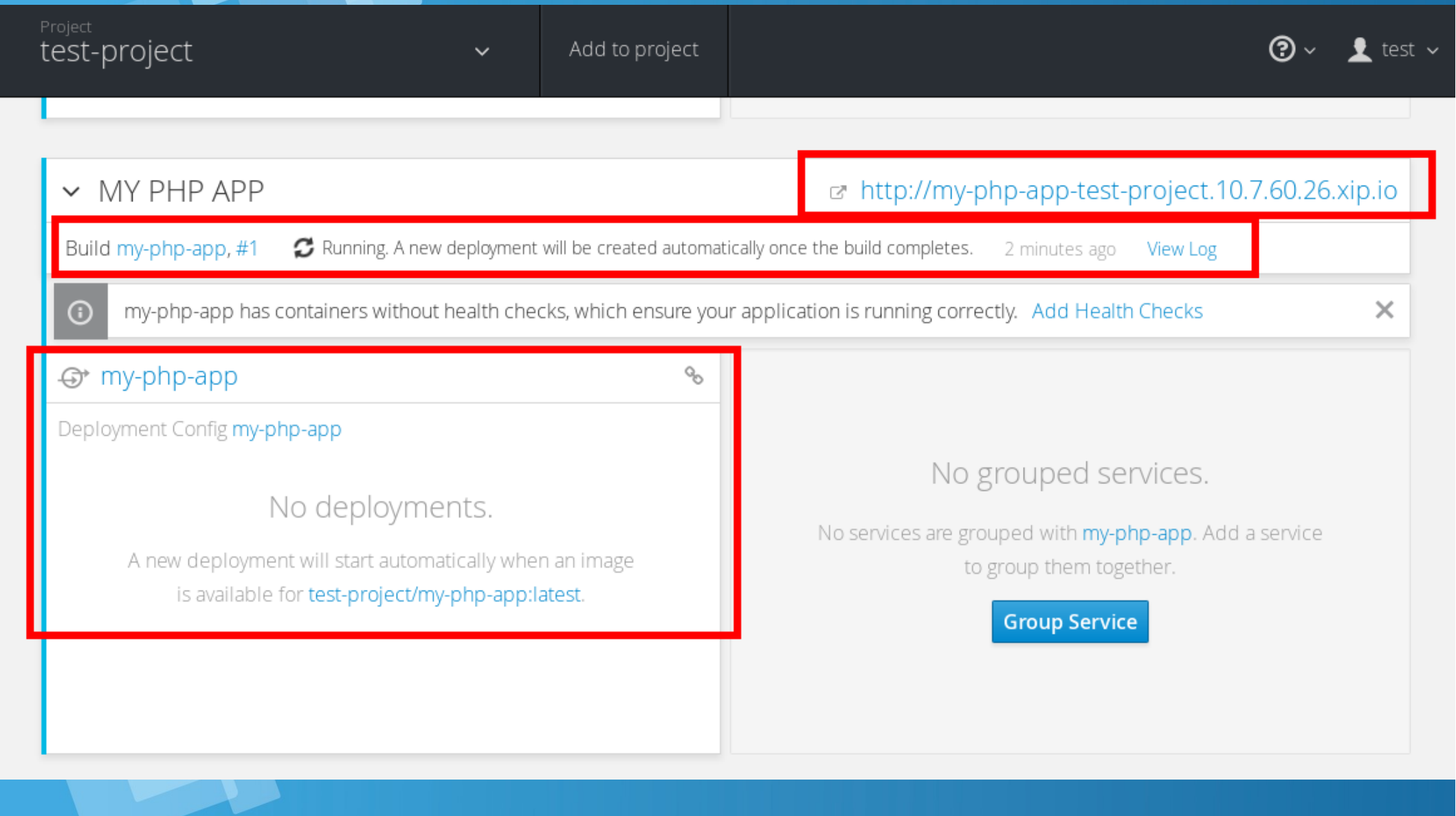
Making code changes

A GitHub webhook trigger has been created for the **my-php-app** build config.

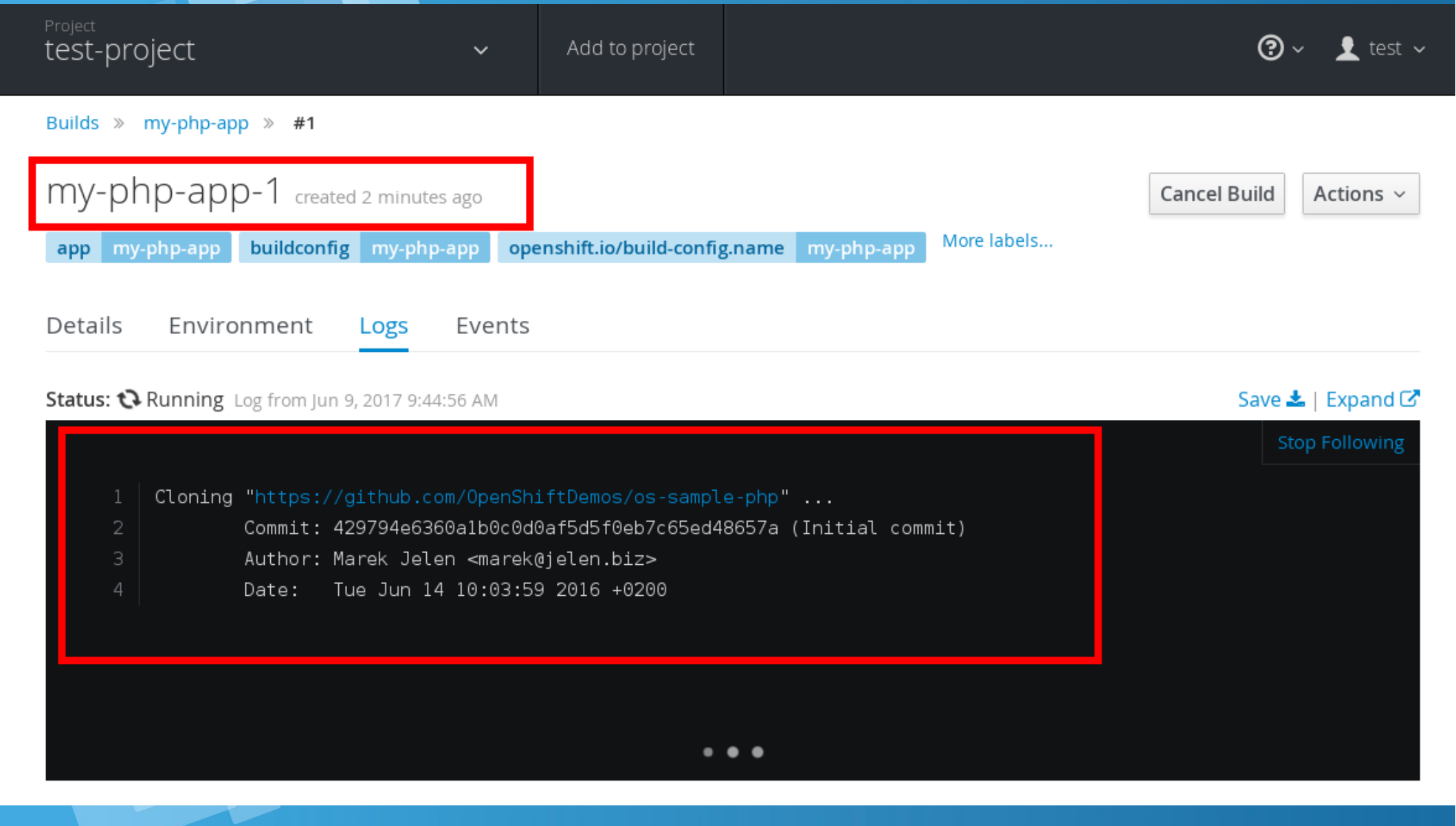
You can now set up the webhook in the GitHub repository settings if you own it, in https://github.com/OpenShiftDemos/os-sample-php/settings/hooks, using the following payload URL:

https://10.7.60.26:8443/oapi/v1/namespace

Origin 3rd App Deployment



Origin 3rd App Deployment



Origin 3rd App Deployment

my-php-app-test-project.10.7.60.26.xip.io

PHP Version 7.0.10

System	Linux my-php-app-1-zm6n6 3.10.0-514.21.1.el7.x86_64 #1 SMP Thu May 25 17:04:51 UTC 2017 x86_64
Build Date	Nov 3 2016 08:06:55
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/opt/rh/rh-php70
Loaded Configuration File	/etc/opt/rh/rh-php70/php.ini
Scan this dir for additional .ini files	/etc/opt/rh/rh-php70/php.d
Additional .ini files parsed	/etc/opt/rh/rh-php70/php.d/10-opcache.ini, /etc/opt/rh/rh-php70/php.d/20-bcmath.ini, /etc/opt/rh/rh-php70/php.d/20-bz2.ini, /etc/opt/rh/rh-php70/php.d/20-calendar.ini, /etc/opt/rh/rh-php70/php.d/20-ctype.ini, /etc/opt/rh/rh-php70/php.d/20-curl.ini, /etc/opt/rh/rh-php70/php.d/20-dom.ini, /etc/opt/rh/rh-php70/php.d/20-exif.ini, /etc/opt/rh/rh-php70/php.d/20-fileinfo.ini, /etc/opt/rh/rh-php70/php.d/20-ftp.ini, /etc/opt/rh/rh-php70/php.d/20-gd.ini, /etc/opt/rh/rh-php70/php.d/20-gettext.ini, /etc/opt/rh/rh-php70/php.d/20-gmp.ini, /etc/opt/rh/rh-php70/php.d/20-iconv.ini, /etc/opt/rh/rh-php70/php.d/20-intl.ini, /etc/opt/rh/rh-php70/php.d/20-json.ini, /etc/opt/rh/rh-php70/php.d/20-ldap.ini, /etc/opt/rh/rh-php70/php.d/20-mbstring.ini, /etc/opt/rh/rh-php70/php.d/20-mysqlnd.ini, /etc/opt/rh/rh-php70/php.d/20-pdo.ini, /etc/opt/rh/rh-php70/php.d/20-pgsql.ini, /etc/opt/rh/rh-php70/php.d/20-phar.ini, /etc/opt/rh/rh-php70/php.d/20-posix.ini, /etc/opt/rh/rh-php70/php.d/20-shmop.ini, /etc/opt/rh/rh-php70/php.d/20-simplexml.ini, /etc/opt/rh/rh-php70/php.d/20-soap.ini, /etc/opt/rh/rh-php70/php.d/20-sockets.ini, /etc/opt/rh/rh-php70/php.d/20-sqlite3.ini, /etc/opt/rh/rh-php70/php.d/20-sysvmsg.ini, /etc/opt/rh/rh-php70/php.d/20-sysvsem.ini, /etc/opt/rh/rh-php70/php.d/20-sysvshm.ini, /etc/opt/rh/rh-php70/php.d/20-tokenizer.ini, /etc/opt/rh/rh-php70/php.d/20-xml.ini, /etc/opt/rh/rh-php70/php.d/20-xmlwriter.ini, /etc/opt/rh/rh-php70/php.d/20-xsl.ini, /etc/opt/rh/rh-php70/php.d/20-zip.ini, /etc/opt/rh/rh-php70/php.d/30-mysqli.ini, /etc/opt/rh/rh-php70/php.d/30-pdo_mysql.ini, /etc/opt/rh/rh-php70/php.d/30-pdo_pgsql.ini, /etc/opt/rh/rh-php70/php.d/30-pdo_sqlite.ini, /etc/opt/rh/rh-php70/php.d/30-wddx.ini, /etc/opt/rh/rh-php70/php.d/30-xmlreader.ini

https://blog.shameerc.com/2016/08/my-docker-setup-ubuntu-php7-fpm-nginx-and-mariadb

OpenShift Others Lab

- Image Stream Template
- Pipeline for CI/CD

Install the default image-streams

Enabling Image stream using ansible

```
[root@docker-host]# mkdir /SOURCE ; cd /SOURCE

[root@docker-host SOURCE]# git clone https://github.com/openshift/openshift-ansible.git
Cloning into 'openshift-ansible'...
remote: Counting objects: 53839, done.
remote: Compressing objects: 100% (47/47), done.
remote: Total 53839 (delta 26), reused 43 (delta 11), pack-reused 53775
Receiving objects: 100% (53839/53839), 14.12 MiB | 930.00 KiB/s, done.
Resolving deltas: 100% (32741/32741), done.

[root@docker-host SOURCE]# cd openshift-
ansible/roles/openshift_examples/files/examples/latest/

[root@docker-host latest]# oc login -u system:admin -n default

[root@docker-host latest]# oadm policy add-cluster-role-to-user cluster-admin admin

[root@docker-host latest]# for f in image-streams/image-streams-centos7.json; do cat $f
| oc create -n openshift -f -; done

[root@docker-host latest]# for f in db-templates/*.json; do cat $f | oc create -n
openshift -f -; done

[root@docker-host latest]# for f in quickstart-templates/*.json; do cat $f | oc create
-n openshift -f -; done
```

Install Jenkins Persistent

Install Jenkins using Image stream from ansible

1) **Click “Add to project”**
2) **On Browse catalog, search for jenkins and select “Jenkins Persistent”**
3) **Leave everything default and click “Create”**

Ref:
https://github.com/openshift/origin/blob/master/examples/jenkins/README.md

Install Jenkins Persistent

OPENSHIFT ORIGIN

test-project » Add to Project » Catalog » Jenkins (Persistent)

Jenkins (Persistent)

Images

jenkins:latest from parameter Jenkins ImageStreamTag

Parameters

Jenkins Service Name

jenkins

The name of the OpenShift Service exposed for the Jenkins container.

Jenkins JNLP Service Name

jenkins-jnlp

The name of the service used for master/slave communication.

Enable OAuth in Jenkins

true

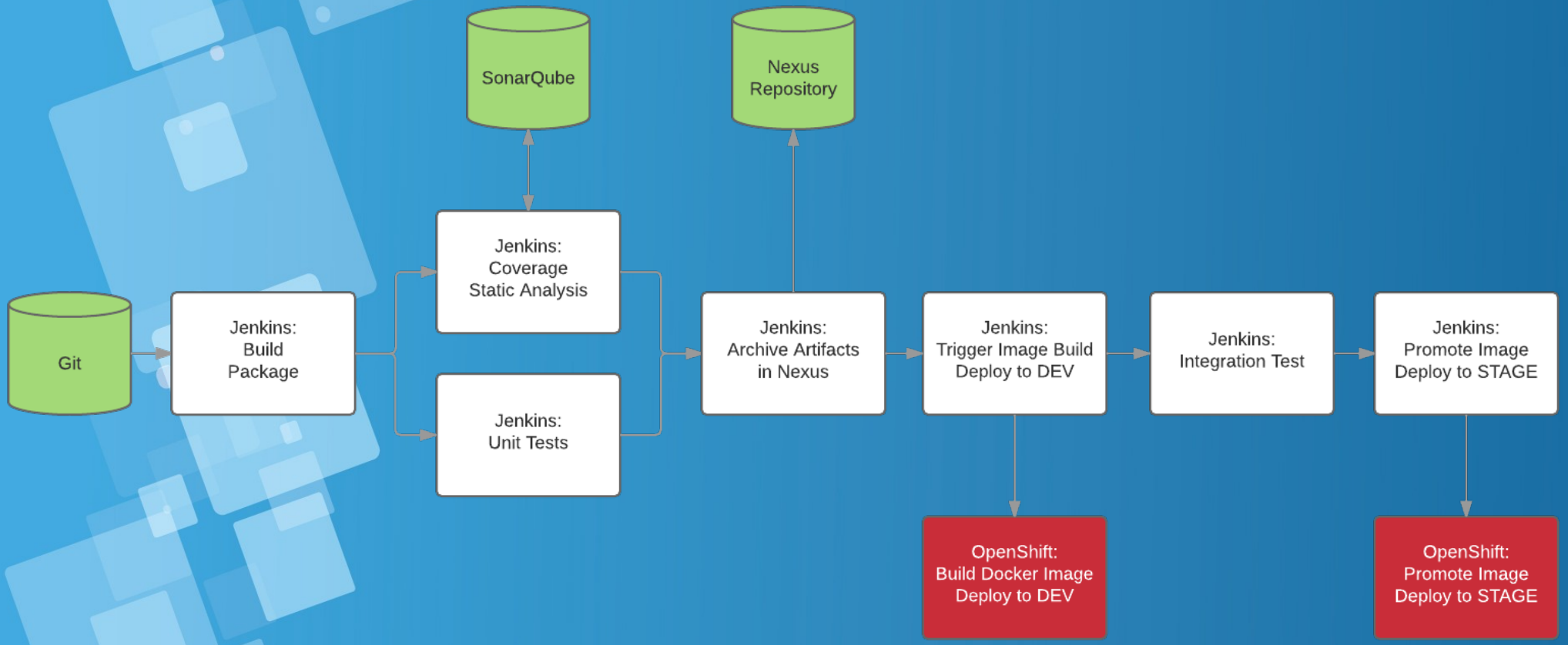
CI/CD Demo – OpenShift Origin

This repository includes the infrastructure and pipeline definition for continuous delivery using Jenkins, Nexus and SonarQube on OpenShift. On every pipeline execution, the code goes through the following steps:

1) Code is cloned from Gogs, built, tested and analyzed for bugs and bad patterns
2) The WAR artifact is pushed to Nexus Repository manager
3) A Docker image (tasks:latest) is built based on the Tasks application WAR artifact deployed on JBoss EAP 6
4) The Tasks Docker image is deployed in a fresh new container in DEV project
5) If tests successful, the DEV image is tagged with the application version (tasks:7.x) in the STAGE project
6) The staged image is deployed in a fresh new container in the STAGE project

CI/CD Demo – OpenShift Origin

The following diagram shows the steps included in the deployment pipeline:



CI/CD Demo – OpenShift Origin

Follow these instructions in order to create a local OpenShift cluster. Otherwise using your current OpenShift cluster, create the following projects for CI/CD components, Dev and Stage environments:

1) **# oc new-project dev --display-name="Tasks - Dev"**
2) **# oc new-project stage --display-name="Tasks - Stage"**
3) **# oc new-project cicd --display-name="CI/CD"**

CI/CD Demo – OpenShift Origin

Follow these instructions in order to create a local OpenShift cluster. Otherwise using your current OpenShift cluster, create the following projects for CI/CD components, Dev and Stage environments:

1) **# oc new-project dev --display-name="Tasks - Dev"**
2) **# oc new-project stage --display-name="Tasks - Stage"**
3) **# oc new-project cicd --display-name="CI/CD"**

Jenkins needs to access OpenShift API to discover slave images as well accessing container images. Grant Jenkins service account enough privileges to invoke OpenShift API for the created projects:

1) **# oc policy add-role-to-user edit system:serviceaccount:cicd:jenkins -n dev**
2) **# oc policy add-role-to-user edit system:serviceaccount:cicd:jenkins -n stage**

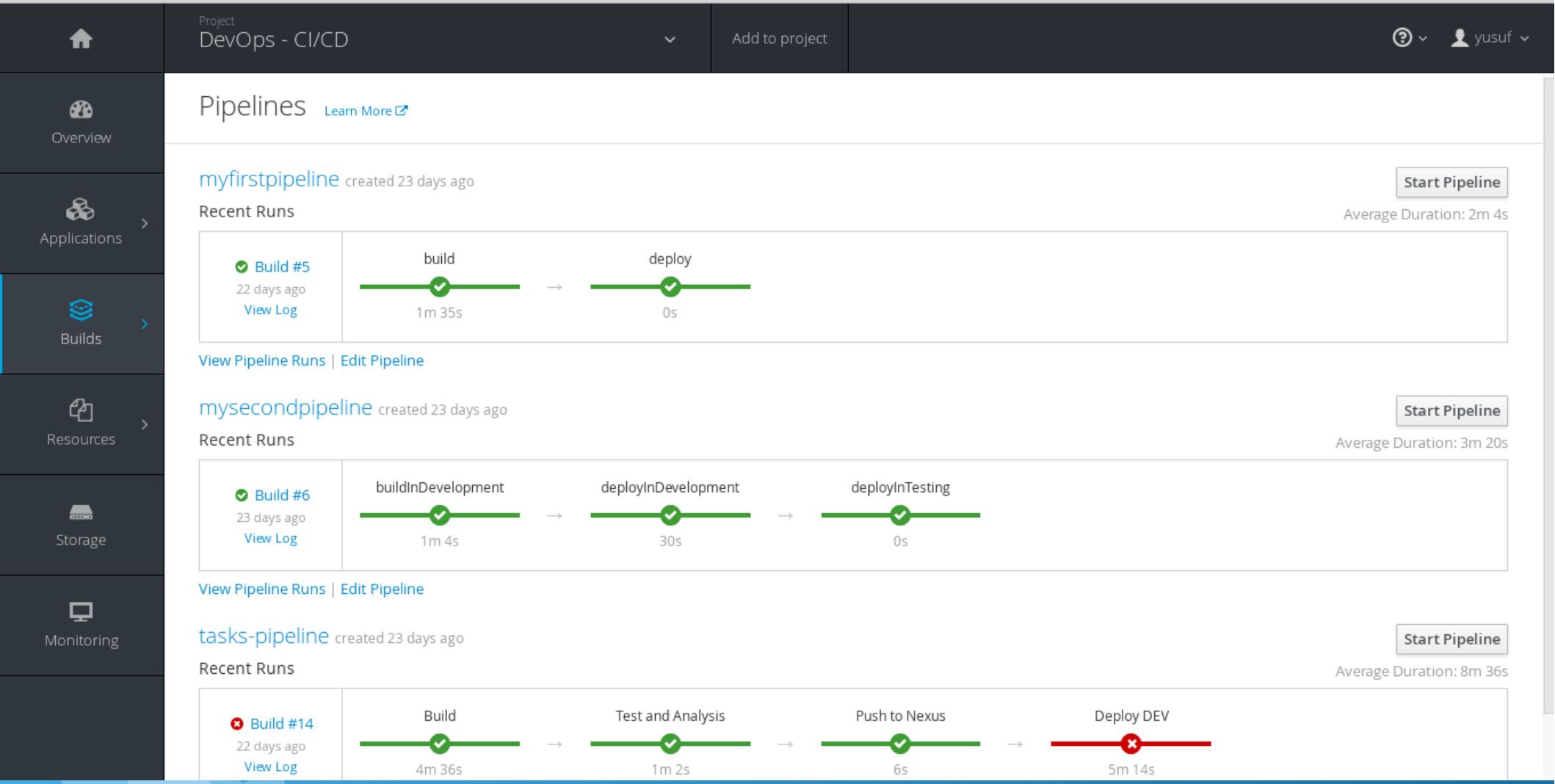
CI/CD Demo – OpenShift Origin

1) Create the CI/CD components based on the provided template
   oc process -f cicd-template.yaml | oc create -f -
2) To use custom project names, change cicd, dev and stage in the above commands to your own names and use the following to create the demo:
   oc process -f cicd-template.yaml -v DEV_PROJECT=dev-project-name -v STAGE_PROJECT=stage-project-name | oc create -f - -n cicd-project-name

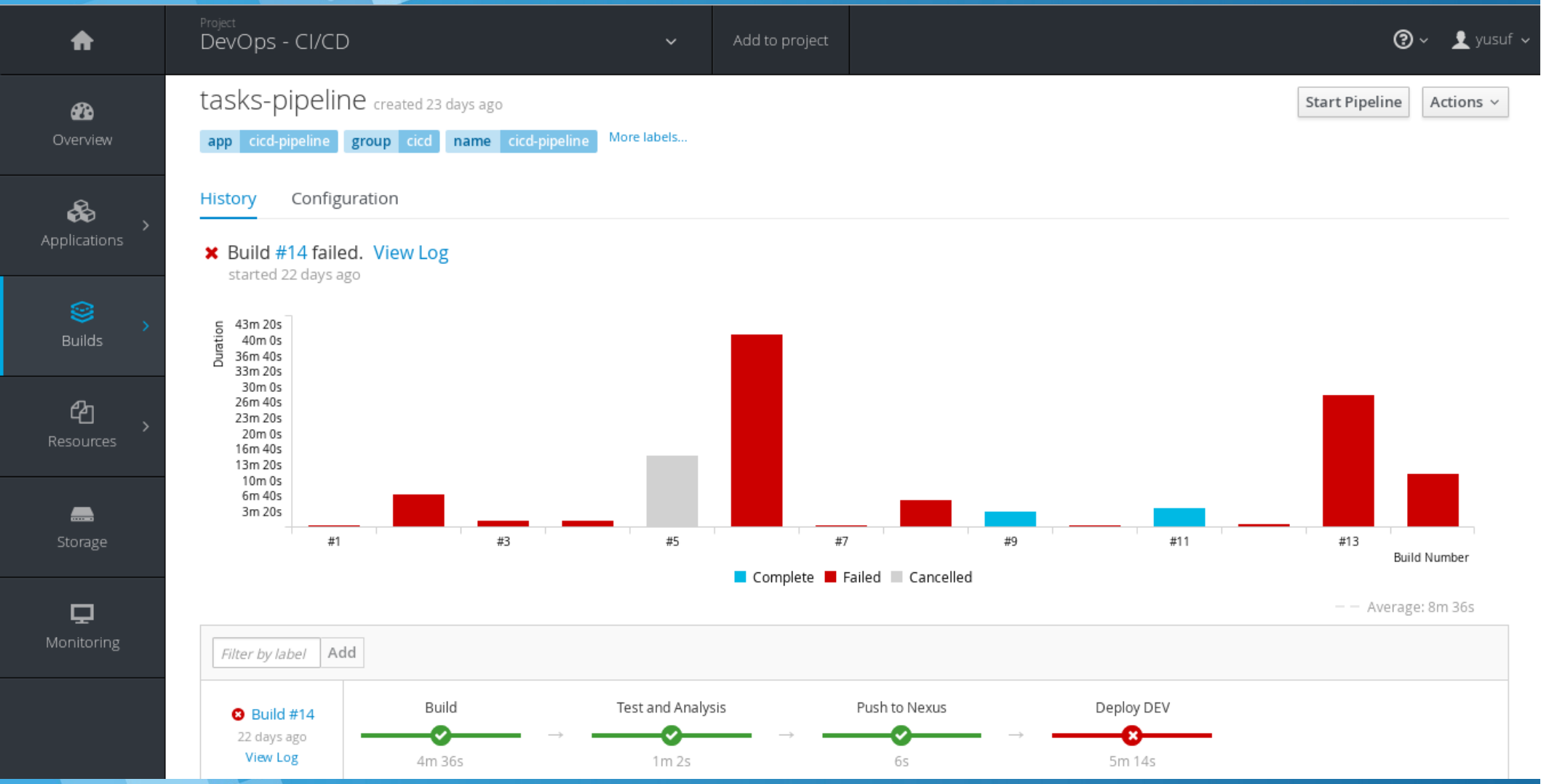
   Note: you need ~8GB memory for running this demo.

Ref: https://github.com/isnuryusuf/openshift-cd-demo

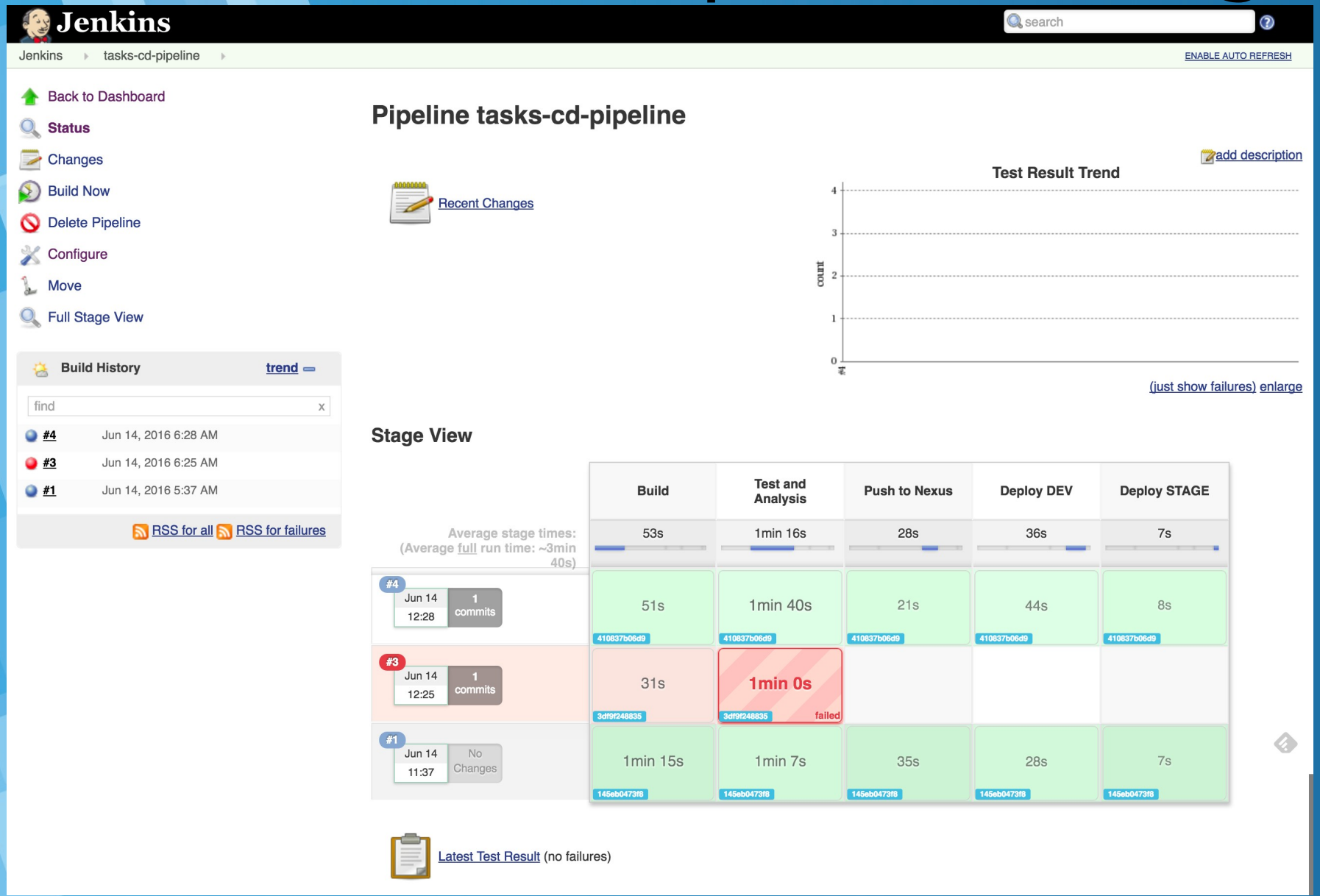
CI/CD Demo – OpenShift Origin



CI/CD Demo – OpenShift Origin



CI/CD Demo – OpenShift Origin



CI/CD Demo – OpenShift Origin

1) Create the CI/CD components based on the provided template
   oc process -f cicd-template.yaml | oc create -f -
2) To use custom project names, change cicd, dev and stage in the above commands to your own names and use the following to create the demo:
   oc process -f cicd-template.yaml -v DEV_PROJECT=dev-project-name -v STAGE_PROJECT=stage-project-name | oc create -f - -n cicd-project-name

   Note: you need ~8GB memory for running this demo.

Ref: https://github.com/isnuryusuf/openshift-cd-demo

Thnk you for Coming

More about me

- https://www.linkedin.com/in/yusuf-hadiwinata-sutandar-3017aa41/
- https://www.facebook.com/yusuf.hadiwinata
- https://github.com/isnuryusuf/

Join me on:

- “Linux administrators” & “CentOS Indonesia Community” Facebook Group
- Docker UG Indonesia: https://t.me/dockerid

Reference

- openshiftenterprise3-160414081118.pptx
- 2017-01-18_-_RedHat_at_CERN_-_Web_application_hosting_with_Openshift_and_Docker.pptx
- DO280 OpenShift Container Platform Administration I
- https://github.com/openshift/origin/

Other Usefull Link

- https://ryaneschinger.com/blog/rolling-updates-kubernetes-replication-controllers-vs-deployments/
- https://kubernetes.io/docs/concepts/storage/persistent-volumes/
- http://blog.midokura.com/2016/08/kubernetes-ready-networking-done-midonet-way/
- https://blog.openshift.com/red-hat-chose-kubernetes-openshift/
- https://blog.openshift.com/chose-not-join-cloud-foundry-foundation-recommendations-2015/
- https://kubernetes.io/docs/concepts/workloads/pods/pod/
- https://blog.openshift.com/enterprise-ready-kubernetes/