

Table of Contents

1. Introduction.....1	3.B Install Minikube3
2. Installation.....2	3.C Install Kubectl4
2.A Installing Kubectl.....2	3.D Getting Started4
2.B Administration.....2	4. Kubectl CLI5
2.C You'll need more than Kubernetes...3	4.A Kubectl Operations5
3. Running Locally via Minikube3	5. About the Author10
3.A Prerequisites3	

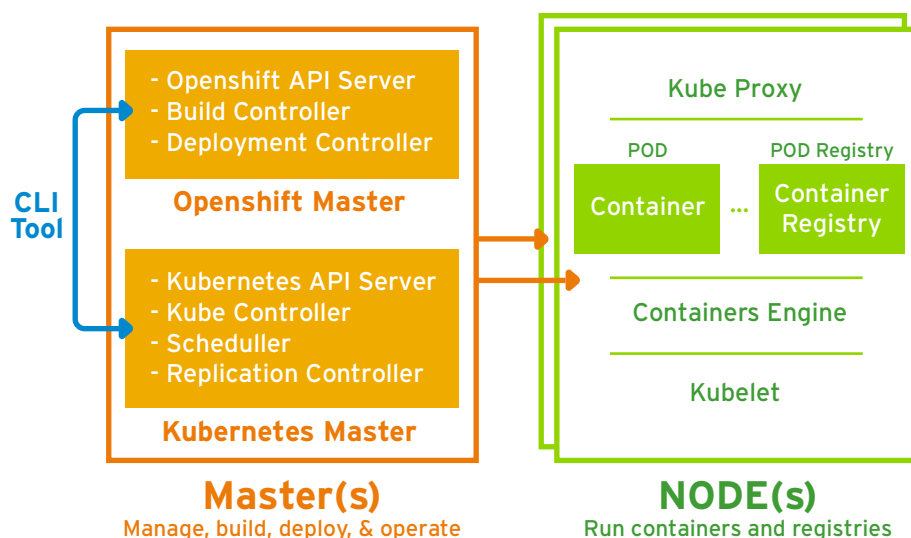
1. Introduction

Linux containers are a technology that allows you to package and isolate applications with their entire runtime environment—all of the files necessary to run. This makes it easy to move the contained application between environments (dev, test, production, etc.) while retaining full functionality.

Containers package applications with the files on which they depend. This reduces the friction between development and operations, simplifies application deployment, and accelerates delivery cycles—allowing you to deliver value to customers faster.

Kubernetes is an open-source platform for automating deployment, scaling, and operations of application **containers** across clusters of hosts, providing container-centric infrastructure.

- Container orchestrator
- Runs Linux containers
 - Describe and launch containers
 - Monitors and maintains container state
 - Performs container oriented networking



2. Installation

2.A Installing kubectl

Download a pre-compiled release[1] and unzip it --- kubectl should be located in the `platforms/<os>/<arch>` directory.

[1] <https://github.com/kubernetes/kubernetes/releases>

Add kubectl to your path. Note, you can simply copy it into a directory that is already in your `$PATH` (e.g. `/usr/local/bin`). For example:

```
# Linux
$ sudo cp kubernetes/platforms/linux/amd64/kubectl /usr/local/bin/kubectl
# OS X
$ sudo cp kubernetes/platforms/darwin/amd64/kubectl /usr/local/bin/kubectl
```

You also need to ensure it's executable:

```
$ sudo chmod +x /usr/local/bin/kubectl
```

2.B Administration

To administer and interact with any given Kubernetes cluster (local or remote), you must set up your `kubeconfig` file. By default, kubectl configuration lives at `~/.kube/config`

You can also create a cluster in your local machine via Minikube (See section 3: Running Locally via Minikube)

```
current-context: federal-context
apiVersion: v1
clusters:
- cluster:
    api-version: v1
    server: http://cow.org:8080
    name: cow-cluster
- cluster:
    certificate-authority: path/to/my/cafile
    server: https://horse.org:4443
    name: horse-cluster
contexts:
- context:
    cluster: horse-cluster
    namespace: chisel-ns
    user: green-user
    name: federal-context
kind: Config
preferences:
  colors: true
users:
- name: green-user
  user:
    client-certificate: path/to/my/client/cert
    client-key: path/to/my/client/key
```

2.C You'll need more than Kubernetes:

Kubernetes operates at the application level rather than at the hardware level, it provides some generally applicable features common to PaaS offerings, such as deployment, scaling, load balancing, logging, monitoring, etc.

However, Kubernetes is not an all-inclusive Platform as a Service (PaaS); therefore, you will still need to consider any needs for DevOps functionality separately:

- Networking
- Image registry
- Metrics and logging
- Complex deployments such as A/B and Blue/Green
- Application lifecycle management
- Application services such as database and messaging
- Self-service portal
- Container security

Much of this additional functionality is provided by the Red Hat OpenShift Container Platform (which includes Kubernetes.)

3. Running Locally via Minikube

Minikube is a tool that makes it easy to run Kubernetes locally --- it runs a single-node Kubernetes cluster inside a virtual machine on your laptop. This is useful for users looking to try out Kubernetes, or develop with it on a day-to-day basis.

3.A Prerequisites

Minikube requires that VT-x/AMD-v virtualization is enabled in BIOS on all platforms. For example:

```
# Linux
$ cat /proc/cpuinfo | grep 'vmx|svm'
# OS X
$ sysctl -a | grep machdep.cpu.features | grep VMX
```

Make sure if the setting is enabled where this command should output something.

Install an x86 virtualization software package in your local machine:

- Linux: The latest [VirtualBox](#)
- OS X: The latest [VirtualBox](#) or [VMware Fusion](#)

3.B Install Minikube

Feel free to leave off the `sudo mv minikube /usr/local/bin` if you would like to add minikube to your path manually.

```
# Linux/
curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.12.2/minikube-
linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/

# OS X
curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.12.2/minikube-
darwin-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

3.C Install Kubectl

You will need to download and install the kubectl client binary to run commands against the cluster. For example:

```
# Linux/amd64
curl -Lo kubectl http://storage.googleapis.com/kubernetes-release/release/v1.3.0/bin/
linux/amd64/kubectl && chmod +x kubectl && sudo mv kubectl /usr/local/bin/

# OS X/amd64
curl -Lo kubectl http://storage.googleapis.com/kubernetes-release/release/v1.3.0/bin/
darwin/amd64/kubectl && chmod +x kubectl && sudo mv kubectl /usr/local/bin/
```

3.D Getting Started

Note that the IP below is dynamic and can change. It can be retrieved with `minikube ip`.

```
$ minikube start
Starting local Kubernetes cluster...
Running pre-create checks...
Creating machine...
Starting local Kubernetes cluster...

$ kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4
--port=8080
deployment "hello-minikube" created
$ kubectl expose deployment hello-minikube --type=NodePort
service "hello-minikube" exposed

# We have now launched an echoserver pod but we have to wait until the pod is up before
curling/accessing it
# via the exposed service.
# To check whether the pod is up and running we can use the following:
$ kubectl get pod
NAME                                READY    STATUS              RESTARTS   AGE
hello-minikube-3383150820-vctvh    1/1     ContainerCreating   0           3s

# We can see that the pod is still being created from the ContainerCreating status
$ kubectl get pod
NAME                                READY    STATUS              RESTARTS   AGE
hello-minikube-3383150820-vctvh    1/1     Running             0           13s

# We can see that the pod is now Running and we will now be able to curl it:
$ curl $(minikube service hello-minikube --url)
CLIENT VALUES:
client_address=192.168.99.1
command=GET
real path=/
...
```

```
# To access the Kubernetes Dashboard, run this command in a shell after starting minikube
to get the address:
$ minikube dashboard
$ minikube stop
Stopping local Kubernetes cluster...
Stopping "minikube"...
```

4. kubectl CLI

```
kubectl [command] [TYPE] [NAME] [flags]
```

- **Command:** Specifies the operation that you want to perform on one or more resources, for example create, get, delete.
- **Type:** Specifies the resource type. Resource types are case-sensitive and you can specify the singular, plural, or abbreviated forms.
- **Name:** Specifies the name of the resource. Names are case-sensitive. If the name is omitted, details for all resources are displayed.

4.A Kubectl Operations

All examples include the general syntax and description for kubectl operations:

Creating Objects

```
# example my-rc.yaml file for creating a object based Replication Controller
```

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80

# create resource(s)
$ kubectl create -f my-rc.yaml
replicationcontroller "nginx" created
```

```
# create resource(s) from url
$ kubectl create -f https://git.io/vPieo
pod "busybox0" created
```

```
# start a single instance of nginx
$ kubectl run nginx --image=nginx
deployment "nginx" created
```

Viewing, Finding Resources

```
# Get commands with basic output
```

```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
busybox-sleep	1/1	Running	0	8m
busybox-sleep-less	1/1	Running	0	8m
busybox0	1/1	Running	0	3m
hello-minikube-3015430129-vfgei	1/1	Running	0	20m
nginx-701339712-tkuma	1/1	Running	0	3m

```
# Get commands with yaml or json file format
```

```
$ kubectl get pod/nginx-cmpmt -o yaml
```

```
apiVersion: v1
```

```
kind: Pod
```

```
metadata:
```

```
  annotations:
```

```
    kubernetes.io/created-by: |
```

```
{“kind”:”SerializedReference”,”apiVersion”:”v1”,”reference”:{“kind”:”ReplicationControl  
ler”,”namespace”:”default”,”name”:”nginx”,”uid”:”01e01208-bb6a-11e6-a905-7eca61497d69”,  
”apiVersion”:”v1”,”resourceVersion”:”58757”}}
```

```
  creationTimestamp: 2016-12-06T04:11:05Z
```

```
  generateName: nginx-
```

```
  labels:
```

```
    app: nginx
```

```
  name: nginx-cmpmt
```

```
  namespace: default
```

```
  ownerReferences:
```

```
  - apiVersion: v1
```

```
    controller: true
```

```
    kind: ReplicationController
```

```
    name: nginx
```

```
    uid: 01e01208-bb6a-11e6-a905-7eca61497d69
```

```
  resourceVersion: “58815”
```

```
  selfLink: /api/v1/namespaces/default/pods/nginx-cmpmt
```

```
  uid: 01e10582-bb6a-11e6-a905-7eca61497d69
```

```
spec:
```

```
containers:
```

```
  - image: nginx
```

```
    imagePullPolicy: Always
```

```
    name: nginx
```

```
    ports:
```

```
    - containerPort: 80
```

```
    protocol: TCP
```

```
    resources: {}
```

```
    terminationMessagePath: /dev/termination-log
```

```
    volumeMounts:
```

```
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
```

```
      name: default-token-xxufg
```

```
      readOnly: true
```

```
  dnsPolicy: ClusterFirst
```

```
  nodeName: minikube
```

```
  restartPolicy: Always
```

```
  securityContext: {}
```

```
  serviceAccount: default
```

```
  serviceAccountName: default
```

```
  terminationGracePeriodSeconds: 30
```

```
  volumes:
```

```
  - name: default-token-xxufg
```

```

    secret:
      secretName: default-token-xxufg
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: 2016-12-06T04:11:05Z
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: 2016-12-06T04:11:23Z
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: 2016-12-06T04:11:05Z
    status: "True"
    type: PodScheduled
containerStatuses:
  - containerID:
      docker://46cdf4314702cc368cf76b46d690134bc78e0de313eb324409fefe088753ed78
        image: nginx
        imageID: docker://
sha256:abf312888d132e461c61484457ee9fd0125d666672e22f972f3b8c9a0ed3f0a1
    lastState: {}
    name: nginx
    ready: true
    restartCount: 0
    state:
      running:
        startedAt: 2016-12-06T04:11:23Z
  hostIP: 192.168.99.100
  phase: Running
  podIP: 172.17.0.13
  startTime: 2016-12-06T04:11:05Z

```

Describe commands with verbose output

\$ kubectl describe pods busybox-sleep

```

Name:          busybox-sleep
Namespace:     default
Node:          minikube/192.168.99.100
Start Time:    Sun, 27 Nov 2016 23:11:35 +0900
Labels:        <none>
Status:        Running
IP:            172.17.0.5
Controllers:   <none>
Containers:
  busybox:
    Container ID:
      docker://4f599b509de0e8504b151e2dfef98c14082ee149ec8da9132824e38095a6b86f
    Image:          busybox
    Image ID:       docker://
sha256:e02e811dd08fd49e7f6032625495118e63f597eb150403d02e3238af1df240ba
    Port:
    Args:
    sleep
    1000000
    State:          Running
    Started:        Sun, 27 Nov 2016 23:11:43 +0900
    Ready:          True
    Restart Count:  0
    Environment Variables:  <none>
Conditions:
  Type          Status
  Initialized    True
  Ready         True
  PodScheduled   True

```

```

Volumes:
  default-token-xxufg:
    Type:      Secret (a volume populated by a Secret)
    SecretName: default-token-xxufg
QoS Tier:      BestEffort

# List Services Sorted by Name
$ kubectl get services --sort-by=.metadata.name
NAME             CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
hello-minikube   10.0.0.38    <nodes>       8080/TCP    51m
kubernetes       10.0.0.1     <none>        443/TCP    53m

# Get ExternalIPs of all nodes
$ kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP")].address}'

```

Viewing, Finding Resources

```

# Add a Label
$ kubectl label pods busybox-sleep new-label=new-busybox-sleep
pod "busybox-sleep" labeled

# Add an annotation
$ kubectl annotate pods busybox-sleep icon-url=http://goo.gl/XXBTWq
pod "busybox-sleep" annotated

# Auto scale a deployment "nginx"
$ kubectl autoscale deployment nginx --min=2 --max=5
deployment "nginx" autoscaled

# Rolling update pods of frontend-v1
$ kubectl rolling-update frontend-v1 -f frontend-v2.json

# Force replace, delete and then re-create the resource. Will cause a service outage
$ kubectl replace --force -f ./pod.json

# Create a service for a replicated nginx, which serves on port 80 and connects to the
containers on port 8000
$ kubectl expose rc nginx --port=80 --target-port=8000

```

Patching Resources

```

# Partially update a node
$ kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'
"k8s-node-1" patched

# Update a container's image; spec.containers[*].name is required because it's a merge key
$ kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-hostname","image":"new image"}]}}'
"k8s-node-1" patched

```

Editing Resources

```

# Edit the service named docker-registry
$ kubectl edit svc/docker-registry
service "docker-registry" edited

```


Scaling Resources

```
# Scale a replicaset named nginx-701339712 to 5
$ kubectl scale --replicas=5 rs/nginx-701339712
replicaset "nginx-701339712" scaled

# Scale multiple replication controllers
$ kubectl scale --replicas=5 rc/foo rc/bar rc/baz
```

Deleting Resources

```
# Delete a pod using the type and specific name
$ kubectl delete pod/nginx-701339712-tkuma
pod "nginx-701339712-tkuma" deleted

# Delete pods and services with same names "baz" and "foo"
$ kubectl delete pod,service baz foo
pod "baz" deleted
service "foo" deleted

# Delete pods and services with label name=myLabel
$ kubectl delete pods,services -l name=myLabel

# Delete all pods and services in namespace my-ns
$ kubectl -n my-ns delete po,svc --all
```

Interacting with running pods

```
# dump pod logs (stdout)
$ kubectl logs busybox-sleep

# stream pod logs (stdout)
$ kubectl logs -f hello-minikube-3015430129-vfgei

# Run pod as interactive shell
$ kubectl run -i --tty busybox --image=busybox -- sh

# Attach to Running Container
$ kubectl attach my-pod -i

# Forward port to service
$ kubectl port-forward my-svc 6000
```

Interacting with running pods

```
# Mark a specific node as unschedulable
$ kubectl cordon minikube
node "minikube" cordoned

# Mark a specific as schedulable
$ kubectl uncordon minikube
node "minikube" uncordoned

# Display addresses of the master and services
$ kubectl cluster-info
Kubernetes master is running at https://192.168.99.100:8443
KubeDNS is running at https://192.168.99.100:8443/api/v1/proxy/namespaces/
kube-system/services/kube-dns
kubernetes-dashboard is running at https://192.168.99.100:8443/api/v1/proxy/
namespaces/kube-system/services/kubernetes-dashboard
```

```
# Dump current cluster state to stdout  
$ kubectl cluster-info dump
```

```
# Dump current cluster state to /path/to/cluster-state  
$ kubectl cluster-info dump --output-directory=/path/to/cluster-state
```

5. About the Author



Daniel Oh is an AppDev Solution Architect, Agile & DevOps CoP Manager at Red Hat and has specialty about JBoss middleware, Java EE, Containers, Agile methodology, DevOps, PaaS(OpenShift), Containerized application design, MSA, and Mobile application platform.

[Twitter @danieloh30](#)

[in](#) LinkedIn

Table of Contents

1. What is OpenShift?.....	1	5. Simple routing overview	4
2. Cheat sheet guide.....	1	6. Examples.....	5
3. Command overview.....	2	7. Additional operations.....	10
4. Simple build and deploy overview.....	4	8. About the author.....	11

What is Openshift?

From OpenShift.com:

OpenShift is Red Hat's Platform-as-a-Service (PaaS) that allows developers to quickly develop, host, and scale applications in a cloud environment.

Openshift makes use of the Kubernetes upstream project to provide a secure, robust, and extendable manner for orchestrating applications. Openshift works to further the access management and build/deploy services that are provided in the upstream Kubernetes project. Development teams are empowered to own and maintain their applications through production environments, while operations teams can provide the guide rails for developers to have that application ownership in a multi-tenant environment.

Cheatsheet Guide

This guide is mostly focused on the developer experience, however several administrator tasks are detailed below. A high-level listing of operations from command line interface is provided, followed by a walkthrough of an example application build, deploy, and management. The command list is not exhaustive, but does cover the majority of operations a developer will need to understand to manage an application's lifecycle.

Command Overview

Login/User management

<code>oc login</code>	authenticate to an openshift cluster
<code>oc logout</code>	end the current session
<code>oc whoami</code>	show the current user context

Project management

<code>oc project</code>	show the current project context
<code>oc get projects</code>	show all project current login has access to
<code>oc status</code>	show overview of current project resources
<code>oc new-project</code>	create a new project in Openshift and change to that context

Resource management

<code>oc new-app</code>	create a new application from from source code, container image, or OpenShift template
<code>oc new-build</code>	create a new build configuration from source code
<code>oc label</code>	add/update/remove labels from an Openshift resource
<code>oc annotate</code>	add/update/remove annotations from an Openshift resource
<code>oc create</code>	create a new resource from filename or stdin
<code>oc get</code>	retrieve a resource (use -o for additional output options)
<code>oc replace</code>	replace an existing resource from filename or stdin
<code>oc delete</code>	delete a resource
<code>oc edit</code>	modify a resource from text editor
<code>oc describe</code>	retrieve a resource with details

Cluster management

<code>oc adm</code>	administrative functions for an openshift cluster
<code>oc adm router registry</code>	install a router or registry
<code>oc adm policy</code>	manage role/scc to user/group bindings, as well as additional policy administration
<code>oc adm diagnostics</code>	run tests/validation against a cluster
<code>oc adm cordon/uncordon/drain</code>	unschedule/schedule/drain a node
<code>oc adm groups</code>	manage groups
<code>oc adm top</code>	show usage statistics of resources

Additional resource management

<code>oc patch</code>	Update fields for a resource with JSON or YAML segments
<code>oc extract</code>	get configmaps or secrets and save to disk
<code>oc set</code>	Modify miscellaneous application resources
<code>oc set probe</code>	Add a readiness/liveness probe on pod template/deployment configuration
<code>oc set volumes</code>	Manage volume types on a pod template/deployment configuration
<code>oc set build-hook</code>	Set a script/command to execute as part of the build process
<code>oc set build-secret</code>	set a secret to be included as part of the build process
<code>oc set env</code>	set environment variables on a pod template/deployment configuration/build configuration
<code>oc set image</code>	update the image for deployment configurations/daemonsets
<code>oc set triggers</code>	set triggers for deployment configurations/build configurations

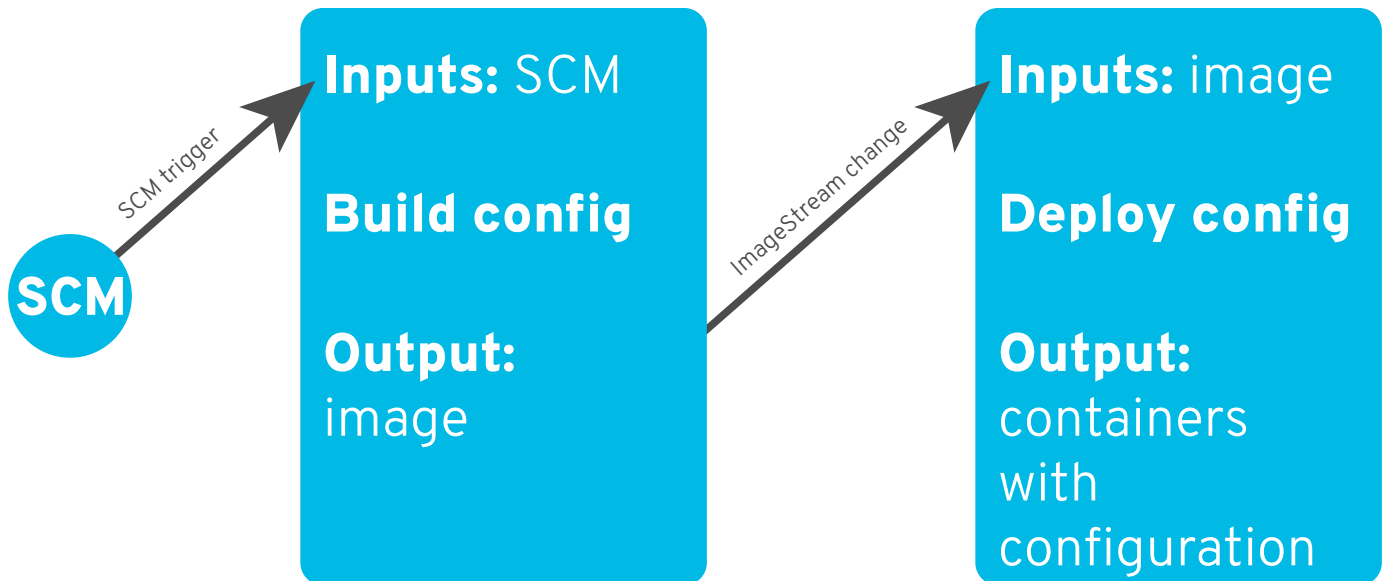
Operational commands

<code>oc logs</code>	retrieve the logs for a resource (build configurations, deployment configurations, and pods)
<code>oc rsh</code>	remote shell into a container
<code>oc rsync</code>	copy files to or from a container
<code>oc exec</code>	execute a command in a container
<code>oc run</code>	create a deployment configuration from image
<code>oc idle</code>	scale resources to zero replicas

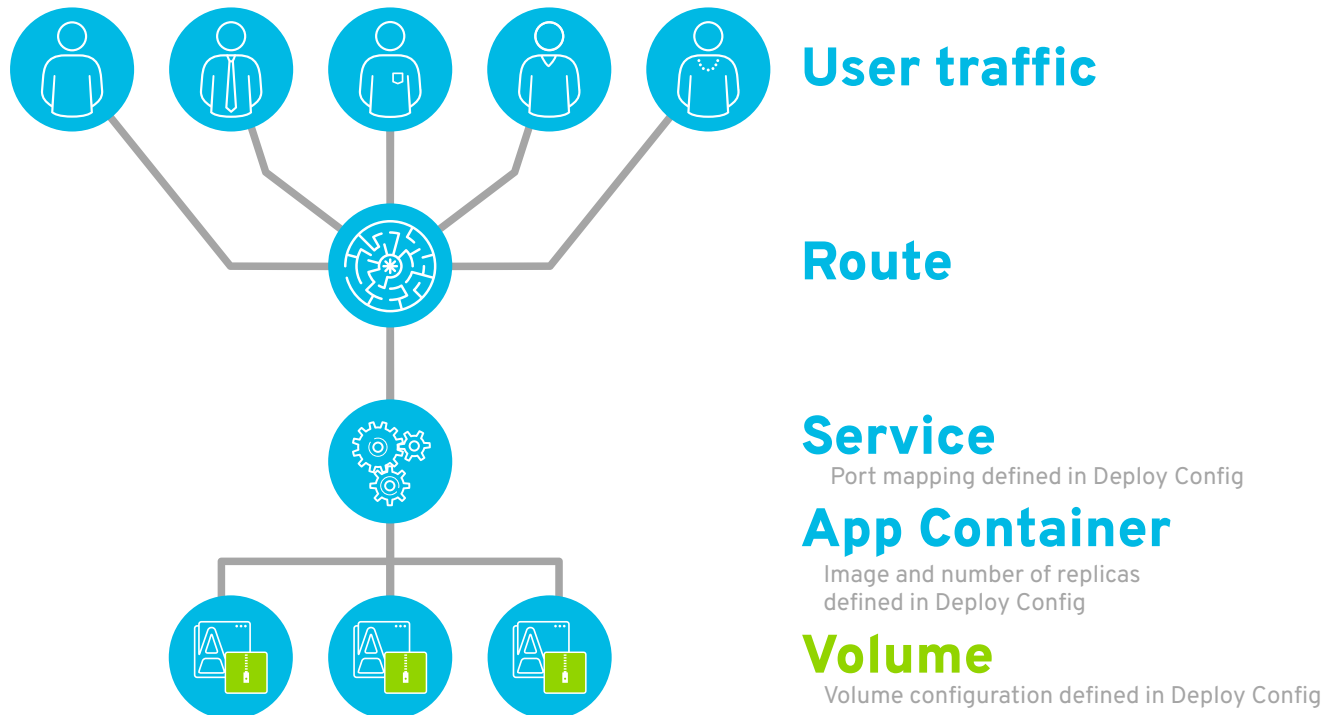
Build / Deploy

<code>oc rollout</code>	manage deployments from deployment configuration
<code>oc rollout latest</code>	start a new deployment with the latest state
<code>oc rollout undo</code>	perform a rollback operation
<code>oc rollout history</code>	oc rollout history - View historical information for a deployment configuration
<code>oc rollout status</code>	watch the status of a rollout until complete
<code>oc tag</code>	tag existing images into image streams
<code>oc start-build</code>	start a new build from a build configuration
<code>oc cancel-build</code>	cancel a build in progress
<code>oc import-image</code>	pull in images and tags from an external Docker registry
<code>oc scale</code>	change the number of pod replicas for a deployment

Simple Build/Deploy Overview



Simple Routing Overview



Examples

Login

First, we can login to the cluster to interact with Openshift via CLI

```
$ oc login -u myuser https://openshift.example.com
Authentication required for https://openshift.example.com
Username: myuser
Password:
```

Note that leaving the `-p` option off of login prompts for password. Additionally we can verify our user context:

```
$ oc whoami
myuser
```

Create Project

Let's list out our current available projects (those that we have at least view access for):

```
$ oc get projects
```

If this is our first login and no one has added us to any existing projects, there shouldn't be any projects listed. Let's create a project (allowed by self-provisioner role to all authenticated users, in the default Openshift policy installation).

```
$ oc new-project myproject --display-name='My Project' --description='cool project owned by myuser'
```

Now using project "myproject" on server "https://openshift.example.com:443".

To build a new example application on Ruby 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
```

If you want to view the specifics of the project definition, output the full spec to YA

```
$ oc get project myproject1 -o yaml
apiVersion: v1
kind: Project
metadata:
  annotations:
    openshift.io/description: A really cool project owned by myuser
    openshift.io/display-name: My Project
    openshift.io/requester: myuser
    openshift.io/sa.scc.mcs: s0:c51,c20
    openshift.io/sa.scc.supplemental-groups: 1000000000/10000
    openshift.io/sa.scc.uid-range: 1000000000/10000
    creationTimestamp: 2017-02-10T15:36:18Z
  labels:
    name: myproject
    resourceVersion: "32381158"
    selfLink: /oapi/v1/projects/myproject
    uid: aa94c906-efa6-11e6-af71-02a55ffb157d
spec:
  finalizers:
    - openshift.io/origin
    - kubernetes
status:
  phase: Active
```

Add users to project

We can add additional users to our project by default, since self-provisioners get the "admin" role for any project they create:

```
$ oc adm policy add-role-to-user edit anotheruser
```

This allows anotheruser to edit resources within the project, but not manage policy

Create app from code and image

```
$ oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git
--> Found Docker image 06f0cdc (2 days old) from Docker Hub for "centos/ruby-22-centos7"
Ruby 2.2
-----
Platform for building and running Ruby 2.2 applicationsTags: builder, ruby, ruby22
* An image stream will be created as "ruby-22-centos7:latest" that will track the source image
* A source build using source code from https://github.com/openshift/ruby-ex.git will be created
* The resulting image will be pushed to image stream "ruby-ex:latest"
* Every time "ruby-22-centos7:latest" changes a new build will be triggered
* This image will be deployed in deployment config "ruby-ex"
* Port 8080/tcp will be load balanced by service "ruby-ex"
* Other containers can access this service through the hostname "ruby-ex"
--> Creating resources with label app=ruby-ex ...
imagestream "ruby-22-centos7" created
imagestream "ruby-ex" created
buildconfig "ruby-ex" created
deploymentconfig "ruby-ex" created
service "ruby-ex" created
--> Success
Build scheduled, use 'oc logs -f bc/ruby-ex' to track its progress.
Run 'oc status' to view your app.
```

The new-app command handles the majority of resource creation via template. Notice that deploymentconfig/buildconfig/service/imagestream were all set up.

Get resources

We can view the resources that were created as part of the new-app command, as well as the build/deploy resources that were created automatically. Notice that the new-app automatically started a new build of our code, and the deployment config watches successful builds to know when to next rollout/deploy. A good place to start with viewing application status is checking the pods in your project:

```
$ oc get pods
NAME READY STATUS RESTARTS AGE
ruby-ex-1-a7y56 1/1 Running 0 24m
ruby-ex-1-build 0/1 Completed 0 26m
```

This shows us the build pod completed successfully. Additionally we can see that there is one ready and running pod deployed with our application.

The status command shows us similar results:

```
$ oc status -v
In project My Project (myproject1) on server https://openshift.example.com:443
svc/ruby-ex - 172.30.36.21:8080
dc/ruby-ex deploys istag/ruby-ex:latest <-
bc/ruby-ex source builds https://github.com/openshift/ruby-ex.git on istag/ruby-
22-centos7:latest
deployment #1 deployed 26 minutes ago - 1 pod
Warnings:
* dc/ruby-ex has no readiness probe to verify pods are ready to accept traffic or
ensure deployment is successful.
try: oc set probe dc/ruby-ex --readiness ...
View details with 'oc describe <resource>/<name>' or list everything with 'oc get
all'.
```

Add a volume

If we want to attach a volume to our pods, the oc set volume command can be used:

```
$ oc set volume dc/ruby-ex --add --mount-path=/mnt/emptydir
info: Generated volume name: volume-7d1e8
deploymentconfigs/ruby-ex

$ oc get pods
NAME READY STATUS RESTARTS AGE
ruby-ex-1-a7y56 1/1 Running 0 2h
ruby-ex-1-build 0/1 Completed 0 2h
ruby-ex-2-deploy 0/1 ContainerCreating 0 5s
```

In this example, a simple emptyDir volume was attached, though the same command can be used for Persistent Volumes. Also notice that the deployment configuration has a ConfigChange trigger, so adding this volume automatically started a new deployment.

Edit resource

Making a change to any Openshift resource is simple. Let's change the `/mnt/emptydir` mountpath above to `/mnt/appdata`:

```
$ oc edit dc ruby-ex
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file
# will be reopened with the relevant failures.
#
...
volumeMounts:
- mountPath:/mnt/emptydir /mnt/appdata
  name: volume-7d1e8
...
```

Saving the file in your text editor will update the resource, or report errors if validation did not succeed. Note that this change on the deployment config kicks off another deployment for our app.

Start build

If a new build from source is desired:

```
$ oc start-build ruby-ex
build "ruby-ex-2" started
```

Watch build

The build logs can be watched with the `oc logs` command (including `-f` option for follow):

```
$ oc logs -f bc/ruby-ex
Cloning "https://github.com/openshift/ruby-ex.git" ...
Commit: 855ab2de53ff897a19e1055f7554c64d19e02c50 (Merge pull request #6 from aj07/
typo)
Author: Ionut Palade <PI-Victor@users.noreply.github.com>
Date: Mon Dec 12 14:37:32 2016 +0100
----> Installing application source ...
----> Building your Ruby application from source ...
----> Running 'bundle install --deployment --without development:test' ...
Fetching gem metadata from https://rubygems.org/.....
Installing puma 3.4.0
Installing rack 1.6.4
Using bundler 1.7.8
Your bundle is complete!
Gems in the groups development and test were not installed.
It was installed into ./bundle
----> Cleaning up unused ruby gems ...

Pushing image 172.30.114.236:5000/myproject/ruby-ex:latest ...
Pushed 7/9 layers, 78% complete
Pushed 8/9 layers, 89% complete
Pushed 9/9 layers, 100% complete
Push successful
```

Start Deploy

Most configuration or image changes will automatically start a new deploy by default, but new deployments can be started manually as well:

```
$ oc rollout latest ruby-ex
deploymentconfig "ruby-ex" rolled out
```

Watch Deploy

The overall deployment status can be watched via `oc logs` command:

```
$ oc logs -f dc/ruby-ex
--> Scaling up ruby-ex-5 from 0 to 1, scaling down ruby-ex-4 from 1 to 0 (keep 1
pods available, don't exceed 2 pods)
Scaling ruby-ex-5 up to 1
Scaling ruby-ex-4 down to 0
--> Success
```

Additionally container logs can be viewed with `oc logs`:

```
$ oc logs ruby-ex-5-kgzvd
[1] Puma starting in cluster mode...
[1] * Version 3.4.0 (ruby 2.2.2-p95), codename: Owl Bowl Brawl
[1] * Min threads: 0, max threads: 16
[1] * Environment: production
[1] * Process workers: 8
[1] * Phased restart available
[1] * Listening on tcp://0.0.0.0:8080
[1] Use Ctrl-C to stop
[1] - Worker 2 (pid: 29) booted, phase: 0
[1] - Worker 1 (pid: 25) booted, phase: 0
[1] - Worker 5 (pid: 41) booted, phase: 0
[1] - Worker 3 (pid: 33) booted, phase: 0
[1] - Worker 0 (pid: 21) booted, phase: 0
[1] - Worker 4 (pid: 37) booted, phase: 0
[1] - Worker 6 (pid: 45) booted, phase: 0
[1] - Worker 7 (pid: 60) booted, phase: 0
```

Remote shell

Interacting directly with the container is simple with `oc rsh`:

```
$ oc rsh ruby-ex-5-kgzvd
sh-4.2$ ls
Gemfile Gemfile.lock README.md bundle config.ru
```

Create route

```
$ oc expose service ruby-ex
route "ruby-ex" exposed
```

With no other options defined this will create a route for your application using the default route naming (ex: `$appname-$projectname.openshift.example.com`)

Idle app

We're done testing our application, so we can idle the service in order to save resources. This interacts with a Kubernetes service to set the pod replicas to 0, and when the service is next accessed will automatically boot up the pods again:

```
$ oc idle ruby-ex
Marked service myproject1/ruby-ex to unidle resource DeploymentConfig myproject1/
ruby-ex (unidle to 1 replicas)
Idled DeploymentConfig myproject1/ruby-ex
```

Delete app

If we're completely done with our application, we can delete resources within the project (or the project itself) to clean up:

```
$ oc delete services -l app=ruby-ex
service "ruby-ex" deleted


$ oc delete all -l app=ruby-ex
buildconfig "ruby-ex" deleted
imagestream "ruby-22-centos7" deleted
imagestream "ruby-ex" deleted
deploymentconfig "ruby-ex" deleted

$ oc delete project myproject
project "myproject" deleted
```

About the author



BEN PRITCHETT is a Systems Engineer within Red Hat's IT department, deploying Red Hat's internal offerings of Platform-as-a-Service technologies. His focus is to bridge the gap between development and operations teams using automation and training.

 <http://github.com/bjpritch>