Using oc to manage OpenShift

The Basics

Examples of the basic need to know oc commands to deploy and manage containers on OpenShift. Set a few variables to prevent sensitive information in the terminal history.

```
export REGISTRY_USERNAME=YourUsername
export REGISTRY_PASSWORD=SomePassword
export REGISTRY_HOST=quay.io
export REGISTRY_EMAIL=mail@example.com
export OSU=developer
export OSP=SuperAw3SomePassrd
```

Pods

Using `oc explain` it is simple to get the ducumentation for the running version of OpenShift. Here are a few basics.

Get built-in documentation for Pods

```
oc explain pod
```

Get the pod spec

```
oc explain pod.spec
```

Details on the pod's containers

```
oc explain pod.spec.containers
```

Details about the pod's containers images

```
oc explain pod.spec.containers.image
```

Example of a pod file

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-world-pod
  labels:
    app: hello-world-pod
spec:
  containers:
  - env:
    - name: MESSAGE
      value: Hi there! You've run a pod.
    image: docker.io/mcleary/helloworld-go
    imagePullPolicy: Always
    name: hello-world-override
    resources: {}
```

Create a Pod on OpenShift based on a file

```
oc create -f pod.yaml
```

Use `oc get` to information from OpenShift

Get pods

```
oc get pods
```

Watch pods deploy

```
oc get pods --watch
```

Get all resources

```
oc get all
```

Access the shell of a running container. Use `oc get pods` to get the pod name.

```
oc rsh <pod-name>
```

Use port forwards to interact with the pod on the local machine. Get the pod name from the `oc get pods`.

```
oc port-forward <pod-name> <local_port>:pod_port>
```

Delete OpenShift resources use the following syntax

```
oc delete <resource type> <resource name>
```

Delete a pod

```
oc delete pod <pod-name>
```

Deployments

Deploy an existing image based on its tag

```
oc new-app mcleary/helloworld-go:latest --as-deployment-config
```

Deploy an application from Git

```
oc new-app https://github.com/clusterapps/helloworld-go.git --as-deployment-config
```

Follow build progress when using

```
oc logs -f bc/helloworld-go
```

Set the name for the DeploymentConfig

```
oc new-app mcleary/helloworld-go --name hello-app --as-deployment-config
```

DeploymentConfig with a parameter

```
oc new-app MESSAGE="This is a parameter" mcleary/helloworld-go --name hello-app --as-deployment-config
```

DeploymentConfig with many patameters

```
oc new-app mysql MYSQL_USER=user MYSQL_PASSWORD=pass MYSQL_DATABASE=testdb -l db=mysql
```

Get information about a DeploymentConfig

Describe the DC to get its labels

```
oc describe dc/hello-app
```

Get the full YAML definition

```
oc get -o yaml dc/hello-app
```

Roll out the latest version of the application

```
oc rollout latest dc/hello-app
```

Roll back to the previous version of the application

```
oc rollback dc/hello-app
```

Scaling

General Syntax `oc scale dc/<dc name> --replicas=<desired replicas>`

Manual scale to 3 pods

```
oc scale dc/helloworld-go --replicas=3
```

Scale down to one

```
oc scale dc/helloworld-go --replicas=1
```

General Syntax to create a HorizontalPodAutoscaler

```
oc autoscale dc/<dc name> \
  --min <desired minimum pods> \
  --max <desired maximum pods> \
  --cpu-percent=<desiredTargetCPU>
```

Auto scaling between 1 and 10 pods with an 80% CPU target

```
oc autoscale dc/helloworld-go \
  --min 1 \
  --max 10 \
  --cpu-percent=80
```

Show the HPA

```
oc get hpa
```

Information on the HPA

```
oc describe hpa/helloworld-go
```

YAML for the HPA

```
oc get -o yaml hpa/helloworld-go
```

Deleting resources

Delete a single resource

```
oc delete <resource> <name>
oc delete dc hello-app # Delete deployment config
oc delete svc hello-app # Delete a service
```

Delete all application resources using labels (get labels from oc describe)

```
oc delete all -l app=app=hello-app
```

Delete everything in a project

```
oc delete all --all
```

Templates

Get a list of templates

Templates in the working namespace

```
oc get template
```

Get built-in templates

```
oc get templates -n openshift
```

Get template details

Full details of the template.

```
oc describe template -n openshift mariadb-persistent
```

Get just the parameters for a template.

```
oc process --parameters mysql-persistent -n openshift
```

Create a new application from a template

```
oc new-app --template=mariadb-persistent \
 -p MYSQL_USER=jack -p MYSQL_PASSWORD=password \
 -p MYSQL_DATABASE=jack
```

Check the status of the deployment and get the pod name

```
oc get all
```

Full details of a pod from a template.

```
oc describe pod/mariadb-1-qlvrj
```

Networking

Access oc explain documentation

`oc explain service`

Get information about Service spec

```
oc explain service.spec
```

Get YAML definition for a service

```
oc get -o yaml service/hello-world
```

Get YAML definition for a route

```
oc get -o yaml route/hello-world
```

Creating services

Create a service for a single pod

```
oc expose --port 8080 pod/hello-world-pod
```

Create a service for a DeploymentConfig

```
oc expose --port 8080 dc/hello-world
```

Check that the service and pod are connected properly

```
oc status
```

Using Pod environment variables to find service Virtual IPs

Log into a pod. Get pod name from `oc get pods`

```
oc rsh pod/helloworld-2
```

Inside the pod, get all environment variables

```
env
```

Testing connectivity using environment variables with wget

```
wget -qO- $HELLOWORLD_GO_PORT_8080_TCP_ADDR:$HELLOWORLD_GO_PORT_8080_TCP_PORT
```

Creating Routes

Create a Route based on a Service. Get the service name from `oc get svc`

```
oc expose svc/helloworld-go
```

Get the Route URL

```
oc status
```

Check the route

```
curl helloworld-go-lab1.apps.okd4.example.com
```

ConfigMaps

Create a ConfigMap using literal command line arguments

```
oc create configmap helloconfig --from-literal KEY="VALUE"
```

Create from a file

```
oc create configmap helloconfig --from-file=MESSAGE.txt
```

Create from a file with a key override

```
oc create configmap helloconfig --from-file=MESSAGE=MESSAGE.txt
```

Create using `--from-file` with a directory

```
oc create configmap helloconfig --from-file pods
```

Verify

```
oc get -o yaml configmap/helloconfig
```

Consuming ConfigMaps as Environment Variables

Set environment variables

```
oc set env dc/hello-app --from cm/helloconfig
```

Secrets

Secrets use simalar syntax as ConfigMaps. Secrets are base64 encoded ConfigMaps

There are a few main types.

- Opaque
- Service Account Tokens
- Registry Authentication
- Simple Auth Types

A simple generic (Opaque) Secret. Key-Value pairs

```
oc create secret generic <secret-name> --from-literal KEY="VALUE"
```

Check the Secret

```
oc get -o yaml secret/<secret-name>
```

Consume the Secret as Environment Variables the same as ConfigMaps

```
oc set env dc/<dc-name> --from secret/<secret-name>
```

Create a default registry secret

```
oc create secret docker-registry secret_name
  --docker-server=$REGISTRY_HOST \
  --docker-username=$REGISTRY_USERNAME \
  --docker-password=$REGISTRY_PASSWORD \
  --docker-email=$REGISTRY_EMAIL
```

Link the secret to the service account named "default"

```
oc secrets link default secret_name --for=pull
```

Check that the service account has the secret associated

```
oc describe serviceaccount/default
```

Decoding secrets. First get the hash.

- mysecret = name of secret
- password = is the entry to decode

```
oc get secret mysecret -o yaml |grep password
```

Result simalar to `password: cGFzc3dvcmQ=`

Decode as base64

```
echo "cGFzc3dvcmQ=" |base64 -d
```

Images

ImageStreams

List ImageStreams

```
oc get is
```

List tags

```
oc get istag
```

Create the ImageStream but don't deploy

```
oc import-image --confirm quay.io/clearyme/helloworld-go
```

Importing any new images from the repository using the same command as importing a new.

```
oc import-image --confirm quay.io/clearyme/helloworld-go
```

Creating a new local tag : `oc tag <original> <destination>`

```
oc tag $REGISTRY_HOST/$REGISTRY_USERNAME/helloworld-go:latest helloworld-go:local-1
```

Deploy an application based on your new ImageStream (lab1 is the name of the oc project with the newly tagged image)

```
oc new-app lab1/helloworld-go:local-1
```

Build a custom image

From within the directory where the `Dockerfile` is, build the image and tag it for the registry.

```
podman build -t $REGISTRY_HOST/$REGISTRY_USERNAME/helloworld-go:latest .
```

Log into a registry

```
podman login $REGISTRY_HOST
```

Push the image ti the registery

```
podman push $REGISTRY_HOST/$REGISTRY_USERNAME/helloworld-go
```

Builds and BuildConfigs

Create a new BuildConfig from a Git repository URL

`oc new-build <git URL>`

Example

```
oc new-build https://github.com/clusterapps/helloworld-go.git
```

Example new build from s branch

```
oc new-build https://github.com/clusterapps/helloworld-go.git#update
```

Example using --context-dir to build from a subdirectory

```
oc new-build https://github.com/clusterapps/openshift-playing.git --context-dir hello-world-go
```

Working with existing BuildConfigs

Get a list of BuildConfigs

```
oc get bc
```

Start a build

```
oc start-build bc/helloworld
```

Get the list of builds

```
oc get build
```

Get logs for a single build

```
oc logs -f build/helloworld-go-1
```

Get logs for the latest build for a BuildConfig

```
oc logs -f bc/helloworld-go
```

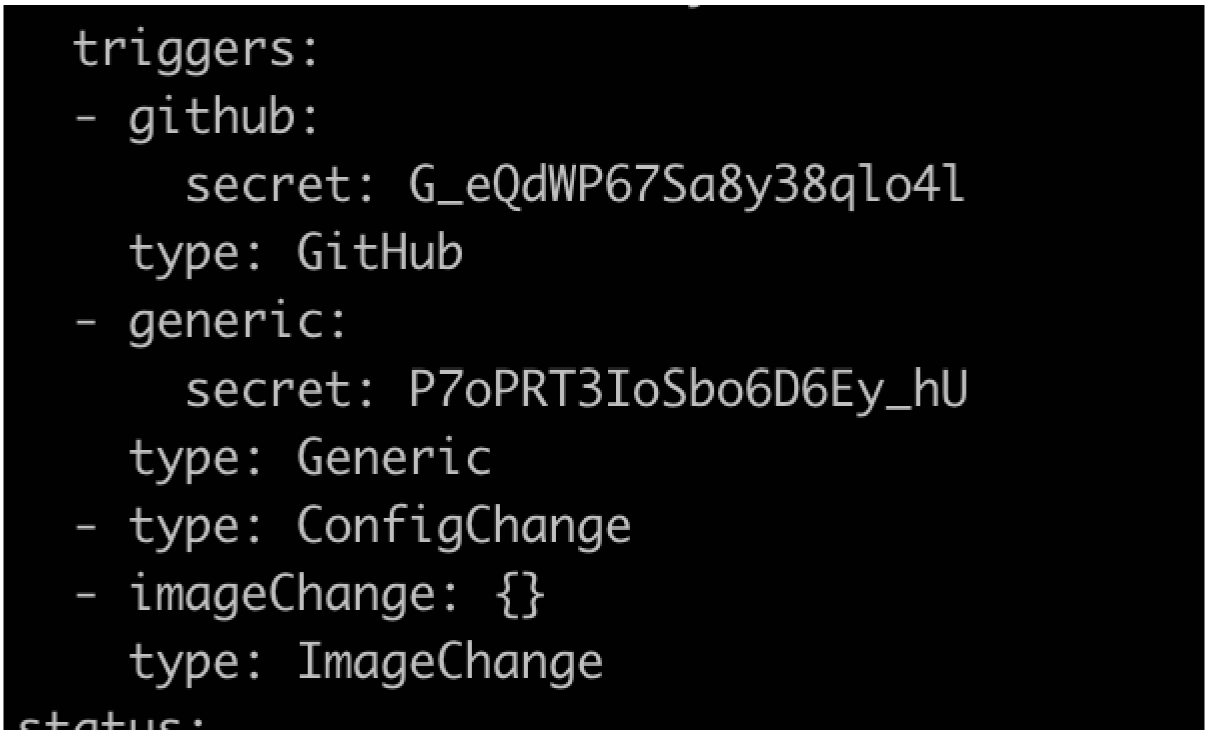
Use `oc cancel-build` to stop a build ``

```
oc cancel-build build/helloworld-go-1
```

Working with WebHooks

Get the secret token

```
oc get -o yaml buildconfig/helloworld-go
```



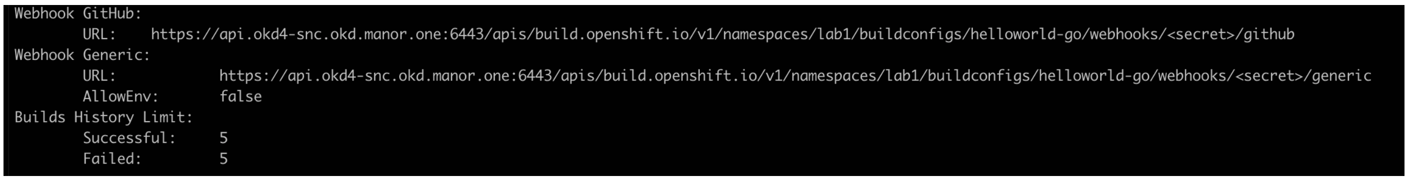
```
  triggers:
  - github:
      secret: G_eQdWP67Sa8y38qlo4l
    type: GitHub
  - generic:
      secret: P7oPRT3IoSbo6D6Ey_hU
    type: Generic
  - type: ConfigChange
  - imageChange: {}
    type: ImageChange
status:
```

Export the secret as a variable

```
export GENERIC_SECRET=G_eQdWP67Sa8y38qlo4l
```

Get the webhook URL

```
oc describe buildconfig/helloworld-go
```



```
Webhook GitHub:
        URL:    https://api.okd4-snc.okd.manor.one:6443/apis/build.openshift.io/v1/namespaces/lab1/buildconfigs/helloworld-go/webhooks/<secret>/github
Webhook Generic:
        URL:            https://api.okd4-snc.okd.manor.one:6443/apis/build.openshift.io/v1/namespaces/lab1/buildconfigs/helloworld-go/webhooks/<secret>/generic
        AllowEnv:       false
Builds History Limit:
        Successful:     5
        Failed:         5
```

Copy the webhook URL and replace `<secret>` with $GENERIC_SECRET

```
curl -X POST -k <webhook URL with secret replaced with $GENERIC_SECRET>
```

Set build hooks

Set a post-commit hook

```
oc set build-hook bc/helloworld-go \
  --post-commit \
  --script="echo Hello from build hook"
```

Check the logs output for "Hello from a build hook"

```
oc logs -f bc/helloworld-go
```

Check the events to see if it ran

```
oc get events
```

Remove a build hook

```
oc set build-hook bc/helloworld-go \
  --post-commit \
  --remove
```

Source to Image

The syntax is the same as normal builds. OpenShift uses S2I to guess the language when there is no Dockerfile. A language can also be specified at build time durring the `oc new-app`.

Works with Java, Ruby, Node, PHP, Python and PERL,

Overriding S2I Scripts Assemble and Run are the two main scripts Overrides go in your source at `.s2i/bin/assemble` or `.s2i/bin/run` They need to call the original scripts, which are at `/usr/libexec/s2i/assemble` or `/usr/libexec/s2i/run`

New app without a `Dockerfile`.

```
oc new-app https://github.com/clusterapps/openshift-playing.git --context-dir s2i/ruby
```

New app specifying the language by adding *language tilda* to the new-app command.

```
oc new-app ruby~https://github.com/clusterapps/openshift-playing.git --context-dir s2i/ruby
```

New app from a git branch

```
oc new-app https://github.com/clusterapps/helloworld-go.git#updates
```

Volumes

Check out the built-in documentation

`oc explain persistentvolume.spec` or the The official Kubernetes Documentation for Volumes

Most deployent of k8s and OpenShift may have a dynamic storageclass. You can just the dynamic storage by specifying the mountpoint and size. For manual storage, here are some examples.

Storage basics for containers.

Mount Volumes

emptyDir

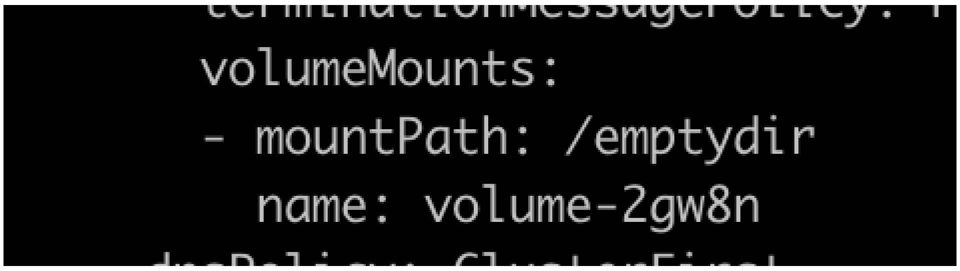
`oc set volume dc/<dc name> --add --type emptyDir --mount-path <path inside container>`

Example: Add an emptyDir volume. An emptyDir is ephemeral. It will survive a pod reboot, but will be deleted when the pod is deleted. Only good for testing.

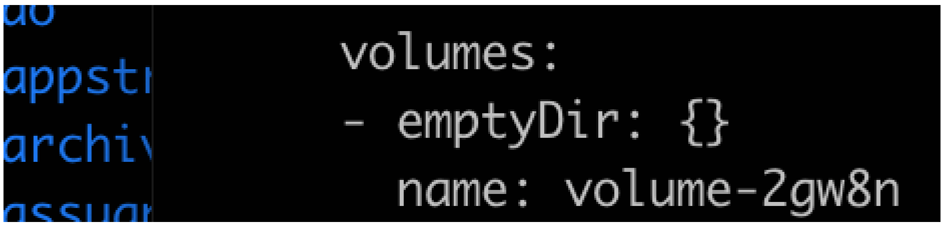
```
oc set volume dc/helloworld-go --add \
  --type emptyDir \
  --mount-path /emptydir
```

View the DeploymentConfig to view the volume information. Look for container.volumeMounts and volumes.

```
oc get -o yaml dc/helloworld-go
```



```
    volumeMounts:
    - mountPath: /emptydir
      name: volume-2gw8n
```



```
      volumes:
      - emptyDir: {}
        name: volume-2gw8n
```

ConfigMaps

Basic example

`oc set volume <DC name> --add --configmap-name <configmap name> -mount-path <path inside container>`

Create the configmap to use as a Volume

```
oc create configmap volume-file --from-literal file.txt="Contents"
```

Mount the ConfigMap

```
oc set volume dc/helloworld-go --add --configmap-name volume-file --mount-path /configs/
```

NFS

NFS is a very common storage method. It's cheap and easy to manage, but does have its own pitfalls.

Create an NFS persistent volume(PV) defination. A regular user cannot create PV.

```
cat > nfs.yml <<EOF
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfspv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  nfs:
    path: /nfsshare
    server: 172.20.255.2
  persistentVolumeReclaimPolicy: Retain
EOF
```

Map a PVC to a PV

Create the PVC YAML

```
cat > nfspvc.yml <<EOF
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: nfs001
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  volumeName: nfspv
  storageClassName: ''
  volumeMode: Filesystem
```

```
EOF
```

Mount the pvc to a directory

```
oc set volume dc/helloworld-go --add --name=nfs001 -t pvc --claim-name=nfs001 -m /data
```

Basic troubleshooting

Retrieve the logs from a build configuration

```
oc logs bc/<application-name>
```

If a build fails, after finding and fixing the issues, run the following command to request a new build:

```
oc start-build <application-name>
```

Deployment logs

```
oc logs dc/<application-name>
```

Temporarily access some of these missing commands is mounting the host binaries folders, such as /bin, /sbin, and /lib, as volumes inside the container

```
sudo podman run -it -v /bin:/bin image /bin/bash
```

Revision #16
Created 18 February 2022 01:29:13 by Michael Cleary
Updated 12 March 2022 21:05:40 by Michael Cleary