Kubernetes

Pods: Basics Challenge Solutions

2.1. CHALLENGE: pod exploration

- Run `kubectl run apache --image docker.io/httpd:latest` to create another `apache` pod

```
~$ kubectl run apache --image docker.io/httpd:latest

pod/apache created

~$
```

- Use `kubectl describe` to view the new `apache` pod
    - What image is this new apache pod running? How is it different from the initial `apache` pod?

```
~$ kubectl describe pod apache | grep Image

    Image:          docker.io/httpd:latest
    Image ID:
docker.io/library/httpd@sha256:5123fb6e039b83a4319b668b4fe1ee04c4fbd7c4c8d
1d6ef843e8a943a9aed3f

~$
```

- Are there any labels?

```
~$ kubectl describe pod apache | grep Labels

Labels:       run=apache

~$
```

The `run` label is attached to every pod created using `kubectl run` with the pod's name as the value.

- Using the `kubectl delete -h` help page, try to delete the new `apache` pod using any labels you find

```
~$ kubectl delete -h | grep label

Delete resources by file names, stdin, resources and names, or by
resources and label selector.
 JSON and YAML formats are accepted. Only one type of argument may be
specified: file names, resources and names, or resources and label
selector.
  # Delete pods and services with label name=myLabel
  -l, --selector='': Selector (label query) to filter on, not including
uninitialized ones.
  kubectl delete ([-f FILENAME] | [-k DIRECTORY] | TYPE [(NAME | -l label
| --all)]) [options]

~$ kubectl delete pod -l run=apache

pod "apache" deleted

~$
```

5. CHALLENGE: a complex pod

Next let's try creating a pod with a more complex specification.

Create a pod config that describes a pod called `hello` with a:

- container based on an `docker.io/ubuntu:14.04` image
- with an environment variable called `MESSAGE` and a value `hello world`
- Runs the command: `/bin/sh -c "echo $MESSAGE"`
- make sure that the container is *never restarted*

This challenge can be completed imperatively with `kubectl run` flags:

```
~$ kubectl run hello \
--image ubuntu:14.04 \
--env MESSAGE="hello world" \
--restart Never \
-o yaml --dry-run=client \
--command -- /bin/sh -c "echo \$MESSAGE" \
> complex-pod.yaml
```

`complex-pod.yaml` declaratively may also look like:

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: hello
```

```
    name: hello
  spec:
    containers:
    - command:
      - /bin/sh
      - -c
      - echo $MESSAGE
      env:
      - name: MESSAGE
        value: hello world
      image: ubuntu:14.04
      name: hello
      resources: {}
    dnsPolicy: ClusterFirst
    restartPolicy: Never
  status: {}
```

Separating the `command` and `args` arrays is a best practice for readability; however there is no flag for `args` in the `kubectl run` command so the entire shell and echo is stated with the `--command` flag.

Check for success by running `kubectl logs hello` after applying.