Lab 5.1 - Configuring the Deployment

Overview

In this lab, we will add resources to our deployment with further configuration you may need for production. We'll also work with updating deployed applications and automation of batch jobs and regular tasks.

Save a copy of your **~/app1/simpleapp.yaml** file, in case you would like to repeat portions of the labs, or you find your file difficult to use due to typos and whitespace issues.

```
student@ckad-1:~$ cp ~/app1/simpleapp.yaml ~/beforeLab5.yaml
```

Secrets and ConfigMap

There are three different ways a ConfigMap can ingest data:

- From a literal value
- From a file
- From a directory of files.

Create a ConfigMap containing primary colors. We will create a series of files to ingest into the ConfigMap. First, create a directory **primary** and populate it with four files. Then, we create a file in our home directory with our favorite color:

```
student@ckad-1:~/app1$ cd
student@ckad-1:~$ mkdir primary
student@ckad-1:~$ echo c > primary/cyan
student@ckad-1:~$ echo m > primary/magenta
```

```
student@ckad-1:~$ echo y > primary/yellow
student@ckad-1:~$ echo k > primary/black
student@ckad-1:~$ echo "known as key" >> primary/black
student@ckad-1:~$ echo blue > favorite
```

Generate a **configmap** using each of the three methods:

```
student@ckad-1:~$ kubectl create configmap colors \
 --from-literal=text=black \
 --from-file=./favorite \
 --from-file=./primary/
configmap/colors created
```

View the newly created **configmap**. Note the way the ingested data is presented:

```
student@ckad-1:~$ kubectl get configmap colors
NAME      DATA      AGE
colors    6         11s
```

```
student@ckad-1:~$ kubectl get configmap colors -o yaml
apiVersion: v1
data:
  black: |
    k
    known as key
  cyan: |
    c
  favorite: |
    blue
  magenta: |
    m
  text: black
  yellow: |
    y
kind: ConfigMap
metadata:
  creationTimestamp: 2018-04-05T19:49:59Z
  name: colors
  namespace: default
  resourceVersion: "13491"
  selfLink: /api/v1/namespaces/default/configmaps/colors
  uid: 86457ce3-390a-11e8-ba73-42010a800003
```

Update the YAML file of the application to make use of the **configmap** as an environmental parameter. Add the six lines from the **env:** line to **key:favorite**.

```
student@ckad-1:~$ vim app1/simpleapp.yaml
….
  spec:
   containers:
   - image: 10.105.119.236:5000/simpleapp:latest
    env:                                    #Add from here
    - name: ilike
     valueFrom:
      configMapKeyRef:
       name: colors
       key: favorite                        #To here
    imagePullPolicy: Always
….
```

Delete and re-create the deployment with the new parameters:

```
student@ckad-1-lab-7xtx:~$ kubectl delete deployment try1
deployment.extensions "try1" deleted

student@ckad-1-lab-7xtx:~$ kubectl create -f app1/simpleapp.yaml
deployment.extensions/try1 created
```

Even though the **try1** container is not in a **ready** state, it is running and useful. Use **kubectl exec** to view a variable's value. View the pod state, then verify you can see the **ilike** value within.

```
student@ckad-1:~$ kubectl get po
<output_omitted>
student@ckad-1:~$ kubectl exec -c try1 -it try1-5db9bc6f85-whxbf -- \
   /bin/bash -c 'echo $ilike'
blue
```

Edit the YAML file again, this time adding the third method of using a ConfigMap. Edit the file to add three lines. **envFrom** should be indented the same amount as **env** earlier in the file, and **configMapRef** should be indented the same as **configMapKeyRef**.

```
student@ckad-1:~$ vim app1/simpleapp.yaml
….

     configMapKeyRef:
       name: colors
```

```
              key: favorite
        envFrom:                  #Add this and the following two lines
        - configMapRef:
            name: colors
        imagePullPolicy: Always
….
```

Again delete and recreate the deployment. Check that the pods restart:

```
student@ckad-1:~$ kubectl delete deployment try1
deployment.extensions "try1" deleted

student@ckad-1:~$ kubectl create -f app1/simpleapp.yaml
deployment.extensions/try1 created

student@ckad-1:~$ kubectl get pods
NAME                         READY     STATUS       RESTARTS    AGE
nginx-6b58d9cdfd-9fnl4       1/1       Running      1           23h
registry-795c6c8b8f-hl5wf    1/1       Running      2           23h
try1-d4fbf76fd-46pkb         1/2       Running      0           40s
try1-d4fbf76fd-9kw24         1/2       Running      0           39s
try1-d4fbf76fd-bx9j9         1/2       Running      0           39s
try1-d4fbf76fd-jw8g7         1/2       Running      0           40s
try1-d4fbf76fd-lppl5         1/2       Running      0           39s
try1-d4fbf76fd-xtfd4         1/2       Running      0           40s
```

View the settings inside the **try1** container of a pod. The following output is truncated in a few places. Omit the container name, to observe the behavior. Also, execute a command to see all environmental variables instead of logging into the container first:

```
student@ckad-1:~$ kubectl exec -it try1-d4fbf76fd-46pkb -- /bin/bash -c
'env'
Defaulting container name to try1.
Use 'kubectl describe pod/try1-d4fbf76fd-46pkb -n default' to see all of
the containers in this pod.
REGISTRY_PORT_5000_TCP_ADDR=10.105.119.236
HOSTNAME=try1-d4fbf76fd-46pkb
TERM=xterm
yellow=y
<output_omitted>
REGISTRY_SERVICE_HOST=10.105.119.236
KUBERNETES_SERVICE_PORT=443
REGISTRY_PORT_5000_TCP=tcp://10.105.119.236:5000
```

```
KUBERNETES_SERVICE_HOST=10.96.0.1
text=black
REGISTRY_SERVICE_PORT_5000=5000
<output_omitted>
black=k
known as key

<output_omitted>
ilike=blue
<output_omitted>
magenta=m

cyan=c
<output_omitted>
```

For greater flexibility and scalability, ConfigMaps can be created from a YAML file, then deployed and redeployed as necessary. Once ingested into the cluster, the data can be retrieved in the same manner as any other object. Create another ConfigMap, this time from a YAML file:

```
student@ckad-1:~$ vim car-map.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: fast-car
  namespace: default
data:
  car.make: Ford
  car.model: Mustang
  car.trim: Shelby

student@ckad-1:~$ kubectl create -f car-map.yaml
configmap/fast-car created
```

View the ingested data, and note that the output is just as in file created:

```
student@ckad-1:~$ kubectl get configmap fast-car -o yaml
apiVersion: v1
data:
  car.make: Ford
  car.model: Mustang
  car.trim: Shelby
kind: ConfigMap
metadata:
  creationTimestamp: 2018-07-26T16:36:32Z
  name: fast-car
  namespace: default
  resourceVersion: "105700"
  selfLink: /api/v1/namespaces/default/configmaps/fast-car
  uid: aa19f8f3-39b8-11e8-ba73-42010a800003
```

Add the **configMap** settings to the **simpleapp.yaml** file as a volume. Both containers in the **try1** deployment can access to the same volume, using the **volumeMounts** statements. Remember that the **volume** stanza is of equal depth to the **containers** stanza, and should probably come after for readability:

```
student@ckad-1:~$ vim app1/simpleapp.yaml
….
    spec:
      containers:
      - image: 10.105.119.236:5000/simpleapp:latest
        volumeMounts:
        - mountPath: /etc/cars
          name: car-vol
       name: car-vol
        imagePullPolicy: Always
        name: try1
….
          initialDelaySeconds: 15
          periodSeconds: 20
      volumes:                    #Add this and the following four lines
      - configMap:
          defaultMode: 420
          name: fast-car
        name: car-vol
         dnsPolicy: ClusterFirst
         restartPolicy: Always
….
```

Delete and recreate the deployment:

```
student@ckad-1:~$ kubectl delete deployment try1
deployment.extensions "try1" deleted

student@ckad-1:~$ kubectl create -f app1/simpleapp.yaml
deployment.extensions/try1 created
```

Verify the deployment is running. Note that we still have not automated the creation of the **`/tmp/healthy`** file inside the container; as a result, the **`AVAILABLE`** count remains zero until we use the for loop to create the file. We will remedy this in the next step.

```
student@ckad-1:~$ kubectl get deployment
NAME        DESIRED    CURRENT    UP-TO-DATE    AVAILABLE    AGE
nginx       1          1          1             1            1d
registry    1          1          1             1            1d
try1        6          6          6             0            39s
```

Our health check was the successful execution of a command. We will edit the command of the existing *readinessProbe* to check for the existence of the mounted **`configMap`** file and re-create the deployment. After a minute, both containers should become available for each pod in the deployment:

```
student@ckad-1:~$ kubectl delete deployment try1
deployment.extensions "try1" deleted


student@ckad-1:~$ vim app1/simpleapp.yaml
….
        readinessProbe:
          exec:
            command:
            - ls                       #Add this and the following line.
            - /etc/cars
          periodSeconds: 5
….

student@ckad-1:~$ kubectl create -f app1/simpleapp.yaml
deployment.extensions/try1 created
```

Wait about a minute and view the deployment and pods. All six replicas should be running and report that 2/2 containers are in a **`ready`** state within:

```
student@ckad-1:~$ kubectl get deployment
NAME        DESIRED    CURRENT    UP-TO-DATE    AVAILABLE    AGE
nginx       1          1          1             1            1d
registry    1          1          1             1            1d
try1        6          6          6             6            1m

student@ckad-1:~$ kubectl get pods
NAME                           READY      STATUS     RESTARTS    AGE
nginx-6b58d9cdfd-9fnl4         1/1        Running    1           1d
```

```
registry-795c6c8b8f-hl5wf   1/1       Running   2          1d
try1-7865dcb948-2dzc8       2/2       Running   0          1m
try1-7865dcb948-7fkh7       2/2       Running   0          1m
try1-7865dcb948-d85bc       2/2       Running   0          1m
try1-7865dcb948-djrcj       2/2       Running   0          1m
try1-7865dcb948-kwlv8       2/2       Running   0          1m
try1-7865dcb948-stb2n       2/2       Running   0          1m
```

View a file within the new volume mounted in a container. It should match the data we created inside the **`configMap`**. Because the file did not have a carriage-return, it will appear prior to the following prompt:

```
student@ckad-1:~$ kubectl exec -c try1  -it try1-7865dcb948-stb2n --
/bin/bash \
-c 'cat /etc/cars/car.trim'
Shelbystudent@ckad-1:~$
```

Attaching Storage

There are several types of storage which can be accessed with Kubernetes, with flexibility of storage being essential to scalability. In this exercise, we will configure an NFS server. With the NFS server, we will create a new *persistent volume (pv)* and a *persistent volume claim (pvc)* to use it.

Use the **`CreateNFS.sh`** script from the tarball to set up NFS on your master node. This script will configure the server, export **`/opt/sfw`** and create a file **`/opt/sfw/hello.txt`**.

```
student@ckad-1:~$ bash lfd259/CreateNFS.sh
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu xenial InRelease
Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu xenial-updates
InRelease [102 kB]

<output_omitted>

Should be ready. Test here and second node

Export list for localhost:
/opt/sfw *
```

Test by mounting the resource from your **second** node. Begin by installing the client software:

```
student@ckad-2:~$ sudo apt-get -y install nfs-common nfs-kernel-server
<output_omitted>
```

Test that you can see the exported directory using **showmount** from your second node:

```
student@ckad-2:~$ showmount -e ckad-1   ## First node’s name or IP
Export list for ckad-1:
/opt/sfw *
```

Mount the directory. Be aware that, unless you edit **/etc/fstab**, this is not a persistent mount. Change out the node name for that of your master node:

```
student@ckad-2:~$ sudo mount ckad-1:/opt/sfw /mnt
```

Verify the **hello.txt** file created by the script can be viewed:

```
student@ckad-2:~$ ls -l /mnt
total 4
-rw-r--r-- 1 root root 9 Sep 28 17:55 hello.txt
```

Return to the master node and create a YAML file for an object with kind **PersistentVolume**. The included example file needs an edit to the server parameter. Use the hostname of the master server and the directory you created in the previous step. Only syntax is checked, an incorrect name or directory will not generate an error, but a Pod using the incorrect resource will not start. Note that the **accessModes** do not currently affect actual access, and are typically used as labels instead:

```
student@ckad-1:~$ cd lfd259; vim PVol.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pvvol-1
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /opt/sfw
    server: ckad-1   #<-- Edit to match your master node name
    readOnly: false
```

Create and verify you have a new 1Gi volume named **pvvol-1**. Note the status shows as **Available**. Remember we made two persistent volumes for the image registry earlier.

```
student@ckad-1:~/lfd259$ kubectl create -f PVol.yaml
persistentvolume/pvvol-1 created

student@ckad-1:~/lfd259$ kubectl get pv
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                        STORAGECLASS   REASON     AGE
pvvol-1             1Gi        RWX            Retain           Available
4s
registryvm          200Mi      RWO            Retain           Bound
default/nginx-claim0                                   4d
task-pv-volume      200Mi      RWO            Retain           Bound
default/registry-claim0                                4d
```

Now that we have a new volume, we will use a persistent volume claim (pvc) to use it in a Pod. We should have two existing claims from our local registry:

```
student@ckad-1:~/lfd259$ kubectl get pvc
NAME                 STATUS     VOLUME           CAPACITY   ACCESS MODES
STORAGECLASS    AGE
nginx-claim0         Bound      registryvm       200Mi      RWO
4d
registry-claim0      Bound      task-pv-volume   200Mi      RWO
4d
```

Create a YAML file with the kind **PersistentVolumeClaim**.

```
student@ckad-1:~/lfd259$ vim pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-one
spec:
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 200Mi
```

Create and verify the new **pvc** status is **bound**. Note the size is 1Gi, even though 200Mi was suggested. Only a volume of at least that size could be used, so the smallest available was chosen.

```
student@ckad-1:~/lfd259$ kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-one created

student@ckad-1:~/LFD259$ kubectl get pvc
NAME                 STATUS    VOLUME           CAPACITY   ACCESS MODES
STORAGECLASS   AGE
nginx-claim0         Bound     registryvm       200Mi      RWO
4d
pvc-one              Bound     pvvol-1          1Gi        RWX
4s
registry-claim0      Bound     task-pv-volume   200Mi      RWO
4d
```

Now, look at the status of the physical volume. It should also show as **bound**.

```
student@ckad-1:~/lfd259$ kubectl get pv ; cd
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS    CLAIM
STORAGECLASS   REASON   AGE
pvvol-1             1Gi        RWX            Retain           Bound
default/pvc-one                       14m
registryvm          200Mi      RWO            Retain           Bound
default/nginx-claim0                  4d
task-pv-volume      200Mi      RWO            Retain           Bound
default/registry-claim0               4d
```

Edit the **simpleapp.yaml** file to include two new sections. While one section for the container will use the volume mount point, you should have an existing entry for **car-vol.** The other section adds a volume to the deployment in general, which you can put after the **configMap volume** section.

```
student@ckad-1:~$ vim app1/simpleapp.yaml
<output_omitted>
     volumeMounts:
     - mountPath: /etc/cars
       name: car-vol
     - name: nfs-vol          ## Add these two lines
       mountPath: /opt        ##

<output_omitted>

     volumes:
      - configMap:
          defaultMode: 420
          name: fast-car
        name: car-vol
      - name: nfs-vol               ## Add these three lines
        persistentVolumeClaim:    ##
          claimName: pvc-one      ##
<output_omitted>
```

Delete and re-create the deployment:

```
student@ckad-1:~/app1$ kubectl delete deployment try1 ; kubectl create -f \
 simpleapp.yaml
deployment.extensions "try1" deleted
deployment.extensions/try1 created
```

View the details for any of the pods in the deployment: you should see **`nfs-vol`** mounted under **`/opt`**. The use to command line completion with the **`<Tab>`** key can be helpful for using a pod name.

```
student@ckad-1:~/app1$ kubectl describe pod try1-594fbb5fc7-5k7sj
<output_omitted>
    Mounts:
      /etc/cars from car-vol (rw)
      /opt from nfs-vol (rw)
<output_omitted>
```

Rolling Updates and Rollbacks

When we started working with **`simpleapp`**, we used a Docker tag called **`latest`**. While this is the default tag when pulling an image, and commonly used, it remains just a string, and it may not be the actual latest version of the image.

Make a slight change to our source and create a new image. We will use updates and rollbacks with our application. Adding a comment to the last line should be enough for a new image to be generated:

```
student@ckad-1:~$ cd ~/app1
student@ckad-1:~/app1$ vim simple.py
<output_omitted>
## Sleep for five seconds then continue the loop
```

```
  time.sleep(5)

## Adding a new comment so image is different.
```

Build the image again. A new container and image will be created. Verify when successful. There should be a different image ID and a recent creation time:

```
student@ckad-1:~/app1$ sudo docker build -t simpleapp .
Sending build context to Docker daemon 7.168 kB
Step 1/3 : FROM python:2
 ---> 2863c80c418c
Step 2/3 : ADD simple.py /
 ---> cde8ecf8492b
Removing intermediate container 3e908b76b5b4
Step 3/3 : CMD python ./simple.py
 ---> Running in 354620c97bf5
 ---> cc6bba0ea213
Removing intermediate container 354620c97bf5
Successfully built cc6bba0ea213

student@ckad-1:~/app1$ sudo docker images
REPOSITORY                                               TAG
IMAGE ID            CREATED             SIZE
simpleapp                                                latest
cc6bba0ea213        8 seconds ago       679 MB
10.105.119.236:5000/simpleapp                            latest
15b5ad19d313        4 days ago          679 MB
<output_omitted>
```

Tag and push the updated image to your locally hosted registry. A reminder that your IP address will be different than the example below. Use the tag **`v2`** this time, instead of **`latest`**.

```
student@ckad-1:~/app1$ sudo docker tag simpleapp
10.105.119.236:5000/simpleapp:v2

student@ckad-1:~/app1$ sudo docker push 10.105.119.236:5000/simpleapp:v2
The push refers to a repository [10.105.119.236:5000/simpleapp]
d6153c8cc7c3: Pushed
ca82a2274c57: Layer already exists
de2fbb43bd2a: Layer already exists
4e32c2de91a6: Layer already exists
6e1b48dc2ccc: Layer already exists
ff57bdb79ac8: Layer already exists
```

```
6e5e20cbf4a7: Layer already exists
86985c679800: Layer already exists
8fad67424c4e: Layer already exists
v2: digest:
sha256:6cf74051d09463d89f1531fceb9c44cbf99006f8d9b407dd91d8f07baeee7e9c
size: 2218
```

Connect to a terminal running on your **second** node. Pull the **`latest`** image, then pull **`v2`**. Note the **`latest`** did not pull the new version of the image. Again, remember to use the IP for your locally hosted registry. You'll note the digest is different:

```
student@ckad-2:~$ sudo docker pull 10.105.119.236:5000/simpleapp
Using default tag: latest
latest: Pulling from simpleapp
Digest:
sha256:cefa3305c36101d32399baf0919d3482ae8a53c926688be3386f9bbc04e490a5
Status: Image is up to date for 10.105.119.236:5000/simpleapp:latest

student@ckad-2:~$ sudo docker pull 10.105.119.236:5000/simpleapp:v2
v2: Pulling from simpleapp
f65523718fc5: Already exists
1d2dd88bf649: Already exists
c09558828658: Already exists
0e1d7c9e6c06: Already exists
c6b6fe164861: Already exists
45097146116f: Already exists
f21f8abae4c4: Already exists
1c39556edcd0: Already exists
fa67749bf47d: Pull complete
Digest:
sha256:6cf74051d09463d89f1531fceb9c44cbf99006f8d9b407dd91d8f07baeee7e9c
Status: Downloaded newer image for 10.105.119.236:5000/simpleapp:v2
```

Use **`kubectl edit`** to update the image for the **`try1`** deployment to use **`v2`**. As we are only changing one parameter, we could also use the **`kubectl set`** command. Note that the configuration file has not been updated, so a **`delete`** or a **`replace`** command would not include the new version. It can take the pods up to a minute to delete and to recreate each pod in sequence.

```
student@ckad-1:~/app1$ kubectl edit deployment try1
<output_omitted>
      containers:
      - image: 10.105.119.236:5000/simpleapp:v2
        imagePullPolicy: Always
```

```
<output_omitted>
```

Verify each of the pods has been recreated and is using the new version of the image. Note that some messages will show the scaling down of the old **replicaset**, others should show the scaling up using the new image:

```
student@ckad-1:~/app1$ kubectl get events
LAST SEEN   FIRST SEEN   COUNT     NAME
KIND         SUBOBJECT                  TYPE      REASON
SOURCE                  MESSAGE
4s          4s           1         try1-594fbb5fc7-nxhfx.152422073b7084da
Pod          spec.containers{goproxy}   Normal    Killing
kubelet, ckad-2-wdrq    Killing container with id docker://goproxy:Need to
kill Pod
<output_omitted>
2m          2m           1         try1.1524220c35a0d0fb
Deployment                              Normal    ScalingReplicaSet
deployment-controller   Scaled up replica set try1-895fccfb to 5
2m          2m           3         try1.1524220e0d69a94a
Deployment                              Normal    ScalingReplicaSet
deployment-controller   (combined from similar events): Scaled down replica
set try1-594fbb5fc7 to 0
```

View the images of a Pod in the deployment. Narrow the output to just view the images. The **goproxy** remains unchanged, but the **simpleapp** should now be **v2**:

```
student@ckad-1:~/app1$ kubectl describe pod try1-895fccfb-ttqdn |grep Image
    Image:          10.105.119.236:5000/simpleapp:v2
    Image ID:
docker-pullable://10.105.119.236:5000/simpleapp@sha256:6cf74051d09463d89f15
31fceb9c44cbf99006f8d9b407dd91d8f07baeee7e9c
    Image:          k8s.gcr.io/goproxy:0.1
    Image ID:
docker-pullable://k8s.gcr.io/goproxy@sha256:5334c7ad43048e3538775cb09aaf184
f5e8acf4b0ea60e3bc8f1d93c209865a5
```

View the update history of the deployment:

```
student@ckad-1:~/app1$ kubectl rollout history deployment try1
deployments "try1"
REVISION  CHANGE-CAUSE
```

```
1         <none>
2         <none>
```

Compare the output of the rollout history for the two revisions. Images and labels should be different, with the image **v2** being the change we made:

```
student@ckad-1:~/app1$ kubectl rollout history deployment try1 \
  --revision=1 > one.out

student@ckad-1:~/app1$ kubectl rollout history deployment try1 \
  --revision=2 > two.out

student@ckad-/app11:~$ diff one.out two.out
1c1
< deployments "try1" with revision #1
---
> deployments "try1" with revision #2
3c3
<   Labels:      pod-template-hash=1509661973
---
>   Labels:      pod-template-hash=45197796
7c7
<     Image:     10.105.119.236:5000/simpleapp:latest
---
>     Image:     10.105.119.236:5000/simpleapp:v2
```

View what would be undone using the **--dry-run** option while undoing the rollout. This allows us to see the new template prior to using it:

```
student@ckad-1:~/app1$ kubectl rollout undo --dry-run=true deployment/try1
deployment.extensions/try1
Pod Template:
  Labels:  pod-template-hash=1509661973
      run=try1
  Containers:
   try1:
    Image: 10.105.119.236:5000/simpleapp:latest
    Port:  <none>
<output_omitted>
```

View the pods. Depending on how fast you type, the **try1** pods should be about 2 minutes old:

```
student@ckad-1:~/app1$ kubectl get pods
```

```
NAME                        READY     STATUS    RESTARTS   AGE
nginx-6b58d9cdfd-9fnl4      1/1       Running   1          5d
registry-795c6c8b8f-hl5wf   1/1       Running   2          5d
try1-594fbb5fc7-7dl7c       2/2       Running   0          2m
try1-594fbb5fc7-8mxlb       2/2       Running   0          2m
try1-594fbb5fc7-jr7h7       2/2       Running   0          2m
try1-594fbb5fc7-s24wt       2/2       Running   0          2m
try1-594fbb5fc7-xfffg       2/2       Running   0          2m
try1-594fbb5fc7-zfmz8       2/2       Running   0          2m
```

In our case, there are only two revisions. Were there more, we could choose a particular version. The following command would have the same effect as the previous, without the **`--dry-run`** option.

```
student@ckad-1:~/app1$ kubectl rollout undo deployment try1 --to-revision=1
deployment.extensions/try1
```

Again, it can take a bit for the pods to be terminated and re-created. Keep checking back until they are all running again.

```
student@ckad-1:~/app1$ kubectl get pods
NAME                        READY     STATUS        RESTARTS   AGE
nginx-6b58d9cdfd-9fnl4      1/1       Running       1          5d
registry-795c6c8b8f-hl5wf   1/1       Running       2          5d
try1-594fbb5fc7-7dl7c       2/2       Terminating   0          3m
try1-594fbb5fc7-8mxlb       0/2       Terminating   0          2m
try1-594fbb5fc7-jr7h7       2/2       Terminating   0          3m
try1-594fbb5fc7-s24wt       2/2       Terminating   0          2m
try1-594fbb5fc7-xfffg       2/2       Terminating   0          3m
try1-594fbb5fc7-zfmz8       1/2       Terminating   0          2m
try1-895fccfb-8dn4b         2/2       Running       0          22s
try1-895fccfb-kz72j         2/2       Running       0          10s
try1-895fccfb-rxxtw         2/2       Running       0          24s
try1-895fccfb-srwq4         1/2       Running       0          11s
try1-895fccfb-vkvmb         2/2       Running       0          31s
try1-895fccfb-z46qr         2/2       Running       0          31s
```

Working with Jobs

We will create a simple cron job to explore how to create them and view their execution. We will run a regular job and view both the job status and output. Note that the jobs are expected to be idempotent, so should not be used for tasks that require strict timings to run. The **sleep 30** command will cause some jobs to finish in the next minute, as the job could start at any time during the minute.

Begin by creating a YAML file for the cron job. Set the time interval to be every minute. Use the **busybox** container and pass it the **date** command. We could just as easily use a **copy** command to backup output files from our **simpleapp**.

```
student@ckad-1:~/app1$ vim cron-job.yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: date
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name:  dateperminute
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; sleep 30
          restartPolicy: OnFailure
```

View the **cronjob**. Depending on the speed you type, one may not have run yet, as seen below:

```
student@ckad-1:~/app1$ kubectl get cronjob date
NAME      SCHEDULE      SUSPEND   ACTIVE    LAST SCHEDULE   AGE
date      */1 * * * *   False     0         <none>          4s
```

View the jobs as they run. Give it a couple of minutes. Note the successful jobs completed within the timeframe of the minute, but each eventually did finish. Use **<ctrl>-c** to stop the **--watch** option.

```
student@ckad-1:~/app1$ kubectl get jobs --watch
NAME              DESIRED   SUCCESSFUL   AGE
date-1523426280   1         0             2s
date-1523426280   1         1            32s
date-1523426340   1         0             0s
date-1523426340   1         0             0s
```

```
^C
```

View the pods; you should see at least a couple of completed pods:

```
student@ckad-1:~/app1$ kubectl get pods  | grep date
date-1523426280-zjwfm       0/1       Completed   0          1m
date-1523426340-hk897       0/1       Completed   0          42s
```

View the output of the job; you should see a recent time:

```
student@ckad-1:~/app1$ kubectl logs date-1523426340-hk897
Wed Apr 11 05:59:10 UTC 2018
```

Clean up by deleting the **cronjob**.

```
student@ckad-1:~/app1$ kubectl delete cronjob date
cronjob.batch "date" deleted
```