Lab 3.1 - Deploying a New Application

Overview

In this lab, we will deploy a very simple Python application, test it using Docker, ingest it into Kubernetes, and configure probes to ensure it continues to run. This lab requires the completion of the previous lab, the installation and configuration of a Kubernetes cluster.

Working with Python

Install Python on you master node. It may already be installed, as is shown in the output below:

```
student@ckad-1:~$ sudo apt-get -y install python
Reading package lists... Done
Building dependency tree
Reading state information... Done
python is already the newest version (2.7.12-1~16.04).
python set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
student@ckad-1:~$
```

Locate the Python binary on your system:

```
student@ckad-1:~$ which python
/usr/bin/python
```

Create a new directory and change into it. The **`docker build`** process pulls everything from the current directory into the image file by default. Make sure the chosen directory is empty:


```
student@ckad-1:~$ mkdir app1

student@ckad-1:~$ cd app1

student@ckad-1:~/app1$ ls -l
total 0
```

Create a simple Python script which prints the time and hostname every 5 seconds. There are six commented parts to this script, which should explain what each part is meant to do. The script is included with others in the course tar file, though you are encouraged to create the file by hand if you are not already familiar with the process:

```
student@ckad-1:~/app1$ vim simple.py
#!/usr/bin/python
## Import the necessary modules
import time
import socket

## Use an ongoing while loop to generate output
while True :

## Set the hostname and the current date
  host = socket.gethostname()
  date = time.strftime("%Y-%m-%d %H:%M:%S")

## Convert the date output to a string
  now = str(date)

## Open the file named date in append mode
## Append the output of hostname and time
  f = open("date.out", "a" )
  f.write(now + "\n")
  f.write(host + "\n")
  f.close()

## Sleep for five seconds then continue the loop
  time.sleep(5)
```

Make the file executable and test that it works. Use **<ctrl-c>** to interrupt the **while** loop after 20 or 30 seconds. The output will be sent to a newly created file in your current directory called **date.out**.

```
student@ckad-1:~/app1$ chmod +x simple.py
```



```
student@ckad-1:~/app1$ ./simple.py
^CTraceback (most recent call last):
  File "./simple.py", line 42, in <module>
    time.sleep(5)
KeyboardInterrupt
```

View the **date.out** file. It should contain the hostname and timedate stamps:

```
student@ckad-1:~/app1$   cat date.out
2018-03-22 15:51:38
ckad-1
2018-03-22 15:51:43
ckad-1
2018-03-22 15:51:48
ckad-1
<output_omitted>
```

Create a Dockerfile. Note the name is important; it cannot have a suffix. We will use three statements: **FROM** to declare which version of Python to use, **ADD** to include our script, and **CMD** to indicate the action of the container. Should you be including more complex tasks, you may need to install extra libraries, shown commented out as **RUN pip install** in the following example:

```
student@ckad-1:~/app1$ vim Dockerfile
FROM python:2
ADD simple.py /
## RUN pip install pystrich
CMD [ "python", "./simple.py" ]
```

Build the container. The output below shows mid-build, as necessary software is downloaded. You will need to use **sudo** in order to run this command. After the three-step process completes, the last line of output should indicate success.

```
student@ckad-1:~/app1$ sudo docker build -t simpleapp .
Sending build context to Docker daemon 3.072 kB
Step 1/3 : FROM python:2
2: Pulling from library/python
4176fe04cefe: Pull complete
851356ecf618: Pull complete
6115379c7b49: Pull complete
aaf7d781d601: Extracting [==================>                    ]
54.03 MB/135 MB
40cf661a3cc4: Download complete
```



```
c582f0b73e63: Download complete
6c1ea8f72a0d: Download complete
7051a41ae6b7: Download complete
<output_omitted>
Successfully built c4e0679b9c36
```

Verify you can see the new image among others downloaded during the build process, installed to support the cluster, or you may have already worked with. The newly created **simpleapp** image should be listed first:

```
student@ckad-1:~/app1$ sudo docker images
REPOSITORY              TAG       IMAGE ID         CREATED          SIZE
simpleapp               latest    c4e0679b9c36     2 minutes ago    681 MB
quay.io/calico/node     v2.6.8    e96a297310fd     13 days ago      282 MB
python                  2         d8690ef56706     2 weeks ago      681 MB

<output_omitted>
```

Use Docker to run a container using the new image. While the script is running, you won't see any output and the shell will be occupied running the image in the background. After 30 seconds, use **<ctrl>-c** to interrupt. The local **date.out** file will not be updated with new times; instead, that output will be a file of the container image.

```
student@ckad-1:~$ sudo docker run simpleapp
^CTraceback (most recent call last):
  File "./simple.py", line 24, in <module>
    time.sleep(5)
KeyboardInterrupt
```

Locate the newly created **date.out** file. The following command should show two files of this name, the one created when we ran **simple.py** and another under **/var/lib/docker** when run via a Docker container:

```
student@ckad-1:~/app1$ sudo find / -name date.out
/home/student/app1/date.out
/var/lib/docker/aufs/diff/ee814320c900bd24fad0c5db4a258d3c2b78a19cde629d7de
7d27270d6a0c1f5/date.out
```


View the contents of the **date.out** file created via Docker. Note the need for **sudo**, as Docker created the file this time, and the owner is root. The long name is shown on several lines in the example, but would be a single line when typed or copied.

```
student@ckad-1:~/app1$ sudo tail \
/var/lib/docker/aufs/diff/ee814320c900bd24fa\
d0c5db4a258d3c2b78a19cde629d7de7d27270d6a0c1f5/date.out
2018-03-22 16:13:46
53e1093e5d39
2018-03-22 16:13:51
53e1093e5d39
2018-03-22 16:13:56
53e1093e5d39
```

Configure A Local Docker Repository

While we could create an account and upload our application to hub.docker.com, thus sharing it with the world, we will instead create a local repository and make it available to the nodes of our cluster.

We'll need to complete a few steps with special permissions; for ease of use, we'll become root using **sudo**:

```
student@ckad-1:~/app1$ cd
student@ckad-1:~$ sudo -i
```

Install the **docker-compose** software and utilities to work with the **nginx** server, which will be deployed with the registry:

```
root@ckad-1:~# apt-get install -y docker-compose apache2-utils
<output_omitted>
```

Create a new directory for configuration information. We'll be placing the repository in the root filesystem. A better location may be chosen in a production environment.

```
root@ckad-1:~# mkdir -p /localdocker/data

root@ckad-1:~# cd /localdocker/
```


Create a **docker-compose** file. Inside is an entry for the **nginx** web server to handle outside traffic, and a **registry** entry listening to loopback port 5000 for running a local Docker registry.

```
root@ckad-1:/localdocker# vim docker-compose.yaml
nginx:
  image: "nginx:1.12"
  ports:
    - 443:443
  links:
    - registry:registry
  volumes:
    - /localdocker/nginx/:/etc/nginx/conf.d
registry:
  image: registry:2
  ports:
    - 127.0.0.1:5000:5000
  environment:
    REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY: /data
  volumes:
    - /localdocker/data:/data
```

Use the **docker-compose up** command to create the containers declared in the previous step YAML file. This will capture the terminal and run until you use **<ctrl>-c** to interrupt. There should be five **registry_1** entries with **info** messages about memory and which port is being listened to. Once we're sure the **docker** file works, we'll convert to a Kubernetes tool.

```
root@ckad-1:/localdocker# docker-compose up
Pulling nginx (nginx:1.12)...
1.12: Pulling from library/nginx
2a72cbf407d6: Pull complete
f37cbdc183b2: Pull complete
78b5ad0b466c: Pull complete
Digest:
sha256:edad623fc7210111e8803b4359ba4854e101bcca1fe7f46bd1d35781f4034f0c
Status: Downloaded newer image for nginx:1.12
Creating localdocker_registry_1
Creating localdocker_nginx_1
Attaching to localdocker_registry_1, localdocker_nginx_1
registry_1  | time="2018-03-22T18:32:37Z" level=warning msg="No HTTP secret
provided - generated ran
<output_omitted>
```


Test that you can access the repository. Open a second terminal to the master node. Use the **curl** command to test the repository. It should return **{},** but does not have a carriage-return, so will be on the same line as the following prompt. You should also see the GET request in the first captured terminal, without error. Don't forget the trailing slash. You'll see a **"Moved Permanently"** message if the path doesn't match exactly.

```
student@ckad-1:~/localdocker$ curl http://127.0.0.1:5000/v2/
{}student@ckad-1:~/localdocker$
```

Now that we know that **docker-compose** format is working, ingest the file into Kubernetes using **kompose**. Use **<ctrl-c>** to stop the previous **docker-compose.**

```
^CGracefully stopping... (press Ctrl+C again to force)
Stopping localdocker_nginx_1 ... done
Stopping localdocker_registry_1 ... done
```

Download the **kompose** binary and make it executable:

```
root@ckad-1:/localdocker# curl -L
https://github.com/kubernetes/kompose/releases/download/v1.1.0/kompose-linu
x-amd64 -o kompose

  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
100   609    0   609    0     0   1963      0 --:--:-- --:--:-- --:--:--
1970
100 45.3M  100 45.3M    0     0  16.3M      0  0:00:02  0:00:02 --:--:--
25.9M

root@ckad-1:/localdocker# chmod +x kompose
```

Move the binary to a directory in our **$PATH**. Then, return to your non-root user:

```
root@ckad-1:/localdocker# mv ./kompose /usr/local/bin/kompose

root@ckad-1:/localdocker# exit
```

Create two physical volumes in order to deploy a local registry for Kubernetes. 200mi for each should be enough for each of the volumes. More details on how persistent volumes and persistent volume claims are covered in an upcoming chapter.


```
student@ckad-1:~$ vim vol1.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    type: local
  name: task-pv-volume
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 200Mi
  hostPath:
    path: /tmp/data
  persistentVolumeReclaimPolicy: Retain

student@ckad-1:~$ vim vol2.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    type: local
  name: registryvm
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 200Mi
  hostPath:
    path: /tmp/nginx
  persistentVolumeReclaimPolicy: Retain
```

Create both volumes:

```
student@ckad-1:~$ kubectl create -f vol1.yaml
persistentvolume "task-pv-volume" created

student@ckad-1:~$ kubectl create -f vol2.yaml
persistentvolume "registryvm" created
```

Verify that both volumes have been created. They should show an **`Available`** status:


```
student@ckad-1:~$ kubectl get pv
NAME              CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM     STORAGECLASS   REASON    AGE
registryvm        200Mi      RWO            Retain           Available
27s
task-pv-volume    200Mi      RWO            Retain           Available
32s
```

Go to the configuration file for a **localdocker** registry:

```
student@ckad-1:~$ cd /localdocker/

student@ckad-1:~/localdocker$ ls
data  docker-compose.yaml  nginx
```

Convert the Docker file into a single YAML file for use with Kubernetes. Not all objects convert exactly from Docker to kompose; you will get errors about the mount syntax for the new volumes. They can be safely ignored.

```
student@ckad-1:~/localdocker$ sudo kompose convert -f docker-compose.yaml \
-o localregistry.yaml
WARN Volume mount on the host "/localdocker/nginx/" isn't supported -
ignoring path on the host
WARN Volume mount on the host "/localdocker/data" isn't supported -
ignoring path on the host
```

Review the file. You'll find that multiple objects will be created as well:

```
student@ckad-1:/localdocker$ less localregistry.yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: Service
  metadata:
    annotations:
      kompose.cmd: kompose convert -f docker-compose.yaml -o
localregistry.yaml
      kompose.version: 1.1.0 (36652f6)
    creationTimestamp: null
    labels:
<output_omitted>
```


View the cluster resources prior to deploying the registry. Only the cluster service and two available persistent volumes should exist in the **default** namespace:

```
student@ckad-1:~/localdocker$ kubectl get pods,svc,pvc,pv,deploy
NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   4h

NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM     STORAGECLASS   REASON    AGE
registryvm          200Mi      RWO            Retain           Available
15s
task-pv-volume      200Mi      RWO            Retain           Available
17s
```

Use **kubectl** to create a local Docker registry:

```
student@ckad-1:~/localdocker$ kubectl create -f localregistry.yaml
service "nginx" created
service "registry" created
deployment.extensions "nginx" created
persistentvolumeclaim "nginx-claim0" created
deployment.extensions "registry" created
persistentvolumeclaim "registry-claim0" created
```

View the newly deployed resources. The persistent volumes should now show as **Bound**. Find the service IP for the registry. It should be sharing port 5000. In the example below, the IP address is 10.110.186.162, but yours may be different:

```
student@ckad-1:~/localdocker$ kubectl get pods,svc,pvc,pv,deploy
NAME                              READY     STATUS    RESTARTS   AGE
pod/nginx-6b58d9cdfd-95zxq        1/1       Running   0          1m
pod/registry-795c6c8b8f-b8z4k     1/1       Running   0          1m

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP
1h
service/nginx        ClusterIP   10.106.82.218    <none>        443/TCP
1m
service/registry     ClusterIP   10.110.186.162   <none>         5000/TCP
1m
```



```
NAME                                        STATUS    VOLUME           CAPACITY
ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/nginx-claim0          Bound     registryvm       200Mi
RWO                           1m
persistentvolumeclaim/registry-claim0       Bound     task-pv-volume   200Mi
RWO                           1m

NAME                                CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS    CLAIM                       STORAGECLASS   REASON    AGE
persistentvolume/registryvm         200Mi      RWO            Retain
Bound     default/nginx-claim0                                  5m
persistentvolume/task-pv-volume     200Mi      RWO            Retain
Bound     default/registry-claim0                               6m

NAME                              DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
AGE
deployment.extensions/nginx       1         1         1            1
1m
deployment.extensions/registry    1         1         1            1
1m
```

Verify you get the same `{}` response using the Kubernetes deployed registry, as we did when using `docker-compose`.

Note you must use the trailing slash after `v2`. Please also note that, if the connection hangs, it may be due to a firewall issue. If running your nodes using GCE, ensure your instances are using VPC setup and all ports are allowed. If using AWS, also make sure all ports are being allowed.

```
student@ckad-1:~/localdocker$ curl http://10.110.186.162:5000/v2/
{}student@ckad-1:~/localdocker$
```

Edit the Docker configuration file to allow insecure access to the registry. In a production environment, steps should be taken to create and use TLS authentication instead. Use the IP and port of the registry:

```
student@ckad-1:~$ sudo vim /etc/docker/daemon.json
{ "insecure-registries":["10.110.186.162:5000"] }
```

Restart Docker on the local system. It can take up to a minute for the restart to take place:

```
student@ckad-1:~$ sudo systemctl restart docker.service
```


Download and tag a typical image from hub.docker.com. Tag the image using the IP and port of the registry. We will also use the **latest** tag.

```
student@ckad-1:~$ sudo docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
<output_omitted>
Digest:
sha256:9ee3b83bcaa383e5e3b657f042f4034c92cdd50c03f73166c145c9ceaea9ba7c
Status: Downloaded newer image for ubuntu:latest

student@ckad-1:~$ sudo docker tag ubuntu:latest 10.110.186.162:5000/tagtest
```

Push the newly tagged image to your local registry. If you receive an error about an HTTP request to an HTTPS client, check that you edited the **/etc/docker/daemon.json** file correctly and restarted the service:

```
student@ckad-1:~$ sudo docker push 10.110.186.162:5000/tagtest
The push refers to a repository [10.110.186.162:5000/tagtest]
db584c622b50: Pushed
52a7ea2bb533: Pushed
52f389ea437e: Pushed
88888b9b1b5b: Pushed
a94e0d5a7c40: Pushed
latest: digest:
sha256:0847cc7fed1bfafac713b0aa4ddfb8b9199a99092ae1fc4e718cb28e8528f65f
size: 1357
```

We will test to make sure we can also pull images from our local repository. Begin by removing the local cached images:

```
student@ckad-1:~$ sudo docker image remove ubuntu:latest
Untagged: ubuntu:latest
Untagged:
ubuntu@sha256:e348fbbea0e0a0e73ab0370de151e7800684445c509d46195aef73e090a49
bd6

student@ckad-1:~$ sudo docker image remove 10.110.186.162:5000/tagtest
Untagged: 10.110.186.162:5000/tagtest:latest
Untagged:
10.110.186.162:5000/tagtest@sha256:0847cc7fed1bfafac713b0aa4ddfb8b9199a9909
2ae1fc4e718cb28e8528f65f
```



```
Deleted:
sha256:f975c50357489439eb9145dbfa16bb7cd06c02c31aa4df45c77de4d2baa4e232
Deleted:
sha256:0bd983fc698ee9453dd7d21f8572ea1016ec9255346ceabb0f9e173b4348644f
Deleted:
sha256:08fe90e1a1644431accc00cc80f519f4628dbf06a653c76800b116d3333d2b6d
Deleted:
sha256:5dc5eef2b94edd185b4d39586e7beb385a54b6bac05d165c9d47494492448235
Deleted:
sha256:14a40a140881d18382e13b37588b3aa70097bb4f3fb44085bc95663bdc68fe20
Deleted:
sha256:a94e0d5a7c404d0e6fa15d8cd4010e69663bd8813b5117fbad71365a73656df9
```

Pull the image from the local registry. It should report the download of a newer image:

```
student@ckad-1:~$ sudo docker pull 10.110.186.162:5000/tagtest
Using default tag: latest
latest: Pulling from tagtest
bf8f2f4f7b8b: Pull complete
4288a6810024: Pull complete
bc5512367466: Pull complete
aa9fdb4f8e2a: Pull complete
665607941289: Pull complete
Digest:
sha256:0847cc7fed1bfafac713b0aa4ddfb8b9199a99092ae1fc4e718cb28e8528f65f
Status: Downloaded newer image for 10.110.186.162:5000/tagtest:latest
```

Use **`docker tag`** to assign the **`simpleapp`** image, and then **`push`** it to the local registry. The image and dependent images should be pushed to the local repository:

```
student@ckad-1:~$ sudo docker tag simpleapp 10.110.186.162:5000/simpleapp

student@ckad-1:~$ sudo docker push 10.110.186.162:5000/simpleapp
The push refers to a repository [10.110.186.162:5000/simpleapp]
321938b97e7e: Pushed
ca82a2274c57: Pushed
de2fbb43bd2a: Pushed
4e32c2de91a6: Pushed
6e1b48dc2ccc: Pushed
ff57bdb79ac8: Pushed
6e5e20cbf4a7: Pushed
86985c679800: Pushed
8fad67424c4e: Pushed
```



```
latest: digest:
sha256:67ea3e11570042e70cdcbad684a1e2986f59aaf53703e51725accdf5c70d475a
size: 2218
```

Configure the second minion node to use the local registry running on the master server. Connect to the minion node. Edit the Docker file with the same values from the master node, and restart the service:

```
student@ckad-2:~$ sudo vim /etc/docker/daemon.json
{ "insecure-registries":["10.110.186.162:5000"] }

student@ckad-2:~$ sudo systemctl restart docker.service
```

Pull the recently pushed image from the registry running on the master node:

```
student@ckad-2:~$ sudo docker pull 10.110.186.162:5000/simpleapp
Using default tag: latest
latest: Pulling from simpleapp
f65523718fc5: Pull complete
1d2dd88bf649: Pull complete
c09558828658: Pull complete
0e1d7c9e6c06: Pull complete
c6b6fe164861: Pull complete
45097146116f: Pull complete
f21f8abae4c4: Pull complete
1c39556edcd0: Pull complete
85c79f0780fa: Pull complete
Digest:
sha256:67ea3e11570042e70cdcbad684a1e2986f59aaf53703e51725accdf5c70d475a
Status: Downloaded newer image for 10.110.186.162:5000/simpleapp:latest
```

Return to the master node and deploy the **simpleapp** in **kubernetes** with several replicas. We will name the deployment **try1**. With multiple replicas, the scheduler should run some containers on each node:

```
student@ckad-1:~$ kubectl run try1 \
 --image=10.110.186.162:5000/simpleapp:latest \
--replicas=6
deployment.apps "try1" created
```

View the running pods. You should see six replicas of **simpleapp**, as well as two running the locally hosted image repository:


```
student@ckad-1:~$ kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
nginx-6b58d9cdfd-j6jm6        1/1       Running   1          13m
registry-795c6c8b8f-5jnpn     1/1       Running   1          13m
try1-857bdcd888-6klrr         1/1       Running   0          25s
try1-857bdcd888-9pwnp         1/1       Running   0          25s
try1-857bdcd888-9xkth         1/1       Running   0          25s
try1-857bdcd888-tw58z         1/1       Running   0          25s
try1-857bdcd888-xj9lk         1/1       Running   0          25s
try1-857bdcd888-znpm8         1/1       Running   0          25s
```

On the second node, use **docker ps** to verify containers of **simpleapp** are running. The scheduler will try to deploy an equal number to both nodes:

```
student@ckad-2:~$ sudo docker ps | grep simple

3ae4668d71d8
10.110.186.162:5000/simpleapp@sha256:67ea3e11570042e70cdcbad684a1e2986f59aa
f53703e51725accdf5c70d475a                          "python ./simple.py"
48 seconds ago      Up 48 seconds
k8s_try1_try1-857bdcd888-9xkth_default_2e94b97e-322a-11e8-af56-42010a800004
_0

ef6448764625
10.110.186.162:5000/simpleapp@sha256:67ea3e11570042e70cdcbad684a1e2986f59aa
f53703e51725accdf5c70d475a                          "python ./simple.py"
48 seconds ago      Up 48 seconds
k8s_try1_try1-857bdcd888-znpm8_default_2e99f356-322a-11e8-af56-42010a800004
_0
```

Return to the master node. Save the **try1** deployment as YAML:

```
student@ckad-1:~/app1$ cd ~/app1/
student@ckad-1:~/app1$ kubectl get deployment try1 -o yaml > simpleapp.yaml
```

Edit the YAML file to remove **creationTimestamp**, **selfLink**, **uid**, **resourceVersion**, and all the **status** information. In newer versions of Kubernetes it seems to no longer be necessary to remove these values in order to deploy again. Be aware that older versions would error if these values were found in the YAML file. For backwards compatibility, we will continue to remove these entries:

```
student@ckad-1:~/app1$ vim simpleapp.yaml
<output_omitted>
```


Delete and recreate the **try1** deployment using the YAML file. Verify the deployment is running with the expected number of replicas:

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

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

student@ckad-1:~/app1$ kubectl get deployment
NAME        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx       1         1         1            1           17m
registry    1         1         1            1           17m
try1        6         6         6            6           7s
```

Configure Probes

When large datasets need to be loaded or a complex application launched prior to client access, a **readinessProbe** can be used. The Pod will not become available to the cluster until a test is met. **readinessProbes** and **livenessProbes** use the same syntax and are identical, other than the name. Where the **readinessProbe** is checked prior to being ready, then not again, the **livenessProbe** continues to be checked. There are three types of liveness probes:

- A command returns a zero exit value, meaning success
- An HTTP request returns a response code in the 200 to 500 range
- The third probe uses a TCP socket.

In this example, we'll use a command, **cat**, which will return a zero exit code when the file **/tmp/healthy** has been created and can be accessed.

Edit the YAML deployment file and add the stanza for a readiness probe. Remember that, when working with YAML, whitespace matters. Indentation is used to parse where information should be associated within the stanza and the entire file. If you get an error about validating data, check the indentation. It can also be helpful to paste the file to this website to see how indentation affects the JSON value, which is actually what Kubernetes ingests: https://www.json2yaml.com/:

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



```
      - image: 10.111.235.60:5000/simpleapp:latest
        imagePullPolicy: Always
        name: try1
        readinessProbe:
          exec:
            command:
            - cat
            - /tmp/healthy
          periodSeconds: 5
        resources: {}
….
```

Delete and recreate the **try1** deployment:

```
student@ckad-1:~/app1$ kubectl delete deployment try1
deployment.extensions "try1" deleted

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

The new **try1** deployment should reference six pods, but show zero available. They are all missing the **/tmp/healthy** file:

```
student@ckad-1:~/app1$ kubectl get deployment
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx      1         1         1            1           39m
registry   1         1         1            1           39m
try1       6         6         6            0           5s
```

Take a closer look at the pods. Choose one of the **try1** pods as a test to create the health check file:

```
student@ckad-1:~/app1$ kubectl get pods
NAME                        READY     STATUS    RESTARTS   AGE
nginx-6b58d9cdfd-g7lnk      1/1       Running   1          40m
registry-795c6c8b8f-7vwdn   1/1       Running   1          40m
try1-9869bdb88-2wfnr        0/1       Running   0          26s
try1-9869bdb88-6bknl        0/1       Running   0          26s
try1-9869bdb88-786v8        0/1       Running   0          26s
try1-9869bdb88-gmvs4        0/1       Running   0          26s
try1-9869bdb88-lfvlx        0/1       Running   0          26s
try1-9869bdb88-rtchc        0/1       Running   0          26s
```


Run the bash shell interactively and touch the **/tmp/healthy** file:

```
student@ckad-1:~/app1$ kubectl exec  -it try1-9869bdb88-rtchc -- /bin/bash
root@try1-9869bdb88-rtchc:/# touch /tmp/healthy
root@try1-9869bdb88-rtchc:/# exit
exit
```

Wait at least five seconds, then check the pods again. Once the probe runs again, the container should show available quickly. The pod with the existing **/tmp/healthy** file should be running and show **1/1** in a **READY** state. The rest will continue to show **0/1**.

```
student@ckad-1:~/app1$ kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
nginx-6b58d9cdfd-g7lnk        1/1       Running   1          44m
registry-795c6c8b8f-7vwdn     1/1       Running   1          44m
try1-9869bdb88-2wfnr          0/1       Running   0          4m
try1-9869bdb88-6bknl          0/1       Running   0          4m
try1-9869bdb88-786v8          0/1       Running   0          4m
try1-9869bdb88-gmvs4          0/1       Running   0          4m
try1-9869bdb88-lfvlx          0/1       Running   0          4m
try1-9869bdb88-rtchc          1/1       Running   0          4m
```

Touch the file in the remaining pods. Consider a for loop, as an easy method to update each pod:

```
student@ckad-1:~$ for name in try1-9869bdb88-2wfnr try1-9869bdb88-6bknl
try1-9869bdb88-786v8 try1-9869bdb88-gmvs4 try1-9869bdb88-lfvlx
> do
> kubectl exec $name touch /tmp/healthy
> done
```

It may take a short while for the probes to check, for the file and the health checks to succeed:

```
student@ckad-1:~/app1$ kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
nginx-6b58d9cdfd-g7lnk        1/1       Running   1          1h
registry-795c6c8b8f-7vwdn     1/1       Running   1          1h
try1-9869bdb88-2wfnr          1/1       Running   0          22m
try1-9869bdb88-6bknl          1/1       Running   0          22m
try1-9869bdb88-786v8          1/1       Running   0          22m
try1-9869bdb88-gmvs4          1/1       Running   0          22m
```



```
try1-9869bdb88-lfvlx        1/1       Running   0          22m
try1-9869bdb88-rtchc        1/1       Running   0          22m
```

Now that we know when a Pod is healthy, we may want to keep track that it stays healthy, using a **livenessProbe**. You could use one probe to determine when a Pod becomes available and a second probe, to a different location, to ensure ongoing health.

Edit the Deployment again. Add in a **livenessProbe** section as seen below. This time we will add a new container to the pod running a simple application which will respond to port 8080. Note that the dash (-) in front of the **name: goproxy** is indented the same amount as the - in front of the **image:** line for **simpleapp** earlier in the file. In this example that would be seven spaces:

```
student@ckad-1:~/app1$ vim simpleapp.yaml
….
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
       - name: goproxy
         image: k8s.gcr.io/goproxy:0.1
         ports:
         - containerPort: 8080
         readinessProbe:
           tcpSocket:
             port: 8080
           initialDelaySeconds: 5
           periodSeconds: 10
         livenessProbe:
           tcpSocket:
             port: 8080
           initialDelaySeconds: 15
           periodSeconds: 20
       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 simpleapp.yaml
deployment.extensions "try1" created
```


View the newly created Pods. You'll note that there are two containers per pod, and only one is running. The new **simpleapp** containers will not have the **/tmp/healthy** file, so they will not become available until we touch the **/tmp/healthy** file again. We could include a command which creates the file into the container arguments. The output below shows it can take a bit for the old pods to terminate.

```
student@ckad-1:~$ kubectl get pods
NAME                          READY     STATUS              RESTARTS   AGE
nginx-6b58d9cdfd-g7lnk        1/1       Running             1          13h
registry-795c6c8b8f-7vwdn     1/1       Running             1          13h
try1-76cc5ffcc6-4rjvh         1/2       Running             0          3s
try1-76cc5ffcc6-bk5f5         1/2       Running             0          3s
try1-76cc5ffcc6-d8n5q         0/2       ContainerCreating   0          3s
try1-76cc5ffcc6-mm6tw         1/2       Running             0          3s
try1-76cc5ffcc6-r9q5n         1/2       Running             0          3s
try1-76cc5ffcc6-tx4dz         1/2       Running             0          3s
try1-9869bdb88-2wfnr          1/1       Terminating         0          12h
try1-9869bdb88-6bknl          1/1       Terminating         0          12h
try1-9869bdb88-786v8          1/1       Terminating         0          12h
try1-9869bdb88-gmvs4          1/1       Terminating         0          12h
try1-9869bdb88-lfvlx          1/1       Terminating         0          12h
try1-9869bdb88-rtchc          1/1       Terminating         0          12h
```

Create the health check file for the **readinessProbe**. You can use a for loop again for each action, with updated Pod names. As there are now two containers in the Pod, you should include the container name for where the command will execute. If no name is given, it will default to the first container. Depending on how you edited the YAML file, **try1** should be the first pod and **goproxy** the second. To ensure the correct container is updated, add **-c try1** to the **kubectl** command. Your Pod names will be different. Use the names of the newly started containers from the **kubectl get pods** command output.

```
student@ckad-1:~$ for name in try1-76cc5ffcc6-4rjvh try1-76cc5ffcc6-bk5f5
try1-76cc5ffcc6-d8n5q try1-76cc5ffcc6-mm6tw try1-76cc5ffcc6-r9q5n
try1-76cc5ffcc6-tx4dz
do
kubectl exec $name -c try1 touch /tmp/healthy
done
```

In the next minute or so, the second container in each Pod, which was not running, will change status to **Running**. Each should show **2/2** containers running:

```
student@ckad-1:~$ kubectl get pods
NAME                          READY     STATUS              RESTARTS   AGE
```



```
nginx-6b58d9cdfd-g7lnk      1/1       Running             1          13h
registry-795c6c8b8f-7vwdn   1/1       Running             1          13h
try1-76cc5ffcc6-4rjvh       2/2       Running             0          3s
try1-76cc5ffcc6-bk5f5       2/2       Running             0          3s
try1-76cc5ffcc6-d8n5q       2/2       Running            0          3s
try1-76cc5ffcc6-mm6tw       2/2       Running            0          3s
try1-76cc5ffcc6-r9q5n       2/2       Running           0          3s
try1-76cc5ffcc6-tx4dz       2/2       Running           0          3s
```

View the events for a particular pod. Even though both containers are currently running and the pod is in good shape, note the events show the last issue:

```
student@ckad-1:~/app1$ kubectl describe pod try1-76cc5ffcc6-tx4dz | tail
 Normal   SuccessfulMountVolume  9m                kubelet, ckad-1-lab-x6dj
MountVolume.SetUp succeeded for volume "default-token-jf69w"
  Normal   Pulling                 9m                kubelet,
ckad-1-lab-x6dj  pulling image "10.108.143.90:5000/simpleapp"
  Normal   Pulled                  9m                kubelet,
ckad-1-lab-x6dj  Successfully pulled image "10.108.143.90:5000/simpleapp"
  Normal   Created                 9m                kubelet,
ckad-1-lab-x6dj  Created container
  Normal   Started                 9m                kubelet,
ckad-1-lab-x6dj  Started container
  Normal   Pulling                 9m                kubelet,
ckad-1-lab-x6dj  pulling image "k8s.gcr.io/goproxy:0.1"
  Normal   Pulled                  9m                kubelet,
ckad-1-lab-x6dj  Successfully pulled image "k8s.gcr.io/goproxy:0.1"
  Normal   Created                 9m                kubelet,
ckad-1-lab-x6dj  Created container
  Normal   Started                 9m                kubelet,
ckad-1-lab-x6dj  Started container
  Warning  Unhealthy               4m (x60 over 9m)  kubelet,
ckad-1-lab-x6dj  Readiness probe failed: cat: /tmp/healthy: No such file or
directory
```

If you look for the status of each container in the pod, they should show that both are running and ready.

```
student@ckad-1:~/app1$ kubectl describe pod try1-76cc5ffcc6-tx4dz | \
grep -E 'State|Ready'
    State:          Running
    Ready:          True
    State:          Running
```



```
Ready:          True
Ready           True
```

