Exercise 3.2: Configure A Local Docker Repo

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.

1. 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
```

2. 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>
```

3. 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/
```

4. 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
```

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
```

5. 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. **Let it run. You will use ctrl-c in a few steps.**

```
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>
```

6. 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 does not match exactly.

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

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

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

8. Download the kompose binary and make it executable. The command can run on a single line. Note that the option following the dash is the letter as in **o**utput. The short URL goes here: https://github.com/kubernetes/kompose/releases/download/v1.1.0/kompose-linux-amd64

```
root@ckad-1:/localdocker# curl -L https://bit.ly/2tN0bEa -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
```

9. 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
```

10. Create two physical volumes in order to deploy a local registry for Kubernetes. 200Mi for each should be enough for each of the volumes. Use the **hostPath** `storageclass` for the volumes.

    More details on how persistent volumes and persistent volume claims are covered in an upcoming chapter, `Deployment Configuration`.

```
student@ckad-1:~$ vim vol1.yaml
```

vol1.yaml

```
1 apiVersion: v1
2 kind: PersistentVolume
3 metadata:
4   labels:
5     type: local
6   name: task-pv-volume
```

```
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 200Mi
  hostPath:
    path: /tmp/data
  persistentVolumeReclaimPolicy: Retain
```

```
student@ckad-1:~$ vim vol2.yaml
```

vol2.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    type: local
  name: registryvm
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 200Mi
  hostPath:
    path: /tmp/nginx
  persistentVolumeReclaimPolicy: Retain
```

11. 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
```

12. Verify 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
```

13. Go to the configuration file directory for the local Docker registry.

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

14. Convert the Docker file into a single YAML file for use with Kubernetes. Not all objects convert exactly from Docker to **kompose**, you may 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
```

15. Review the file. You'll find that multiple Kubernetes objects will have been created such as `Services`, `Persistent Volume Claims` and `Deployments` using environmental parameters and volumes to configure the container within.

```
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>
```

16. 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
persistentvolume/registryvm     200Mi      RWO            Retain
 Available                                      15s
persistentvolume/task-pv-volume 200Mi      RWO            Retain
 Available                                      17s
```

17. To illustrate the fast changing nature of Kubernetes you will show that the API has changed for `Deployments`. Use the `--dry-run` option to see what the API now requires. View the YAML output so we can see what we need to edit for the local registry.

```
student@ckad-1:~/localdocker$ kubectl create deployment drytry --image=nginx --dry-run -o yaml
```

drytry

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    creationTimestamp: null
5    labels:
6      app: drytry
7    name: drytry
8  spec:
9    replicas: 1
10   selector:
11     matchLabels:
12       app: drytry
13   strategy: {}
14   template:
15 <output_omitted>
```

18. From this output we can see that we need to change the `apiVersion`, add `selector`, and add `matchLabels` and a label line. The three lines to add will be part of the replicaSet information, right after the `replicas` line.

    Following is a **diff** output. Use the man page to decode the output if you are not already familiar with the command.

```
student@ckad-1:~/localdocker$ sudo vim localregistry.yaml
<make edits>

student@ckad-1:~/localdocker$ diff edited-localregistry.yaml localregistry.yaml

41c41
< - apiVersion: apps/v1
---
> - apiVersion: extensions/v1beta1
53,55d52
<     selector:
<       matchLabels:
<         io.kompose.service: nginx
93c90
< - apiVersion: apps/v1
---
> - apiVersion: extensions/v1beta1
105,107d101
<     selector:
<       matchLabels:
<         io.kompose.service: registry
```

    Use **kubectl** to create the local docker registry.

19.
```
student@ckad-1:~/localdocker$ kubectl create -f localregistry.yaml

service/nginx created
service/registry created
deployment.apps/nginx created
persistentvolumeclaim/nginx-claim0 created
deployment.apps/registry created
persistentvolumeclaim/registry-claim0 created
```

20. View the newly deployed resources. The persistent volumes should now show as `Bound`. Be aware that due to the manner that volumes are bound it is possible that the registry claim may not to be bound to the registry volume. 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, 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                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx      1/1     1            1           12s
deployment.apps/registry   1/1     1            1           12s
```

21. 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.

    Edit the IP address to that of your `registry` service.

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

22. 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 you verified in the previous step.

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

23. Restart docker on the local system. It can take up to a minute for the restart to take place. Ensure the service is `active`. It should report that the service recently became status as well.

```
student@ckad-1:~$ sudo systemctl restart docker.service
student@ckad-1:~$ sudo systemctl status docker.service | grep Active
   Active: active (running) since Tue 2019-09-24 15:24:36 UTC; 40s ago
```

24. 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
```

25. 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
```

26. 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:e348fbbea0e0a0e73ab0370de151e7800684445c509d46195aef73e090a49bd6

student@ckad-1:~$ sudo docker image remove 10.110.186.162:5000/tagtest
Untagged: 10.110.186.162:5000/tagtest:latest
<output_omitted>
```

27. 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
Digest: sha256:0847cc7fed1bfafac713b0aa4ddfb8b9199a99092ae1fc4e718cb28e8528f65f
Status: Downloaded newer image for 10.110.186.162:5000/tagtest:latest
```

28. 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
```

29. Configure the worker (second) node to use the local registry running on the master server. Connect to the worker node. Edit the Docker `daemon.json` file with the same values as 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
```

30. 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
```

31. Return to the master node and deploy the `simpleapp` in Kubernetes with several replicas. We will name the deployment `try1`. Scale to have six replicas. Multiple replicas the scheduler should run some containers on each node.

```
student@ckad-1:~$ kubectl create deployment try1 --image=10.110.186.162:5000/simpleapp:latest
deployment.apps/try1 created

student@ckad-1:~$  kubectl scale deployment try1 --replicas=6
deployment.apps/try1 scaled
```

32. 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
```

33. On the `second node` use **sudo docker ps** to verify containers of `simpleapp` are running. The scheduler will try to deploy an equal number to both nodes by default.

```
student@ckad-2:~$ sudo docker ps | grep simple
3ae4668d71d8     \
  10.110.186.162:5000/simpleapp@sha256:67ea3e11570042e70cdcbad684a1e2986f59aaf53703e51725accdf5c70d475a \
            "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@sha256:67ea3e11570042e70cdcbad684a1e2986f59aaf53703e51725accdf5c70d475a \
            "python ./simple.py"     48 seconds ago      Up 48 seconds \
             k8s_try1_try1-857bdcd888-znpm8_default_2e99f356-322a-11e8-af56-42010a800004_0
```

34. 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
```

35. Delete and recreate the `try1` deployment using the YAML file. Verify the deployment is running with the expected six replicas.

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

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

student@ckad-1:~/app1$ kubectl get deployment
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
nginx      1/1     1            1           15m
registry   1/1     1            1           15m
try1       6/6     6            6           5s
```