Docker-Registry

Note

This guide for installing docker registry on kubernetes is perfectly valid if you are not going to use OpenFaas. If you are, jump to the *Docker-Registry TLS* section Docker-Registry TLS!

We could install the Docker registry from Arkade, but for the life of me I could not figure out how to tell it to use persistent storage. Helm on other hand could be used to install it with persistent storage, but honestly I don't remember why I did not use it in the end.

Namespace

I will install everything related to Docker registry into its own `namespace` called `docker-registry`. So we create that first:

```
kubectl create namespace docker-registry
```

Storage

Since we are going to store docker images in our personal registry to be later used with OpenFaaS, it would be a shame if they disappeared every time the pod reschedules to another node.

We need persistent storage that would follow our pods around and provide them with the same data all the time.

If you followed my setup you should have longhorn installed.

persistentVolumeClaim

> A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumeClaims are a way for users to "claim" durable storage (such as a GCE PersistentDisk or an iSCSI volume) without knowing the details of the particular cloud environment.

We will create a ***new folder called*** ***`docker-registry`*** and a new file `pvc.yaml` inside it:

```
cd
mkdir docker-registry
cd  docker-registry
nano pvc.yaml
```

In our `pvc.yaml`

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: longhorn-docker-registry-pvc
  namespace: docker-registry
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: longhorn
  resources:
    requests:
      storage: 10Gi
```

We are telling Kubernetes to use Longhorn as our storage class, and to claim/create 10 GB of disk space for persistent storage. We will call it `longhorn-docker-registry-pvc`, and reference it by this name later.

Important

Notice I have specified `namespace`, This is important since only pods/deployment in that namespace would be able to see the disk.

To learn more about volumes check out the official documentation here: https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Apply our `pvc.yaml`

```
kubectl apply -f pvc.yaml
```

And check

```
root@control01:/home/ubuntu/docker-registry# kubectl get pvc -n docker-registry
NAME                           STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
longhorn-docker-registry-pvc   Bound    pvc-39662498-535a-4abd-9153-1c8dfa74749b   10Gi       RWO            longhorn       5d6h

#longhorn should also create automatically PV ( physical volume )
root@control01:/home/ubuntu/docker-registry# kubectl get pv -n docker-registry
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                                          STORAGECLASS   REASON
AGE
pvc-39662498-535a-4abd-9153-1c8dfa74749b   10Gi       RWO            Delete           Bound    docker-registry/longhorn-docker-registry-pvc   longhorn
5d6h
```

Cool, cool: now we have storage!

Deployment

Now we will create a simple deployment of docker registry and let it loose on our Kubernetes cluster.

Create a file in your docker-registry directory called `docker.yaml`

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: registry
  namespace: docker-registry
spec:
  replicas: 1
  selector:
    matchLabels:
      app: registry
  template:
    metadata:
      labels:
        app: registry
        name: registry
    spec:
      nodeSelector:
        node-type: worker
      containers:
      - name: registry
        image: registry:2
        ports:
        - containerPort: 5000
        volumeMounts:
        - name: volv
          mountPath: /var/lib/registry
          subPath: registry
      volumes:
        - name: volv
          persistentVolumeClaim:
            claimName: longhorn-docker-registry-pvc
```

What to pay attention to:

- ***namespace*** - I specified `docker-registry`.
- ***replicas*** - I'm using 1, so there will be only one docker-registry running.
- ***nodeSelector*** - As mentioned before in setting up my Kubernetes, I have labeled worker nodes with node-type=worker. This will make it so that the deployment prefers those nodes.
- ***image*** - This will tell Kubernetes to download registry:2 from official docker hub.
- ***containerPort*** - Which port the container will expose/use.
- ***volumeMounts*** - Definition of where in the pod we will mount our persistent storage.
- ***volumes*** - Definition where we refer back to PVC we created before.

Apply the deployment and wait a little for everything to come online.

```
kubectl apply -f docker.yaml
```

Check with

```
# Deployment
root@control01:/home/ubuntu/docker-registry# kubectl get deployments -n docker-registry
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
registry   1/1     1            1           5d6h
# Pods ( should be 1 )
root@control01:/home/ubuntu/docker-registry# kubectl get pods -n docker-registry
NAME                        READY   STATUS    RESTARTS   AGE
registry-69f76f7f97-zf4v4   1/1     Running   0          5d6
```

Technically, we are done, but we need to also create a service to make the registry available cluster-wide, and ideally on the same IP/name all the time, no matter on what node it runs.

Service

Again, if you followed my network setting, we have set up metalLB to provide us with external IPs for pods. Therefore, we use this as a LoadBalancer service for our Docker registry.

In your folder `docker-registry` create `service.yaml` and paste in the following:

```
apiVersion: v1
kind: Service
metadata:
  name: registry-service
  namespace: docker-registry
```

```
spec:
  selector:
    app: registry
  type: LoadBalancer
  ports:
    - name: docker-port
      protocol: TCP
      port: 5000
      targetPort: 5000
  loadBalancerIP: 192.168.0.232
```

What to pay attention to:

- ***kind*** - Servicem, just to let Kubernetes know what we are creating.
- ***name*** - Just a name for our service.
- ***namespace*** - I specified `docker-registry` because the deployment we are targeting is in that name space.
- ***selector and app*** - The value for this is lifted from our deployment where this is set: `app: registry`.
- ***type*** - Here, we tell Kubernetes that we want LoadBalancer (MetalLB).
- ***ports*** - we define `port` on that would be on our external IP and `targetPort` (that's the port inside the app).
- ***loadBalancerIP*** - This is optional, but I have included it here. This will allow us to specify which IP we want for the external IP. If you remove that line, MetalLB will assign the next free IP from the pool we allocated to it.

Apply the service

```
kubectl apply -f service.yaml
```

Give it a few seconds to get the IP and check.

```
root@control01:/home/ubuntu/docker-registry# kubectl get svc -n docker-registry
NAME               TYPE           CLUSTER-IP   EXTERNAL-IP     PORT(S)          AGE
registry-service   LoadBalancer   10.43.5.16   192.168.0.232   5000:32096/TCP   7m48s
```

Fantastic! The service seems to be up and running with external port 5000. About the 32096 port behind it: this might be different for you. It is assigned on the node where the pod is running. In essence it's like this: External IP:5000 -> Node where the Pod/Container is:32096 -> container inside:5000. I hope that make sense 🙂

To get more info about the service we can ask Kubectl to describe it to us:

```
root@control01:/home/ubuntu/docker-registry# kubectl get svc -n docker-registry
NAME               TYPE           CLUSTER-IP   EXTERNAL-IP     PORT(S)          AGE
registry-service   LoadBalancer   10.43.5.16   192.168.0.232   5000:32096/TCP   7m48s
root@control01:/home/ubuntu/docker-registry# kubectl describe svc registry-service  -n docker-registry
Name:                     registry-service
Namespace:                docker-registry
Labels:                   <none>
Annotations:              <none>
Selector:                 app=registry
Type:                     LoadBalancer
IP:                       10.43.5.16
IP:                       192.168.0.232
LoadBalancer Ingress:     192.168.0.232
Port:                     docker-port  5000/TCP
TargetPort:               5000/TCP
NodePort:                 docker-port  32096/TCP
Endpoints:                10.42.8.13:5000
Session Affinity:         None
External Traffic Policy:  Cluster
Events:
  Type    Reason        Age                  From                Message
  ----    ------        ----                 ----                -------
  Normal  IPAllocated   77s (x537 over 11m)  metallb-controller  Assigned IP "192.168.0.232"
  Normal  nodeAssigned  76s (x539 over 11m)  metallb-speaker     announcing from node "cube06"
```

Docker registry test

We are going to perform a simple test of whether our docker register is working.

First, get docker.io; you will need it on one node on the cluster, since you will also have to build all your images on arm64.

```
#For ubuntu
apt install docker.io
```

Next, you need to edit `/etc/docker/daemon.json`, since we (well, I 🙂) do not have an external IP and therefore no ssl certificate to use HTTPS. So, as it is now, our docker registry server is only `HTTP`. If you tried to push an image to it, it would say this:

```
The push refers to repository [192.168.0.232:5000/my-ubuntu]
Get https://192.168.0.232:5000/v2/: http: server gave HTTP response to HTTPS client
```

Therefore, we tell the Docker about our insecure registry in `/etc/docker/daemon.json`

Note

You might not have this file, and will need to create it. In my installation it was already present, and I only had to add a comma after the `storage-driver` line and a new line with `insecure-registries`.

```
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2",
  "insecure-registries" : ["192.168.0.232:5000"]
}
```

Reload service:

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

We will download an Ubuntu container from the official docker registry, re-tag it and push to our registry:

```
root@control01:~# docker pull ubuntu:16.04
16.04: Pulling from library/ubuntu
3e30c5e4609a: Pull complete
be82da0c7e99: Pull complete
bdf04dffef88: Pull complete
2624f7934929: Pull complete
Digest: sha256:3355b6e4ba1b12071ba5fe9742042a2f10b257c908fbdfac81912a16eb463879
Status: Downloaded newer image for ubuntu:16.04
docker.io/library/ubuntu:16.04

root@control01:~# docker tag ubuntu:16.04 192.168.0.232:5000/my-ubuntu
root@control01:~# docker push 192.168.0.232:5000/my-ubuntu
The push refers to repository [192.168.0.232:5000/my-ubuntu]
3660514ed6c6: Pushed
2f33c1b8271f: Pushed
753fcdb98fb4: Pushed
1632f6712b3f: Pushed
latest: digest: sha256:2e459e7ec895eb5f94d267fb33ff4d881699dcd6287f27d79df515573cd83d0b size: 1150

# Check with curl:
root@control01:~# curl 192.168.0.232:5000/v2/_catalog
{"repositories":["my-ubuntu"]}
```

Yay! It worked!

Last step

I know, I know. This is taking forever. The last step is to let our K3s cluster know about our private Docker registry.

Here is where I got my info from: https://rancher.com/docs/k3s/latest/en/installation/private-registry/

Add a dns name to `/etc/hosts` on ***every node***, I named it like this:

```
192.168.0.232 docker-registry docker-registry.local
```

It is a good idea to have the `/etc/hosts` nice and synced between all nodes, so I will add it once into control01 node and, using Ansible, move it to all nodes:

```
echo "192.168.0.232 docker-registry docker-registry.local" >> /etc/hosts
ansible cube -b -m copy -a "src=/etc/hosts dest=/etc/hosts"
```

Now tell k3s about it. As root, create file `/etc/rancher/k3s/registries.yaml`:

```
nano /etc/rancher/k3s/registries.yaml
```

Add the following:

```
mirrors:
  docker-registry:
    endpoint:
      - "http://docker-registry.local:5000"
```

Send it to every control node of the cluster:

```
# Make sure the directory exists
ansible cube -b -m file -a "path=/etc/rancher/k3s state=directory"

# Copy the file
ansible cube -b -m copy -a "src=/etc/rancher/k3s/registries.yaml dest=/etc/rancher/k3s/registries.yaml"
```

And hopefully this is it. Congratulation getting this far. Now, get some coffee or drink of your choosing and maybe get me one too 🙂

Liked it ? Buy me a drink :)

Last update: October 20, 2021

Comments

What do you think?

8 Responses

Upvote Funny Love Surprised Angry Sad

0 Comments https://rpi4cluster.com Disqus' Privacy Policy Login

Favorite Tweet Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name