Docker-Registry TLS

> **Note**
>
> This guide for installing a Docker registry with TLS, enabling HTTPS, with self-signed certificate.

This guide in general is a copy of the non TLS version with some tweeks.

Namespace

I will install everything related to Docker-registry into its own `namespace` called `docker-registry`. So, we will 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 follows our pod around and provides it with the same data all the time.

If you followed my setup, you should have longhorn installed already.

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 ***new folder called*** ***`docker-registry`*** and a new file `pvc.yaml` in it.

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

In our `pvc.yaml`:

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

and another one where we will store certificates:

```
nano pvc_cert.yaml
```

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

We are telling Kubernetes to use Longhorn as our storage class, and to claim/create 10 GB and 10MB of disk space for persistent storage. We will call it `longhorn-docker-registry-pvc` and `longhorn-docker-registry-pvc-cert`, and we will 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` and `pvc_cert.yaml`:

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

> **Note**
>
> I know that you can fit both into one yaml file, but I like stuff separated so if I want to delete one, I can just call the appropriate 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                                       STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
longhorn-docker-registry-pvc               Bound    pvc-700169df-2996-4031-bef5-e4cb7a560264   10Gi       RWO            longhorn       10d
longhorn-docker-registry-pvc-cert          Bound    pvc-520e979f-2531-4b3e-8f0e-be8ee9bf0915   10Mi       RWO            longhorn       28h
```

Cool, cool: now we have storage! (Status might be different, something like: Unattached)

Creating certificates for docker registry

I literally spent over 24 hours to get this to work with the setup I have, including OpenFaas and so on... so hopefully I did not forget a step :smile:

Generate certificates in your docker-register directory:

```
#install opnessl if its not
sudo apt-get install openssl

# generate certificate and key
openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes -keyout registry.key \
-out registry.crt -subj "/CN=registry.cube.local" \
-addext "subjectAltName=DNS:registry.cube.local,DNS:*.cube.local,IP:192.168.0.232"
```

> **Important**
>
> This is very important: my entry in /etc/hosts for the registry will be `192.168.0.232 registry registry.cube.local`. I know what IP it will be, as we will set it later (remember metalLB?), and there is no DNS server, so every node will have to have this in /etc/hosts. Call it whatever you want, but add the correct names into `subjectAltName` parameters. Without it, there might be issues where some tools complain about incorrectly signed certificates and missing SAN. Something like: `x509: cannot validate certificate for <IP> because it doesn't contain any IP SANs`

Now you have two new files in your `docker-registry` directory:

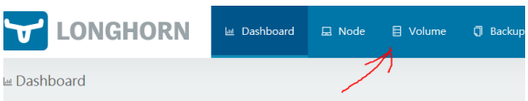
```
ubuntu@control01:~/docker-registry$ ls | grep regis
registry.crt
registry.key
```

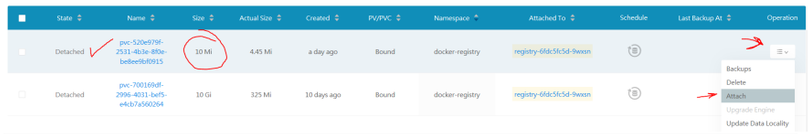
Longhorn copy data to disk

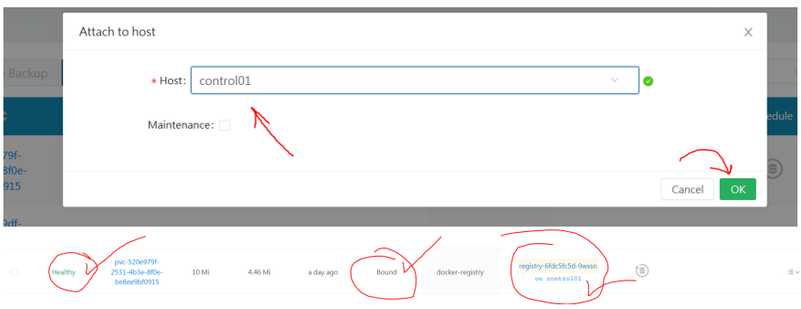
What we need to do now to tell our upcoming docker registry in kubernetes about the certificates, and where they are.

Copy our certificates to the longhorn disk we created. Remember Longhorn UI? Get to it now. (I'm not sure how to do it from CLI right now; if you know, comment below!)

Follow the picture guide. Basically, we tell Longhorn to attach the disk to your node (How freaking cool and simple is that?!)







This will create a new `VIRTUAL-DISK` on the node you attached it:

```
# in my case its /dev/sdc
Disk /dev/sdc: 10 MiB, 10485760 bytes, 20480 sectors
Disk model: VIRTUAL-DISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

Now, create filesystem and mount the volume:

```
mkdir /tmp/disk
# Skipp mkfs if the disk was used before and there are data...
mkfs.ext4 /dev/sdc
mount /dev/sdc /tmp/disk
```

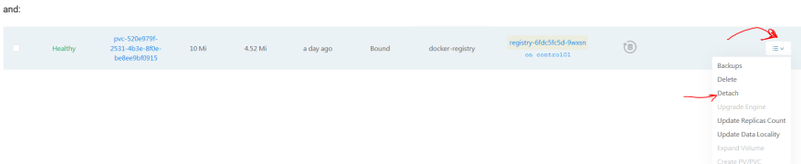
Copy the certificate and key to /tmp/disk:

```
cp registry.* /tmp/disk
# check
root@control01:/home/ubuntu/docker-registry# ll /tmp/disk
total 32
drwxr-xr-x  3 root root  4096 Feb  1 17:06 ./
drwxrwxrwt 11 root root  4096 Feb  2 22:00 ../
drwx------  2 root root 16384 Feb  1 16:57 lost+found/
-rw-r--r--  1 root root  1907 Feb  2 22:00 registry.crt
-rw-------  1 root root  3272 Feb  2 22:00 registry.key
```

Unmount the disk and detach in Longhorn UI:

```
umount /tmp/disk
```

and:



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`:

```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
        env:
        - name: REGISTRY_HTTP_TLS_CERTIFICATE
          value: "/certs/registry.crt"
        - name: REGISTRY_HTTP_TLS_KEY
          value: "/certs/registry.key"
        volumeMounts:
        - name: lv-storage
          mountPath: /var/lib/registry
          subPath: registry
        - name: lv-certs
          mountPath: /certs
      volumes:
        - name: lv-storage
          persistentVolumeClaim:
            claimName: longhorn-docker-registry-pvc
        - name: lv-certs
          persistentVolumeClaim:
            claimName: longhorn-docker-registry-pvc-cert
```

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 the 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.
- ***env*** - This will be passed as environmental variables into the container, and used by docker registry.

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           21s
# Pods ( should be 1 )
root@control01:/home/ubuntu/docker-registry# kubectl get pods -n docker-registry
NAME                        READY   STATUS    RESTARTS   AGE
registry-6fdc5fc5d-npslq    1/1     Running   0          29s
```

We are not done yet; we also need to create a service to make the registry available cluster-wide, and ideally on the same IP/name all the time no matter what node it runs on.

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:

```yaml
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*** - Service, 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 our external IP and `targetPort` (that's the port inside the app/container).
- ***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 to a 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 :smile:

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"
```

Add root certificate to nodes

Currently, we have SSL or TLS (Fuck it, I will now call it HTTPS) enabled docker registry running on Kubernetes. However, since we create the certificate we need to add it as a root certificate to our nodes, every single one! If we don't, any service that try to use it will complain about a certificate signed by an unknown authority or something similar. So, we will make ourselves the authority like this:

```
#For ubuntu
ansible cube -b -m copy -a "src=registry.crt dest=/usr/local/share/ca-certificates/registry.crt"
ansible cube -b -m copy -a "src=registry.key dest=/usr/local/share/ca-certificates/registry.key"
ansible all -b -m shell -a "update-ca-certificates"
```

In essence, on every node you need to copy our `registry.crt` into `/usr/local/share/ca-certificates/`, and execute `update-ca-certificates`, which will add our certificate as a root certificate. This will fool the verification into thinking that we are the authority for the certificate (which we are) and it won't complain.

An example of doing it manually:

```
ubuntu@control01:~/docker-registry$ sudo cp registry.* /usr/local/share/ca-certificates/
ubuntu@control01:~/docker-registry$ sudo update-ca-certificates
Updating certificates in /etc/ssl/certs...
1 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
```

Making K3s use private docker registry

I know, I know: this is taking forever.

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 registry registry.cube.local
```

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

```
echo "192.168.0.232 registry registry.cube.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:

```yaml
mirrors:
  registry.cube.local:5000:
    endpoint:
      - "https://registry.cube.local:5000"
configs:
  registry.cube.local:
    tls:
      ca_file: "/usr/local/share/ca-certificates/registry.crt"
      key_file: "/usr/local/share/ca-certificates/registry.key"
```

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"
```

Docker registry test

Follow the guide how to install docker from here:

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 registry.cube.local: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 https:///registry.cube.local:5000/v2/_catalog
{"repositories":["my-ubuntu"]}
```

Yay ! it worked

Hopefully, this is it; congratulations getting this far. Now, get some coffee and maybe get me one too 🙂

Liked it ? Buy me a drink :)

Last update: May 26, 2022

Comments

What do you think?

2 Responses

Upvote Funny Love Surprised Angry Sad

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

Favorite Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Lâm Phúc Tài • 3 months ago • edited

Hi, I'm trying to test after all.
But I have some problem about "x509: certificate signed by unknown authority"

Follow this thread is restart Docker to reload new cert. Hope it help u ^^
https://forums.docker.com/t...

Thanks!

• Reply • Share ›

Gary Murphy • 5 months ago

I am not running Longhorn, but I installed the TLS certificates for the Docker registry by mounting the PVC on a Rocky Linux container and just generating the certificates directly to the volume

• Reply • Share ›