Docker Private Registry

We will be doing following steps to create our Private registry:

Creating files for authentication

root@master1:~# **mkdir -p /registry && cd "$\_"**

root@master1:/registry# **mkdir certs**

root@master1:/registry# **openssl req -x509 -newkey rsa:4096 -days 365 -nodes -sha256 -keyout certs/tls.key -out certs/tls.crt -subj "/CN=docker-registry" -addext "subjectAltName = DNS:docker-registry"**

**Let’s use htpasswd to add user authentication for registry access. My credentials for the private registry would be myuser/mypasswd.**

**root@master1:/registry# mkdir auth**

**root@master1:/registry# docker run --rm --entrypoint htpasswd registry:2.7.0 -Bbn myuser mypasswd > auth/htpasswd**

**At this point, our /registry directory looks like this:**

**root@master1:/# ls -R /registry/**

**/registry/:**

**auth certs**

**/registry/auth:**

**htpasswd**

**/registry/certs:**

**tls.crt tls.key**

**root@master1:/#**

**2. Using Secrets to mount the certificates**

In Kubernetes, a Secret is a resource that will enable you to inject sensitive data into a container when it starts up. This data can be anything like password, OAuth tokens or ssh keys. They can be exposed inside a container as mounted files or volumes or environment variables.

The below command creates a Secret of type tls named certs-secret in the default namespace from the pair of public/private keys we just created.

root@master1:/# **kubectl create secret tls certs-secret --cert=/registry/certs/tls.crt --key=/registry/certs/tls.key**

secret/certs-secret created

The Secret auth-secret that we create from the htpasswd file is of type generic which means the Secret was created from a local file.

root@master1**:/# kubectl create secret generic auth-secret --from-file=/registry/auth/htpasswd**

secret/auth-secret created

Creating Persistent Volume and Claim for repository storage

The images that are pushed to our registry should be placed in a consistent storage location. Hence, we will be using a Persistent Volume of 1 GB hosted at a temporary location in the node where our registry Pod will be running. The Pod uses a Persistent Volume Claim which will be bound to the newly created volume as a one-to-one mapping.

apiVersion: v1

kind: PersistentVolume

metadata:

name: docker-repo-pv

spec:

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

hostPath:

path: /tmp/repository

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: docker-repo-pvc

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 1Gi

Copy the above content into a yaml file, say repository-volume.yaml and execute the below command:

root@master1:/# **kubectl create -f repository-volume.yaml**

persistentvolume/docker-repo-pv created

persistentvolumeclaim/docker-repo-pvc created

Creating the Registry Pod

Next, let us create the actual Pod and a corresponding Service to access it. In the yaml file docker-registry-pod.yaml below, the image that we use for our registry is called registry which is downloaded from DockerHub. The images pushed to this registry will be saved in /var/lib/registry directory internally, hence we mount our Persistent Volume using the Claim docker-repo-pvc to persist the images permanently. The environment variables which are required by the registry container are taken care by the Secrets that we mount as volumes.

The Service is named docker-registry, with which we want to access our docker private registry. Note that this was the exact name that was given in the Common Name “/CN=” field while generating the TLS certificates. The registry container by default is exposed at port 5000 and we bind our Service to this port accordingly.

apiVersion: v1

kind: Pod

metadata:

name: docker-registry-pod

labels:

app: registry

spec:

containers:

- name: registry

image: registry:2.7.0

volumeMounts:

- name: repo-vol

mountPath: "/var/lib/registry"

- name: certs-vol

mountPath: "/certs"

readOnly: true

- name: auth-vol

mountPath: "/auth"

readOnly: true

env:

- name: REGISTRY\_AUTH

value: "htpasswd"

- name: REGISTRY\_AUTH\_HTPASSWD\_REALM

value: "Registry Realm"

- name: REGISTRY\_AUTH\_HTPASSWD\_PATH

value: "/auth/htpasswd"

- name: REGISTRY\_HTTP\_TLS\_CERTIFICATE

value: "/certs/tls.crt"

- name: REGISTRY\_HTTP\_TLS\_KEY

value: "/certs/tls.key"

volumes:

- name: repo-vol

persistentVolumeClaim:

claimName: docker-repo-pvc

- name: certs-vol

secret:

secretName: certs-secret

- name: auth-vol

secret:

secretName: auth-secret

---

apiVersion: v1

kind: Service

metadata:

name: docker-registry

spec:

selector:

app: registry

ports:

- port: 5000

targetPort: 5000

Create the Registry Pod and the Service using the following command:

root@master1:/# kubectl create -f docker-registry-pod.yaml

pod/docker-registry-pod created

service/docker-registry created

root@master1:/# kubectl get all

NAME READY STATUS RESTARTS AGE

pod/docker-registry-pod 1/1 Running 0 36s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/docker-registry ClusterIP 10.107.59.73 <none> 5000/TCP 36s

service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 65d

Allowing access to the registry from all the nodes in the cluster

We observe from the above step that our registry can be accessed at 10.107.59.73:5000, since the ip-address of our Service turned out to be 10.107.59.73. Note that this value will be different in your case.

Let’s make a note of the registry name and its ip-address as environment variables.

**root@master1:/# export REGISTRY\_NAME="docker-registry"**

**root@master1:/# export REGISTRY\_IP="10.107.59.73"**

**Now, we shall append the entry “10.107.59.73 docker-registry” to the /etc/hosts file of all the nodes in our cluster so that this ip-address is resolved to the name docker-registry.**

**The above step can also be done using a single command from the master node (Enter password, if prompted):\**

**root@master1:/# for x in $(kubectl get nodes -o jsonpath='{ $.items[\*].status.addresses[?(@.type=="InternalIP")].address }'); do ssh root@$x "echo '$REGISTRY\_IP $REGISTRY\_NAME' >> /etc/hosts"; done**

**Next, we must copy the tls.crt that we created earlier as “ca.crt” into a custom /etc/docker/certs.d/docker-registry:5000 directory in all the nodes in our cluster to make sure that our self-signed certificate is trusted by Docker. Note that the directory that is created inside /etc/docker/certs.d should be having the name of the format<registry\_name>:<registry\_port>.**

**This step can be done manually or with the help of a single command from the master node as follows:**

**root@master1:/# for x in $(kubectl get nodes -o jsonpath='{ $.items[\*].status.addresses[?(@.type=="InternalIP")].address }'); do ssh root@$x "rm -rf /etc/docker/certs.d/$REGISTRY\_NAME:5000;mkdir -p /etc/docker/certs.d/$REGISTRY\_NAME:5000"; done**

**root@master1:/#**

**root@master1:/# for x in $(kubectl get nodes -o jsonpath='{ $.items[\*].status.addresses[?(@.type=="InternalIP")].address }'); do scp /registry/certs/tls.crt root@$x:/etc/docker/certs.d/$REGISTRY\_NAME:5000/ca.crt; done**

**tls.crt 100% 1822 2.7MB/s 00:00**

**tls.crt 100% 1822 2.4MB/s 00:00**

**tls.crt 100% 1822 1.6MB/s 00:00**

**tls.crt 100% 1822 2.1MB/s 00:00**

**Testing our Private Docker Registry**

Now, let us try to login to the registry from the master node, using the same credentials we created earlier:

root@master1:/# **docker login docker-registry:5000 -u myuser -p mypasswd**

WARNING! Using --password via the CLI is insecure. Use --password-stdin.

WARNING! Your password will be stored unencrypted in /root/.docker/config.json.

Configure a credential helper to remove this warning. See

https://docs.docker.com/engine/reference/commandline/login/#credentials-store

**Login Succeeded**

**Hurray! This worked!!**

**Before we start using our registry, let us create a Secret of type docker-registry which uses the credentials myuser/mypasswd for enabling all the nodes in our cluster to authenticate with our private Docker registry.**

**root@master1:/# kubectl create secret docker-registry registry-credential --docker-server=docker-registry:5000 --docker-username=** **myuser --docker-password=mypassword-n cctns-new**

**secret/ registry-credential created**

**Let us try to push a custom image to our private Docker registry.**

**root@master1:/# docker pull nginx**

**Using default tag: latest**

**latest: Pulling from library/nginx**

**bf5952930446: Pull complete**

**cb9a6de05e5a: Pull complete**

**9513ea0afb93: Pull complete**

**b49ea07d2e93: Pull complete**

**a5e4a503d449: Pull complete**

**Digest: sha256:b0ad43f7ee5edbc0effbc14645ae7055e21bc1973aee5150745632a24a752661**

**Status: Downloaded newer image for nginx:latest**

**docker.io/library/nginx:latest**

**root@master1:/#**

**root@master1:/# docker tag nginx:latest docker-registry:5000/mynginx:v1**

**root@master1:/#**

**root@master1:/# docker push docker-registry:5000/mynginx:v1**

**The push refers to repository [docker-registry:5000/mynginx]**

**550333325e31: Layer already exists**

**22ea89b1a816: Layer already exists**

**a4d893caa5c9: Layer already exists**

**0338db614b95: Layer already exists**

**d0f104dc0a1f: Layer already exists**

**v1: digest: sha256:179412c42fe3336e7cdc253ad4a2e03d32f50e3037a860cf5edbeb1aaddb915c size: 1362**

**root@master1:/#**