Working With CongfigMap and Private Registry Using Secrets

[Edition 1]

[Last Update 200910]

For any issues/help contact : ***support@k21academy.com***

Contents

1 INTRODUCTION

A **Kubernetes cluster** uses the Secret of **docker**-**registry** type to authenticate with a container **registry** to pull a **private** image. If you need more control (for example, to set a namespace or a label on the new secret) then you can customise the Secret before storing it.

To share access to your private container images across multiple services and revisions, you create a list of Kubernetes secrets (imagePullSecrets) using your registry credentials, add that imagePullSecrets to your default service account, and then deploy those configurations to your Knative cluster.

This guide Covers:

1. Configure Private Registry in Kubernetes Cluster

    - Logging to docker registry
    - Create a Secret based on Docker credentials
    - Create a Pod that uses the Secret to pull Image

2 DOCUMENTATION

2.1 Kubernetes Documentation

1. Docker Registry
   https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/
2. Kubernetes Secrets
   https://kubernetes.io/docs/concepts/configuration/secret/
3. Credentials Securely using Secrets
   https://kubernetes.io/docs/tasks/inject-data-application/distribute-credentials-secure/
4. Private Registry
   https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/
5. ConfigMaps
   https://kubernetes.io/docs/concepts/configuration/configmap/

2.2 Linux Commands and VIM Commands

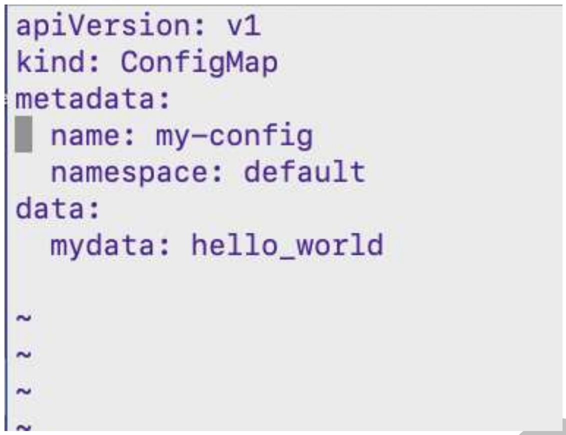
1. Basic Linux Commands
   https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners
   https://www.hostinger.in/tutorials/linux-commands
2. Basic VIM Commands
   https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started
3. Popular VIM Commands
   https://www.keycdn.com/blog/vim-commands

3 WORKING WITH CONFIGMAP

3.1 Setting Container Environment Variables using ConfigMap

1. Create a ConfigMap from the yaml file and enter the contents given below

```
$ vi config-map.yaml
```

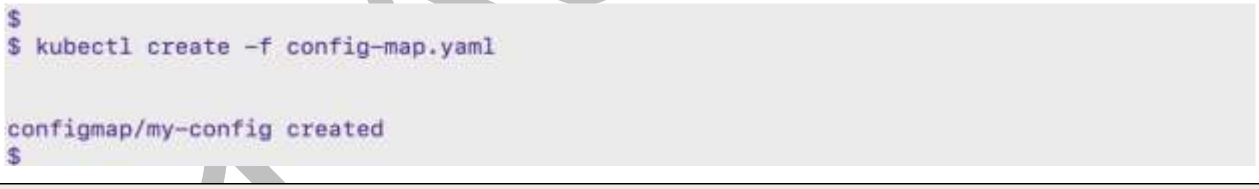


```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: default
data:
  mydata: hello_world

~
~
~
~
```

2. Create the ConfigMap using the yaml

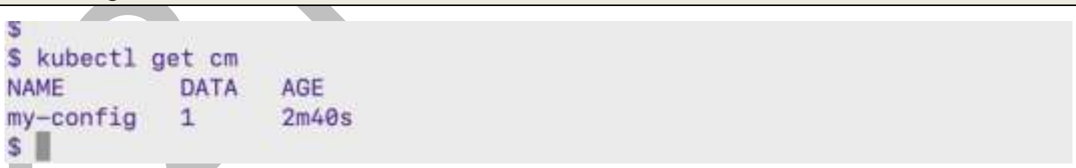
```
$ kubectl create -f config-map.yaml
```



```
$
$ kubectl create -f config-map.yaml


configmap/my-config created
$
```

```
$ kubectl get cm
```

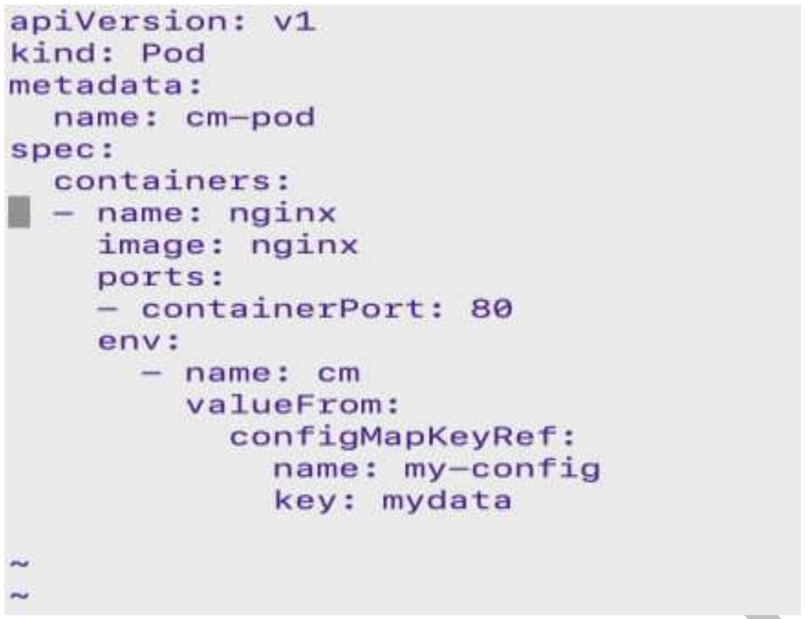


```
$
$ kubectl get cm
NAME        DATA   AGE
my-config   1      2m40s
$
```

3.2 Create Pod that Uses ConfigMap

1. View and create the pod from configmap-pod.yaml file

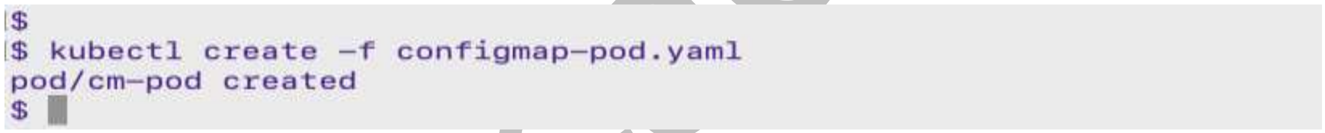
```
$ vi configmap-pod.yaml
```



```
apiVersion: v1
kind: Pod
metadata:
  name: cm-pod
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    env:
      - name: cm
        valueFrom:
          configMapKeyRef:
            name: my-config
            key: mydata

~
~
```

```
$ kubectl create -f configmap-pod.yaml
```

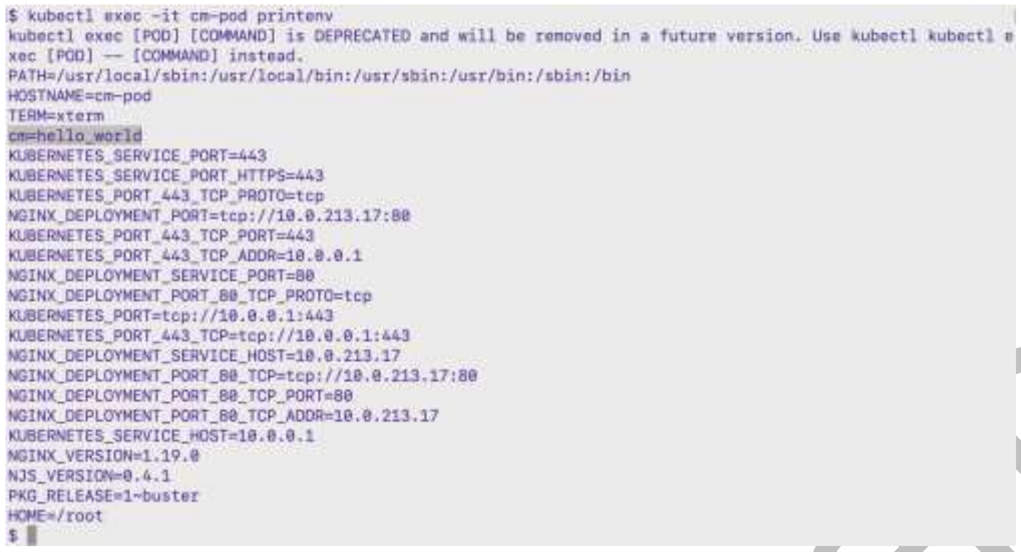


```
$
$ kubectl create -f configmap-pod.yaml
pod/cm-pod created
$
```

3.3 Verify Pod uses ConfigMap to set the Environmental Variable

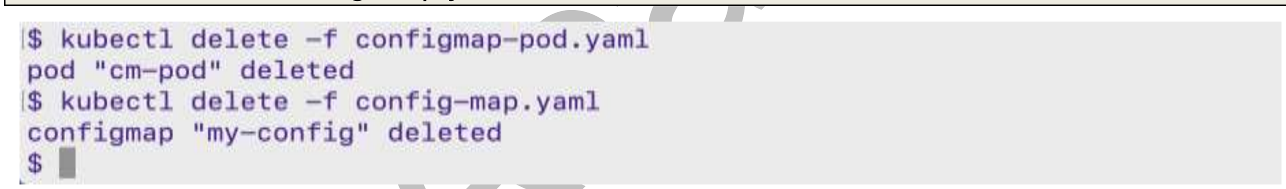
1. Once the Pod is up, verify that the environment variable specified in the ConfigMap is set in the container

```
$ kubectl exec -it cm-pod printenv
```


```
$ kubectl exec -it cm-pod printenv
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl e
xec [POD] -- [COMMAND] instead.
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=cm-pod
TERM=xterm
cm=hello_world
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP_PROTO=tcp
NGINX_DEPLOYMENT_PORT=tcp://10.0.213.17:80
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_ADDR=10.0.0.1
NGINX_DEPLOYMENT_SERVICE_PORT=80
NGINX_DEPLOYMENT_PORT_80_TCP_PROTO=tcp
KUBERNETES_PORT=tcp://10.0.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.0.0.1:443
NGINX_DEPLOYMENT_SERVICE_HOST=10.0.213.17
NGINX_DEPLOYMENT_PORT_80_TCP=tcp://10.0.213.17:80
NGINX_DEPLOYMENT_PORT_80_TCP_PORT=80
NGINX_DEPLOYMENT_PORT_80_TCP_ADDR=10.0.213.17
KUBERNETES_SERVICE_HOST=10.0.0.1
NGINX_VERSION=1.19.0
NJS_VERSION=0.4.1
PKG_RELEASE=1~buster
HOME=/root
$
```

3.4 Clean-up the Resources

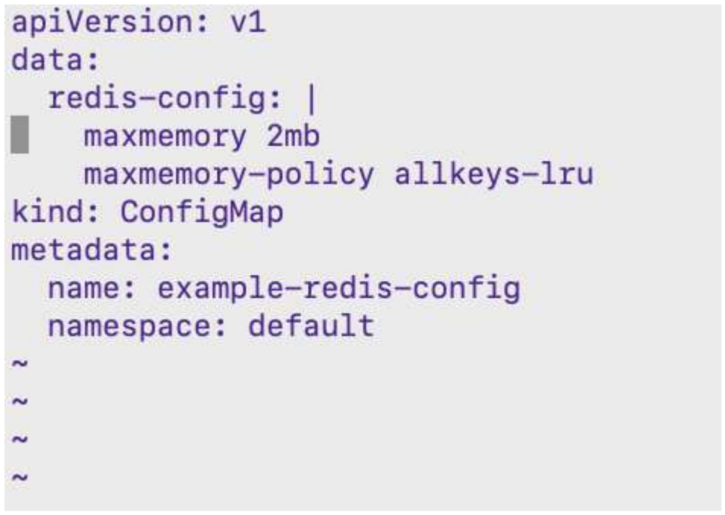
```
$ kubectl delete -f configmap-pod.yaml
$ kubectl delete -f config-map.yaml
```


```
$ kubectl delete -f configmap-pod.yaml
pod "cm-pod" deleted
$ kubectl delete -f config-map.yaml
configmap "my-config" deleted
$
```

3.5 Setting Configuration File with Volume using ConfigMap

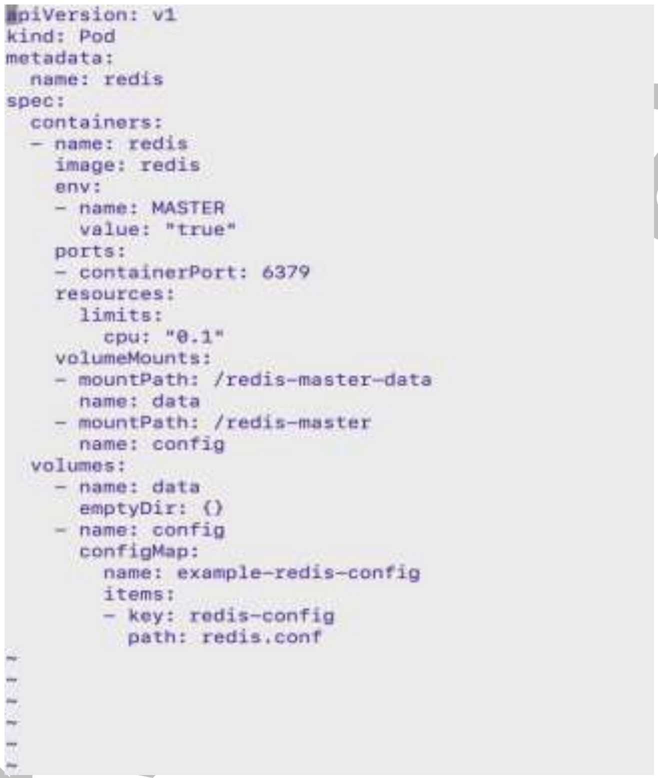
1. View the below ConfigMap and Pod yaml files and create the resources using them.

```
$ vim redis-cm.yaml
```



```
apiVersion: v1
data:
  redis-config: |
    maxmemory 2mb
    maxmemory-policy allkeys-lru
kind: ConfigMap
metadata:
  name: example-redis-config
  namespace: default
~
~
~
~
```

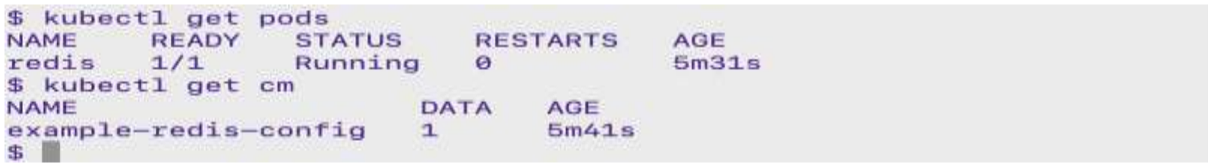
```
$ vim redis-pod.yaml
```



```
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
  - name: redis
    image: redis
    env:
    - name: MASTER
      value: "true"
    ports:
    - containerPort: 6379
    resources:
      limits:
        cpu: "0.1"
    volumeMounts:
    - mountPath: /redis-master-data
      name: data
    - mountPath: /redis-master
      name: config
  volumes:
    - name: data
      emptyDir: {}
    - name: config
      configMap:
        name: example-redis-config
        items:
        - key: redis-config
          path: redis.conf
~
~
~
~
~
~
```

2. Verify by listing the created resources

```
$ kubectl get pods
$ kubectl get cm
```

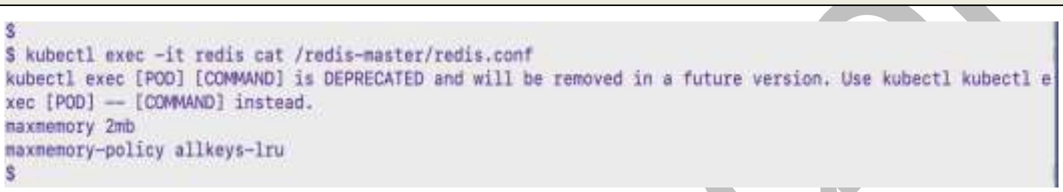


```
$ kubectl get pods
NAME     READY    STATUS     RESTARTS    AGE
redis    1/1      Running    0           5m31s
$ kubectl get cm
NAME                       DATA    AGE
example-redis-config       1       5m41s
$
```

3.6 Verify Mounting of ConfigMap as Volume

1. See that the config file redis.conf is present at /redis-master/ and is having the contents specified in the ConfigMap

```
$ kubectl exec -it redis cat /redis-master/redis.conf
```



```
$
$ kubectl exec -it redis cat /redis-master/redis.conf
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl e
xec [POD] -- [COMMAND] instead.
maxmemory 2mb
maxmemory-policy allkeys-lru
$
```

3.7 Delete all the resources created in this task

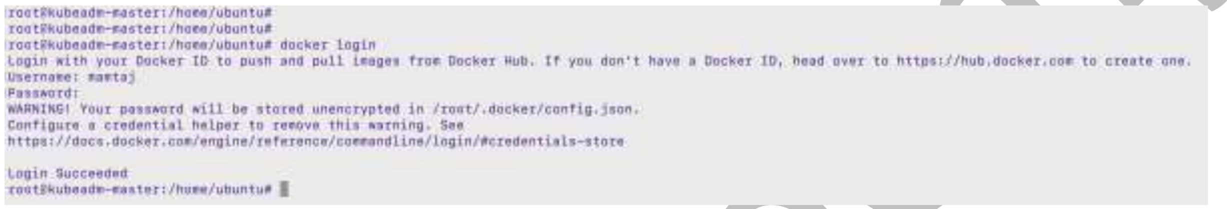
```
$ kubectl delete -f redis-pod.yaml
$ kubectl delete -f redis-cm.yaml
```

4 CONFIGURE PRIVATE REGISTRY IN KUBERNETES CLUSTER

4.1 Logging to docker registry

1. Login to Docker Registry in order to pull a private image. When prompted, enter your Docker username and password.

```
$ docker login
```

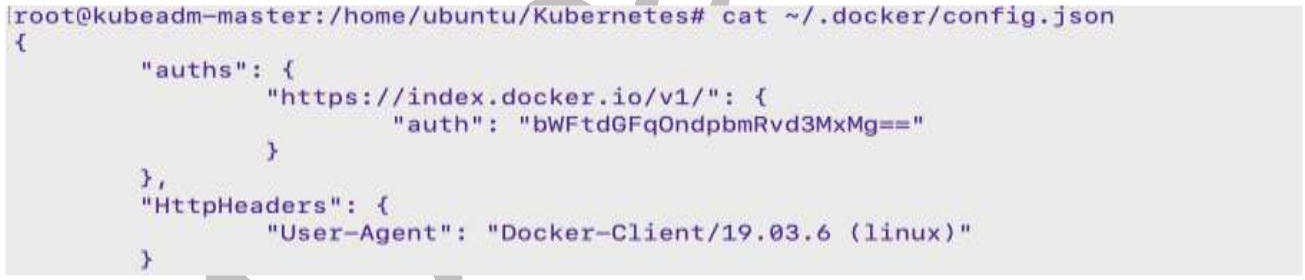


```
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: mamtaj
Password:
WARNING! 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
root@kubeadm-master:/home/ubuntu#
```

2. The login process creates or updates a config.json file that holds an authorization token. View the config.json file:

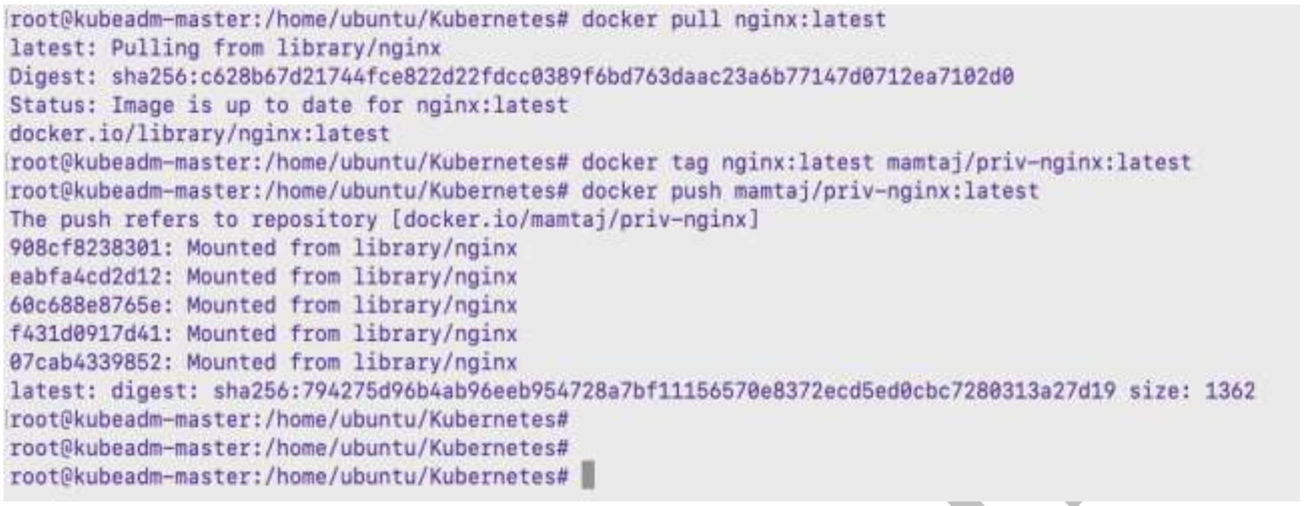
```
$ cat ~/.docker/config.json
```



```
root@kubeadm-master:/home/ubuntu/Kubernetes# cat ~/.docker/config.json
{
        "auths": {
                "https://index.docker.io/v1/": {
                        "auth": "bWFtdGFqOndpbmRvd3MxMg=="
                }
        },
        "HttpHeaders": {
                "User-Agent": "Docker-Client/19.03.6 (linux)"
        }
```

3. The pull nginx:latest image, tag it as per your private registry and push it to the docker hub private registry server

```
$ docker pull nginx:latest
$ docker tag nginx:latest mamtaj/priv-nginx:latest
$ docker push mamtaj/priv-nginx:latest
```



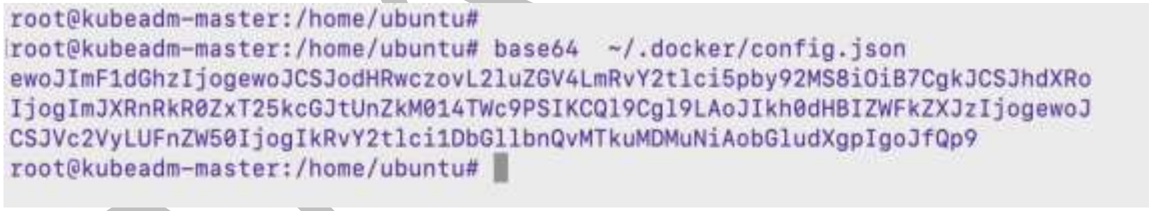
```
root@kubeadm-master:/home/ubuntu/Kubernetes# docker pull nginx:latest
latest: Pulling from library/nginx
Digest: sha256:c628b67d21744fce822d22fdcc0389f6bd763daac23a6b77147d0712ea7102d0
Status: Image is up to date for nginx:latest
docker.io/library/nginx:latest
root@kubeadm-master:/home/ubuntu/Kubernetes# docker tag nginx:latest mamtaj/priv-nginx:latest
root@kubeadm-master:/home/ubuntu/Kubernetes# docker push mamtaj/priv-nginx:latest
The push refers to repository [docker.io/mamtaj/priv-nginx]
908cf8238301: Mounted from library/nginx
eabfa4cd2d12: Mounted from library/nginx
60c688e8765e: Mounted from library/nginx
f431d0917d41: Mounted from library/nginx
07cab4339852: Mounted from library/nginx
latest: digest: sha256:794275d96b4ab96eeb954728a7bf11156570e8372ecd5ed0cbc7280313a27d19 size: 1362
root@kubeadm-master:/home/ubuntu/Kubernetes#
root@kubeadm-master:/home/ubuntu/Kubernetes#
root@kubeadm-master:/home/ubuntu/Kubernetes# 
```

4.2 Create a Secret based on Docker credentials

3. A Kubernetes cluster uses the Secret of docker-registry type to authenticate with a container registry to pull a private image

4. base64 encode the docker file and save the string without breaking the value

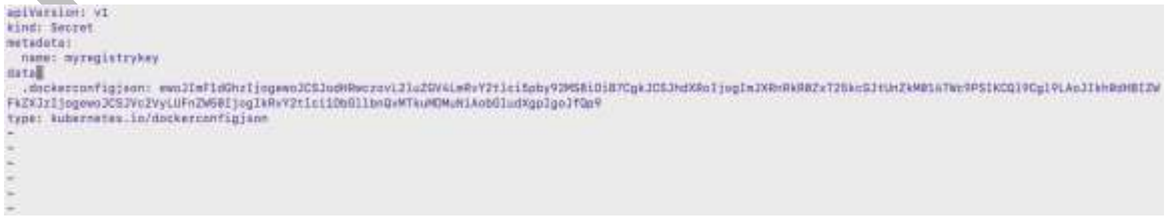
```
$ base64  ~/.docker/config.json
```



```
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu# base64  ~/.docker/config.json
ewoJImF1dGhzIjogewoJCSJodHRwczovL2luZGV4LmRvY2tlci5pby92MS8iOiB7CgkJCSJhdXRo
IjogImJXRnRkR0ZxT25kcGJtUnZkM014TWc9PSIKCQl9Cgl9LAoJIkh0dHBIZWFkZXJzIjogewoJ
CSJVc2VyLUFnZW50IjogIkRvY2tlci1DbG1lbnQvMTkuMDMuNiAobGludXgpIgoJfQp9
root@kubeadm-master:/home/ubuntu# 
```

5. Create yaml file to create Secret, set the name of the data item to .dockerconfigjson, set type to kubernetes.io/dockerconfigjson and paste the base64 encoded string, unbroken as the value for field data[".dockerconfigjson"]

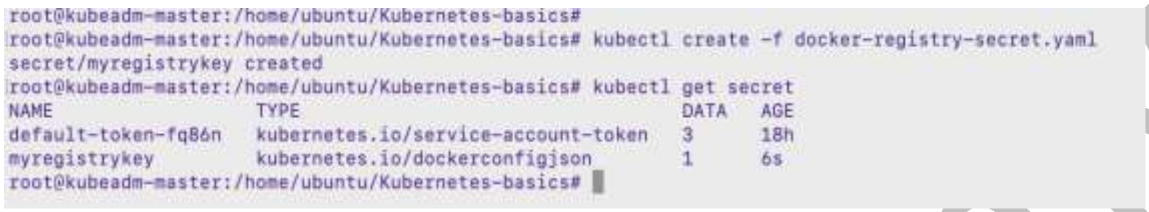
```
$ vim docker-registry-secret.yaml
```



```
apiVersion: v1
kind: Secret
metadata:
  name: myregistrykey
data:
  .dockerconfigjson: ewoJImF1dGhzIjogewoJCSJodHRwczovL2luZGV4LmRvY2tlci5pby92MS8iOiB7CgkJCSJhdXRoIjogImJXRnRkR0ZxT25kcGJtUnZkM014TWc9PSIKCQl9Cgl9LAoJIkh0dHBIZW
FkZXJzIjogewoJCSJVc2VyLUFnZW50IjogIkRvY2tlci1DbG1lbnQvMTkuMDMuNiAobGludXgpIgoJfQp9
type: kubernetes.io/dockerconfigjson
~
~
~
~
~
~
```

4. Create secret from the above yaml file and list it

```
$ kubectl create -f docker-registry-secret.yaml
$ kubectl get secret
```



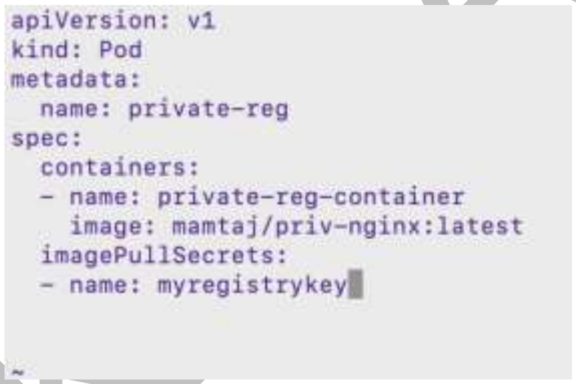
```
root@kubeadm-master:/home/ubuntu/Kubernetes-basics#
root@kubeadm-master:/home/ubuntu/Kubernetes-basics# kubectl create -f docker-registry-secret.yaml
secret/myregistrykey created
root@kubeadm-master:/home/ubuntu/Kubernetes-basics# kubectl get secret
NAME                  TYPE                                  DATA   AGE
default-token-fq86n   kubernetes.io/service-account-token   3      18h
myregistrykey         kubernetes.io/dockerconfigjson        1      6s
root@kubeadm-master:/home/ubuntu/Kubernetes-basics#
```

4.3 Create a Pod that uses the Secret to pull Image

1. View the yaml file to create pod that needs access the Docker Credentials on run time to pull the image

2. To pull the image from the private registry, Kubernetes needs credentials. The **imagePullSecrets** field in the configuration file specifies that Kubernetes should get the credentials from a Secret named **myregistrykey**

```
$ vim priv-reg-pod.yaml
```

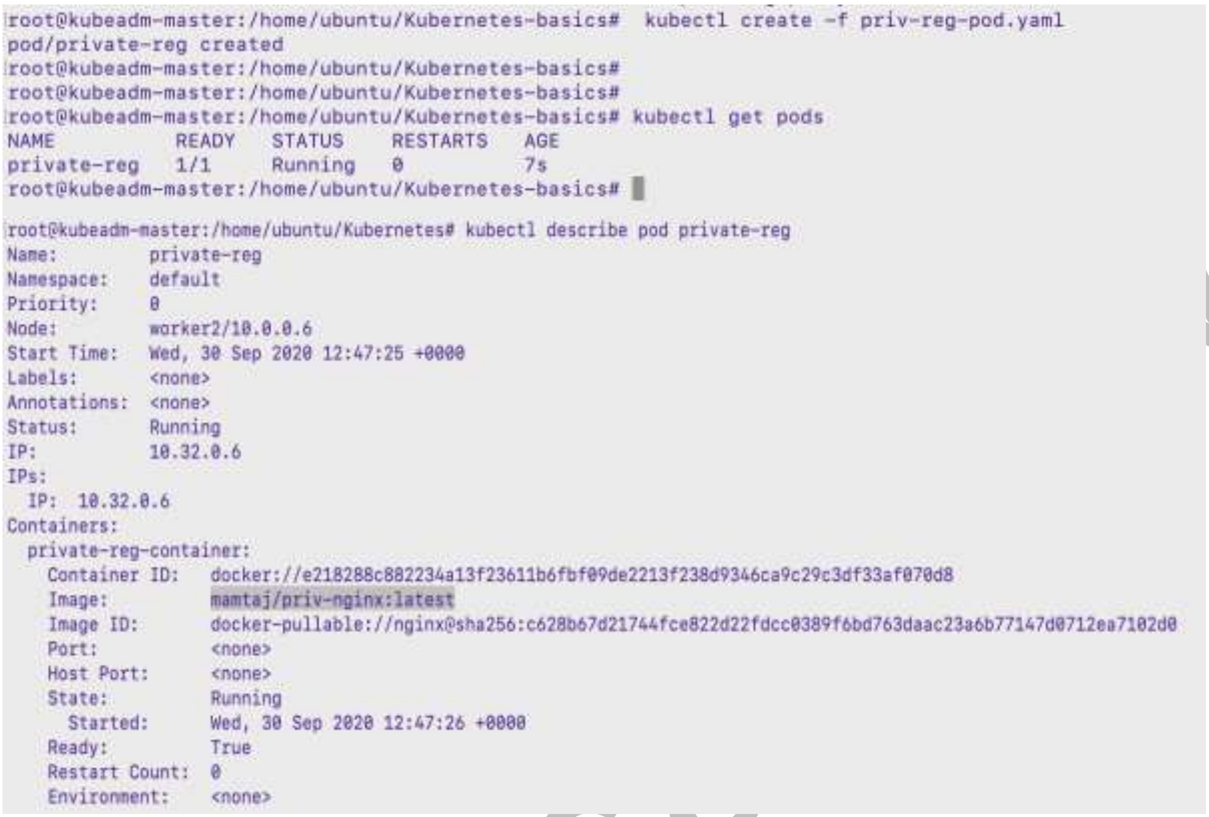


```
apiVersion: v1
kind: Pod
metadata:
  name: private-reg
spec:
  containers:
  - name: private-reg-container
    image: mamtaj/priv-nginx:latest
  imagePullSecrets:
  - name: myregistrykey


~
```

4. Create pod with above configuration yaml file and verify that container image is successfully pulled using Docker credentials and Pod goes to running state

```
$ kubectl create -f priv-reg-pod.yaml
$ kubectl get pods
$ kubectl describe pod private-reg
```



```
root@kubeadm-master:/home/ubuntu/Kubernetes-basics#  kubectl create -f priv-reg-pod.yaml
pod/private-reg created
root@kubeadm-master:/home/ubuntu/Kubernetes-basics#
root@kubeadm-master:/home/ubuntu/Kubernetes-basics#
root@kubeadm-master:/home/ubuntu/Kubernetes-basics# kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
private-reg   1/1     Running   0          7s
root@kubeadm-master:/home/ubuntu/Kubernetes-basics#
```

```
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl describe pod private-reg
Name:         private-reg
Namespace:    default
Priority:     0
Node:         worker2/10.0.0.6
Start Time:   Wed, 30 Sep 2020 12:47:25 +0000
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           10.32.0.6
IPs:
  IP:  10.32.0.6
Containers:
  private-reg-container:
    Container ID:   docker://e218288c882234a13f23611b6fbf09de2213f238d9346ca9c29c3df33af070d8
    Image:          mamtaj/priv-nginx:latest
    Image ID:       docker-pullable://nginx@sha256:c628b67d21744fce822d22fdcc0389f6bd763daac23a6b77147d0712ea7102d0
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Wed, 30 Sep 2020 12:47:26 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
```

4.4 CleanUp the RESOURCES

1. Delete the secret and pod created during this lab exercise

```
$ kubectl delete -f priv-reg-pod.yaml
$ kubectl delete -f docker-registry-secret.yaml
```

5 SUMMARY

In this guide we Covered:

1. Configure Private Registry in Kubernetes Cluster

   - Logging to docker registry
   - Create a Secret based on Docker credentials
   - Create a Pod that uses the Secret to pull Image