Jonathan

Follow 29 Followers About

Certified Kubernetes Security Specialist (CKS) Preparation Part 7 — Supply Chain Security

Jonathan Mar 17 · 5 min read

If you have not yet checked the previous parts of this series, please go ahead and check Part1, Part2, Part3, Part4, Part5 and Part6.

In this article, I would focus on the preparation around **supply chain security** in CKS certification exam.

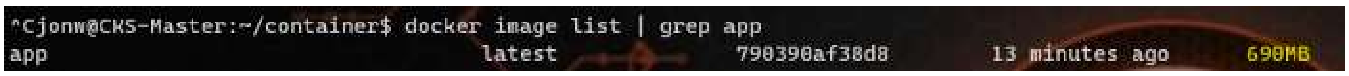
Docker Image Management

To be honest, I have never really gotten familiar with Docker image as I seldom get the chance to go to that stage of the pipeline. However, in this exam, we would need to have the general idea of how Docker image works and what would be the practice of making this process safe and efficient.

Basically, the two principles below are for reducing Docker image size. This article provides great explanation around this.

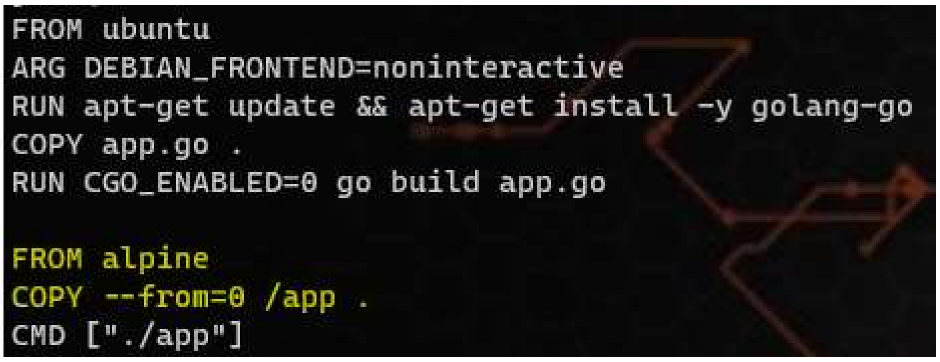
- Light-weight Image as base
- Multi-stage Build

In practice, we could test buy creating a container running with original Docker file and we are seeing the image consists around 690 MB.



```
^Cjonw@CKS-Master:~/container$ docker image list | grep app
app                       latest              790390af38d8        13 minutes ago      690MB
```

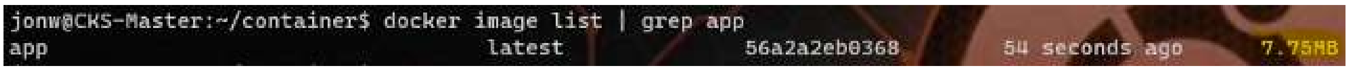
If we apply the 2 principles above,



```
FROM ubuntu
ARG DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y golang-go
COPY app.go .
RUN CGO_ENABLED=0 go build app.go

FROM alpine
COPY --from=0 /app .
CMD ["./app"]
```

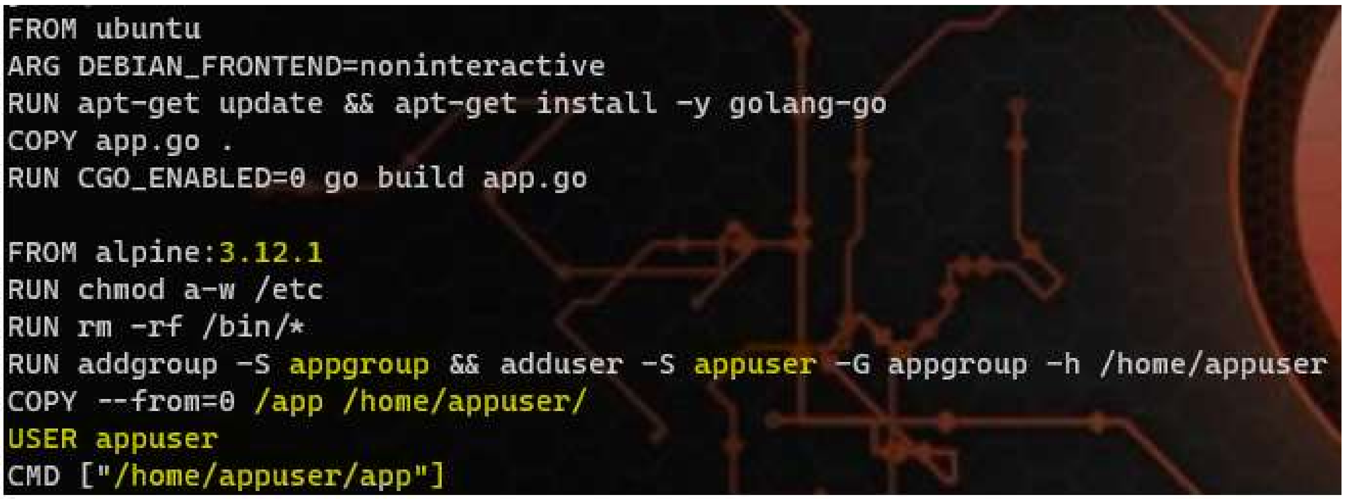
we would see Docker image size being hugely reduced. Mainly, this is because from the second stage (stage 1), it just acquires that components needed from the first stage (stage 0) and that removes a lot of unnecessary files.



```
jonw@CKS-Master:~/container$ docker image list | grep app
app                                      latest              56a2a2eb0368        54 seconds ago      7.75MB
```

With 4 principles below, we could make Docker image more secure.

- Use specific package versions
- Do not run as root
- Make file system read only
- Remove shell access



```
FROM ubuntu
ARG DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y golang-go
COPY app.go .
RUN CGO_ENABLED=0 go build app.go

FROM alpine:3.12.1
RUN chmod a-w /etc
RUN rm -rf /bin/*
RUN addgroup -S appgroup && adduser -S appuser -G appgroup -h /home/appuser
COPY --from=0 /app /home/appuser/
USER appuser
CMD ["/home/appuser/app"]
```

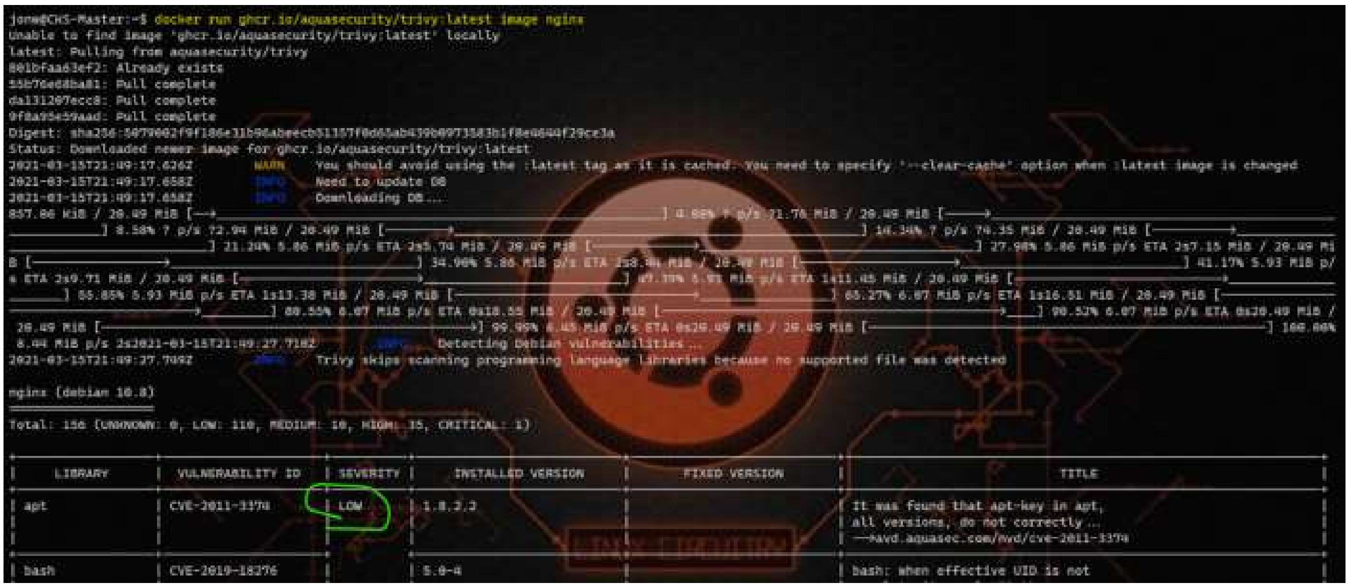
Make sure the Docker image still runs after updating the Dockerfile.

Image Security — Trivy

Since we would be running all kinds of Docker image within containers (Pods), one extremely important thing is to ensure container images are always without security concern. Trivy was developed for this sole purpose and it is extremely easy to use.

Trivy could be installed through OS and run as a service or simply run the service inside a Docker container. General guidance on Trivy commands to scan container images. In this article, we would run Trivy inside a Docker container.

```
docker run ghcr.io/aquasecurity/trivy:latest image nginx
```



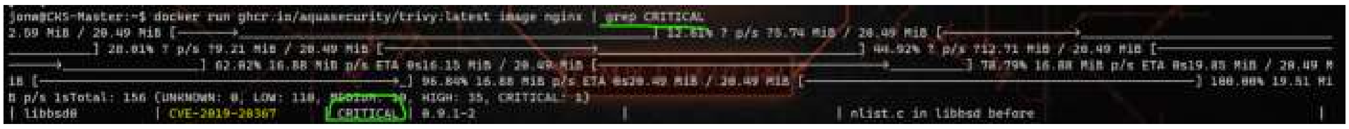
```
jone@CKS-Master:~$ docker run ghcr.io/aquasecurity/trivy:latest image nginx
Unable to find image 'ghcr.io/aquasecurity/trivy:latest' locally
latest: Pulling from aquasecurity/trivy
801bfaa63ef2: Already exists
55b76e68ba81: Pull complete
da131207ecc8: Pull complete
9f8a95e59aad: Pull complete
Digest: sha256:5079002f9f186e31b96abeecb51357f0d65ab439b0973583b1f8e0644f29ce3a
Status: Downloaded newer image for ghcr.io/aquasecurity/trivy:latest
2021-03-15T21:49:17.626Z        WARN    You should avoid using the :latest tag as it is cached. You need to specify '--clear-cache' option when :latest image is changed
2021-03-15T21:49:17.658Z        INFO    Need to update DB
2021-03-15T21:49:17.658Z        INFO    Downloading DB...
2021-03-15T21:49:27.709Z        INFO    Detecting Debian vulnerabilities...
2021-03-15T21:49:27.709Z        INFO    Trivy skips scanning programming language libraries because no supported file was detected

nginx (debian 10.8)
===================
Total: 156 (UNKNOWN: 0, LOW: 110, MEDIUM: 10, HIGH: 35, CRITICAL: 1)

+---------+------------------+----------+-------------------+---------------+--------------------------------------+
| LIBRARY | VULNERABILITY ID | SEVERITY | INSTALLED VERSION | FIXED VERSION |                TITLE                 |
+---------+------------------+----------+-------------------+---------------+--------------------------------------+
| apt     | CVE-2011-3374    | LOW      | 1.8.2.2           |               | It was found that apt-key in apt,    |
|         |                  |          |                   |               | all versions, do not correctly...    |
|         |                  |          |                   |               | -->avd.aquasec.com/nvd/cve-2011-3374 |
+---------+------------------+----------+-------------------+---------------+--------------------------------------+
| bash    | CVE-2019-18276   |          | 5.0-4             |               | bash: when effective UID is not      |
```

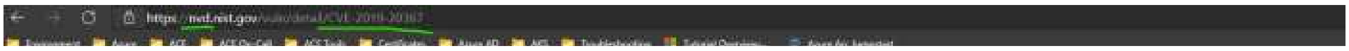
Search container images with CRITICAL severity.

```
docker run ghcr.io/aquasecurity/trivy:latest image nginx | grep
CRITICAL
```

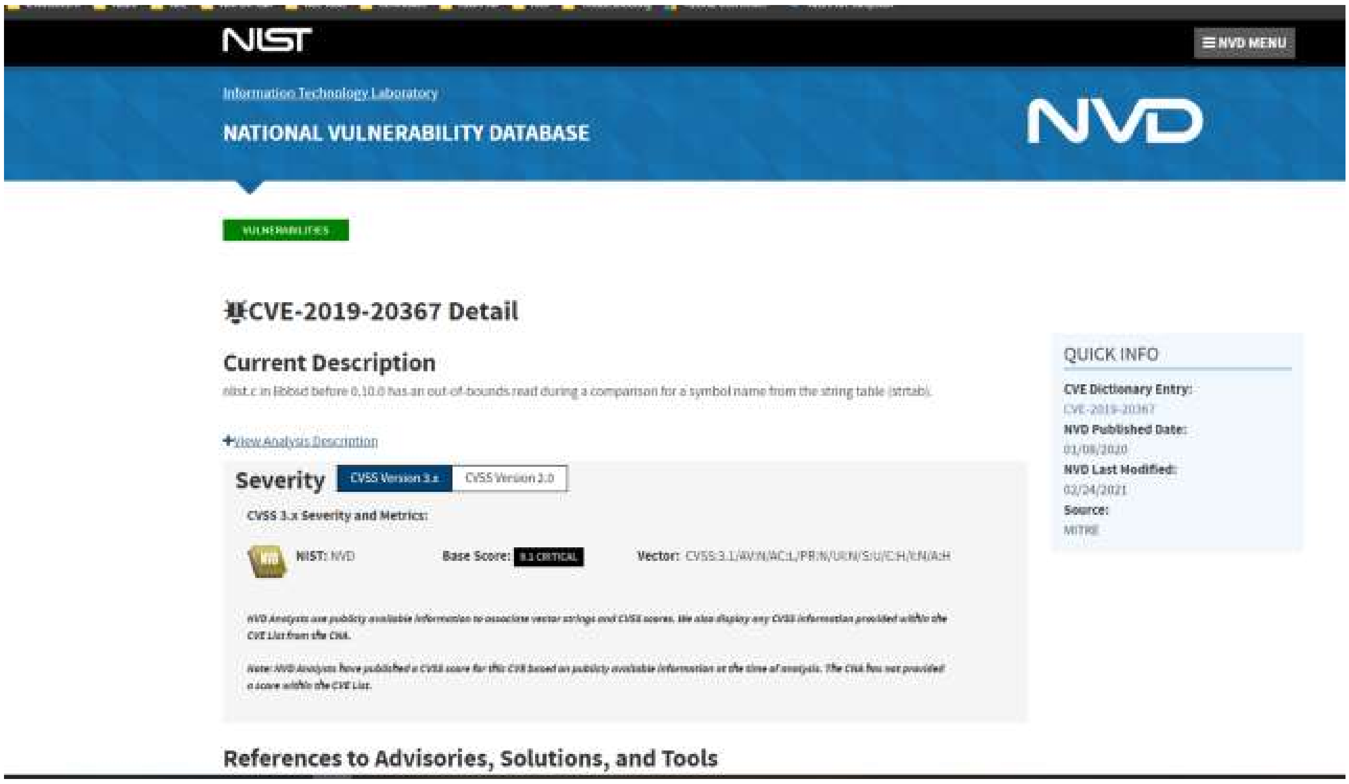


```
jone@CKS-Master:~$ docker run ghcr.io/aquasecurity/trivy:latest image nginx | grep CRITICAL
B p/s 1sTotal: 156 (UNKNOWN: 0, LOW: 110, MEDIUM: 10, HIGH: 35, CRITICAL: 1)
| libbsd0       | CVE-2019-20367      | CRITICAL | 0.9.1-2      |         | nlist.c in libbsd before        |
```

Once we get the details on container image vulnerability, we could look it up in the national vulnerability database.



https://nvd.nist.gov/vuln/detail/CVE-2019-20367


NIST
NVD MENU
Information Technology Laboratory
NATIONAL VULNERABILITY DATABASE
NVD
VULNERABILITIES

CVE-2019-20367 Detail

Current Description

nlist.c in libbsd before 0.10.0 has an out-of-bounds read during a comparison for a symbol name from the string table (strtab).

+View Analysis Description

Severity CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD Base Score: 9.1 CRITICAL Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

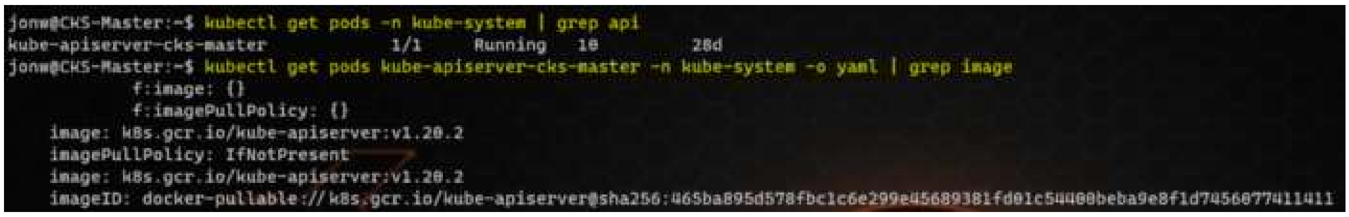
Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

QUICK INFO
CVE Dictionary Entry: CVE-2019-20367
NVD Published Date: 01/08/2020
NVD Last Modified: 02/24/2021
Source: MITRE

References to Advisories, Solutions, and Tools

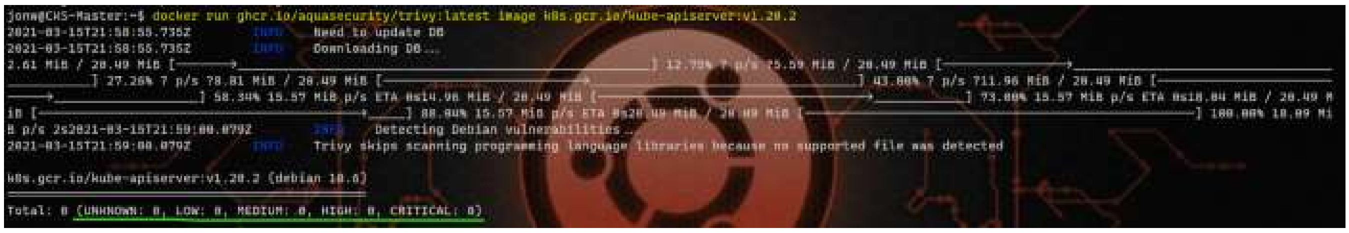
Maybe we could take a look at the container images kube-apiserver is using.

```
- kubectl get pods -n kube-system | grep api
- kubectl get pods kube-apiserver-cks-master -n kube-system -o yaml
| grep image
```



```
jonw@CKS-Master:~$ kubectl get pods -n kube-system | grep api
kube-apiserver-cks-master            1/1     Running   10         28d
jonw@CKS-Master:~$ kubectl get pods kube-apiserver-cks-master -n kube-system -o yaml | grep image
          f:image: {}
          f:imagePullPolicy: {}
    image: k8s.gcr.io/kube-apiserver:v1.20.2
    imagePullPolicy: IfNotPresent
    image: k8s.gcr.io/kube-apiserver:v1.20.2
    imageID: docker-pullable://k8s.gcr.io/kube-apiserver@sha256:465ba895d578fbc1c6e299e45689381fd01c54400beba9e8f1d7456077411411
```

```
docker run ghcr.io/aquasecurity/trivy:latest image k8s.gcr.io/kube-
apiserver:v1.20.2
```



```
jonw@CKS-Master:~$ docker run ghcr.io/aquasecurity/trivy:latest image k8s.gcr.io/kube-apiserver:v1.20.2
2021-03-15T21:58:55.735Z        INFO    Need to update DB
2021-03-15T21:58:55.735Z        INFO    Downloading DB...
2.61 MiB / 20.49 MiB [------>______________________________] 12.75% ? p/s ?5.59 MiB / 20.49 MiB [---------->______________
__________] 27.28% ? p/s ?8.81 MiB / 20.49 MiB [----------------->______________________] 43.00% ? p/s ?11.96 MiB / 20.49 MiB [-------
--->_______________] 58.34% 15.57 MiB p/s ETA 0s14.96 MiB / 20.49 MiB [------------------->________] 73.00% 15.57 MiB p/s ETA 0s18.04 MiB / 20.49 M
iB [----------------------->____] 88.04% 15.57 MiB p/s ETA 0s20.49 MiB / 20.49 MiB [----------------------------] 100.00% 10.09 Mi
B p/s 2s2021-03-15T21:59:00.079Z        INFO    Detecting Debian vulnerabilities...
2021-03-15T21:59:00.079Z        INFO    Trivy skips scanning programming language libraries because no supported file was detected

k8s.gcr.io/kube-apiserver:v1.20.2 (debian 10.6)
===============================================
Total: 0 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 0, CRITICAL: 0)
```

At the end of the day, please ensure all services and applications are running on container images with no major security concerns.

Image Policy Webhook

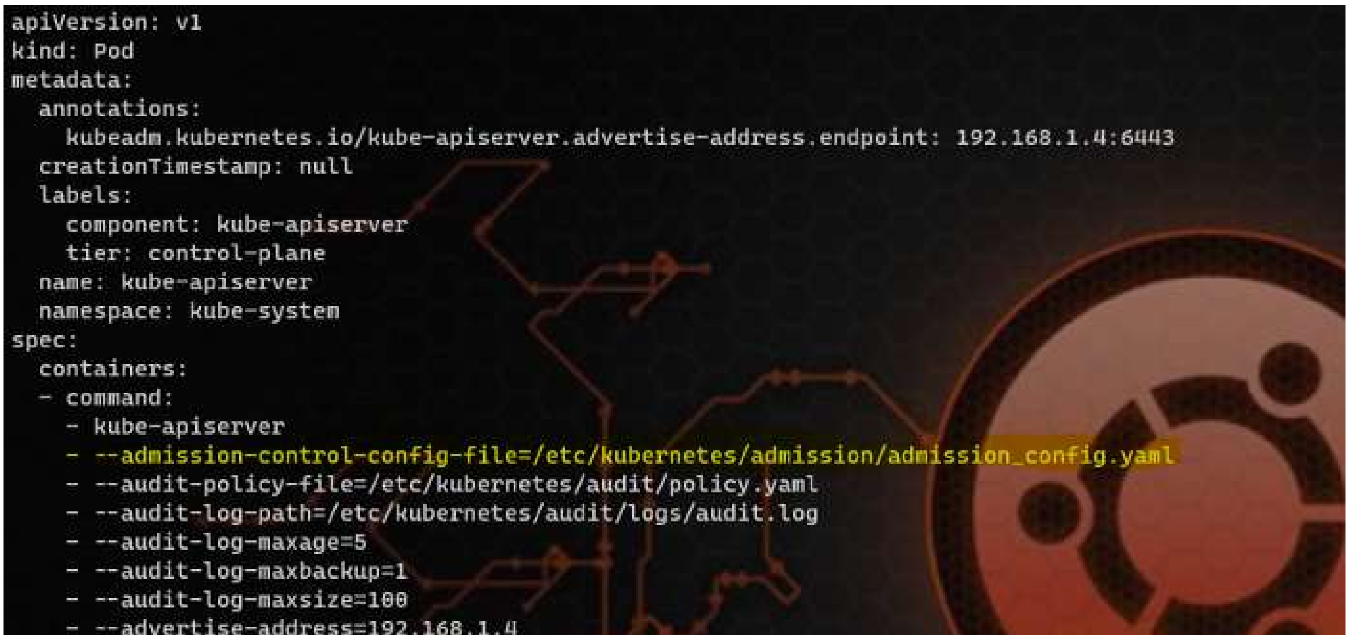
Image Policy Webhook is the connecting piece of letting backend admission controller to go through external server for checking image security and integrity before making any CRUD operation on K8s clusters.

> *An admission controller is a piece of code that intercepts requests to the Kubernetes API server prior to persistence of the object, but after the request is authenticated and authorized. The controllers consist of the list below, are compiled into the `kube-apiserver` binary, and may only be configured by the cluster administrator. In that list, there are two special controllers: MutatingAdmissionWebhook and ValidatingAdmissionWebhook. These execute the mutating and validating (respectively) admission control webhooks which are configured in the API.*

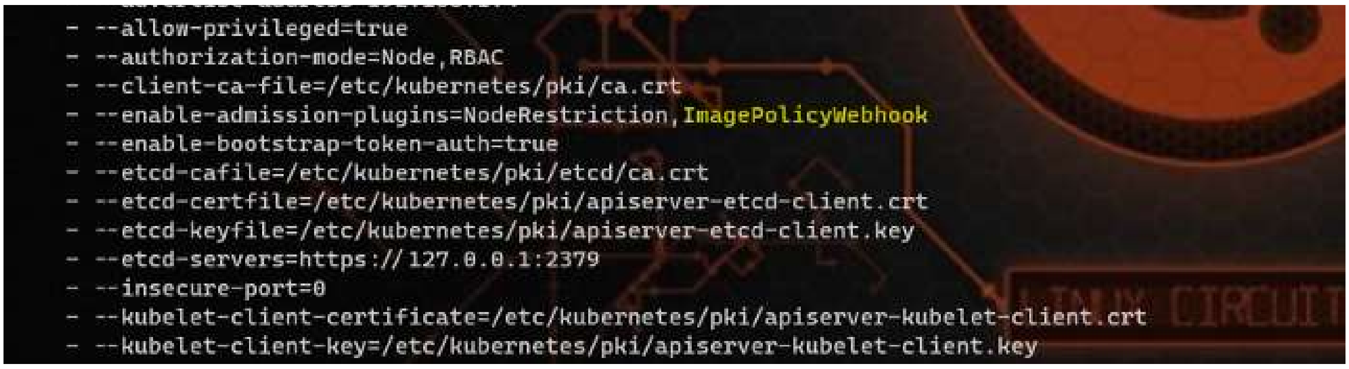
— Quoted from here

First, we add “ImagePolicyWebhook” at the end of admission plugins.

```
sudo nano /etc/kubernetes/manifests/kube-apiserver.yaml
```

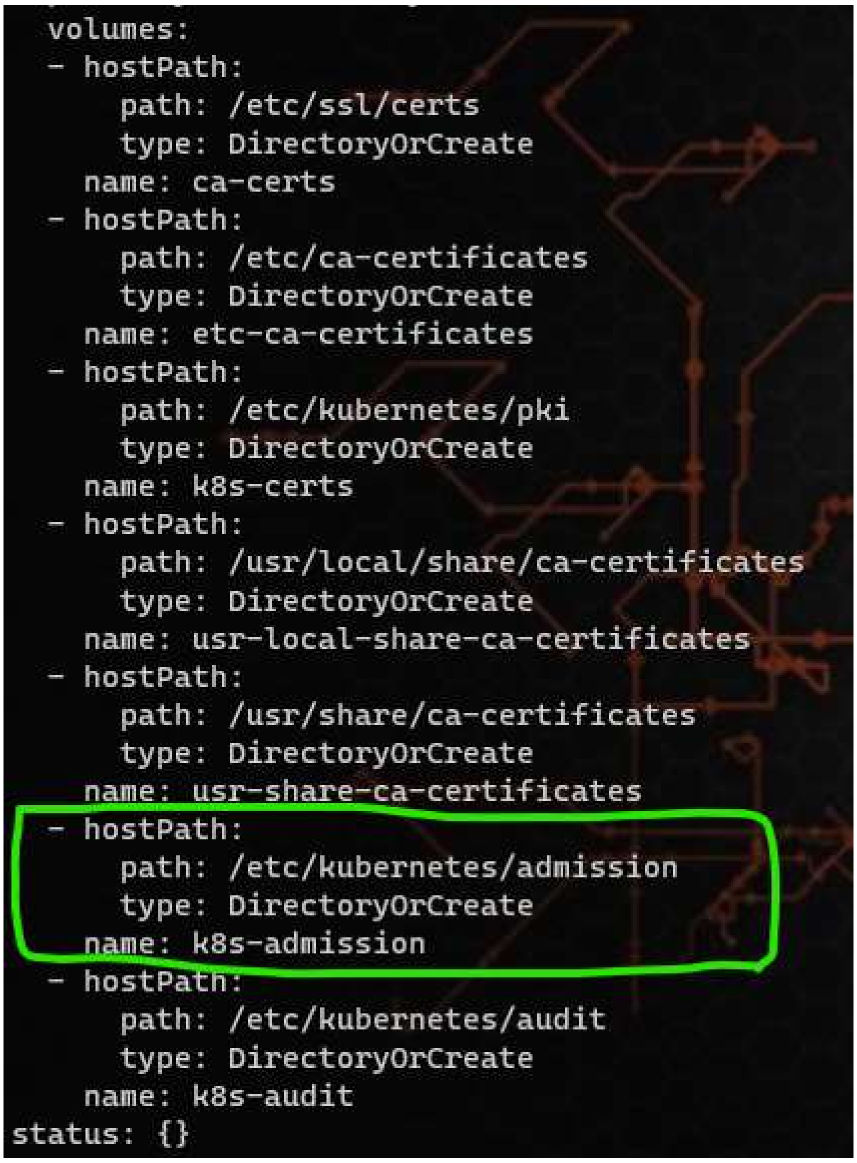


```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 192.168.1.4:6443
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --admission-control-config-file=/etc/kubernetes/admission/admission_config.yaml
    - --audit-policy-file=/etc/kubernetes/audit/policy.yaml
    - --audit-log-path=/etc/kubernetes/audit/logs/audit.log
    - --audit-log-maxage=5
    - --audit-log-maxbackup=1
    - --audit-log-maxsize=100
    - --advertise-address=192.168.1.4
```

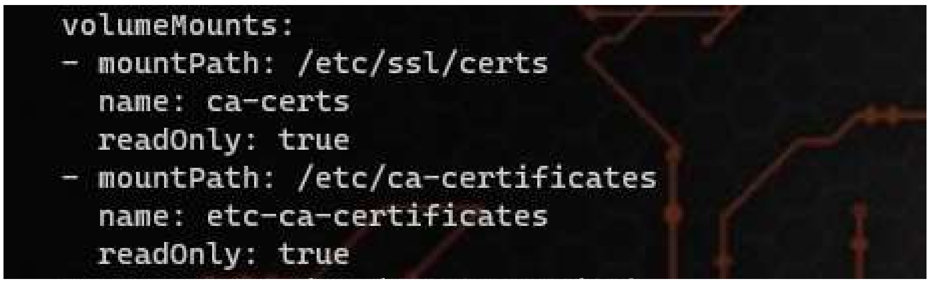


```
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction,ImagePolicyWebhook
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --insecure-port=0
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
```

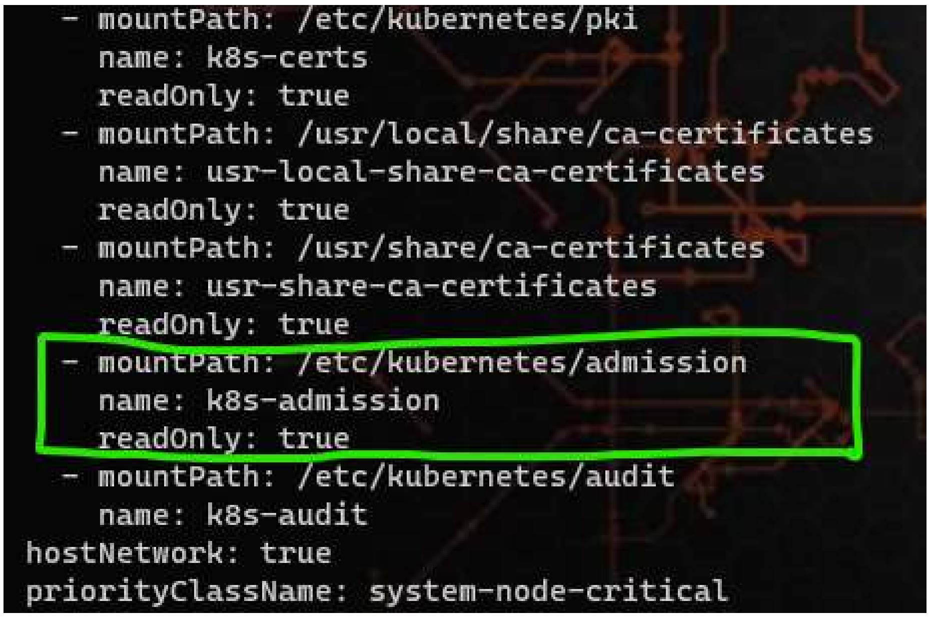
Configure host path to ensure Image Policy Webhook could actually get to the directory holding admission configuration files.



```
  volumes:
  - hostPath:
      path: /etc/ssl/certs
      type: DirectoryOrCreate
    name: ca-certs
  - hostPath:
      path: /etc/ca-certificates
      type: DirectoryOrCreate
    name: etc-ca-certificates
  - hostPath:
      path: /etc/kubernetes/pki
      type: DirectoryOrCreate
    name: k8s-certs
  - hostPath:
      path: /usr/local/share/ca-certificates
      type: DirectoryOrCreate
    name: usr-local-share-ca-certificates
  - hostPath:
      path: /usr/share/ca-certificates
      type: DirectoryOrCreate
    name: usr-share-ca-certificates
  - hostPath:
      path: /etc/kubernetes/admission
      type: DirectoryOrCreate
    name: k8s-admission
  - hostPath:
      path: /etc/kubernetes/audit
      type: DirectoryOrCreate
    name: k8s-audit
status: {}
```

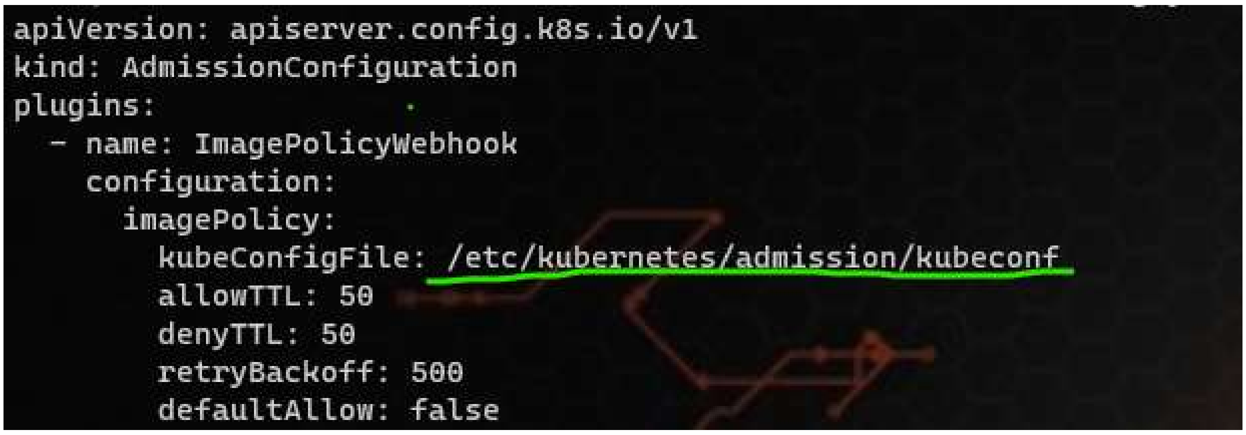


```
    volumeMounts:
    - mountPath: /etc/ssl/certs
      name: ca-certs
      readOnly: true
    - mountPath: /etc/ca-certificates
      name: etc-ca-certificates
      readOnly: true
```



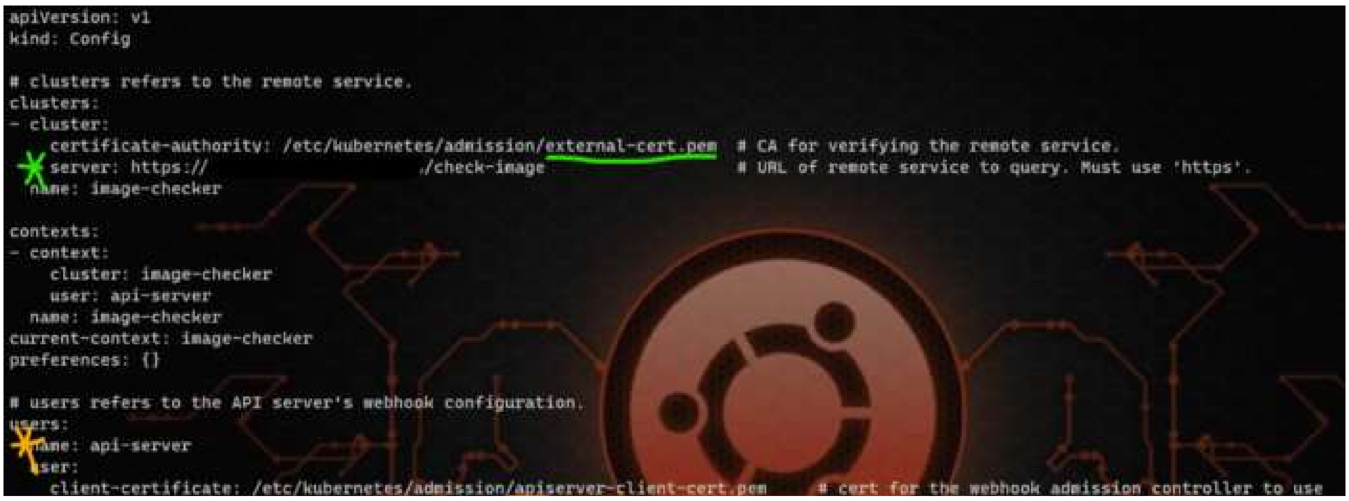
```
  - mountPath: /etc/kubernetes/pki
    name: k8s-certs
    readOnly: true
  - mountPath: /usr/local/share/ca-certificates
    name: usr-local-share-ca-certificates
    readOnly: true
  - mountPath: /usr/share/ca-certificates
    name: usr-share-ca-certificates
    readOnly: true
  - mountPath: /etc/kubernetes/admission
    name: k8s-admission
    readOnly: true
  - mountPath: /etc/kubernetes/audit
    name: k8s-audit
hostNetwork: true
priorityClassName: system-node-critical
```

After setting kube-apiserver to point to the right path for setting up admission configuration, please head over to that specific folder and file. The information within the file should show how Image Policy would be operating on this cluster.



```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
  - name: ImagePolicyWebhook
    configuration:
      imagePolicy:
        kubeConfigFile: /etc/kubernetes/admission/kubeconf
        allowTTL: 50
        denyTTL: 50
        retryBackoff: 500
        defaultAllow: false
```

We notice there is K8s configuration setup, which is for admission controller to have corresponding credentials to communicate with external image-checking server and kube-apiserver.



```
apiVersion: v1
kind: Config

# clusters refers to the remote service.
clusters:
- cluster:
    certificate-authority: /etc/kubernetes/admission/external-cert.pem  # CA for verifying the remote service.
    server: https://[redacted]/check-image                              # URL of remote service to query. Must use 'https'.
  name: image-checker

contexts:
- context:
    cluster: image-checker
    user: api-server
  name: image-checker
current-context: image-checker
preferences: {}

# users refers to the API server's webhook configuration.
users:
- name: api-server
  user:
    client-certificate: /etc/kubernetes/admission/apiserver-client-cert.pem     # cert for the webhook admission controller to use
```

```
  client-key:  /etc/kubernetes/admission/apiserver-client-key.pem          # key matching the cert
```

After setup everything correctly, if we try to deploy an image without security-compliant container image, we should be error messages similar to below

```
Error from server (Forbidden): pods "test" is forbidden: Post
"https://xxxx:xxxx/xxxx?timeout=30s": dial tcp:....
```

For more details on how to setup and what to setup in Image Policy Webhook, please check this section of the official documentation.

That briefly covers the how to avoid having vulnerable container image in K8s cluster. One thing I did not include in this article is using Open Policy Agent (OPA) for only using whitelisted container images for deploying applications/services, because I think if you have followed this series, you already have a general idea on how to setup that in OPA! Happy learning!

Kubernetes Security Preparation Container Images Vulnerability