

[Open in app](#)

Jonathan

[Follow](#)

29 Followers About

Certified Kubernetes Security Specialist (CKS) Preparation Part 3— Cluster Setup



Jonathan Feb 17 · 9 min read

If you have not yet checked the previous parts of this series, please go ahead and check [Part1](#) and [Part2](#).

In this article, I would focus on the preparation around **cluster setup** in CKS certification exam. Some focus would be on CIS Kubernetes Benchmark, kube-bench and verify platform binaries as these 3 are new to me.

Network Policy

Let's first look at Kubernetes network policy. The concept is pretty similar to firewall, just that it could manage pod-to-pod, pod-to-service and external-to-service communication. Administrators identify what resources could access and could be accessed by Pod labels, namespaces or IP addresses. Pretty straightforward. Take the default example from official documentation, it basically means

- Pods with label “role: db” could be accessed with TCP, and port 6379
- Pods and Services with label “role: frontend”, in namespace “project: myproject” and reside in 172.17.0.0/16 would be able to access Pods with label “role: db”.
- Resources reside in 172.17.1.0/24 would not be able to access Pods with label “role: db”.
- The same concept applies to egress rules.

[Open in app](#)

```

metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
      - ipBlock:
        cidr: 172.17.0.0/16
        except:
          - 172.17.1.0/24
      - namespaceSelector:
        matchLabels:
          project: myproject
      - podSelector:
        matchLabels:
          role: frontend
  ports:
    - protocol: TCP
      port: 6379
  egress:
    - to:
      - ipBlock:
        cidr: 10.0.0.0/24
  ports:
    - protocol: TCP
      port: 5978

```

Reference: [Network Policies | Kubernetes](#)

With proper network security rules, administrators could ensure only the allowed resources are accessing the resources that made accessible with the accessible protocols and ports. Pay extra attention on default deny all and allow all rules.

Deny All Ingress

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy

```


[Open in app](#)

```
spec:
  podSelector: {}
  policyTypes:
    - Ingress
```

Allow All Ingress

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-ingress
spec:
  podSelector: {}
  ingress:
    - {}
  policyTypes:
    - Ingress
```

Let's try to see this in practice. We do

- Allow ingress from Pod in namespace label “cks: allow” to Pod with label “cks: denymost”
- Allow ingress from Pod label “cks: allow” to Pod with label “cks: denymost”

Network Policy

```
jonw@CKS-Master:~$ cat networkpolicy-allowaccessors.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-accessors
  namespace: default
spec:
  podSelector:
    matchLabels:
      cks: denymost
```


[Open in app](#)

```
- from:
  - namespaceSelector:
      matchLabels:
        cks: allow
  - podSelector:
      matchLabels:
        cks: allow
```

Test accessing to Pod with label “cks: denymost” from Pod with label “cks: denymost”

```
jonw@CKS-Master:~$ kubectl get pods denymost -o wide --show-labels
NAME    READY   STATUS    RESTARTS   AGE   IP           NODE   NOMINATED-NODE   READINESS   GATES   LABELS
denymost 1/1    Running   0          19m   10.44.0.2   cks-worker   <none>   <none>     <none>   cks=denymost
jonw@CKS-Master:~$ kubectl get pods -A -o wide --show-labels | grep allow
allowingress _accessor2_           1/1    Running   0          28m   10.44.0.1   cks-worker   <none>   <none>   run=accessor2
default _accessor1_                1/1    Running   0          29m   10.44.0.4   cks-worker   <none>   <none>
jonw@CKS-Master:~$ kubectl get ns --show-labels
NAME    STATUS   AGE   LABELS
allowingress   Active   34m   cks=allow
default        Active   41h   <none>
kube-node-lease  Active   41h   <none>
kube-public    Active   41h   <none>
kube-system    Active   41h   <none>
```

```
jonw@CKS-Master:~$ kubectl exec _accessor1_ -- curl 10.44.0.2
% Total    % Received % Xferd  Average Speed   Time   Time  Current
          Dload  Upload Total   Spent    Left  Speed
100  612  100  612    0     0   597k      0 --:--:-- --:--:-- --:--:--  597k
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

```
jonw@CKS-Master:~$ kubectl exec _accessor2_ -n allowingress -- curl 10.44.0.2
% Total    % Received % Xferd  Average Speed   Time   Time  Current
          Dload  Upload Total   Spent    Left  Speed
```


[Open in app](#)

```

<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

```

What about trying to access Pod with label “cks: denymost” from elsewhere? Denied.

```

jonw@CKS-Master:~$ kubectl get pods accessor3 -o wide --show-labels
NAME      READY   STATUS    RESTARTS   AGE     IP           NODE      NOMINATED-NODE   READINESS-GATES   LABELS
accessor3  1/1     Running   0          17m    10.44.8.3   cks-worker   <none>   <none>   run=accessor3
jonw@CKS-Master:~$ kubectl exec accessor3 -- curl 10.44.8.2
% Total % Received % Xferd Average Speed Time Time Current
          Dload Upload Total Spent Left Speed
 0       0       0      0  --:--:--  0:00:28 --:--:--  0" C

```

Kubernetes Dashboard

Although most administrators would prefer to use Kubernetes CLI commands to administrate all resources within K8s clusters, some still prefer to have visual display. Kubernetes Dashboard is created for this reason.

By executing the command below (more details in [this website](#)), people easily setup the all Kubernetes Dashboard required resource under the namespace of “kubernetes-dashboard”.

- `kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.1.0/aio/deploy/recommended.yaml`

```

jonw@CKS-Master:~$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.1.0/aio/deploy/recommended.yaml

```



[Open in app](#)

```
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
```

We could see all the resources being deployed in the one liner.

- `kubectl -n kubernetes-dashboard get pod,deploy,svc`

```
jonw@CH5-Master:~$ kubectl -n kubernetes-dashboard get pod,deploy,svc
I0203 01:29:18.753138    2900 request.go:655] Throttling request took 1.158446537s, request: GET:https://192.168.1.4:6443/apis/batch/v1?timeout=32s
I0203 01:29:28.778289    2900 request.go:655] Throttling request took 5.182831619s, request: GET:https://192.168.1.4:6443/apis/networking.k8s.io/v1beta1?timeout=32s
NAME                                         READY   STATUS    RESTARTS   AGE
pod/dashboard-metrics-scraper-79c5968bdc-2rk6z  1/1     Running   0          75s
pod/kubernetes-dashboard-7448ffc97b-xfj4d        1/1     Running   0          75s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/dashboard-metrics-scraper       1/1     1           1           75s
deployment.apps/kubernetes-dashboard            1/1     1           1           75s

NAME                                TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/dashboard-metrics-scraper   ClusterIP  10.97.235.319 <none>        8080/TCP   75s
service/kubernetes                  ClusterIP  10.99.237.46  <none>        443/TCP   75s
```

As most of you notice, default Kubernetes Dashboard service is exposed as Cluster IP and it would not be possible for administrators to access this IP address without getting inside a shell inside a Pod. For most cases, administrators use “*kubectl proxy*” to proxy an endpoint within the working machine to the actual Kubernetes Dashboard service. For more information on how to proxy the service, check [here](#).

In some testing environments in less security concern, we could make Kubernetes Dashboard deployments and services to be exposed with Node Port, so administrators could use nodes’ IP address, public or private, and assigned port to access the service. We edit the actual running deployment YAML.

- `kubectl edit deployment kubernetes-dashboard -n kubernetes-dashboard`

```
spec:
  containers:
    - args:
        - --namespace=kubernetes-dashboard
        - --insecure-port=9090
      image: kubernetesui/dashboard:v2.1.0
      imagePullPolicy: Always
      livenessProbe:
        failureThreshold: 3
        httpGet:
          path: /healthz
          port: 9090
        initialDelaySeconds: 5
        periodSeconds: 10
        successThreshold: 1
        timeoutSeconds: 1
```

[Open in app](#)

```
scheme: HTTP
initialDelaySeconds: 30
periodSeconds: 10
successThreshold: 3
timeoutSeconds: 30
```

- Remove the argument of auto generate certificates
- Add the argument of insecure port 9090
- Change livenessProbe to listen on port 9090
- Change scheme to listen on HTTP

After that, we make changes on Kubernetes Dashboard services.

- *kubectl edit service kubernetes-dashboard -n kubernetes-dashboard*

```
spec:
  containers:
    - args:
        - --namespace=kubernetes-dashboard
        - --insecure-port=9090
      image: kubernetesui/dashboard:v2.1.0
      imagePullPolicy: Always
      livenessProbe:
        failureThreshold: 3
        httpGet:
          path: /
          port: 9090
          scheme: HTTP
        initialDelaySeconds: 30
        periodSeconds: 10
        successThreshold: 3
        timeoutSeconds: 30
```

- Change port to 9090
- Change targetPort to 9090
- Change type to NodePort

Right now, try to access Kubernetes Dashboard with nodes' IP address with assigned port.


[Open in app](#)

The screenshot shows the Kubernetes Dashboard interface. On the left, there is a sidebar with a "Workloads" heading and a list of resources: CronJobs, DaemonSets, Deployments, Jobs, Pods, and ReplicaSets. The main content area has a message: "There is nothing to display here" and a note: "Please deploy a non-contaminated app, select other namespace or take the Dashboard Tour [?] to learn more".

Since Kubernetes Dashboard is leveraging service account “default” in namespace “kubernetes-dashboard” for accessing each resource, binding the right permission to this service account would allow the dashboard to show more information in the corresponding namespaces.

```
jonw@CKS-Master:~$ kubectl get sa -n kubernetes-dashboard
NAME           SECRETS   AGE
default        1          100m
kubernetes-dashboard 1          100m
```

Secure Ingress

As most of the readers here already have some practical experience on K8s, one of the most popular ingress being used nowadays in K8s is NGINX Ingress, so we would use it as the example. First thing first, we could install NGINX Ingress with one liner.

- *kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.43.0/deploy/static/provider/baremetal/deploy.yaml>*

The detailed instruction is [here](#). Let’s take a look what have been created. One thing to note down is that NGINX Ingress is now

- Being exposed as Node Port service
- Assigned **high port 30597 over HTTP**
- Assigned **high port 32529 over HTTPS.**

The screenshot shows the Kubernetes Dashboard with the "Ingress" tab selected. It displays three resources: a Pod named "pod/ingress-nginx-admission-create-42486", another Pod named "pod/ingress-nginx-admission-patch-btmcg", and a Deployment named "pod/ingress-nginx-controller-80df779996-mp7w7". Below these, a Service named "service/ingress-nginx-controller" is shown. The dashboard provides details for each resource, including their status, restarts, and creation time. The deployment and service are both in a "Running" state with 1/1 pods ready. The service has a "NodePort" type with port 30597/TCP and port 32529/TCP.


[Open in app](#)


After installation, our goal would be to expose a couple of services to test ingress functionalities. As you see below, 2 Pods (service1, service2) has been created and both being exposed as ClusterIP service (service1, service2).



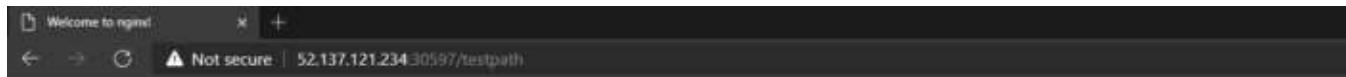
With that, we could head over to official documentation and check [the section about how to set up Ingress rules](#). Basically, what it means in the template is that Ingress would be directing traffic to a service named “test” with the path of “http://<node IP>:<assigned node port>/testpath”. So, let’s give it a try.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

**Note: since we expose service with names as “service1” and “service2” earlier, we would need to make sure the Ingress YAML is actually pointing to service with those

[Open in app](#)

We see we could access “`http://<node IP>:<assigned node port>/testpath`” and view the default NGINX content.



Welcome to nginx!

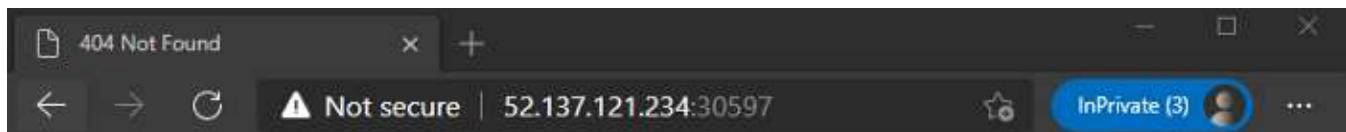
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

We could make Ingress to point to 2 different services with different paths. Since we are making Ingress to point to more than 1 service hosted inside K8s cluster, it makes sense to use path name to correctly show which service we are trying to reach.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /service1
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 80
      - path: /service2
        pathType: Prefix
        backend:
          service:
            name: service2
            port:
              number: 80
```




[Open in app](#)

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

Now we know the service works with HTTP. We have reached the point of making the website to be only accessible over HTTPS, a secure way of accessing the content. Of course, we would first need a certificate to bind the website.

Create self-signed certificate and key with one liner.

- `openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes`

```
jon@jeks-Master:~$ openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes
Can't load /home/jonw/.rnd into RNG
140683424530880:error:2406F079:random number generator:RAND_load_file:cannot open file:../crypto/rand/randfile.c:88:Filename=/home/jonw/.rnd
Generating a RSA private key
-----+
writing new private key to 'key.pem'

You are about to be asked to enter information that will be incorporated
into your certificate request:
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank.
For some fields there will be a default value.
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]: 
State or Province Name (full name) [Some-State]: 
Locality Name (eg, city) []: 
Organization Name (eg, company) [Internet Widgets Pty Ltd]: 
Organizational Unit Name (eg, section) []: 
Common Name (e.g. server FQDN or YOUR name) []:secure-ingress.com
Email Address []:
```


[Open in app](#)

- `kubectl create secret tls secure-ingress — cert=cert.pem — key=key.pem`

```
jonw@CKS-Master:~$ kubectl create secret tls secure-ingress --cert=cert.pem --key=key.pem
secret/secure-ingress created
jonw@CKS-Master:~$ kubectl get secrets
NAME          TYPE           DATA   AGE
default-token-b5fkr  kubernetes.io/service-account-token  3       17h
secure-ingress  kubernetes.io/tls            2       8s
```

Bind the TLS secret to the existing Ingress. For detailed template, please refer to [this site](#).

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  tls:
  - hosts:
    - secure-ingress.com
    secretName: secure-ingress
  rules:
  - host: secure-ingress.com
    http:
      paths:
      - path: /service1
        pathType: Prefix
```

From the screenshot above, we learn Ingress service is listening on 32529 for HTTPS . On Linux environment, use commands like below to visit websites using self-signed certificate

- `curl https://secure-ingress.com:32529 — resolve secure-ingress.com:32529:52.137.121.234 -k`

Resolved public IP is needed since the FQDN resolution would not work in this situation and “-k” is needed for insecure access as it is leveraging self-signed certificate.

```
jonw@JONW-OptiPlex-5090:~$ curl https://secure-ingress.com:32529/service1 --resolve secure-ingress.com:32529:52.137.121.234 -k
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
body {
```


[Open in app](#)

```
<html>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">http://nginx.org/</a> .<br/>
Commercial support is available at
<a href="http://nginx.com/">http://nginx.com/</a> .</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Add “-v” at the end of the curl command to get verbose of the whole process of clients accessing HTTPS website and we would see the access is using self-signed certificate for certain.

- `curl https://secure-ingress.com:32529 --resolve secure-ingress.com:32529:52.137.121.234 -kv`

```
jono@JONW5L3:/mnt/c/Users/jono$ curl https://secure-ingress.com:32529/service2 --resolve secure-ingress.com:32529:52.137.121.234 -kv
* Added secure-ingress.com:32529:52.137.121.234 to DNS cache
* Hostname secure-ingress.com was found in DNS cache
* Trying 52.137.121.234:32529 ...
* TCP_NODELAY set
* Connected to secure-ingress.com (52.137.121.234) port 32529 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/certs/ca-certificates.crt
  CApth: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server accepted to use h2
* Server certificate:
*   subject: C=AU; ST=Some-State; O=Internet Widgits Pty Ltd; CN=secure-ingress.com
*   start date: Feb 3 17:14:01 2021 GMT
*   expire date: Feb 3 17:14:01 2022 GMT
*   issuer: C=AU; ST=Some-State; O=Internet Widgits Pty Ltd; CN=secure-ingress.com
*   SSL certificate verify result: self signed certificate (18), continuing anyway.
```

Node Metadata Protection

Every virtual machines (nodes) hosted in the cloud would need to access node metadata endpoint for various reasons. However, allowing the access to every resource is not ideal. To enhance the security of this access is to apply network policies, so only designated resources within K8s could have the ability of contacting node metadata endpoints. As this is more or less the same concept as section “**Network Policy**”, we would just quick go to the next section.

CIS Benchmarks



[Open in app](#)

target is to see whether administrators could make running K8s cluster to reach that goal. With that in mind, we could head to [this site](#) and download the CIS Benchmarks PDF for K8s.

The PDF file is essentially a reference to let you know what needs to be modified within K8s clusters, but how would we know what needs to be changed? That is where kube-bench come into play. From [this site](#), run

- `docker run --pid=host -v /etc:/etc:ro -v /var:/var:ro -t aquasec/kube-bench:latest [master|node] — version 1.13`

You might need to replace the content of “`master|node`” and “`running K8s version`”:

Then, you would get a result similar to below.

```
jonathan@CKS-Master:~$ docker run --pid=host -v /etc:/etc:ro -v /var:/var:ro -t aquasec/kube-bench:latest master --version 1.28
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 Master Node Configuration Files
[PASS] 1.1.1 Ensure that the API server pod specification file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.2 Ensure that the API server pod specification file ownership is set to root:root (Automated)
[PASS] 1.1.3 Ensure that the controller manager pod specification file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.4 Ensure that the controller manager pod specification file ownership is set to root:root (Automated)
[PASS] 1.1.5 Ensure that the scheduler pod specification file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.6 Ensure that the scheduler pod specification file ownership is set to root:root (Automated)
[PASS] 1.1.7 Ensure that the etcd pod specification file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.8 Ensure that the etcd pod specification file ownership is set to root:root (Automated)
[WARN] 1.1.9 Ensure that the Container Network Interface file permissions are set to 644 or more restrictive (Manual)
[WARN] 1.1.10 Ensure that the Container Network Interface file ownership is set to root:root (Manual)
[PASS] 1.1.11 Ensure that the etcd data directory permissions are set to 700 or more restrictive (Automated)
[PASS] 1.1.12 Ensure that the etcd data directory ownership is set to etcd:etcd (Automated)
[PASS] 1.1.13 Ensure that the admin.conf file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.14 Ensure that the admin.conf file ownership is set to root:root (Automated)
[PASS] 1.1.15 Ensure that the scheduler.conf file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.16 Ensure that the scheduler.conf file ownership is set to root:root (Automated)
[PASS] 1.1.17 Ensure that the controller-manager.conf file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.18 Ensure that the controller-manager.conf file ownership is set to root:root (Automated)
[PASS] 1.1.19 Ensure that the Kubernetes PKI directory and file ownership is set to root:root (Automated)
[PASS] 1.1.20 Ensure that the Kubernetes PKI certificate file permissions are set to 644 or more restrictive (Manual)
[PASS] 1.1.21 Ensure that the Kubernetes PKI key file permissions are set to 600 (Manual)
[INFO] 1.2 API Server
[WARN] 1.2.1 Ensure that the --anonymous-auth argument is set to false (Manual)
[PASS] 1.2.2 Ensure that the --basic-auth-file argument is not set (Automated)
[PASS] 1.2.3 Ensure that the --token-auth-file parameter is not set (Automated)
[PASS] 1.2.4 Ensure that the --kubelet-https argument is set to true (Automated)
[PASS] 1.2.5 Ensure that the --kubelet-client-certificate and --kubelet-client-key arguments are set as appropriate (Automated)
[FAIL] 1.2.6 Ensure that the --kubelet-certificate-authority argument is set as appropriate (Automated)
```

We could ignore the passed ones and focus the ones with [FAIL]. Using [FAIL] 1.2.6 as an example, we first head to the CIS Benchmarks PDF for K8s and search for 1.2.6. Then, we see something like below.

1.2.6 Ensure that the `--kubelet-certificate-authority` argument is set as appropriate (Automated)

Profile Applicability:

Cloud Native Computing Foundation


[Open in app](#)

Verify kubelet's certificate before establishing connection.

Rationale:

The connections from the apiserver to the kubelet are used for fetching logs for pods, attaching (through kubectl) to running pods, and using the kubelet's port-forwarding functionality. These connections terminate at the kubelet's HTTPS endpoint. By default, the apiserver does not verify the kubelet's serving certificate, which makes the connection subject to man-in-the-middle attacks, and unsafe to run over untrusted and/or public networks.

Audit:

Run the following command on the master node:

```
ps -ef | grep kube-apiserver
```

Verify that the `--kubelet-certificate-authority` argument exists and is set as appropriate.

Remediation:

Follow the Kubernetes documentation and setup the TLS connection between the apiserver and kubelets. Then, edit the API server pod specification file

`/etc/kubernetes/manifests/kube-apiserver.yaml` on the master node and set the `--kubelet-certificate-authority` parameter to the path to the cert file for the certificate authority.

```
--kubelet-certificate-authority=<ca-string>
```

Impact:

You require TLS to be configured on apiserver as well as kubelets.

The description matches and it also provides us the way to remediate. Let's try the steps in the cluster. First, we show kubelet configuration with

- `sudo cat /etc/kubernetes/kubelet.conf`

```
jmcginnis-Master:/etc/kubernetes$ sudo cat /etc/kubernetes/kubelet.conf
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data: LS0tLS1CRUdtTiBDRVJUSUZ3Q0FURsAtIPEtci33c2A1NDQkZ9f3SUJB2zlcQur8TjnA3Fo21HOxcmQfcRcBZMEFWTVJNdeVWuWUmU
VFERXdcwRSwvKY201bGRHVnpNojRYRFReE1ESxdNakL6TkhNe11Gb1HEVE1tKTUfTjE1JSUXXQfE18T1L7vdBZ1URVRNQkVHOTFV#RQ08eE1LYTNNAVpY5nVawFJzY3pQ0#TSXdeUVlK52
9a5Wh2Y956UUVCQ1FB#GdnRVBBRENDQVfQ2dmRuJBYy9qcLNlVbHxYe9uTfzAaEKhWmBUTPjTNI4d18t1VnWuxkU1mccl883FXdEnmMytKzdtTHJWd6ZPSS9r2LEia2k0u12FNM3
3TXNaMwQ5TzWqL3KTwfrY1RneUFM#e9K9puuTnLbLU3Rng253Zpkjd2ZLJ2mpjJchdUddyRds-cOpfTz1VnJgvR1RuuuFeywNr#2p{jDjJwH3MHBLTAZwOnpSC9S2M1Z#D8sTg9T
WVYXTEpHSMNEckUrK@ltSTdOCl10Ok1cTRhZFdWePgbp23ZmdqAHpV220V1kLdn8LjWwH432CRQH5T2Nk02J0Y#FytJNWW1RCOOV1nnJNemgkZUtackN2H13RDRjRntdveE1dnI2e
VlnyzTzT2g2RvNqzBppMktZNA2Tz1YwVpb22481BndlBpvRst2F67gpiVqkx53ftA#D0TxxzcrzyH1erQF3RUFByU5DTUvbd0RnWURWjBQOVFIIBJBUURBZ0trTUE4R8ExVmRfdB
VCC193UzNQULCQy4dahhWURWjuBpqkJ2RuzCn3A3WfprWzTun1D0j2R0fkwZ=xxJfGRP#UNQTMQ1M4LbUyJMKRFFQkH3VUFBNELCQVFBMFdPfd9LeVfW5GsySE1YQ32Lz3Jq52p
VRT3dU9rJv2R0TQrMEkb#FTQmljJuhGQpWdE1YXFwV3hIOXbjYjVwNkdmhE102nNQhE3cje3elFpRDFc21poYec3aWZQOFNpc1Q3nVJUJtRLhvdpnOcj1zUGJCShpSYnUzeErp
V2x3VUvrL113B#5zeFhiMURinjR6cwoybxcSw1Vxbk4ZnyWV1WxpQjRdzEYOH#uEckH1rbwuv1Qj1emH#2Vkk7kzsdq1Jh59w#Uwhab2lEcuszzVsS6gxXNNPSHFIJXJ1cExyOGRbc
nh55Et8Vhp0vWpVW#0zCudWVTVhd1VNa3px031WUGtmMU15b1YUk24dnNjBTR22m00kQ0S1JaWTB2d3NSVJClump4ewtRdlVICJRDZJ5akpDanhMQU535hdm5Vc#0Ct0Q8h#e0hRj
ZvRzVUcaOtlSetLuV0RCb0DvJUSUZJ0#FURs8tLs8tCg==
  server: https://192.168.1.4:6443
  name: kubernetes
  contexts:
```


[Open in app](#)

```
preferences: {}
users:
- name: system:node:cks-master
  user:
    client-certificate: /var/lib/kubelet/pki/kubelet-client-current.pem
    client-key: /var/lib/kubelet/pki/kubelet-client-current.pem
```

We copied the content in “certificate-authority-data” and come outside of the text editor to decode the base 64 encoding on this content. Also, we save the content to /etc/kubernetes/pki/apiserver-kubelet-ca.crt. The command should be similar to

- *echo <copied base64-encoded content> | base64 -d >/etc/kubernetes/pki/apiserver-kubelet-ca.crt*

```
jonw@CKS-Master:/etc/kubernetes/pki$ cat apiserver-kubelet-ca.crt
-----BEGIN CERTIFICATE-----
MIIC5zCCA...[REDACTED]...DQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAO/j
SKWN1cOnE0ZkLGXCzQ1pNB8wZlWUgYLdRXfriAGqWtCp3+LC7kLrVwFOI/kfQ5k9
SVE4rwMsZZd916pBRdMhQcTgyALoOd5Zni3bnU7Fh6KviZ7vfRv4h4ECtP7rEwkq
EOKV+x/GTnRqDbCk+jcuRIXy70pKU06X3iH/RemYD0lLoSUf1LJJicDrE++ImI7C
We895q4adWVXJFojwfgPhzOWbDRGKRpK6uF7bBGc90cJCbCc1rN3V6TB85b6rMzh
eKfrCY+YwD4c33muQ5vr6yYgc6SWh6ESjgJi1ky4p6MfuaeOofxoPLvPiQTL/azf
nTgqKxSjbCMrYrfr60kCAwEEAaNCMEAwdgYDVR0PAQH/BAQDAgKkMA8GA1UdEwEB
/wQFMAMBAf8wHQYDVR0OBByEFB7p7UatEfSRyCScA8Y0flqb1FDWMA0GCSqGSIB3
DQEBCwUAA4IBAQAOWLk/KyQJHk2HMXCvKgrjkjUE97u0kQVQ94+0I9xQmBicRxF5
Vua5aqpWxH9pcb5ZZGVxKtfCp207r17zQiD1jfZhb7ifP8S0sT7BufRBmFxovzN
9sPbBHrBnu3xDiclwUEk/Yw3NYxXb1DbF4Fqj2mw9ZUqnFgbEHYzWB4v7128ppRg
6TMYEuB9tDzvVI7+7lwRGKI0Qhp0iGqk3/E1Hlq5sOHqHMrupLr8darxyHKATzEW
UEm6qGJU5GwUJkzWSyVPig1FeKMXRFxvsqm4vehNnD4KRZY0vwsRabBRjxykQVUH
4CeRyjJCjxLANwJGLIW08+NCHtx8aF6oG5Tp
-----END CERTIFICATE-----
```

Now, we head over to modify kube-apiserver.yaml in /etc/kubernetes/manifests/kube-apiserver.yaml, so the communication between kube-apiserver and kubelet could leverage the newly added CA information.

```
jonw@CKS-Master:/etc/kubernetes$ sudo cat 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
```



[Open in app](#)

```
- --advertise-address=192.168.1.4
- --allow-privileged=false
- --anonymous-auth=true
- --authorization-mode=Node,RBAC
- --client-ca-file=/etc/kubernetes/pki/ca.crt
- --enable-admission-plugins=NodeRestriction
- --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-certificate-authority=/etc/kubernetes/pki/apiserver-kubelet-ca.crt
- --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
- --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
- --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
- --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
```

After kube-apiserver restarts, check by running

- `kubectl get pods -n kube-system | grep kube-apiserver`

```
jonw@CKS-Master:/etc/kubernetes$ kubectl get pods -n kube-system | grep kube-apiserver
kube-apiserver-cks-master   1/1     Running   0          8m43s
```

We rerun the kube-bench testing to ensure now [FAIL] 1.2.6 turns to [PASS] 1.2.6.

```
jonw@CKS-Master:/etc/kubernetes$ docker run --pid=host -v /etc:/etc:ro -v /var:/var:ro -t aquasec/kube-bench:latest master --version 1.28
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 Master Node Configuration Files
[PASS] 1.1.1 Ensure that the API server pod specification file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.2 Ensure that the API-server pod specification file ownership is set to root:root (Automated)
[PASS] 1.1.3 Ensure that the controller manager pod specification file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.4 Ensure that the controller manager pod specification file ownership is set to root:root (Automated)
[PASS] 1.1.5 Ensure that the scheduler pod specification file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.6 Ensure that the scheduler pod specification file ownership is set to root:root (Automated)
[PASS] 1.1.7 Ensure that the etcd pod specification file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.8 Ensure that the etcd pod specification file ownership is set to root:root (Automated)
[WARN] 1.1.9 Ensure that the Container Network Interface file permissions are set to 644 or more restrictive (Manual)
[WARN] 1.1.10 Ensure that the Container Network Interface file ownership is set to root:root (Manual)
[PASS] 1.1.11 Ensure that the etcd data directory permissions are set to 700 or more restrictive (Automated)
[PASS] 1.1.12 Ensure that the etcd data directory ownership is set to root:root (Automated)
[PASS] 1.1.13 Ensure that the admin.conf file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.14 Ensure that the admin.conf file ownership is set to root:root (Automated)
[PASS] 1.1.15 Ensure that the scheduler.conf file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.16 Ensure that the scheduler.conf file ownership is set to root:root (Automated)
[PASS] 1.1.17 Ensure that the controller-manager.conf file permissions are set to 644 or more restrictive (Automated)
[PASS] 1.1.18 Ensure that the controller-manager.conf file ownership is set to root:root (Automated)
[PASS] 1.1.19 Ensure that the Kubernetes PKI directory and file ownership is set to root:root (Automated)
[PASS] 1.1.20 Ensure that the Kubernetes PKI certificate file permissions are set to 644 or more restrictive (Manual)
[PASS] 1.1.21 Ensure that the Kubernetes PKI key file permissions are set to 600 (Manual)
1.2 API Server
[WARN] 1.2.1 Ensure that the --anonymous-auth argument is set to false (Manual)
[PASS] 1.2.2 Ensure that the --basic-auth-file argument is not set (Automated)
[PASS] 1.2.3 Ensure that the --token-auth-file parameter is not set (Automated)
[PASS] 1.2.4 Ensure that the --kubelet-https argument is set to true (Automated)
[PASS] 1.2.5 Ensure that the --kubelet-client-certificate and --kubelet-client-key arguments are set as appropriate (Automated)
[PASS] 1.2.6 Ensure that the --kubelet-certificate-authority argument is set as appropriate (Automated)
```

Verify Platform Binaries

[Open in app](#)

comparing the downloaded file sha512sum with the string provided in GitHub page.

First, we head to [this site](#) and search for the version we would desire to install in K8s clusters.

The screenshot shows the GitHub repository for kubernetes/kubernetes. The top navigation bar includes 'Search or jump to...', 'Pull requests', 'Issues' (1.9k), 'Marketplace', and 'Explore'. Below the repository name 'kubernetes / kubernetes', there are tabs for 'Code' (selected), 'Issues', 'Pull requests' (975), 'Actions', 'Projects' (6), 'Security', and 'Insights'. A secondary tab bar below shows 'Releases' (selected) and 'Tags'. The main content area displays a list of tags: 'v1.21.0-alpha.2', 'v1.21.0-alpha.1', 'v1.20.3-rc.0', and 'v1.20.2'. Each tag entry includes a timestamp, a commit hash, download links for 'zip' and 'tar.gz', and 'Notes' and 'Downloads' buttons.

For me, I would need to know v1.20.2 details, so I click on *v1.20.2*.

The screenshot shows the Kubernetes v1.20.2 release page. At the top, it says 'Kubernetes v1.20.2' and credits 'k8s-release-robot' for the release. It mentions '1435 commits to master since this release'. Below this, it says 'See [kubernetes-announce@](#). Additional binary downloads are linked in the CHANGELOG.' and 'See the CHANGELOG for more details.' Under the heading 'Release Assets', it lists 'Kubernetes Source Code: kubernetes.tar.gz' with its SHA256 and SHA512 checksums.

	Kubernetes Source Code: kubernetes.tar.gz
SHA256	2c9f49cef1e82a3798673a5a1c0da860e4f0ff268a7d0b4d5b63b32862e6750b
SHA512	cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f63b931bd4

Then, click on *CHANGELOG* and find *Server binaries*.


[Open in app](#)

server-linux-amd64.tar.gz	65abf178782e43bc21e8455fbfdadfc6064dbeae3ff704ccf9e13e8acee18235c280b06778e5de4bd702f5507e1870fe38c561366c
kubernetes-server-linux-arm.tar.gz	4cac058ade494999292fa4ff0f46c2f3f614d8a6f63d3f0991ee942c0d0aacf6af9ae8b091ffae23809d5281fb4ae0d0743a94a96d8
kubernetes-server-linux-arm64.tar.gz	f5ef9bc1013b638cdc17071e60c987572185d9aea796c44ddd2c791770debe1f7225898c8704f3484c4802e452bb25590bf0523c
kubernetes-server-linux-ppc64le.tar.gz	1d9ad46458137e5490832465c6ed997e6d986746e02985c525a0780c9e91857024d7c7fc2fb61dfbabd4613214e3e9abf2c560c
kubernetes-server-linux-s390x.tar.gz	d9c907cd4e75441970409a9801cff97529af8517c85d2a1c5e9ab055c1886b0e0fd62457644c1c52998e84e7c91920ee09040be5

Depends on the processor K8s clusters are running on, different server binaries would have different sha512sum. The next step is to download the binaries into server and compare it to the string shown on the site.

- *echo “<copied sha512sum>” > compare*
- *wget <v1.20.2 server binaries>*
- *sha512sum <downloaded file> >> compare*
- *cat compare | uniq*

```
jone@CKS-Master:~$ echo "f5ef9bc1013b638cdc17071e60c987572185d9aea796c44ddd2c791770debe1f7225898c8704f3484c4802e452bb25590bf05232dffff58da741e7897b6907bfe7b6907bfe" > compare
jone@CKS-Master:~$ wget https://dl.k8s.io/v1.20.2/kubernetes-server-linux-arm64.tar.gz
--2021-02-03 18:39:54-- https://dl.k8s.io/v1.20.2/kubernetes-server-linux-arm64.tar.gz
Resolving dl.k8s.io (dl.k8s.io) ... 34.107.284.286, 26681:1981:0:26f3::.
Connecting to dl.k8s.io (dl.k8s.io)|34.107.284.286|:443 ... connected.
HTTP request sent, awaiting response ... 302 Moved Temporarily
Location: https://storage.googleapis.com/kubernetes-release/release/v1.20.2/kubernetes-server-linux-arm64.tar.gz [following]
--2021-02-03 18:39:54-- https://storage.googleapis.com/kubernetes-release/release/v1.20.2/kubernetes-server-linux-arm64.tar.gz
Resolving storage.googleapis.com (storage.googleapis.com)|74.125.195.120, 74.125.20.120, 74.125.143.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.195.120|:443 ... connected.
HTTP request sent, awaiting response ... 200 OK
Length: 291791433 (278MB) [application/x-tar]
Saving to: ‘kubernetes-server-linux-arm64.tar.gz’

kubernetes-server-linux-arm64.tar.g 100%[=====] 278.27M  114MB/s  in 2.4s
2021-02-03 18:39:57 (114 MB/s) - ‘kubernetes-server-linux-arm64.tar.gz’ saved [291791433/291791433]

jone@CKS-Master:~$ sha512sum kubernetes-server-linux-arm64.tar.gz >> compare
jone@CKS-Master:~$ cat compare
f5ef9bc1013b638cdc17071e60c987572185d9aea796c44ddd2c791770debe1f7225898c8704f3484c4802e452bb25590bf05232dffff58da741e7897b6907bfe
f5ef9bc1013b638cdc17071e60c987572185d9aea796c44ddd2c791770debe1f7225898c8704f3484c4802e452bb25590bf05232dffff58da741e7897b6907bfe kubernetes-server-linux-arm64.tar.gz
```

This article briefly covers the main topics within the CKS certificate exam domain, cluster setup. We will be talking about the next domain, cluster hardening, in the next article.

[Open in app](#)[Ck](#) [Kubernetes](#) [Cluster](#) [Preparation](#) [Exam](#)[About](#) [Help](#) [Legal](#)[Get the Medium app](#)