Jonathan

Follow 29 Followers About

# Certified Kubernetes Security Specialist (CKS) Preparation Part 4— Cluster Hardening

Jonathan Feb 24 · 9 min read

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

In this article, I would focus on the preparation around **cluster hardening** in CKS certification exam.

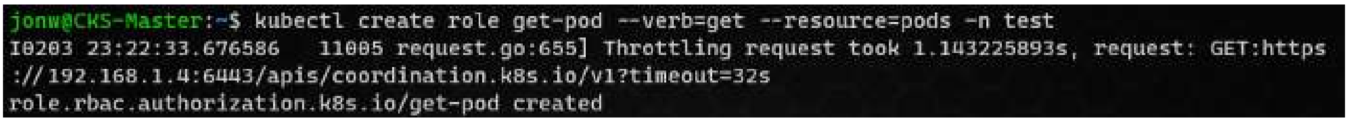
## Role and Role Binding

- Role = the position that could perform actions
- RoleBinding = the binding of user/service account and roles
- Roles are namespace specific.

Let's see an example to get a clearer idea. Our goal is to create a role that could get pods in namespace "test" and bind user "jon" to this role.

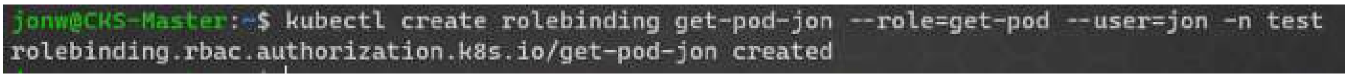
Create a Role called "get-pod" in namespace "test"

- *kubectl create role get-pod — verb=get — resource=pods -n test*



```
jonw@CKS-Master:~$ kubectl create role get-pod --verb=get --resource=pods -n test
I0203 23:22:33.676586   11005 request.go:655] Throttling request took 1.143225893s, request: GET:https
://192.168.1.4:6443/apis/coordination.k8s.io/v1?timeout=32s
role.rbac.authorization.k8s.io/get-pod created
```

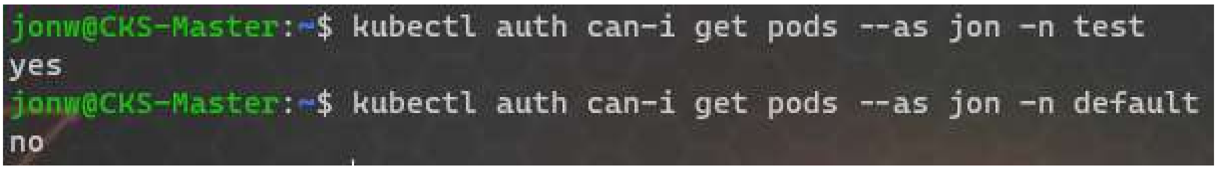
- *kubectl create rolebinding get-pod-jon — role=get-pod — user=jon -n test*



```
jonw@CKS-Master:~$ kubectl create rolebinding get-pod-jon --role=get-pod --user=jon -n test
rolebinding.rbac.authorization.k8s.io/get-pod-jon created
```

Test whether user "jon" could actually get pods in namespace "test"

- *kubectl auth can-i get pods — as jon -n test*
- *kubectl auth can-i get pods — as jon -n default*



```
jonw@CKS-Master:~$ kubectl auth can-i get pods --as jon -n test
yes
jonw@CKS-Master:~$ kubectl auth can-i get pods --as jon -n default
no
```

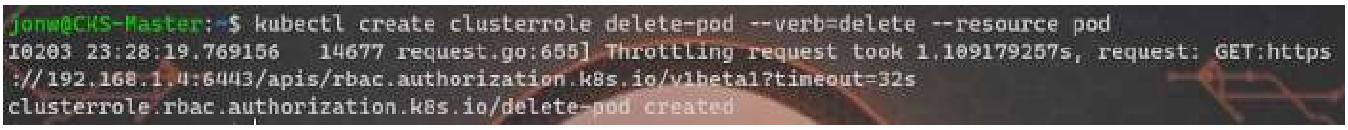
**Cluster Role and Cluster Role Binding**

- ClusterRole = the position that could perform actions across the whole cluster
- ClusterRoleBinding = the binding of user/service account and cluster roles
- ClusterRoles are NOT namespace specific.

Let's see an example to have a clearer idea. Our goal is to create a cluster role that could delete pods in namespace "test2" and bind user "jon" to this cluster role.

Create a ClusterRole called "delete-pod"

- *kubectl create clusterrole delete-pod — verb=delete — resource pod*



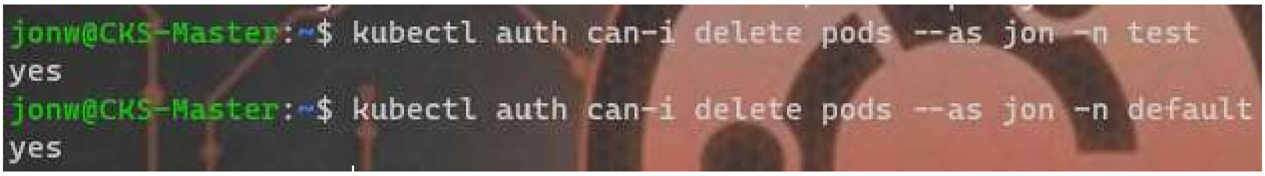
```
jonw@CKS-Master:~$ kubectl create clusterrole delete-pod --verb=delete --resource pod
I0203 23:28:19.769156   14677 request.go:655] Throttling request took 1.109179257s, request: GET:https
://192.168.1.4:6443/apis/rbac.authorization.k8s.io/v1beta1?timeout=32s
clusterrole.rbac.authorization.k8s.io/delete-pod created
```

Create a ClusterRoleBinding called "delete-pod-jon" that connects the ClusterRole "delete-pod" with user "jon"

- *kubectl create clusterrolebinding delete-pod-jon — clusterrole delete-pod — user jon*

Test whether user “jon” could actually delete pods in all namespaces.

- *kubectl auth can-i delete pods — as jon -n test*
- *kubectl auth can-i delete pods — as jon -n default*



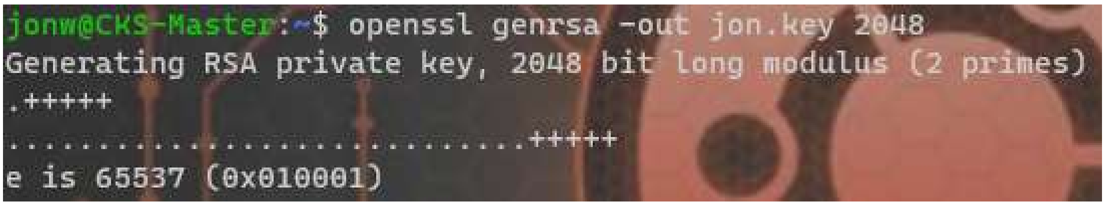
```
jonw@CKS-Master:~$ kubectl auth can-i delete pods --as jon -n test
yes
jonw@CKS-Master:~$ kubectl auth can-i delete pods --as jon -n default
yes
```

### Certificate Signing Requests

Certificate signing requests (CSR) are essentially users or service accounts asking kube-apiserver to provide access for managing K8s clusters with their own identity. This would ensure that users and service accounts would be communicating with kube-apiserver on their own behalf when having interactions. The basic process flow is

User/service account generates a key

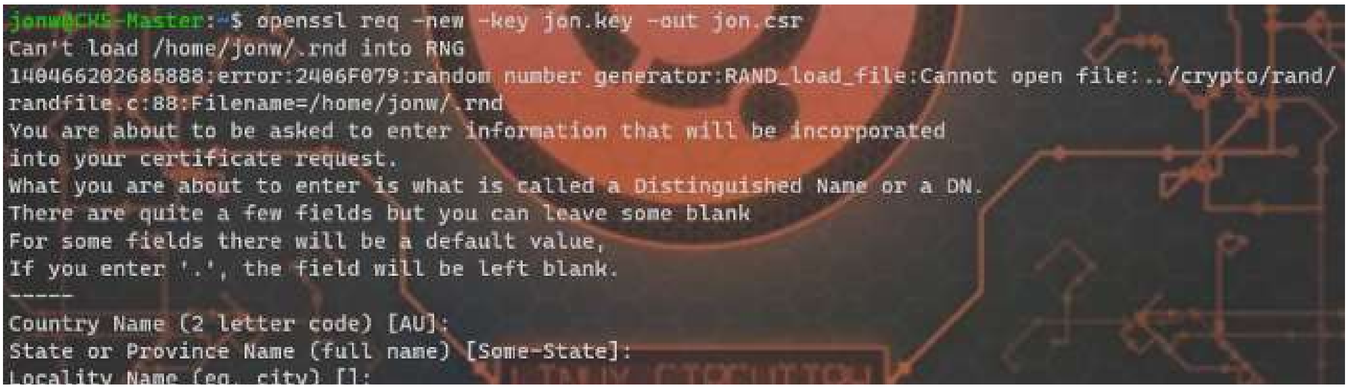
- *openssl genrsa -out jon.key 2048*



```
jonw@CKS-Master:~$ openssl genrsa -out jon.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.+++++
...............................+++++
e is 65537 (0x010001)
```

Use the key to generate a CSR

- *openssl req -new -key jon.key -out jon.csr*

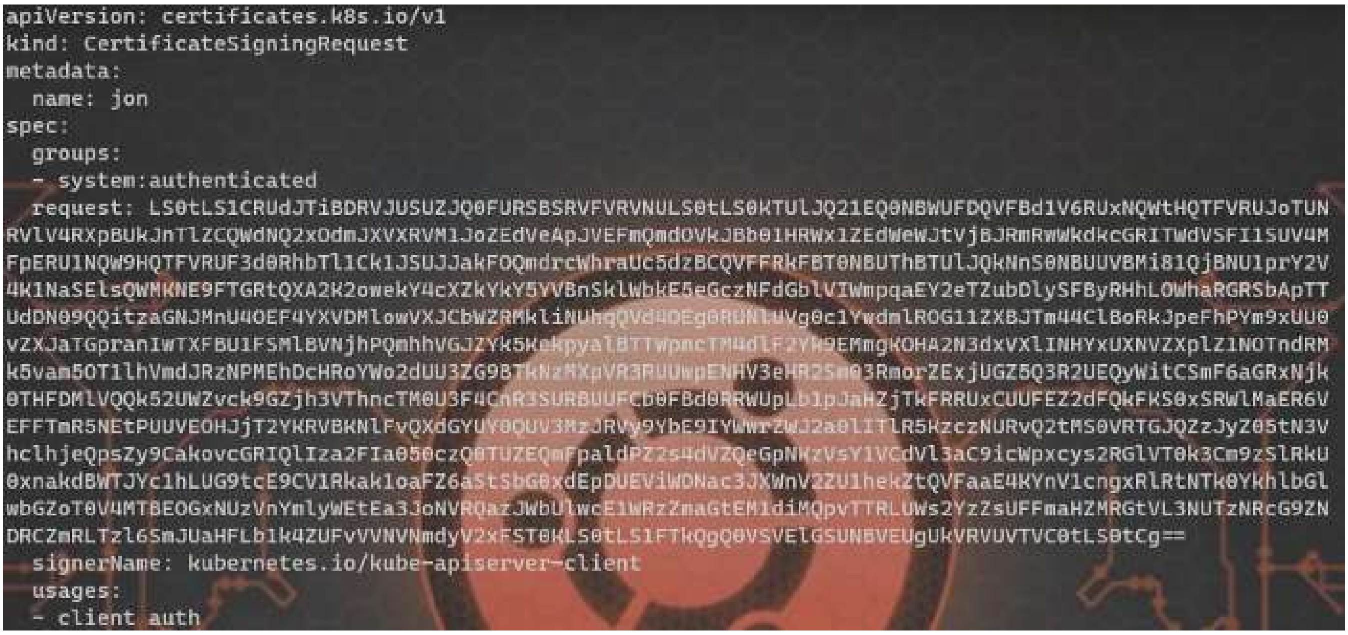


```
jonw@CKS-Master:~$ openssl req -new -key jon.key -out jon.csr
Can't load /home/jonw/.rnd into RNG
140466202685888:error:2406F079:random number generator:RAND_load_file:Cannot open file:../crypto/rand/
randfile.c:88:Filename=/home/jonw/.rnd
You are about to be asked to enter information that will 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) []:
```

```
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Encode the output CSR with base 64 and copy the content to a K8s CSR YAML. Check here for getting the default K8s CSR YAML template.

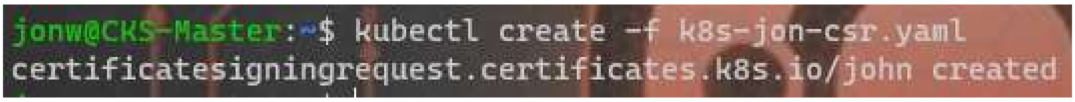
- nano k8s-jonw-csr.yaml
- *cat jon.csr | base64 -w 0*



```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: jon
spec:
  groups:
  - system:authenticated
  request: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0KTUlJQ21EQ0NBWUFDQVFBd1V6RUxNQWtHQTFVRUJoTUN
RVlV4RXpBUkJnTlZCQWdNQ2xOdmJXVXRVM1JoZEdVeApJVEFmQmdOVkJBb01HRWx1ZEdWeWJtVjBJRmRwWkdkcGRITWdVSFI1SUV4M
FpERU1NQW9HQTFVRUF3d0RhbTl1Ck1JSUJJakFOQmdrcWhraUc5dzBCQVFFRkFBT0NBUThBTUlJQkNnS0NBUUVBMi81QjBNU1prY2V
4K1NaSElsQWMKNE9FTGRtQXA2K2owekY4cXZkYkY5YVBnSklWbkE5eGczNFdGblVIWmpqaEY2eTZubDlySFByRHhLOWhaRGRSbApTT
UdDN09QQitzaGNJMnU4OEF4YXVDMlowVXJCbWZRMkliNUhqQVd4OEg0RUNlUVg0c1YwdmlROG11ZXBJTm44ClBoRkJpeFhPYm9xUU0
vZXJaTGpranIwTXFBU1FSMlBVNjhPQmhhVGJZYk5KekpyalBTTwpmcTM4dlF2YW9EMmgKOHA2N3dxVXlINHYxUXNVZXplZ1NOTndRM
k5vam5OT1lhVmdJRzNPMEhDcHRoYWo2dUU3ZG9BTkNzMXpVR3RUUWpENHV3eHR2Sm03RmorZExjUGZ5Q3R2UEQyWitCSmF6aGRxNjk
0THFDMlVQQk52UWZvck9GZjh3VThncTM0U3F4Q2NR3SURBUUFCb0FBd0RRWUpLb1pJaHZjTkFRRUxCUUFFZ2dFQkFKS0xSRWlMaER6V
EFFTmR5NEtPUUVEOHJjT2YKRVBKNlFvQXdGYUY0QUV3MzJRVy9YbE9IYWwrZWJ2a0lITlR5KzczNURvQ2tMS0VRTGJQZzJyZ05tN3V
hclhjeQpsZy9CakovcGRIQlIza2FIa050czQ0TUZEQmFpaldPZ2s4dVZQeGpNWzVsY1VCdVl3aC9icWpxcys2RGlVT0k3Cm9zSlRkU
0xnakdBWTJYc1hLUG9tcE9CV1Rkak1oaFZ6aStSbG0xdEpDUEViWDNac3JXWnV2ZU1hekZtQVFaaE4KYnV1cngxRlRtNTk0YkhlbGl
wbGZoT0V4MTBEOGxNUzVnYmlyWEtEa3JoNVRQazJWbUlwcE1WRzZmaGtEM1diMQpvTTRLUWs2YzZsUFFmaHZMRGtVL3NUTzNRcG9ZN
DRCZmRLTzl6SmJUaHFLb1k4ZUFvVVNVNmdyV2xFST0KLS0tLS1FTkQgQ0VSVElGSUNBVEUgUkVRVUVTVC0tLS0tCg==
  signerName: kubernetes.io/kube-apiserver-client
  usages:
  - client auth
```

Create the K8s CSR

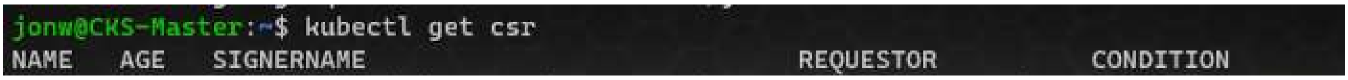
- *kubectl create -f k8s-jon-csr.yaml*



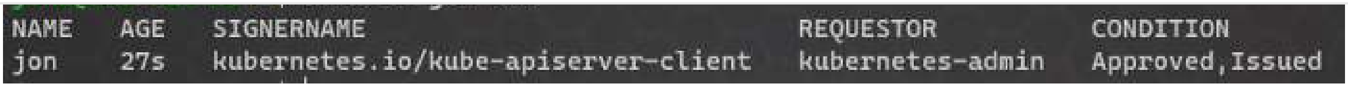
```
jonw@CKS-Master:~$ kubectl create -f k8s-jon-csr.yaml
certificatesigningrequest.certificates.k8s.io/john created
```

Check CSR status and Approve the CSR

- *kubectl get csr*
- *kubectl certificate approve jon*



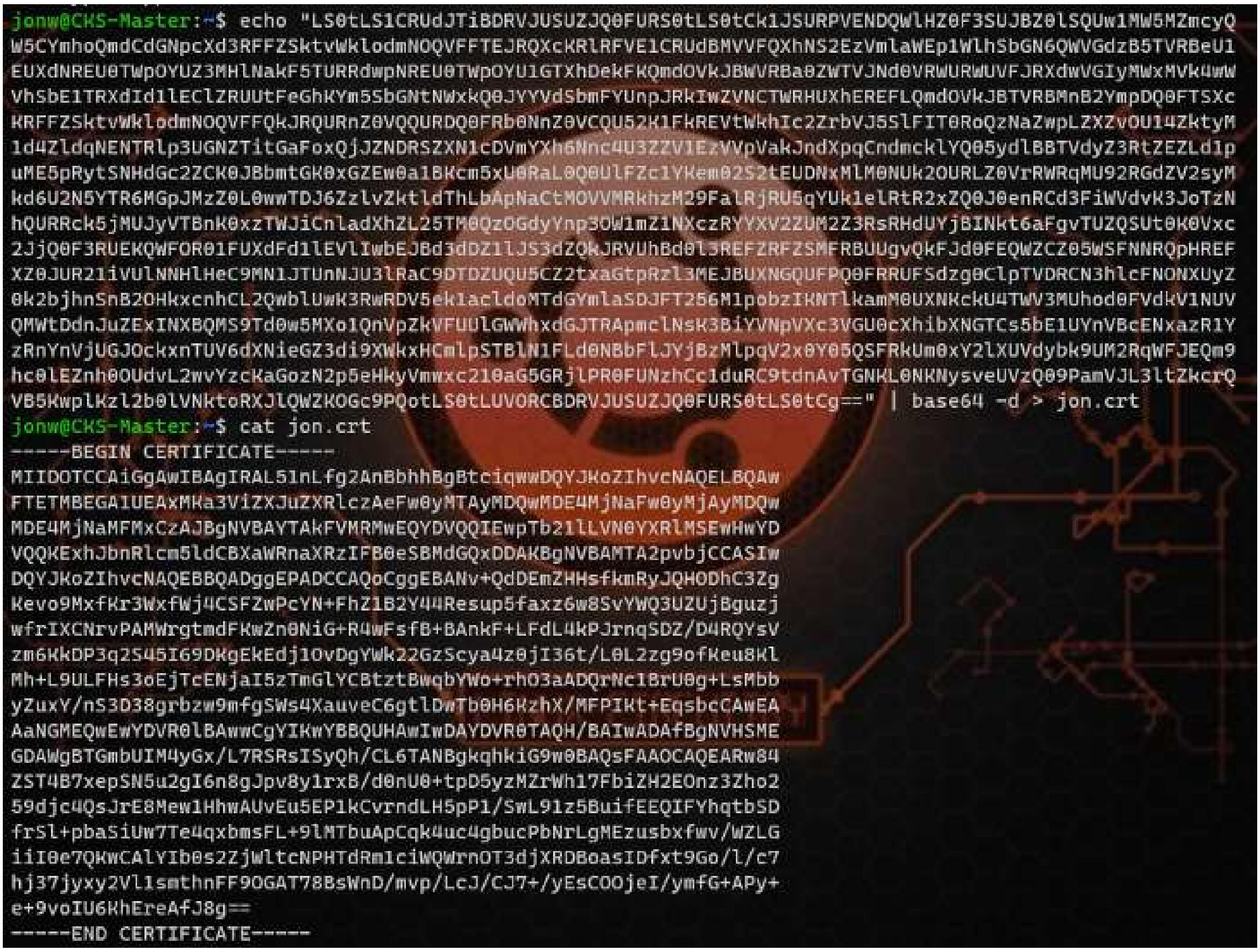
```
jonw@CKS-Master:~$ kubectl get csr
NAME     AGE    SIGNERNAME                              REQUESTOR           CONDITION
```



```
NAME   AGE   SIGNERNAME                            REQUESTOR          CONDITION
jon    27s   kubernetes.io/kube-apiserver-client   kubernetes-admin   Approved,Issued
```

Get client certificate from K8s CSR YAML and Decode it from base 64 and save it to a new file

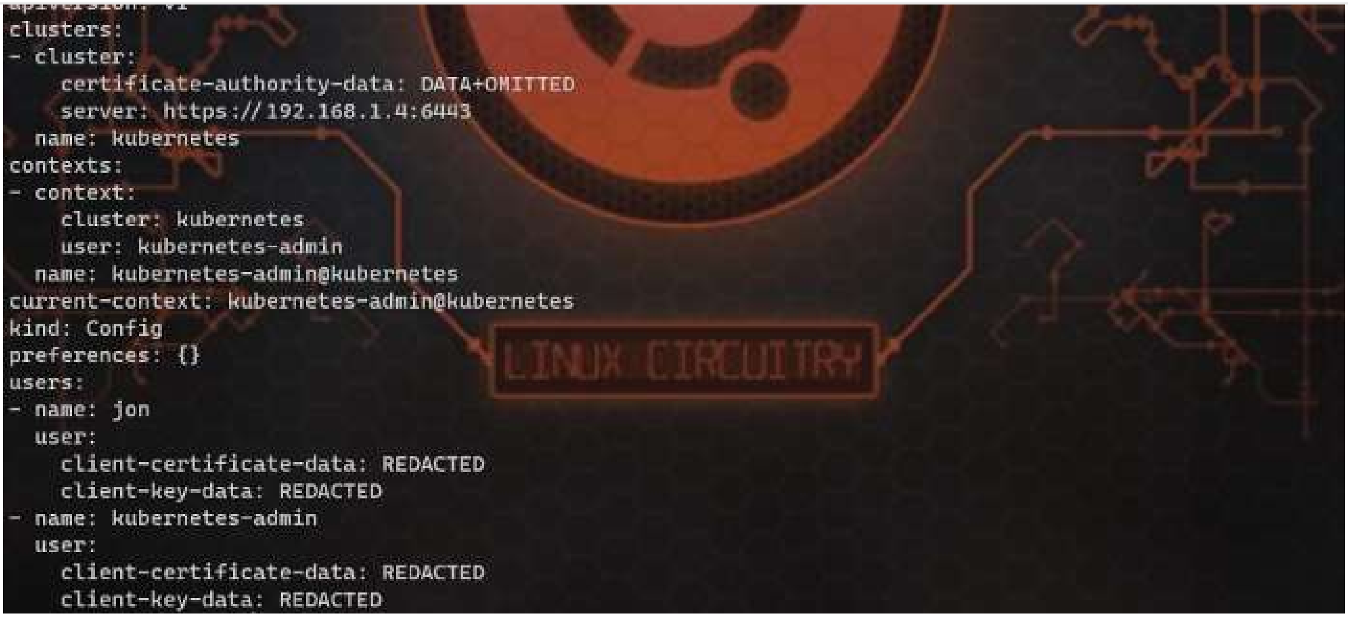
- *kubectl get csr jon -o yaml*
- *echo <certificate content> | base64 -d > jon.crt*
- *cat jon.crt*



```
jonw@CKS-Master:~$ echo "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURPVENDQWlHZ0F3SUJBZ0lSQUw1MW5MZmcyQ
W5CYmhoQmdCdGNpcXd3RFFZSktvWklodmNOQVFFTEJRQXcKRlRFVE1CRUdBMVVFQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB5TVRBeU1
EUXdNREU0TWpOYUZ3MHlNakF5TURRdwpNREU0TWpOYU1GTXhDekFKQmdOVkJBWVRBa0ZWTVJNd0VRWURWUVFJRXdwVGIyMWxMVk4wW
VhSbE1TRXdId1lEClZRUUtFeGhKYm5SbGNtNWxkQ0JYYVdSbmFYUnpJRkIwZVNCTWRHUXhEREFLQmdOVkJBTVRBMnB2YmpDQ0FTSXc
KRFFZSktvWklodmNOQVFFQkJRQURnZ0VQQURDQ0FRb0NnZ0VCQU52K1FkREVtWkhIc2ZrbVJ5SlFIT0RoQzNaZwpLZXZvOU14ZktyM
1d4ZldqNENTRlp3UGNZTitGaFoxQjJZNDRSZXN1cDVmYXh6Nnc4U3ZZV1EzVVpVakJndXpqCndmcklYQ05ydlBBTVdyZ3RtZEZLd1p
uME5pRytSNHdGc2ZCK0JBbmtGK0xGZEw0a1BKcm5xU0RaL0Q0UlFZc1YKem02S2tEUDNxMlM0NUk2OURLZ0VrRWRqMU92RGdZV2syM
kd6U2N5YTR6MGpJMzZ0L0wwTDJ6ZzlvZktldThLbApNaCtMOVVMRkhzM29FalRjRU5qYUk1elRtR2xZQ0J0enRCd3FiWVdvK3JoTzN
hQURRck5jMUJyVTBnK0xzTWJiCnladXhZL25TM0QzOGdyYnp3OW1mZ1NXczRYYXV2ZUM2Z3RsRHdUYjBINkt6aFgvTUZQSUt0K0Vxc
2JjQ0F3RUEKQWFOR01FUXdFd1lEVlIwbEJBd3dDZ1lJS3dZQkJRVUhBd0l3REFZRFZSMFRBUUgvQkFJd0FEQWZCZ05WSFNNRQpHREF
XZ0JUR21iVUlNNHlHeC9MN1JTUnNJU3lRaC9DTDZUQU5CZ2txaGtpRzl3MEJBUXNGQUFPQ0FRRUFSdzg0ClpTVDRCN3hlcFNONXUyZ
0k2bjhnSnB2OHkxcnhCL2QwblUwK3RwRDV5ek1acldoMTdGYmlaSDJFT256M1pobzIKNTlkamM0UXNKckU4TWV3MUhod0FVdkV1NUV
QMWtDdnJuZExINXBQMS9Td0w5MXo1QnVpZkVFUUlGWWhxdGJTRApmclNsK3BiYVNpVXc3VGU0cXhibXNGTCs5bE1UYnVBcENxazR1Y
zRnYnVjUGJOckxnTUV6dXNieGZ3di9XWkxHCmlpSTBlN1FLd0NBbFlJYjBzMlpqV2x0Y05QSFRkUm0xY2lXUVdybk9UM2RqWFJEQm9
hc0lEZnh0OUdvL2wvYzcKaGozN2p5eHkyVmwxc210aG5GRjlPR0FUNzhCc1duRC9tdnAvTGNKL0NKNysveUVzQ09PamVJL3ltZkcrQ
VB5KwplKzl2b0lVNktoRXJlQWZKOGc9PQotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==" | base64 -d > jon.crt
jonw@CKS-Master:~$ cat jon.crt
-----BEGIN CERTIFICATE-----
MIIDOTCCAiGgAwIBAgIRAL51nLfg2AnBbhhBgBtciqwwDQYJKoZIhvcNAQELBQAw
FTETMBEGA1UEAxMKa3ViZXJuZXRlczAeFw0yMTAyMDQwMDE4MjNaFw0yMjAyMDQw
MDE4MjNaMFMxCzAJBgNVBAYTAkFVMRMwEQYDVQQIEwpTb21lLVN0YXRlMSEwHwYD
VQQKExhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQxDDAKBgNVBAMTA2pvbjCCASIw
DQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANv+QdDEmZHHsfkmRyJQHODhC3Zg
Kevo9MxfKr3WxfWj4CSFZwPcYN+FhZ1B2Y44Resup5faxz6w8SvYWQ3UZUjBguzj
wfrIXCNrvPAMWrgtmdFKwZn0NiG+R4wFsfB+BAnkF+LFdL4kPJrnqSDZ/D4RQYsV
zm6KkDP3q2S45I69DKgEkEdj1OvDgYWk22GzScya4z0jI36t/L0L2zg9ofKeu8Kl
Mh+L9ULFHs3oEjTcENjaI5zTmGlYCBtztBwqbYWo+rhO3aADQrNc1BrU0g+LsMbb
yZuxY/nS3D38grbzw9mfgSWs4XauveC6gtlDwTb0H6KzhX/MFPIKt+EqsbcCAwEA
AaNGMEQwEwYDVR0lBAwwCgYIKwYBBQUHAwIwDAYDVR0TAQH/BAIwADAfBgNVHSME
GDAWgBTGmbUIM4yGx/L7RSRsISyQh/CL6TANBgkqhkiG9w0BAQsFAAOCAQEARw84
ZST4B7xepSN5u2gI6n8gJpv8y1rxB/d0nU0+tpD5yzMZrWh17FbiZH2EOnz3Zho2
59djc4QsJrE8Mew1HhwAUvEu5EP1kCvrndLH5pP1/SwL91z5BuifEEQIFYhqtbSD
frSl+pbaSiUw7Te4qxbmsFL+9lMTbuApCqk4uc4gbucPbNrLgMEzusbxfwv/WZLG
iiI0e7QKwCAlYIb0s2ZjWltcNPHTdRm1ciWQWrnOT3djXRDBoasIDfxt9Go/l/c7
hj37jyxy2Vl1smthnFF9OGAT78BsWnD/mvp/LcJ/CJ7+/yEsCOOjeI/ymfG+APy+
e+9voIU6KhEreAfJ8g==
-----END CERTIFICATE-----
```

Set the new credential in kubeconfig for administrators to use

- *kubectl config set-credentials — client-key=jon.key — client-certificate=jon.crt — embed-certs*
- *kubectl config view*



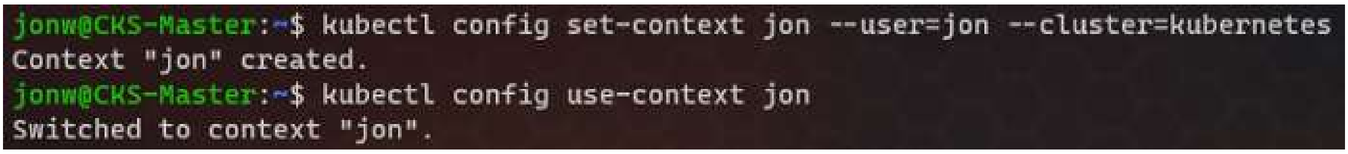
```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.1.4:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: jon
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

Set new context in the cluster

- *kubectl config set-context jon — user=jon — cluster=kubernetes*

Use the context

- *kubectl config use-context jon*



```
jonw@CKS-Master:~$ kubectl config set-context jon --user=jon --cluster=kubernetes
Context "jon" created.
jonw@CKS-Master:~$ kubectl config use-context jon
Switched to context "jon".
```

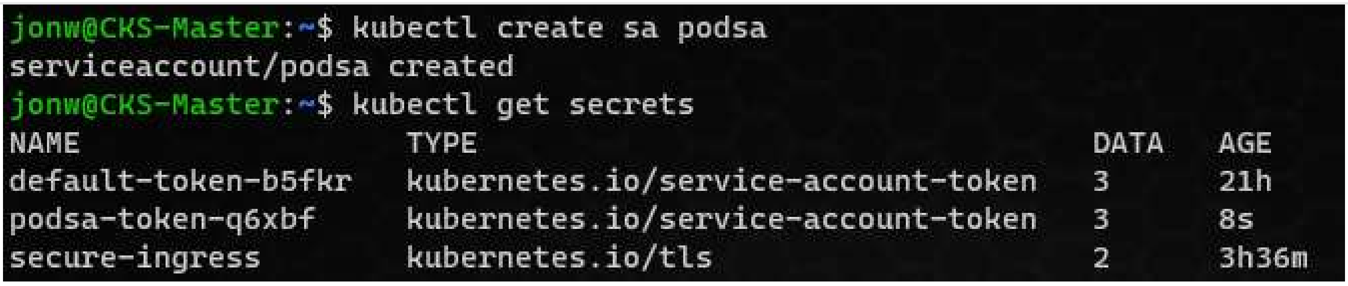
Depending on what permissions have been given to user “jon”, the machine could perform different actions on the credential provided.

### Service Account in Pods

The topic is to ensure administrators are not giving service accounts within Pods to have permissions besides required. For demonstration, we would be creating a service account

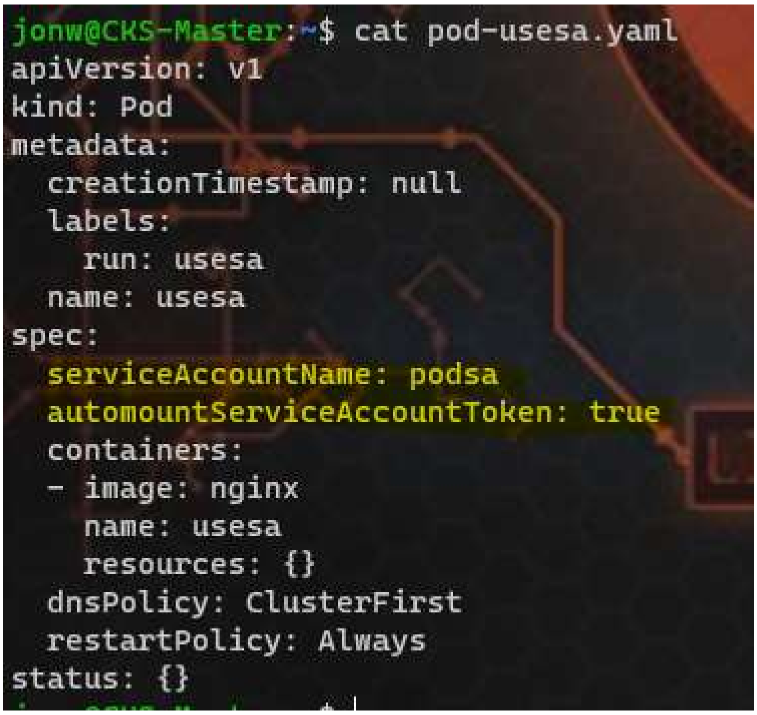
- *kubectl create sa podsa*

and we would see a secret is also auto generated associated with the service account



```
jonw@CKS-Master:~$ kubectl create sa podsa
serviceaccount/podsa created
jonw@CKS-Master:~$ kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-b5fkr   kubernetes.io/service-account-token   3      21h
podsa-token-q6xbf     kubernetes.io/service-account-token   3      8s
secure-ingress        kubernetes.io/tls                     2      3h36m
```

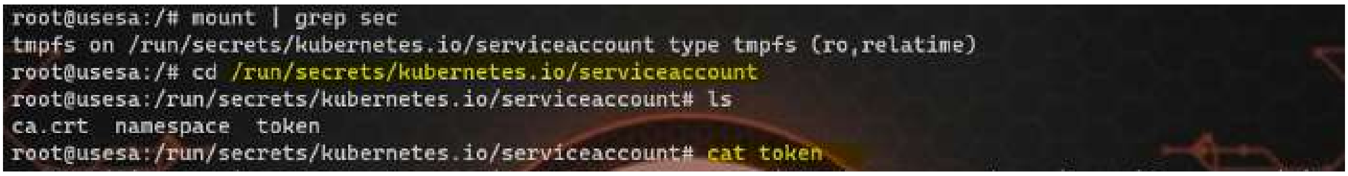
Then, create a Pod that uses that service account and allow the Pod to use its auto-generated service account token (If not explicitly denied, the default is always allow.)



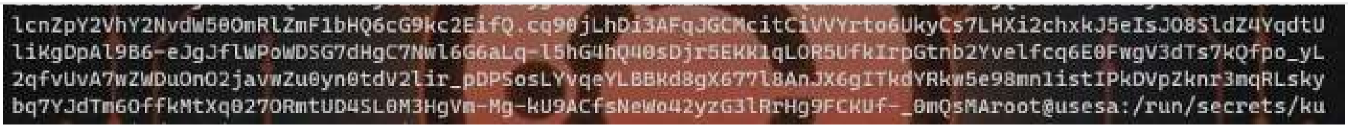
```
jonw@CKS-Master:~$ cat pod-usesa.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: usesa
  name: usesa
spec:
  serviceAccountName: podsa
  automountServiceAccountToken: true
  containers:
  - image: nginx
    name: usesa
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

When we execute inside a shell of the Pod, we could see the auto-generated service account token.

- *kubectl exec usesa -it — bash*
- *mount | grep sec*
- *cd /run/secrets/kubernetes.io/serviceaccount*
- *cat token*

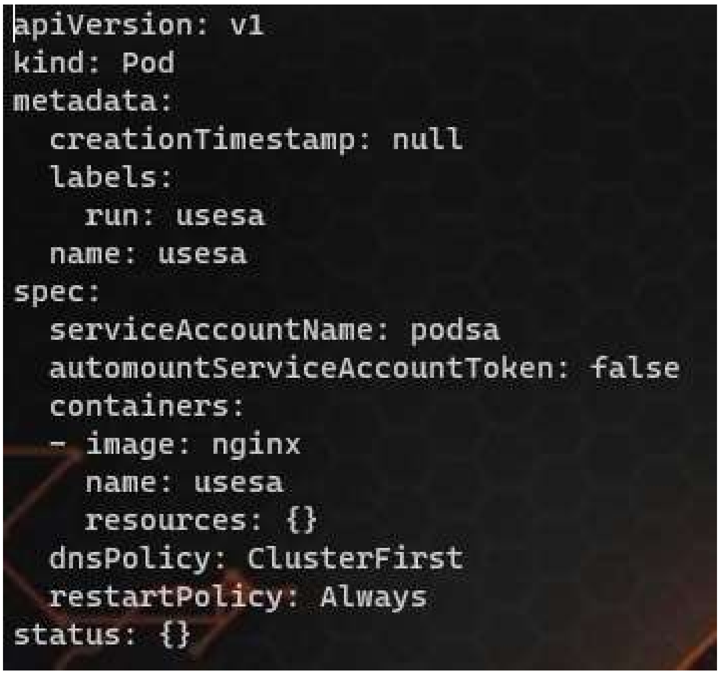


```
root@usesa:/# mount | grep sec
tmpfs on /run/secrets/kubernetes.io/serviceaccount type tmpfs (ro,relatime)
root@usesa:/# cd /run/secrets/kubernetes.io/serviceaccount
root@usesa:/run/secrets/kubernetes.io/serviceaccount# ls
ca.crt  namespace  token
root@usesa:/run/secrets/kubernetes.io/serviceaccount# cat token
```

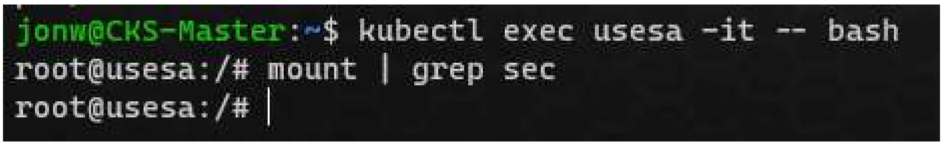


```
lcnZpY2VhY2NvdW50OmRlZmF1bHQ6cG9kc2EifQ.cq90jLhDi3AFqJGCMcitCiVVYrto6UkyCs7LHXi2chxkJ5eIsJO8SldZ4YqdtU
liKgDpAl9B6-eJgJflWPoWDSG7dHgC7Nwl6G6aLq-l5hG4hQ40sDjr5EKK1qLOR5UfkIrpGtnb2YvelfcqbE0FwgV3dTs7kQfpo_yL
2qfvUvA7wZWDuOnO2javwZu0yn0tdV2lir_pDPSosLYvqeYLBBKd8gX677l8AnJX6gITkdYRkw5e98mnlistIPkDVpZknr3mqRLsky
bq7YJdTm6OffkMtXq027ORmtUD4SL0M3HgVm-Mg-kU9ACfsNeWo42yzG3lRrHg9FCKUf-_0mQsMAroot@usesa:/run/secrets/ku
```

If we recreate the Pod with DISALLOWING it to use the auto-generated service account token, we see no auto-generated tokens being mounted in the running container.



```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: usesa
  name: usesa
spec:
  serviceAccountName: podsa
  automountServiceAccountToken: false
  containers:
  - image: nginx
    name: usesa
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```



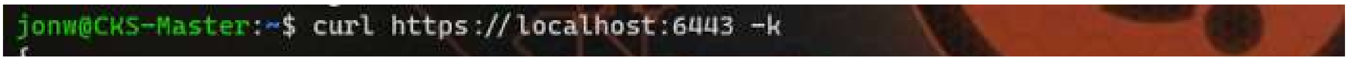
```
jonw@CKS-Master:~$ kubectl exec usesa -it -- bash
root@usesa:/# mount | grep sec
root@usesa:/#
```

Last but not least, we could apply best practices by limit service account permissions with the right roles or cluster roles.

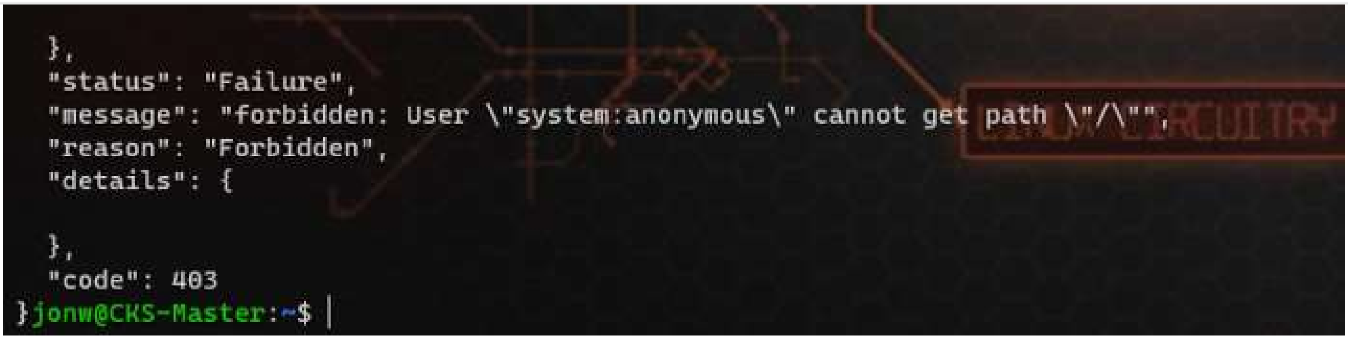
### Kube API Server Access Management

Kube API server is considered the brain of K8s, so we would need to take extra caution what could get access to this core service. By default, kube-apiserver would allow anonymous access as we could see we get HTTP status 403 forbidden access when executing

- *curl https://localhost:6443 -k*



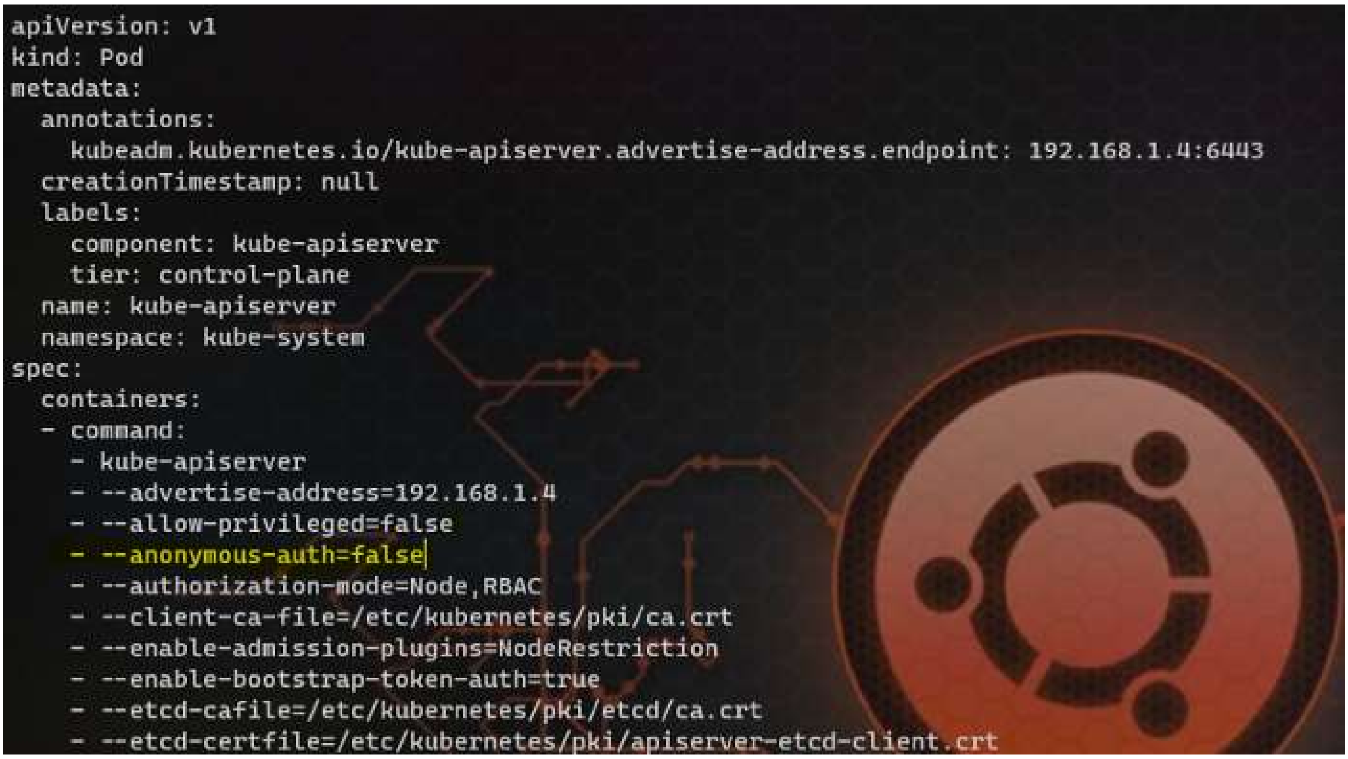
```
jonw@CKS-Master:~$ curl https://localhost:6443 -k
```



```
  },
  "status": "Failure",
  "message": "forbidden: User \"system:anonymous\" cannot get path \"/\"",
  "reason": "Forbidden",
  "details": {

  },
  "code": 403
}jonw@CKS-Master:~$ |
```

If the request cannot even start the authentication, most likely, the console would return HTTP status 401 unauthorized according to this section of official documentation. The way to DISABLE anonymous access is by adding an additional parameter in kube-apiserver

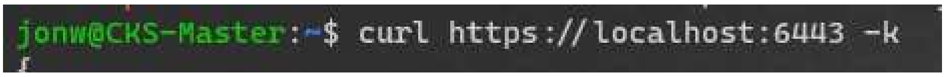
- *sudo nano /etc/kubernetes/manifests/kube-apiserver.yaml*



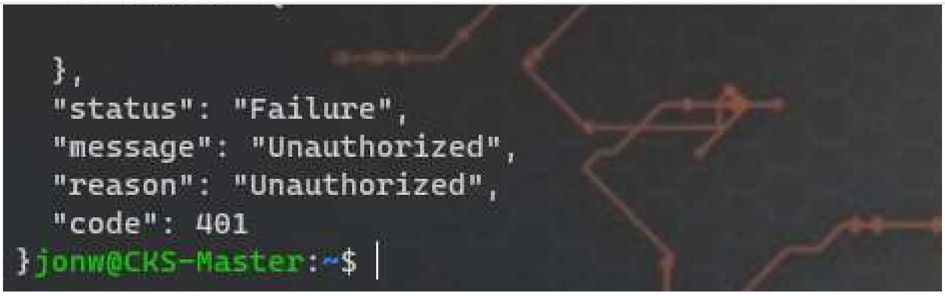
```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 192.168.1.4:6443
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.1.4
    - --allow-privileged=false
    - --anonymous-auth=false
    - --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
```

Once this is done, wait for kube-apiserver to restart and we could again try to access kube-apiserver with anonymous user. As expected, the console is now return HTTP status 401 as the request is not even being authenticated.

- *curl https://localhost:6443 -k*



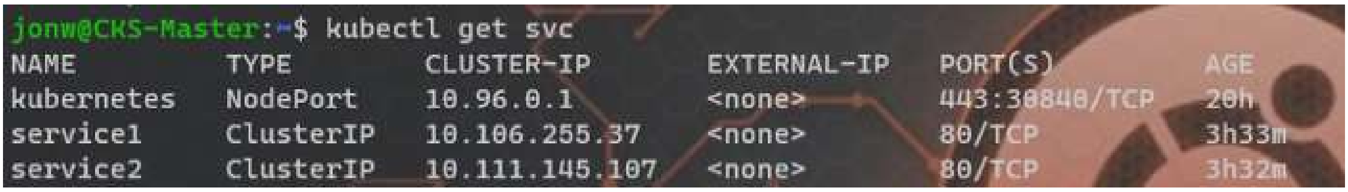
```
jonw@CKS-Master:~$ curl https://localhost:6443 -k
{
```



```
  },
  "status": "Failure",
  "message": "Unauthorized",
  "reason": "Unauthorized",
  "code": 401
}jonw@CKS-Master:~$ 
```

Let’s switch the setting back on allowing anonymous access to kube-apiserver and see how we could expose kube-apiserver service for external access.

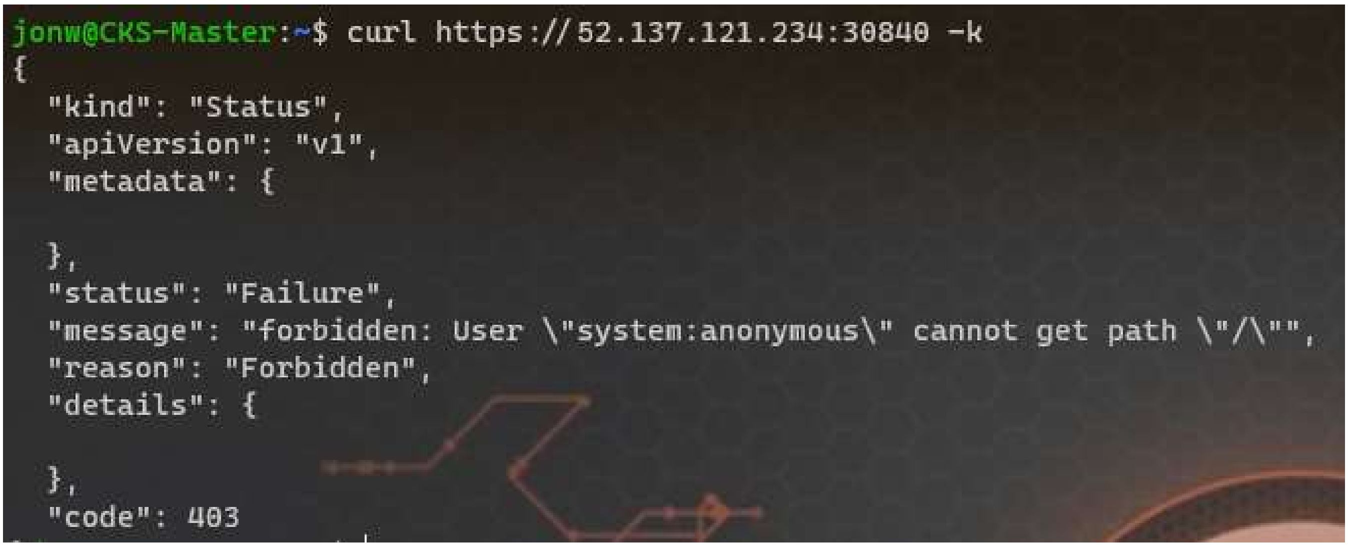
First thing first, change service “kubernetes” to be exposed from ClusterIP to NodePort, so we could use nodes’ public/private IP address and assigned port to access. Since I do not have another VM setup in the same network environment the K8s cluster is in, I would use nodes’ public IP address for demonstration.



```
jonw@CKS-Master:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)         AGE
kubernetes   NodePort    10.96.0.1        <none>        443:30840/TCP   20h
service1     ClusterIP   10.106.255.37    <none>        80/TCP          3h33m
service2     ClusterIP   10.111.145.107   <none>        80/TCP          3h32m
```

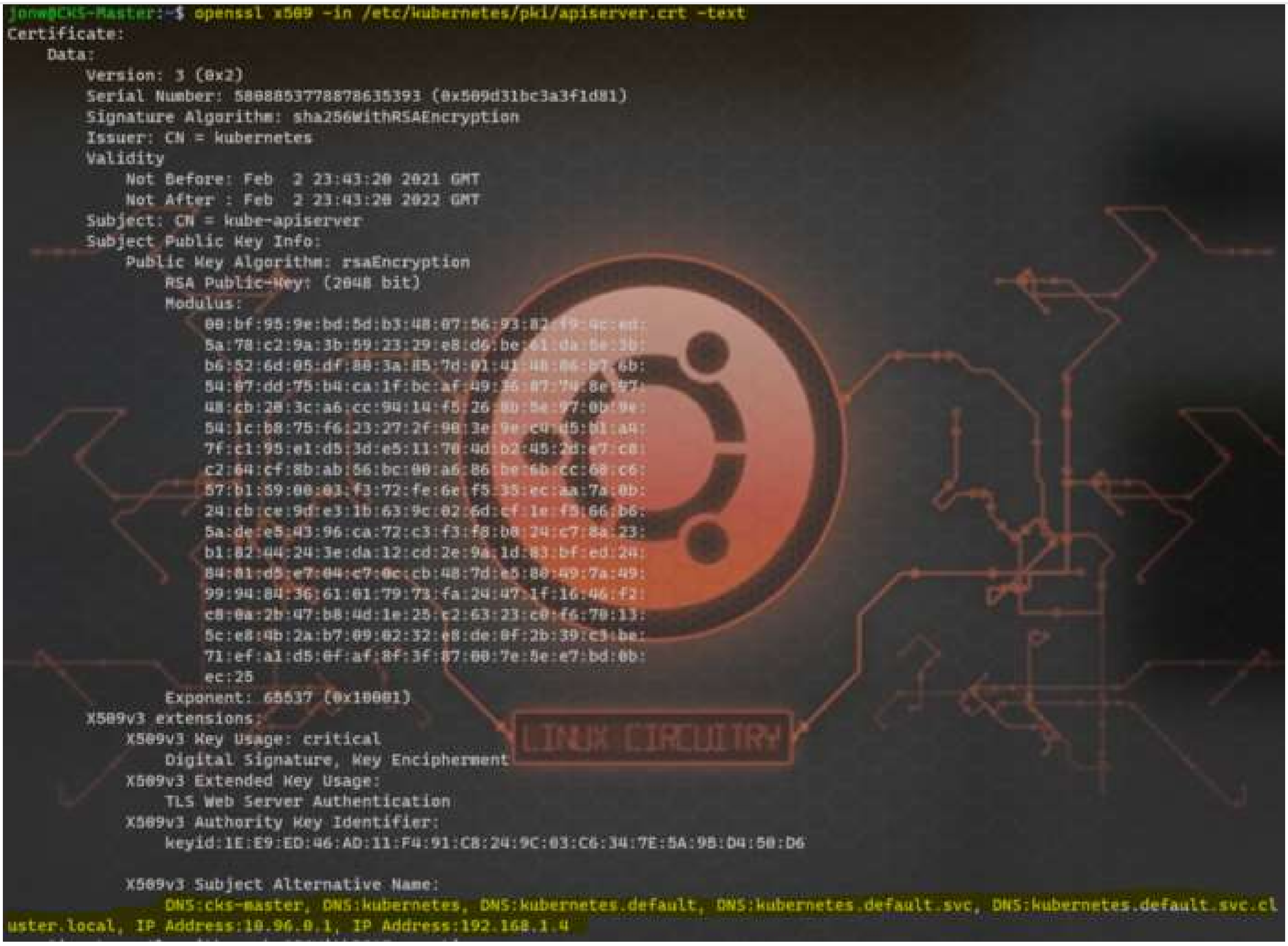
Do a simple curl test to see whether we are getting HTTP status 403

- *curl https://52.137.121.234:30840 -k*



```
jonw@CKS-Master:~$ curl https://52.137.121.234:30840 -k
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "forbidden: User \"system:anonymous\" cannot get path \"/\"",
  "reason": "Forbidden",
  "details": {

  },
  "code": 403
```

Now, we ensure we are using kube-apiserver-recognized FQDN or IP address



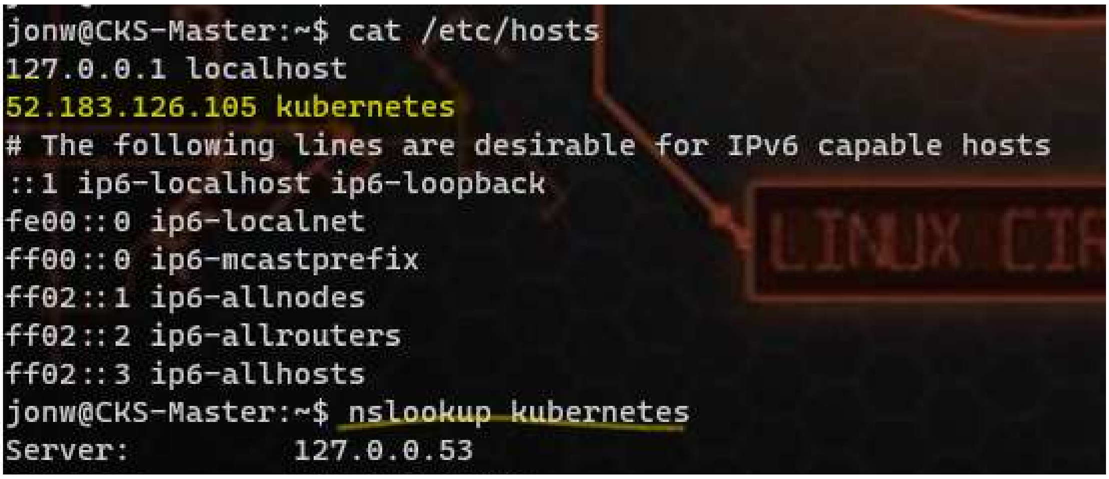
```
jonw@CKS-Master:~$ openssl x509 -in /etc/kubernetes/pki/apiserver.crt -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 5808853778878635393 (0x509d31bc3a3f1d81)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = kubernetes
        Validity
            Not Before: Feb  2 23:43:20 2021 GMT
            Not After : Feb  2 23:43:20 2022 GMT
        Subject: CN = kube-apiserver
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:bf:95:9e:bd:5d:b3:48:07:56:93:82:f9:9c:ed:
                    5a:78:c2:9a:3b:59:23:29:e8:d6:be:61:da:5e:3b:
                    b6:52:6d:05:df:80:3a:85:7d:01:41:48:06:b7:6b:
                    54:07:dd:75:b4:ca:1f:bc:af:49:36:07:74:8e:57:
                    48:cb:20:3c:a6:cc:94:14:f5:26:8b:5e:97:0b:9e:
                    54:1c:b8:75:f6:23:27:2f:90:3e:9e:c4:d5:01:a4:
                    7f:c1:95:e1:d5:3d:e5:11:70:4d:b2:45:2d:e7:c8:
                    c2:64:cf:8b:ab:56:bc:00:a6:86:be:6b:cc:60:c6:
                    57:b1:59:00:03:f3:72:fe:6e:f5:35:ec:aa:7a:0b:
                    24:cb:ce:9d:e3:1b:63:9c:02:6d:cf:1e:f5:66:b6:
                    5a:de:e5:43:96:ca:72:c3:f3:f8:b0:24:c7:8a:23:
                    b1:82:44:24:3e:da:12:cd:2e:9a:1d:83:bf:ed:24:
                    84:01:d5:e7:04:c7:0c:cb:48:7d:e5:80:49:7a:49:
                    99:94:84:36:61:01:79:73:fa:24:47:1f:16:46:f2:
                    c8:0a:2b:47:b8:4d:1e:25:c2:63:23:c0:f6:70:13:
                    5c:e8:4b:2a:b7:09:02:32:e8:de:0f:2b:39:c3:be:
                    71:ef:a1:d5:0f:af:8f:3f:07:00:7e:5e:e7:bd:0b:
                    ec:25
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
            X509v3 Authority Key Identifier:
                keyid:1E:E9:ED:46:AD:11:F4:91:C8:24:9C:03:C6:34:7E:5A:9B:D4:50:D6

            X509v3 Subject Alternative Name:
                DNS:cks-master, DNS:kubernetes, DNS:kubernetes.default, DNS:kubernetes.default.svc, DNS:kubernetes.default.svc.cl
uster.local, IP Address:10.96.0.1, IP Address:192.168.1.4
```

Edit hosts to let any of the FQDN we are seeing in the image above to be resolved in nodes' public IP address required (nodes' public/private IP address).

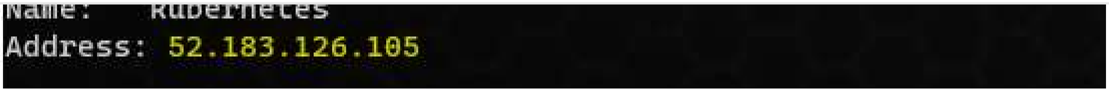
- *sudo nano /etc/hosts*

Look up the kubernetes and see what it resolves into

- *nslookup kubernetes*



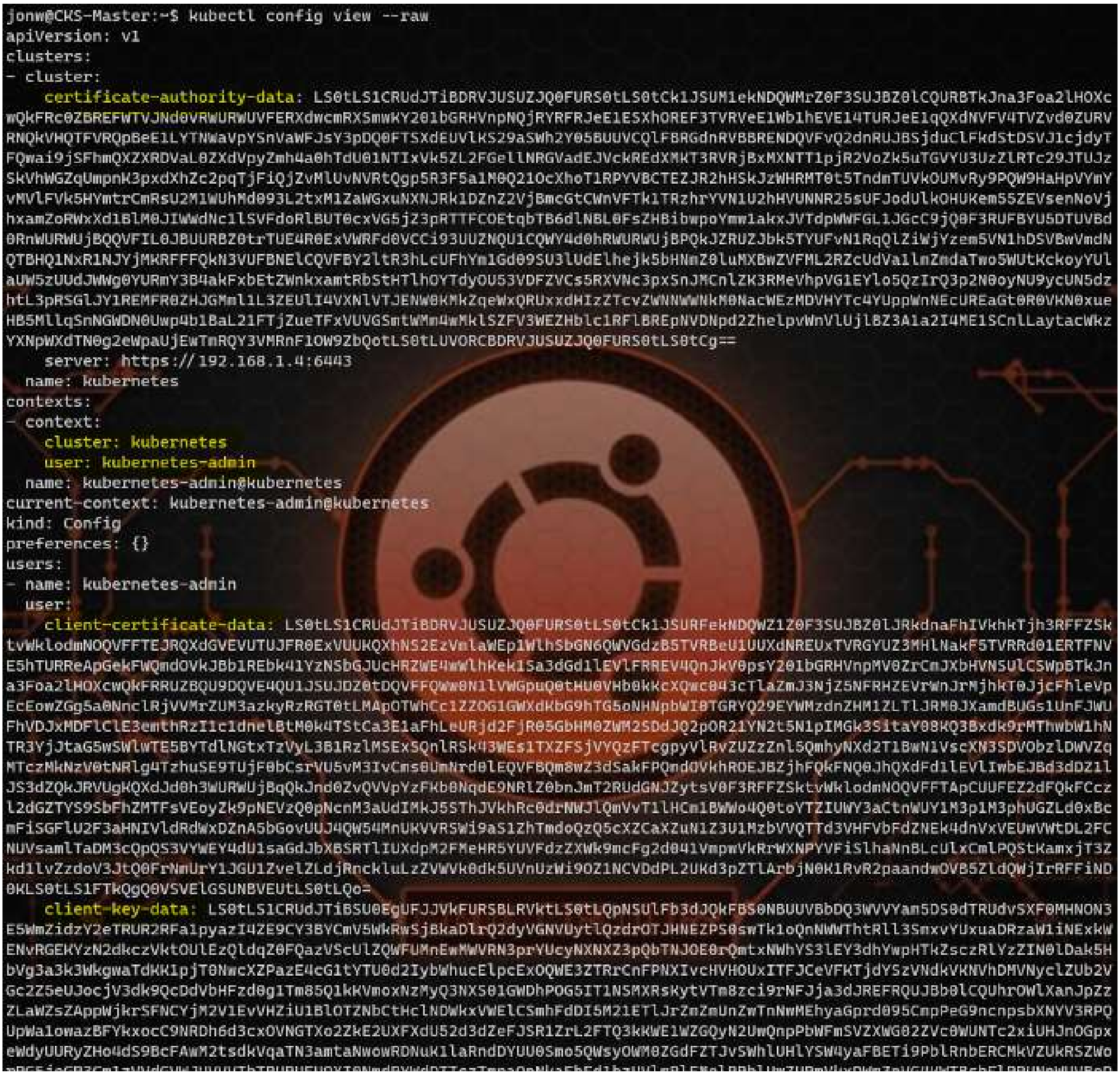
```
jonw@CKS-Master:~$ cat /etc/hosts
127.0.0.1 localhost
52.183.126.105 kubernetes
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
jonw@CKS-Master:~$ nslookup kubernetes
Server:         127.0.0.53
```



```
Name:    kubernetes
Address: 52.183.126.105
```

View the raw data of kubectl config content.

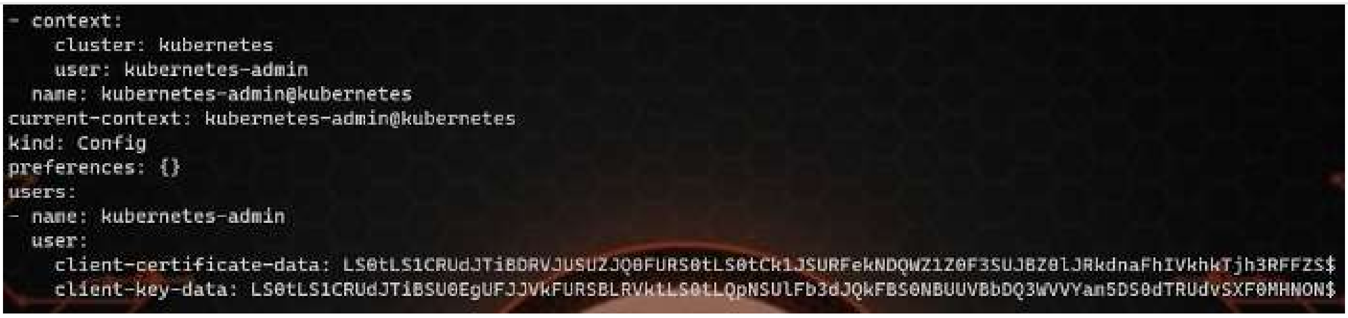
- *kubectl config view — raw > conf*



```
jonw@CKS-Master:~$ kubectl config view --raw
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUM1ekNDQWMrZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXc
wQkFRc0ZBREFWTVJNd0VRWURWUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRJeE1ESXhOREF3TVRVeE1Wb1hEVE14TURJeE1qQXdNVFV4TVZvd0ZURV
RNQkVHQTFVRQpBeE1LYTNWaVpYSnVaWFJsY3pDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBSjduClFkdStDSVJ1cjdyT
FQwai9jSFhmQXZXRDVaL0ZXdVpyZmh4a0hTdU01NTIxVk5ZL2FGellNRGVadEJVckREdXMKT3RVRjBxMXNTT1pjR2VoZk5uTGVYU3UzZlRTc29JTUJz
SkVhWGZqUmpnK3pxdXhZc2pqTjFiQjZvMlUvNVRtQgp5R3F5a1M0Q21OcXhoT1RPYVBCTEZJR2hHSkJzWHRMT0t5TndmTUVkOUMvRy9PQW9HaHpVYmY
vMVlFVk5HYmtrCmRsU2M1WUhMd093L2txM1ZaWGxuNXNJRk1DZnZ2VjBmcGtCWnVFTk1TRzhrYVN1U2hHVUNNR25sUFJodUlkOHUKem55ZEVsenNoVj
hxamZoRWxXd1BlM0JIWWdNc1lSVFdoRlBUT0cxVG5jZ3pRTTFCOEtqbTB6dlNBL0FsZHBibwpoYmw1akxJVTdpWWFGL1JGcC9jQ0F3RUFBYU5DTUVBd
0RnWURWUjBQQVFIL0JBUURBZ0trTUE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0hRWURWUjBPQkJZRUZJbk5TYUFvN1RqQlZiWjYzem5VN1hDSVBwVmdN
QTBHQ1NxR1NJYjMKRFFFQkN3VUFBNElCQVFBY2ltR3hLcUFhYm1Gd09SU3lUdElhejk5bHNmZ0luMXBwZVFML2RZcUdVa1lmZmdaTwo5WUtKckoyYUl
aUW5zUUdJWWg0YURmY3B4akFxbEtZWnkxamtRbStHTlhOYTdyOU53VDFZVCs5RXVNc3pxSnJMCnlZK3RMeVhpVG1EYlo5QzIrQ3p2N0oyNU9ycUN5dz
htL3pRSGlJY1REMFR0ZHJGMml1L3ZEUlI4VXNlVTJENW0KMkZqeWxQRUxxdHIzZTcvZWNNWWNkM0NacWEzMDVHYTc4YUppWnNEcUREaGt0R0VKN0xue
HB5MllqSnNGWDN0Uwp4b1BaL21FTjZueTFxVUVGSmtWMm4wMklSZFV3WEZHblc1RFlBREpNVDNpd2ZhelpvWnVlUjlBZ3A1a2I4ME1SCnlLaytacWkz
YXNpWXdTN0g2eWpaUjEwTmRQY3VMRnF1OW9ZbQotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
    server: https://192.168.1.4:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURFekNDQWZ1Z0F3SUJBZ0lJRkdnaFhIVkhkTjh3RFFZSk
tvWklodmNOQVFFTEJRQXdGVEVUTUJFR0ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB5TVRBeU1UUXdNREUxTVRGYUZ3MHlNakF5TVRRd01ERTFNV
E5hTURReApGekFWQmdOVkJBb1REbk41YzNSbGJUcHRZWE4wWlhKek1Sa3dGd1lEVlFRREV4QnJkV0psY201bGRHVnpMV0ZrCmJXbHVNSUlCSWpBTkJn
a3Foa2lHOXcwQkFRRUZBQU9DQVE4QU1JSUJDZ0tDQVFFQWw0N1lVWGpuQ0tHU0VHb0kkcXQwc043cTlaZmJ3NjZ5NFRHZEVrWnJrMjhkT0JjcFhleVp
EcEowZGg5a0NnclRjVVMrZUM3azkyRzRGT0tLMApOTWhCc1ZZOG1GWXdKbG9hTG5oNHNpbWI0TGRYQ29EYWMzdnZHM1ZLTlJRM0JXamdBUGs1UnFJWU
FhVDJxMDFlClE3emthRzI1c1dnelBtM0k4TStCa3E1aFhLeURjd2FjR05GbHM0ZWM2SDdJQ2pOR21YN2t5N1pIMGk3S1taY08KQ3Bxdk9rMThwbW1hN
TR3YjJtaG5wSWlWTE5BYTdlNGtxTzVyL3B1RzlMSExSQnlRSk43WEs1TXZFSjVYQzFTcgpyVlRvZUZzZnl5QmhyNXd2T1BwN1VscXN3SDVObzlDWVZq
MTczMkNzV0tNRlg4TzhuSE9TUjF0bCsrVU5vM3IvCms0UmNrd0lEQVFBQm8wZ3dSakFPQmdOVkhROEJBZjhFQkFNQ0JhQXdFd1lEVlIwbEJBd3dDZ1l
JS3dZQkJRVUgKQXdJd0h3WURWUjBqQkJnd0ZvQVVpYzFKb0NqdE9NRlZ0bnJmT2RUdGNJZytsV0F3RFFZSktvWklodmNOQVFFTApCUUFFZ2dFQkFCcz
l2dGZTYS9SbFhZMTFsVEoyZk9pNEVzQ0pNcnM3aUdIMkJ5SThJVkhRc0drNWJlQmVvT1lHCm1BWWo4Q0toYTZIUWY3aCtnWUY1M3p1M3phUGZLd0xBc
mFiSGFlU2F3aHNIVldRdWxDZnA5bGovUUJ4QW54MnUKVVRSWi9aS1ZhTmdoQzQ5cXZCaXZuN1Z3U1MzbVVQTTd3VHFVbFdZNEk4dnVxVEUwVWtDL2FC
NUVsamlTaDM3cQpQS3VYWEY4dU1saGdJbXBSRTlIUXdpM2FMeHR5YUVFdzZXWk9mcFg2d041VmpwVkRrWXNPYVFiSlhaNnBLcUlxCmlPQStKamxjT3Z
kd1lvZzdoV3JtQ0FrNmUrY1JGU1ZvelZLdjRnckluLzZVWVk0dk5UVnUzWi9OZ1NCVDdPL2UKd3pZTlArbjN0K1RvR2paandwOVB5ZldQWjIrRFFiND
0KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFb3dJQkFBS0NBUUVBbDQ3WVVYam5DS0dTRUdvSXF0MHNON3
E5WmZidzY2eTRUR2RFa1pyazI4ZE9CY3BYCmV5WkRwSjBkaDlrQ2dyVGNVUytlQzdrOTJHNEZPS0swTk1oQnNWWThtRll3SmxvYUxuaDRzaW1iNExkW
ENvRGEKYzN2dkczVktOUlEzQldqZ0FQazVScUlZQWFUMnEwMWVRN3prYUcyNXNXZ3pQbTNJOE8rQmtxNWhYS3lEY3dhYwpHTkZsczRlYzZIN0lDak5H
bVg3a3k3WkgwaTdKK1pjT0NwcXZPazE4cG1tYTU0d2IybWhucElpVkxOQWE3ZTRrcnFPNXIvcHVHOUxITFJCeVFKTjdYSzVNdkVKNVhDMVNyclZUb2V
Gc2Z5eUJocjV3dk9QcDdVbHFzd0g1Tm85Q1kKVmoxNzMyQ3NXS01GWDhPOG5IT1NSMXRsKytVTm8zci9rNFJja3dJREFRQUJBb0lCQUhrOWlXanJpZz
ZLaWZsZAppWjkrSFNCYjM2V1EvVHZiU1BlOTZNbCtHclNDWkxVWElCSmhFdDI5M21ETlJrZmZmUnZwTnNwMEhyaGprd095CmpPeG9ncnpsbXNYVV3RPQ
UpWa1owazBFYkxocC9NRDh6d3cxOVNGTXo2ZkE2UXFXdU52d3dZeFJSR1ZrL2FTQ3kKWE1WZGQyN2UwQnpPbWFmSVZXWG82ZVc0WUNTc2xiUHJnOGpx
eWdyUURyZHo4dS9BcFAwM2tsdkVqaTN3amtaNwowRDNuK1laRndDYUU0Smo5QWsyOWM0ZGdFZTJvSWhlUHlYSW4yaFBETi9PblRnbERCMkVZUkRSZWo
```

Save the existing raw CA certificate, client certificate and key information into file “conf” and modify the server information to use nodes’ public/private IP address or in this case FQDN if you have configured in hosts record.

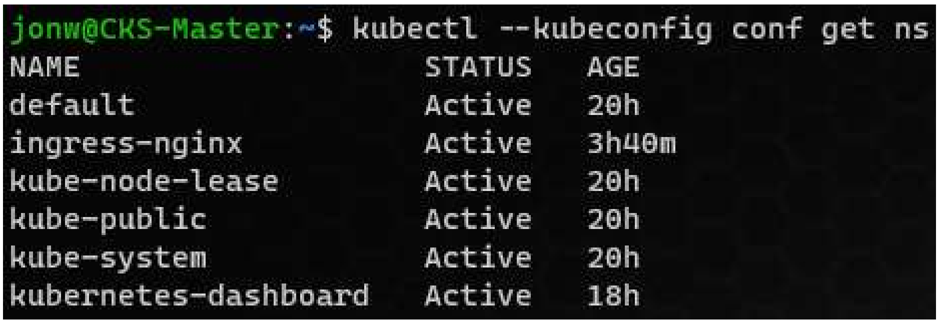
```
apiVersion: v1
clusters:
```



```
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURFekNDQWZ1Z0F3SUJBZ0lJRkdnaFhIVkhkTjh3RFFZS$
    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFb3dJQkFBS0NBUUVBbDQ3WVVYam5DS0dTRUdvSXF0MHNON$
```

Try contacting kube-apiserver, try to get namespaces in this case, with the modified conf file.

- *kubectl — kubeconfig conf get ns*

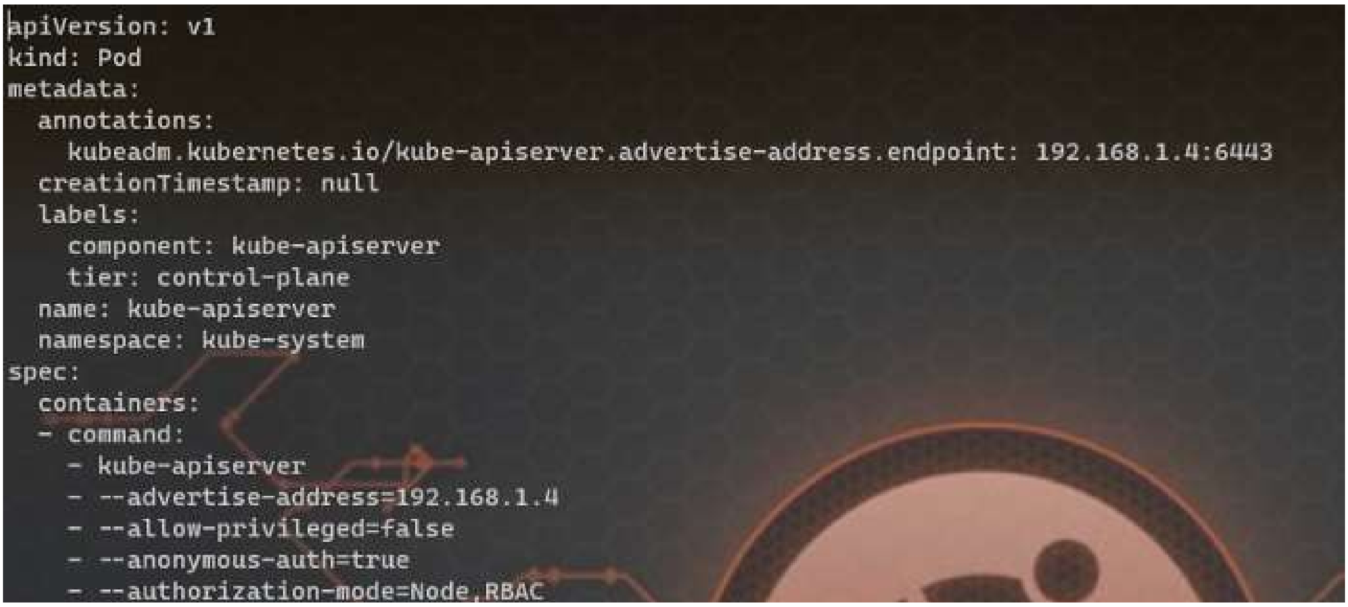


```
jonw@CKS-Master:~$ kubectl --kubeconfig conf get ns
NAME                   STATUS   AGE
default                Active   20h
ingress-nginx          Active   3h40m
kube-node-lease        Active   20h
kube-public            Active   20h
kube-system            Active   20h
kubernetes-dashboard   Active   18h
```

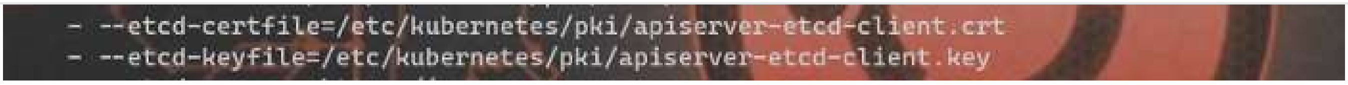
### Node Restriction

Check on K8s master nodes and see whether NodeRestriction admission plug-in is enabled already

- *sudo cat /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
    - --advertise-address=192.168.1.4
    - --allow-privileged=false
    - --anonymous-auth=true
    - --authorization-mode=Node,RBAC
```



```
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
```

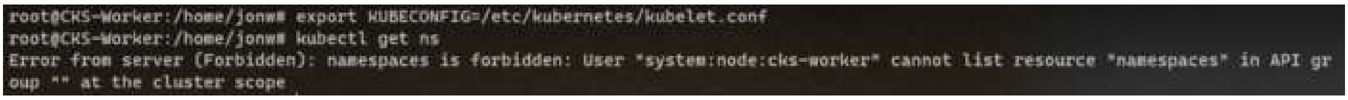
Head over to K8s worker nodes and check whether using worker nodes' kubelet context could modify master nodes labels. First, we would need to associate worker nodes' kubelet context as default context for executing K8s CLI

- *sudo su*
- *export KUBECONFIG=/etc/kubernetes/kubelet.conf*

and test with whatever command works or not

- *kubectl get ns*

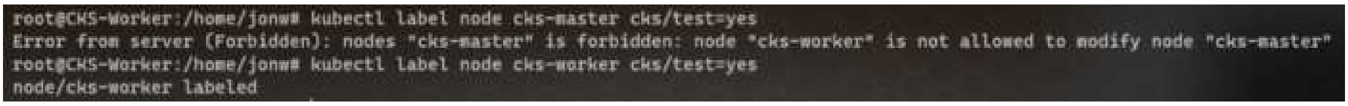
As expected, worker nodes' kubelet does not have permission to get namespaces.



```
root@CKS-Worker:/home/jonw# export KUBECONFIG=/etc/kubernetes/kubelet.conf
root@CKS-Worker:/home/jonw# kubectl get ns
Error from server (Forbidden): namespaces is forbidden: User "system:node:cks-worker" cannot list resource "namespaces" in API gr
oup "" at the cluster scope
```

With worker nodes' kubelet context, it is not authorized to label master nodes but it is able to label worker nodes (itself or other nodes).

- *kubectl label node cks-master cks/test=yes*
- *kubectl label node cks-worker cks/test=yes*



```
root@CKS-Worker:/home/jonw# kubectl label node cks-master cks/test=yes
Error from server (Forbidden): nodes "cks-master" is forbidden: node "cks-worker" is not allowed to modify node "cks-master"
root@CKS-Worker:/home/jonw# kubectl label node cks-worker cks/test=yes
node/cks-worker labeled
```

### Upgrade Kubernetes Clusters with kubeadm

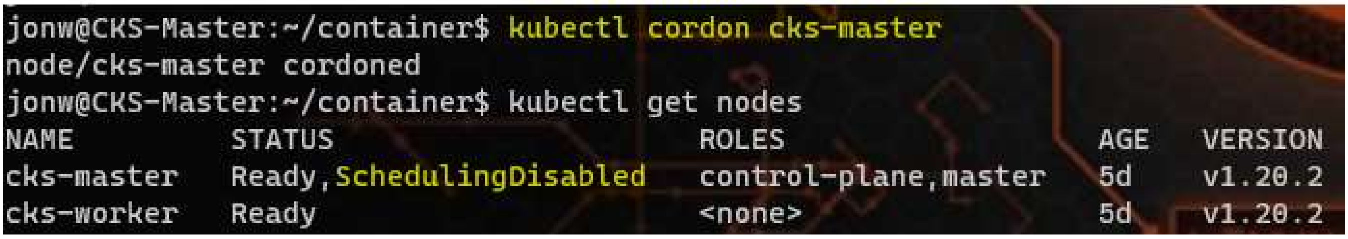
One of the most common tasks every IT administrator would need to do is to update or upgrade the running machines. For K8s administrators, K8s version would also need to be maintain in supported scope and the following would show how to do it.

## Master Nodes

Get all the apt updates

Make sure no more pods being scheduled to master nodes.

- *kubectl cordon cks-master*



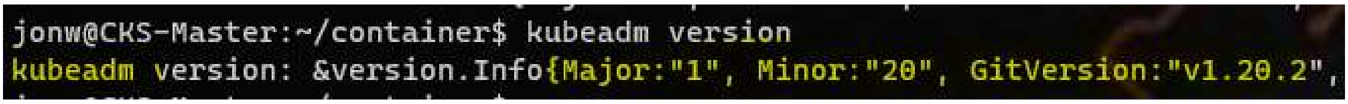
```
jonw@CKS-Master:~/container$ kubectl cordon cks-master
node/cks-master cordoned
jonw@CKS-Master:~/container$ kubectl get nodes
NAME         STATUS                     ROLES                  AGE   VERSION
cks-master   Ready,SchedulingDisabled   control-plane,master   5d    v1.20.2
cks-worker   Ready                      <none>                 5d    v1.20.2
```

Drain all pods, deployment from master nodes.

- *kubectl drain cks-master — ignore-daemonsets*

Check kubeadm version

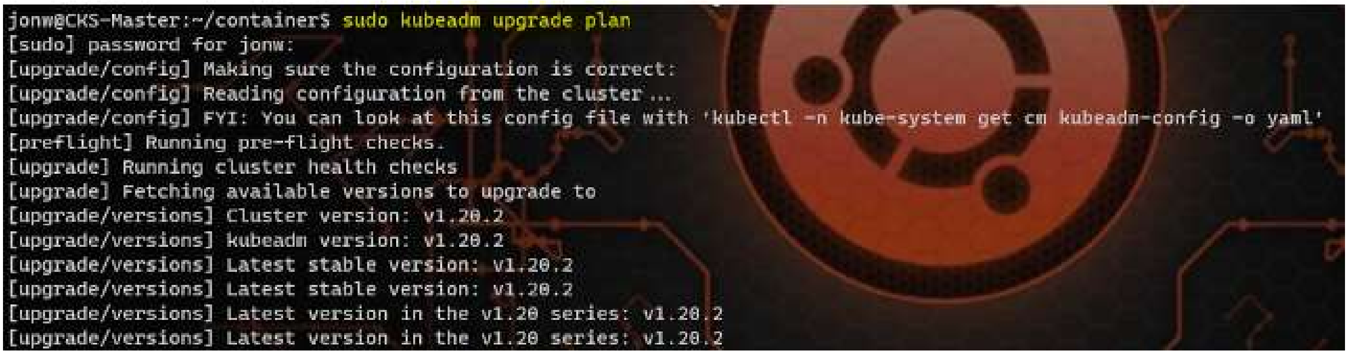
- *kubeadm version*



```
jonw@CKS-Master:~/container$ kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"20", GitVersion:"v1.20.2",
```

Get the upgrade plan

- *kubeadm upgrade plan*

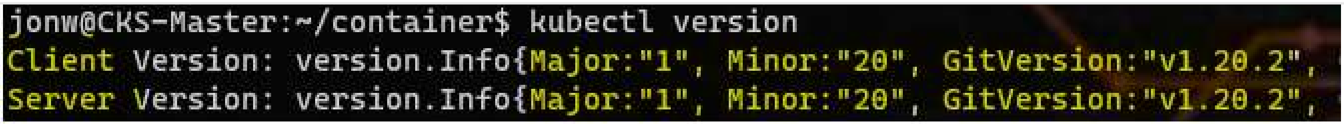


```
jonw@CKS-Master:~/container$ sudo kubeadm upgrade plan
[sudo] password for jonw:
[upgrade/config] Making sure the configuration is correct:
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[preflight] Running pre-flight checks.
[upgrade] Running cluster health checks
[upgrade] Fetching available versions to upgrade to
[upgrade/versions] Cluster version: v1.20.2
[upgrade/versions] kubeadm version: v1.20.2
[upgrade/versions] Latest stable version: v1.20.2
[upgrade/versions] Latest stable version: v1.20.2
[upgrade/versions] Latest version in the v1.20 series: v1.20.2
[upgrade/versions] Latest version in the v1.20 series: v1.20.2
```

Apply the upgrade plan shown before

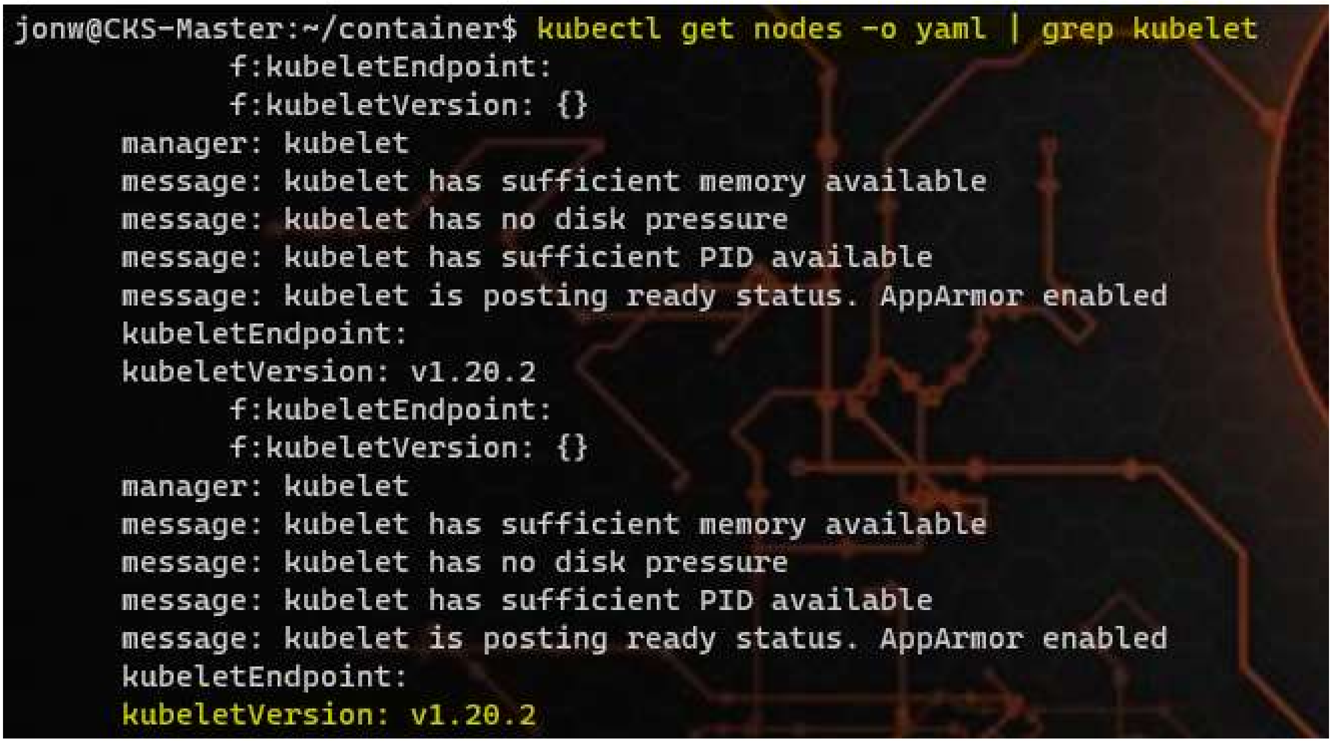
- *kubeadm upgrade apply <K8s version>*

Check kubectl and kubelet version



```
jonw@CKS-Master:~/container$ kubectl version
Client Version: version.Info{Major:"1", Minor:"20", GitVersion:"v1.20.2",
Server Version: version.Info{Major:"1", Minor:"20", GitVersion:"v1.20.2",
```

- *kubectl get nodes -o yaml | grep kubelet*



```
jonw@CKS-Master:~/container$ kubectl get nodes -o yaml | grep kubelet
            f:kubeletEndpoint:
            f:kubeletVersion: {}
      manager: kubelet
      message: kubelet has sufficient memory available
      message: kubelet has no disk pressure
      message: kubelet has sufficient PID available
      message: kubelet is posting ready status. AppArmor enabled
      kubeletEndpoint:
      kubeletVersion: v1.20.2
            f:kubeletEndpoint:
            f:kubeletVersion: {}
      manager: kubelet
      message: kubelet has sufficient memory available
      message: kubelet has no disk pressure
      message: kubelet has sufficient PID available
      message: kubelet is posting ready status. AppArmor enabled
      kubeletEndpoint:
      kubeletVersion: v1.20.2
```

Install all core components to the required version

- *apt-get install kubeadm=<K8s version> kubelet=<K8s version> kubectl=<K8s version>*

Make master nodes available for pod scheduling once again.

- *kubectl uncordon cks-master*

## Worker Nodes

Worker nodes upgrade is slightly different, but mostly the same concept.

Get all the apt updates

- *sudo apt-get update*

Make sure no more pods being scheduled to master nodes.

Drain all pods, deployment from master nodes.

- *kubectl drain cks-worker — ignore-daemonsets*

Check kubeadm version

- *kubeadm version*

Install the required kubeadm version

- *apt-get install kubeadm=<required version>*

Upgrade worker nodes with kubeadm

- *kubeadm upgrade node*

Install all core components to the required version

- *apt-get install kubelet=<K8s version> kubectl=<K8s version>*

Check kubelet and kubectl version to ensure they are running in the required version.

- *kubelet — version*
- *kubectl — version*

Make worker nodes available for pod scheduling once again.

- *kubectl uncordon cks-worker*

For more details on how to use kubeadm to upgrade master nodes and worker nodes, please check this site.

Kubernetes Ck Preparation Cloud Native Security