

GO FOR IT

# LEARN CKS SCENARIOS

SECURITY IS MUST AT ALL LEVELS

BY SAIYAM PATHAK

TWITTER - @SAIYAMPATHAK

YOUTUBE - [YOUTUBE.COM/C/SAIYAM911](https://www.youtube.com/c/Saiyam911)



# CKS

TRYING  
TO BE  
AS NATURAL AS  
POSSIBLE

LET'S  
LEARN  
&  
ENJOY!!

2 HOURS  
15 QUESTIONS [SUBJECT TO CHANGE]  
67% PASSING  
SCORE  
EACH QUESTION SEPERATE WEIGHTAGE  
CKA PREREQUISITE

## CONTENTS !!

- THE BOOK → what is it all about?
- Why CKS?
- FROM WHERE TO LEARN → IN THE END 😊

→ PRACTICE QUESTIONS  
+  
VIDEO SOLUTIONS  
+  
REGULAR UPDATES

THIS IS  
SERIOUS

★ MAIN  
★ HIGHLIGHT  
★ ←

# Contents

<b>Contents</b>	<b>3</b>
<b>The BOOK</b>	<b>5</b>
<b>Katacoda Platform</b>	<b>6</b>
<b>Kind cluster</b>	<b>7</b>
<b>WHY CKS ??</b>	<b>8</b>
<b>Problem 1 - Network Policy - Part 1</b>	<b>9</b>
Solution 1 -	9
<b>Problem 2 - NetworkPolicy - Part2</b>	<b>12</b>
Solution 2 -	12
<b>Problem 3 : AppArmor Profiles</b>	<b>18</b>
Solution 3:	18
<b>Problem 4 - RBAC</b>	<b>21</b>
Solution 4 - (Using Katacoda ubuntu playground)	21
<b>Problem 5: Image Scanning</b>	<b>25</b>
Solution 5:	25
<b>Problem 6 - Audit Policy</b>	<b>27</b>
Solution 6:	27
<b>Problem 7 : Kubernetes Upgrade</b>	<b>32</b>
Solution 7:	32
<b>Problem 8 - CIS Benchmark</b>	<b>39</b>
Solution 8:	39
<b>Problem 9 - Container Runtimes</b>	<b>43</b>
Solution 9 -	43
<b>Problem 10 : Falco</b>	<b>45</b>
Solution 10 :	45
<b>Problem 11 - Secrets</b>	<b>49</b>
Solution 11:	49
<b>Problem 12: PodSecurity Policy</b>	<b>53</b>
Solution 12:	53

<b>Problem 13: Security Context for a Pod or Container</b>	<b>56</b>
Solution 13:	56
<b>Problem14: Privileged Pods</b>	<b>58</b>
Solution 14:	58
<b>Problem 15: Dockerfile best practices and Deployment best practices</b>	<b>59</b>
Solution 15:	59
<b>Problem 16: ImagePolicyWebhook</b>	<b>60</b>
Solution 16 :	60
<b>From Where To Learn CKS Concepts and Bookmarks</b>	<b>63</b>

# The BOOK

This Book focuses on Kubernetes CKS certification where I have provided scenarios, my style explanation, written solution, and the video solutions as well(if you purchase with video solutions). These questions/scenarios should give you enough motivation and practice for appearing in the exam.

This book **does not** cover all the concepts in detail and is focussed on Practice questions with explanation + video solutions. I will also try to provide all the necessary resources (free+paid) from where you can learn the concepts in detail.

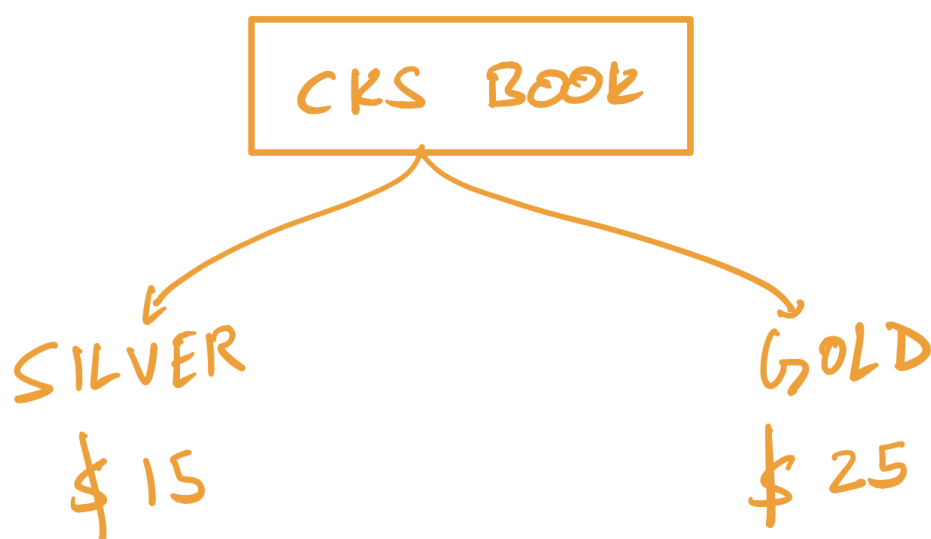
This Book **does not** contain exact questions from the CKS exam (I will not be disclosing them as well EVER). I will try to cover all the topics in the form of questions and then explain how to solve them and the fast approach that will help you save time during the exam. The exam is for 2 hours so you need to make sure that you read, understand, think and solve the problem fast.

I will also cover the bookmarks and some of the tips/tricks I used in the EXAM

There are three packs of this book :

silver = only book

Gold = book and video solutions



# Katacoda Platform

The solutions that I have provided in this book include the cluster setup as well that means the scenario setup is also there. For all the scenarios I have used 2 types of Katacoda Platforms:

- Katacoda Kubernetes Playground  
(<https://www.katacoda.com/courses/kubernetes/playground>)
- Katacoda Ubuntu Playground (here I install K3s for a few scenarios)  
(<https://www.katacoda.com/courses/ubuntu/playground>)
- My Kubernetes 1.22 with containerd playground  
(<https://www.katacoda.com/pathaksaiyam/scenarios/kube122>)  
I have created a playground with a single node Kubernetes cluster 1.22 and containerd for you to play around and do the scenarios.

You can also try these scenarios in your own Kubernetes clusters but to keep things from scratch, the scenario setup is also important so that you are able to see how the scenario gets created and then how to solve a particular problem.

**Ubuntu Playground**

**Katacoda Ubuntu Playground**

Terminal IDE Visualise Host +

Your Interactive Bash Terminal. A safe place to learn and execute commands.

\$

**Kubernetes Playground**

**Launch Cluster**

launch.sh ✓

This will create a two node Kubernetes cluster.

Terminal Host 1 +

controlplane \$ kubectl cluster-info  
Kubernetes master is running at https://172.17.0.13:6443  
KubeDNS is running at https://172.17.0.13:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy  
  
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.  
controlplane \$

# Kind cluster

kind is a tool for running local Kubernetes clusters using Docker container “nodes”.

Though Kind uses Docker but inside the nodes it uses the CRI/Containerd.

You can use Kind clusters for some of the CKS scenarios.

Kind documentation for installing on your laptop-







<https://kind.sigs.k8s.io/docs/user/quick-start/#installation>

```
kind --version  
kind version 0.11.1
```

Create 1.22 cluster using following command

```
~ kind create cluster --name=cks --image=kindest/node:v1.22.0
```

Creating cluster "cks" ...

- ✓ Ensuring node image (kindest/node:v1.22.0) 
- ✓ Preparing nodes 
- ✓ Writing configuration 
- ✓ Starting control-plane 
- ✓ Installing CNI 
- ✓ Installing StorageClass 

Set kubectl context to "kind-cks"

You can now use your cluster with:

```
kubectl cluster-info --context kind-cks
```

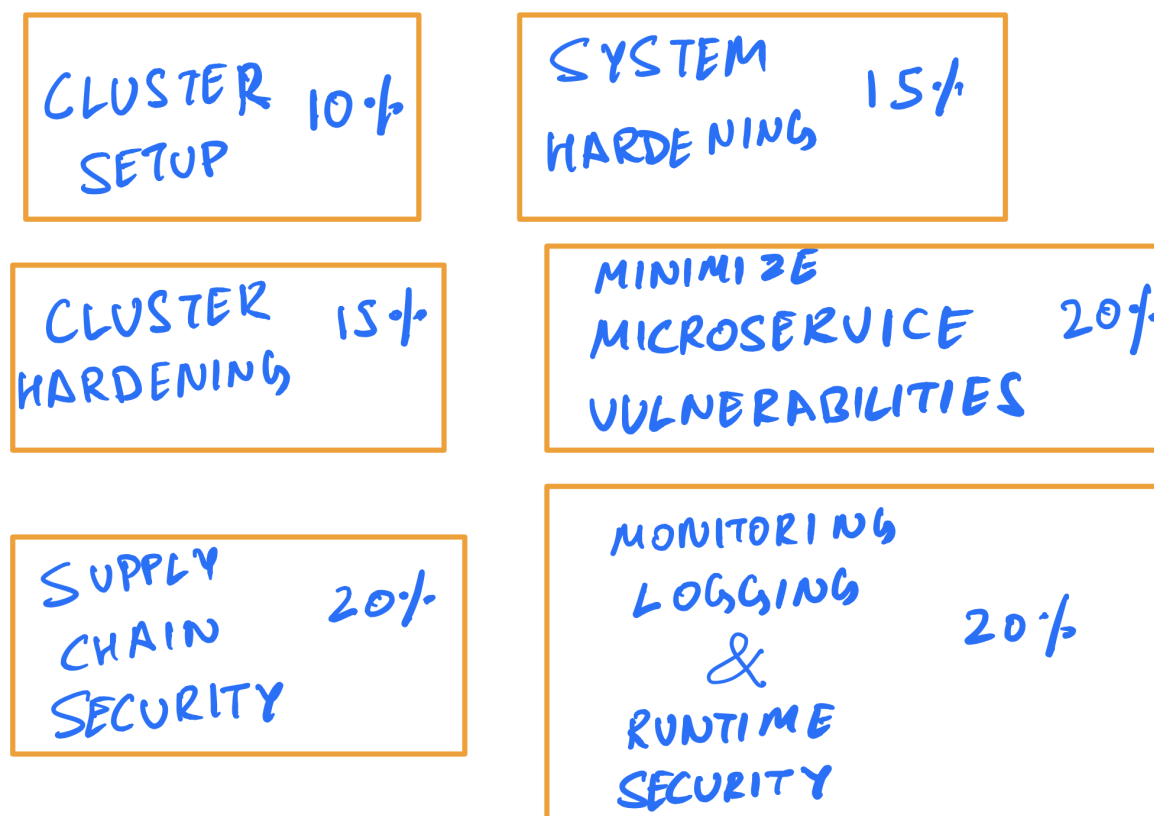
Thanks for using kind! 

# WHY CKS ??

Security is the most important aspect for any system, Kubernetes is no different. CKS is Certified Kubernetes Security exam by LinuxFoundation Launched during Kubecon NA 2020. There is a prerequisite to CKS which is CKA (Certified Kubernetes Administrator), so you have to appear and clear CKA first before even sitting for CKS.

Obviously it's Security a related exam so the exam is oriented towards security but the special thing about this exam is that it includes a lot of external tooling concepts to be understood and even the questions in the exam will be based on those tools.

Things you need to know !!





## Problem 1 - Network Policy - Part 1

- Create a cluster with NetworkPolicy controller enabled
- Create a pod demo1 with nginx image and run in default namespace
- Create a Namespace dev
- Create a pod demo2 with nginx image and run in dev namespace
- Connect from demo2 pod to demo1
- Create a Network Policy that denies all egress traffic in the dev namespace.
- Apply the Network Policy
- Connect from demo2 pod to demo1 and verify that all egress traffic from pods in dev namespace is restricted.

### Solution 1 -

NetworkPolicy in Kubernetes is used to allow/deny traffic in the cluster. You can create a NetworkPolicy for pods with specific labels or all pods in a namespace and apply these policies to set the ingress/egress for them. Even if you are starting to build your cluster you should always think about NetworkPolicy.

Here I am using the Katacoda ubuntu playground.

```
# Create an instance and install K3s using following command
curl -sL https://get.k3s.io |INSTALL_K3S_VERSION="v1.22.5+k3s1" sh -

kubectl get nodes
NAME          STATUS    ROLES          AGE    VERSION
host01        Ready     control-plane,master  42s    v1.22.5+k3s1

# Create dev namespace
kubectl create ns dev
namespace/dev created

# Run pod in default namespace
kubectl run demo1 --image=nginx
pod/demo1 created
```

```
# Run pod in dev namespace
kubectl run demo2 --image=nginx -n dev
pod/demo2 created

# Get the ip of pde demo1 in default namespace
kubectl get po -owide
NAME      READY   STATUS    RESTARTS   AGE   IP           3m15s   10.4.2.8
demo1     1/1     Running   0           3m15s   10.4.2.8

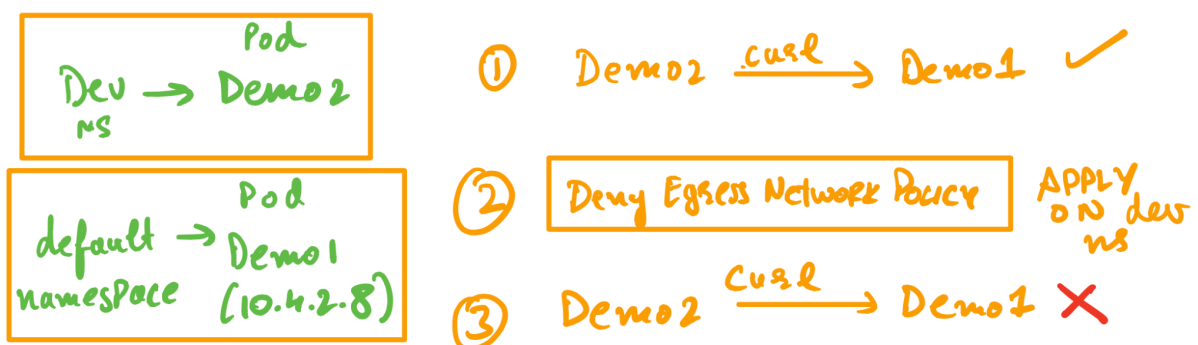
# Exec into pod demo2 and try to connect to demo1 to see the egress
kubectl exec -it demo2 -n dev -- sh
# curl 10.4.2.8
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</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>
```

```
# exit

# Apply the Deny egress policy to the dev namespace
cat << EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-egress
  namespace: dev
spec:
  podSelector: {}
  policyTypes:
    - Egress
EOF
networkpolicy.networking.k8s.io/deny-egress created

# exec into pod demo2 and try to connect to demo1 and this time it
should not connect.
kubectl exec -it demo2 -n dev -- sh
# curl 10.4.2.8
curl: (7) Failed to connect to 10.4.2.8 port 80: Connection timed out
```

Similar to this you can repeat the scenario with Ingress policy



## Problem 2 - NetworkPolicy - Part2

- Create a namespace red
- Create a pod demo1 with nginx image in default namespace
- Create a pod demo2 with nginx image in red namespace
- Create a pod demo3 with nginx image and label `demo:test` in the red namespace
- Create a NetworkPolicy which will restrict access to the pod demo2 in the red namespace but allow traffic from pods in the default namespace and pods with label `demo:test` in red namespace.

### Solution 2 -

For this problem we have to create a Network policy for the pod demo2 where we have to restrict the incoming traffic from everywhere except from the default namespace and the pods with label demo:test.

Using the same setup as above - Katacoda ubuntu playground with K3s

```
curl -sfl https://get.k3s.io |INSTALL_K3S_VERSION="v1.22.5+k3s1" sh -  
  
kubectl get nodes  
NAME          STATUS    ROLES          AGE    VERSION  
host01        Ready     control-plane,master  5m58s  v1.22.5+k3s1
```

Create a pod demo1 in default namespace

```
kubectl run demo1 --image=nginx  
pod/demo1 created
```

Create a Namespace red and create a demo2 pod in it.

```
kubectl create ns red  
namespace/red created  
  
kubectl run demo2 -n red --image=nginx  
pod/demo2 created
```

```
kubectl get pods -n red -owide
```

NAME	READY	STATUS	RESTARTS	AGE	IP
demo2	1/1	Running	0	5m24s	10.42.0.9
host01	<none>		<none>		

Create the network policy(np.yaml) to restrict pod demo2 in the red namespace.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: restrict
  namespace: red
spec:
  podSelector:
    matchLabels:
      run: demo2
  policyTypes:
    - Ingress
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            ns: default
      - from:
          podSelector:
            matchLabels:
              demo: test
```

```
kubectl apply -f np.yaml
networkpolicy.networking.k8s.io/restrict created
```

Right now there is no label on the default namespace so the pod demo1 should not be able to reach demo2.

```
kubectl exec demo1 -- curl 10.42.0.9
  % Total    % Received % Xferd  Average Speed   Time
Time      Time     Current                Dload  Upload    Total
Spent     Left  Speed
  0     0     0     0     0     0     0     0  --:--:--
--:--:-- --:--:--    0curl: (7) Failed to connect to
10.42.0.9 port 80: Connection refused
```

Let's Label the namespace default

```
kubectl label ns default ns=default
namespace/default labeled
```

Check the connectivity again - demo1 should be able to reach demo2

```
kubectl exec demo1 -- curl 10.42.0.9
  % Total    % Received % Xferd  Average Speed   Time
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>

```

YES!! Now this works, so the pods in the default namespace would be able to connect to the demo2 pod in the red namespace.

Lets create a demo3 pod in the red namespace and try to see if it's able to connect.

```

kubectl run demo3 -n red --image=nginx
pod/demo3 created

kubectl exec demo3 -n red -- curl 10.42.0.9
  % Total    % Received % Xferd  Average Speed   Time
Time      Time     Current                    Dload  Upload   Total
Spent     Left  Speed
  0     0     0     0     0     0     0     0  --:--:--
--:--:-- --:--:--          0curl: (7) Failed to connect to
10.42.0.9 port 80: Connection refused
command terminated with exit code 7

```

No this does not work because I added only the pods wil label **demo=test** in red namespace can be allowed to connect.  
Let's label the pod and try to connect again

```
kubectl label pod demo3 -n red demo=test
pod/demo3 labeled

kubectl exec demo3 -n red -- curl 10.42.0.9
<!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>
% Total      % Received % Xferd  Average Speed   Time
Time        Time     Current
```



					Dload	Upload	Total
Spent	Left		Speed				
100	612	100	612	0	0	597k	0 --:--:--
--:--:--	--:--:--	--:--:--	597k				

Awesome! The Scenario is successfully completed.

## Problem 3 : AppArmor Profiles

- Create Apparmor profile and apply it to the pod

**Note** - You can create any profile and use any pod, the main goal is to understand how some AppArmor commands work and how to use it for a Pod.

### Solution 3:

You can use the Ubuntu katacoda playground for this scenario.

AppArmor is a Linux Kernel module using which you can deploy workloads more securely. There are different profiles that you can create in order to restrict the access of a pod or deployment. After creating the profile you can restrict the container's access by adding annotation. There can be different sets of configurations that are set in the profile like you can block the resources or you can set up warnings.

In order to use AppArmor it has to be enabled and installed and by default many distributions support it by default.

You can check by using this command -

```
cat /sys/module/apparmor/parameters/enabled  
Y
```

For this solution I am using a single node K3s install in, you can install it by logging into the node and running the common -> `curl -sfl`

```
https://get.k3s.io |INSTALL_K3S_VERSION="v1.22.5+k3s1" sh -
```

Now create an apparmor profile to deny all writes to the nodes

Create a profile `deny_write`

```
#include <tunables/global>  
  
profile deny_write flags=(attach_disconnected) {
```

```
#include <abstractions/base>

file,

# Deny all file writes.
deny /** w,
}
```

Apply it to the node using following commands

```
# aa-status | grep deny
apparmor_parser -q deny_write
# apparmor_parser -q deny_write
# aa-status | grep deny
    deny_write
```

Now let's create a pod with deny\_write apparmor policy

```
apiVersion: v1
kind: Pod
metadata:
  name: deny
  annotations:
    # add annotation here to apply the AppArmor profile
    "deny_write".
    # format is ->
    container.apparmor.security.beta.kubernetes.io/<container
name>:localhost/<profilename>
    container.apparmor.security.beta.kubernetes.io/deny:
localhost/deny_write
spec:
  containers:
  - name: deny
    image: busybox
```

```
command: [ "sh", "-c", "echo 'Hello AppArmor!' &&
sleep 1h" ]
```

Apply above yaml file and check if the profile is applied correctly.

```
kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
deny      1/1     Running   0           11s

kubectl exec deny -- cat /proc/1/attr/current
deny_write (enforce)
```

Try to create a file

```
kubectl exec deny -- touch /tmp/saiyam
touch: /tmp/saiyam: Permission denied
command terminated with exit code 1
```

So now you have successfully created an apparmor profile, loaded it and applied it to the pod!! You are Awesome!! Let's move ahead.

## Problem 4 - RBAC

- Setup the lab :
    - 1 - Create namespace demo
    - 2 - Create a service account **sam**, **ClusterRole** named **cr** with list, get, watch, delete level access for secrets, pods, deployments and bind it to the same service account by creating **RoleBinding** named **super** in the namespace demo.
- Main scenario:
- Verify that SA sam can delete the deployment but not create it.
  - Limit the SA sam capabilities to view only secrets.
  - Verify that SA sam is only able to view secrets and nothing else.
  - Create a new Role **rr** with create level access for deployments, Rolebinding named **limited** in namespace **demo**.

## Solution 4 - (Using Katacoda ubuntu playground)

RBAC is Role based Access control used to control the access for the user or service account to different kubernetes resources. The

--authorization-mode=RBAC should be enabled in the kube-apiserver.yaml file(located at `/etc/kubernetes/manifests/kube-apiserver.yaml`).

There are three steps in general that are performed in these types of scenarios.

- Create a Service Account
- Create ClusterRole/Role
- Created ClusterRoleBinding/RoleBinding

ClusterRole + RoleBinding - Can be used

Role + RoleBinding - Can be used

ClusterRole + ClusterRoleBinding - Can be used

Role + ClusterRoleBinding - WRONG

For this scenario I am using k3s cluster that can be simply setup by command

```
curl -sfL https://get.k3s.io |INSTALL_K3S_VERSION="v1.22.5+k3s1" sh -
```

### Create Namespace demo

```
kubectl create ns demo  
namespace/demo created
```

### Create SA sam

```
kubectl create sa sam -n demo  
serviceaccount/sam created
```

### Create ClusterRole cr

```
kubectl create clusterrole cr  
--verb=get,list,watch,delete  
--resource=secrets,pods,deployments  
clusterrole.rbac.authorization.k8s.io/cr created
```

### Create RoleBinding super

```
kubectl create rolebinding super  
--serviceaccount=demo:sam -n demo --clusterrole=cr  
rolebinding.rbac.authorization.k8s.io/super created
```

Verify that SA sam can delete the deployment and not create it.

```
$ kubectl auth can-i create deployments --as  
system:serviceaccount:demo:sam -n demo  
no  
  
$ kubectl auth can-i delete deployments --as  
system:serviceaccount:demo:sam -n demo  
yes
```

Edit the ClusterRole cr to limit the SA sam to view only secrets

```
kubectl edit clusterrole cr  
clusterrole.rbac.authorization.k8s.io/cr edited
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: "2021-01-24T16:52:44Z"
  managedFields:
    - apiVersion: rbac.authorization.k8s.io/v1
      fieldsType: FieldsV1
      fieldsV1:
        f:rules: {}
      manager: kubectl-create
      operation: Update
      time: "2021-01-24T16:52:44Z"
  name: cr
  resourceVersion: "509"
  uid: f87c871e-0c16-4951-a777-8caa6c52fe21
rules:
- apiGroups:
  - ""
  resources:
  - secrets
  - pods #REMOVE
  verbs:
  - get #REMOVE
  - list
  - watch #REMOVE
  - delete #REMOVE
- apiGroups: #REMOVE
  - apps #REMOVE
  Resources: #REMOVE
  - deployments #REMOVE
  verbs: #REMOVE
  - get #REMOVE
  - list #REMOVE
  - watch #REMOVE
  - delete #REMOVE
```

Verify using `kubectl auth can-i`

```
kubectl auth can-i create deploy --as system:serviceaccount:demo:sam
-n demo
no
kubectl auth can-i create secrets --as system:serviceaccount:demo:sam
-n demo
no
kubectl auth can-i list secrets --as system:serviceaccount:demo:sam -n
demo
yes
```

New role and rolebinding creation

```
kubectl create role rr --verb=create
--resource=deployments -n demo
role.rbac.authorization.k8s.io/rr created
kubectl create rolebinding limited
--serviceaccount=demo:sam --role=rr -n demo
rolebinding.rbac.authorization.k8s.io/limited created
```

```
kubectl auth can-i create deploy --as system:serviceaccount:demo:sam
-n demo
yes
```

There can be various scenarios based on RBAC where you can be asked to create a new service account in a namespace, different role and role bindings, change the service account of a running pod etc.



## Problem 5: Image Scanning

- Setup the cluster :
  - Have a cluster and on the node, install one of the image scanning tools available. Since trivy docs are accessible during the exam so install trivy.
  - Create 3 pods with different images - nginx, alpine, httpd
- **Scan all the images for the pods present and save the name of the pods that have high or critical vulnerability to a file /opt/badimages.txt on the node.**

### Solution 5:

There are a lot of image scanning tools available out there that you can install and check images for any vulnerabilities. You can have these tools as part of the CI pipeline so that the pods creation is ignored if there is any vulnerability in the images. For this scenario I will use an open source image scanning tool called trivy.

I am using k3s cluster that can be simply setup by command

```
curl -sL https://get.k3s.io |INSTALL_K3S_VERSION="v1.22.5+k3s1" sh -
```

### Trivy Installation

```
curl -sL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh |sh -s -- -b /usr/local/bin

aquasecurity/trivy info checking GitHub for latest tag
aquasecurity/trivy info found version: 0.15.0 for v0.15.0/Linux/64bit
aquasecurity/trivy info installed /usr/local/bin/trivy
```

### Create the pods

```
kubectl run p1 --image=nginx
kubectl run p2 --image=httpd
```

```
kubectl run p3 --image=alpine -- sleep 1000
```

Get list of images used by pods

```
kubectl get pods -o=jsonpath='{range
.items[*]}{"\n"}{.metadata.name}{":\t"}{range
.spec.containers[*]}{.image}{"", "}{end}{end}' |sort

p1:      nginx,
p2:      httpd,
p3:      alpine,
```

Scan the images using trivy

```
trivy image --severity HIGH,CRITICAL nginx
nginx (debian 10.7)
=====
Total: 30 (HIGH: 29, CRITICAL: 1)

trivy image --severity HIGH,CRITICAL httpd
httpd (debian 10.7)
=====
Total: 23 (HIGH: 23, CRITICAL: 0)

trivy image --severity HIGH,CRITICAL alpine
alpine (alpine 3.12.1)
=====
Total: 0 (HIGH: 0, CRITICAL: 0)
```

This shows that you have to store the names of the pod with image httpd and nginx which is p1 and p2 in this scenario.

```
echo p1 $'\n'p2 > /opt/badimages.txt
```

There can be similar scenarios where you have to scan images and take respective actions based on the severity.

## Problem 6 - Audit Policy

- Enable Audit Policy on a Kubernetes cluster.
- Create a policy at `/etc/kubernetes/audit/policy.yaml` (Use from the kubernetes docs)
- Edit the policy to log deployments changes instead of pods at RequestResponse Level.

### Solution 6:

Auditing in Kubernetes provides a record of what's happening inside the cluster at different levels. Different Kubernetes objects can be logged at different levels to provide you the information or security breach if it happens. From the docs Auditing helps you with the following questions.

- what happened?
- when did it happen?
- who initiated it?
- on what did it happen?
- where was it observed?
- from where was it initiated?
- to where was it going?

For this scenario I will use the playground I created:

<https://www.katacoda.com/pathaksaiyam/scenarios/kube122>

Edit the `kube-apiserver.yaml` present at `/etc/kubernetes/manifests` and add below lines

```
spec:
  containers:
  - command:
    - kube-apiserver
    - --audit-policy-file=/etc/kubernetes/audit/policy.yaml
    - --audit-log-path=/etc/kubernetes/audit/logs/audit.log
    - --audit-log-maxsize=3
    - --audit-log-maxbackup=2
```

In the same file add the volume mount and the volumes for the same.

```
volumes:
  - name: audit-log
    hostPath:
      path: /etc/kubernetes/audit/logs/audit.log
      type: FileOrCreate
  - name: audit
    hostPath:
      path: /etc/kubernetes/audit/policy.yaml
      type: File
```

```
volumeMounts:
  - mountPath: /etc/kubernetes/audit/policy.yaml
    name: audit
    readOnly: true
  - mountPath: /etc/kubernetes/audit/logs/audit.log
    name: audit-log
    readOnly: false
```

Now that the apiserver editing is done, create the audit policy(`/etc/kubernetes/audit/policy.yaml`) from kubernetes documentation.

```
apiVersion: audit.k8s.io/v1 # This is required.
kind: Policy
# Don't generate audit events for all requests in
RequestReceived stage.
omitStages:
  - "RequestReceived"
rules:
  # Log pod changes at RequestResponse level
  - level: RequestResponse
    resources:
      - group: ""
        # Resource "pods" doesn't match requests to any
```

```

subresource of pods,
    # which is consistent with the RBAC policy.
    resources: ["pods"]
# Log "pods/log", "pods/status" at Metadata level
- level: Metadata
  resources:
    - group: ""
      resources: ["pods/log", "pods/status"]

# Don't log requests to a configmap called
"controller-leader"
- level: None
  resources:
    - group: ""
      resources: ["configmaps"]
      resourceNames: ["controller-leader"]

# Don't log watch requests by the "system:kube-proxy"
on endpoints or services
- level: None
  users: ["system:kube-proxy"]
  verbs: ["watch"]
  resources:
    - group: "" # core API group
      resources: ["endpoints", "services"]

# Don't log authenticated requests to certain
non-resource URL paths.
- level: None
  userGroups: ["system:authenticated"]
  nonResourceURLs:
    - "/api*" # Wildcard matching.
    - "/version"

# Log the request body of configmap changes in
kube-system.

```

```
- level: Request
  resources:
    - group: "" # core API group
      resources: ["configmaps"]
    # This rule only applies to resources in the
    "kube-system" namespace.
    # The empty string "" can be used to select
    non-namespaced resources.
    namespaces: ["kube-system"]

# Log configmap and secret changes in all other
namespaces at the Metadata level.
- level: Metadata
  resources:
    - group: "" # core API group
      resources: ["secrets", "configmaps"]

# Log all other resources in core and extensions at the
Request level.
- level: Request
  resources:
    - group: "" # core API group
    - group: "extensions" # Version of group should NOT
    be included.

# A catch-all rule to log all other requests at the
Metadata level.
- level: Metadata
  # Long-running requests like watches that fall under
  this rule will not
  # generate an audit event in RequestReceived.
  omitStages:
    - "RequestReceived"
```

Once the apiserver pod is restarted you will be able to see the logs getting generated in the audit.log file.

```
{"kind": "Event", "apiVersion": "audit.k8s.io/v1", "level": "Request", "auditID": "4f7cef69-59c0-451f-aa97-40be7027ebb2", "stage": "ResponseComplete", "requestURI": "/api/v1/namespaces/kube-system/configmaps/vip-configmap", "verb": "get", "user": {"username": "system:serviceaccount:kube-system:default", "uid": "6d7ef50e-dcfa-4069-ba9a-667a83cacc5e", "groups": ["system:serviceaccounts", "system:serviceaccounts:kube-system", "system:authenticated"]}, "sourceIPs": ["172.17.0.64"], "userAgent": "kube-keepalived-vip/v0.0.0 (linux/amd64) kubernetes/$Format", "objectRef": {"resource": "configmaps", "namespace": "kube-system", "name": "vip-configmap", "apiVersion": "v1"}, "responseStatus": {"metadata": {}, "code": 200}, "requestReceivedTimestamp": "2021-01-25T06:55:00.680748Z", "stageTimestamp": "2021-01-25T06:55:00.683244Z", "annotations": {"authorization.k8s.io/decision": "allow", "authorization.k8s.io/reason": "RBAC: allowed by ClusterRoleBinding \"/>

```

Now edit the policy and change the rule to log deployments changes at RequestResponse Level and restart apiserver pod.

```
- level: RequestResponse
  resources:
    - group: ""
      resources: ["deployments"]
```

There can be different scenarios where you are asked to log different Kubernetes objects at different Levels which should just be editing the policy and restarting the apiserver.

## Problem 7 : Kubernetes Upgrade

Upgrade the kubernetes version 1.18 to 1.19

### Solution 7:

Note - This works as it is if you try to upgrade from 1.22 to 1.23 in my Katacoda playground

In this scenario I will be using the katacoda Kubernetes cluster and perform an upgrade on it.

NAME	STATUS	ROLES	AGE	VERSION
controlplane	Ready	master	2m53s	v1.18.0
node01	Ready	<none>	2m21s	v1.18.0

```
Cordon and drain the controlplane node
kubectl cordon controlplane
node/controlplane cordoned

kubectl drain controlplane --ignore-daemonsets
node/controlplane already cordoned
WARNING: ignoring DaemonSet-managed Pods:
kube-system/kube-flannel-ds-amd64-zmxfj,
kube-system/kube-proxy-hgrjx
evicting pod kube-system/coredns-66bff467f8-4g989
evicting pod kube-system/coredns-66bff467f8-xwq5m
pod/coredns-66bff467f8-xwq5m evicted
pod/coredns-66bff467f8-4g989 evicted
node/controlplane evicted
```

Install kubeadm 1.19.0

```
apt-get install kubeadm=1.19.0-00
Reading package lists... Done
Building dependency tree
```



```
Reading state information... Done
The following packages were automatically installed and
are no longer required:
  libc-ares2 libhttp-parser2.7.1 libnetplan0 libuv1
nodejs-doc python3-netifaces
Use 'apt autoremove' to remove them.
The following additional packages will be installed:
  kubernetes-cni
The following packages will be upgraded:
  kubeadm kubernetes-cni
2 upgraded, 0 newly installed, 0 to remove apt-get
install kubeadm=1.19.0-00
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and
are no longer required:
  libc-ares2 libhttp-parser2.7.1 libnetplan0 libuv1
nodejs-doc python3-netifaces
Use 'apt autoremove' to remove them.
The following additional packages will be installed:
  kubernetes-cni
The following packages will be upgraded:
  kubeadm kubernetes-cni
2 upgraded, 0 newly installed, 0 to remove and 86 not
upgraded.
Need to get 32.8 MB of archives.
After this operation, 21.3 MB of additional disk space
will be used.
Do you want to continue? [Y/n] y
Get:1 https://packages.cloud.google.com/apt
kubernetes-xenial/main amd64 kubernetes-cni amd64
0.8.7-00 [25.0 MB]
Get:2 https://packages.cloud.google.com/apt
kubernetes-xenial/main amd64 kubeadm amd64 1.19.0-00
[7,759 kB]
```

```

Fetched 32.8 MB in 4s (9,308 kB/s)
(Reading database ... 145433 files and directories
currently installed.)
Preparing to unpack .../kubernetes-cni_0.8.7-00_amd64.deb
...
Unpacking kubernetes-cni (0.8.7-00) over (0.7.5-00) ...
Preparing to unpack .../kubeadm_1.19.0-00_amd64.deb ...
Unpacking kubeadm (1.19.0-00) over (1.18.0-00) ...
Setting up kubernetes-cni (0.8.7-00) ...
Setting up kubeadm (1.19.0-00) ... 86 not upgraded.
Need to get 32.8 MB of archives.
After this operation, 21.3 MB of additional disk space
will be used.
Do you want to continue? [Y/n] y
Get:1 https://packages.cloud.google.com/apt
kubernetes-xenial/main amd64 kubernetes-cni amd64
0.8.7-00 [25.0 MB]
Get:2 https://packages.cloud.google.com/apt
kubernetes-xenial/main amd64 kubeadm amd64 1.19.0-00
[7,759 kB]
Fetched 32.8 MB in 4s (9,308 kB/s)
(Reading database ... 145433 files and directories
currently installed.)
Preparing to unpack .../kubernetes-cni_0.8.7-00_amd64.deb
...
Unpacking kubernetes-cni (0.8.7-00) over (0.7.5-00) ...
Preparing to unpack .../kubeadm_1.19.0-00_amd64.deb ...
Unpacking kubeadm (1.19.0-00) over (1.18.0-00) ...
Setting up kubernetes-cni (0.8.7-00) ...
Setting up kubeadm (1.19.0-00) ...

```

```

kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.0",
GitCommit:"e19964183377d0ec2052d1f1fa930c4d7575bd50",

```

```
GitTreeState:"clean", BuildDate:"2020-08-26T14:28:32Z",
GoVersion:"go1.15", Compiler:"gc",
Platform:"linux/amd64"}
```

Apply the upgrade:

```
kubeadm upgrade apply v1.19.0
[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
-oyaml'
[preflight] Running pre-flight checks.
[upgrade] Running cluster health checks
[upgrade/version] You have chosen to change the cluster
version to "v1.19.0"
[upgrade/versions] Cluster version: v1.18.0
[upgrade/versions] kubeadm version: v1.19.0
[upgrade/confirm] Are you sure you want to proceed with
the upgrade? [y/N]:
```

After you say yes it will take little time to upgrade and in the end you should see something like

```
[upgrade/successful] SUCCESS! Your cluster was upgraded
to "v1.19.0". Enjoy!
```

Update kubelet and kubectl:

```
apt-get install kubelet=1.19.0-00 kubectl=1.19.0-00
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

The following packages were automatically installed and are no longer required:

libc-ares2 libhttp-parser2.7.1 libnetplan0 libuv1  
nodejs-doc python3-netifaces

Use 'apt autoremove' to remove them.

The following packages will be upgraded:

kubect1 kubelet

2 upgraded, 0 newly installed, 0 to remove and 85 not upgraded.

Need to get 26.6 MB of archives.

After this operation, 4,301 kB disk space will be freed.

Get:1 <https://packages.cloud.google.com/apt>

kubernetes-xenial/main amd64 kubect1 amd64 1.19.0-00  
[8,349 kB]

Get:2 <https://packages.cloud.google.com/apt>

kubernetes-xenial/main amd64 kubelet amd64 1.19.0-00  
[18.2 MB]

Fetched 26.6 MB in 6s (4,672 kB/s)

(Reading database ... 145436 files and directories  
currently installed.)

Preparing to unpack .../kubect1\_1.19.0-00\_amd64.deb ...

Unpacking kubect1 (1.19.0-00) over (1.18.0-00) ...

Preparing to unpack .../kubelet\_1.19.0-00\_amd64.deb ...

Unpacking kubelet (1.19.0-00) over (1.18.0-00) ...

Setting up kubelet (1.19.0-00) ...

Setting up kubect1 (1.19.0-00) ...

Restart Kubelet and uncordon node

```
sudo systemctl daemon-reload  
sudo systemctl restart kubelet
```

```
kubect1 uncordon controlplane  
node/controlplane uncordoned
```

Now lets upgrade the worker node using similar approach

First cordon and drain the node from master

```
kubectl cordon node01
node/node01 cordoned

kubectl drain node01 --ignore-daemonsets
node/node01 already cordoned
WARNING: ignoring DaemonSet-managed Pods:
kube-system/kube-flannel-ds-amd64-vtvgm,
kube-system/kube-keepalived-vip-qsnvp,
kube-system/kube-proxy-hvnjf
evicting pod kube-system/coredns-f9fd979d6-p9sdz
evicting pod kube-system/coredns-f9fd979d6-rqq7t
evicting pod
kube-system/katacoda-cloud-provider-669f978bb6-mmnt5
pod/katacoda-cloud-provider-669f978bb6-mmnt5 evicted
pod/coredns-f9fd979d6-rqq7t evicted
pod/coredns-f9fd979d6-p9sdz evicted
node/node01 evicted
```

Login to the node and run the following commands.

```
apt-get update && apt-get install -y kubeadm=1.19.0-00

kubeadm upgrade node
[upgrade] Reading configuration from the cluster...
[upgrade] FYI: You can look at this config file with
'kubectl -n kube-system get cm kubeadm-config -oyaml'
[preflight] Running pre-flight checks
[preflight] Skipping prepull. Not a control plane node.
[upgrade] Skipping phase. Not a control plane node.
[kubelet-start] Writing kubelet configuration to file
"/var/lib/kubelet/config.yaml"
[upgrade] The configuration for this node was
```

```
successfully updated!  
[upgrade] Now you should go ahead and upgrade the kubelet  
package using your package manager.
```

## Restart Kubelet

```
apt-get install -y kubelet=1.19.0-00 kubect1=1.19.0-00  
  
sudo systemctl daemon-reload  
sudo systemctl restart kubelet
```

## Uncordon the node from Master

```
kubect1 uncordon node01  
node/node01 uncordoned  
  
kubect1 get nodes  
NAME             STATUS    ROLES    AGE    VERSION  
controlplane     Ready    master   20m    v1.19.0  
node01           Ready    <none>   20m    v1.19.0
```

Now that you have seen the upgrade to 1.19, perform the upgrade on my Kubernetes 1.22 katakoda playground from 1.22 to 1.23 (<https://www.katacoda.com/pathaksaiyam/scenarios/kube122>)

## Problem 8 - CIS Benchmark

- Cluster setup:  
Install Kube-bench on the node
- Perform CIS benchmark on the cluster
- Check for various tests performed on master and worker nodes and see for the failed checks.
- Make changes to fix any issue related to PodSecurityPolicy or profiling in controllermanager

### Solution 8:

CIS(Center of internete security) Benchmarks for Kubernetes are the guidelines that includes the best practices of of securely configuring your kubernetes cluster. There are various tools out there that can do CIS benchmark scanning against your cluster and let you know the common vulnerabilities in your cluster and what you should do in order to follow the best practices.

Kube-bench is one such tool that gives a summary view as well as what exactly needs to be done in order to do the right configurations. Now you do not exactly need to run Kube-bench but you need to understand how it works and how you can check/fix issues.

For this scenario I am using a regular katacoda Kubernetes playground. You can use my playground as well and see the differences and try to fix broken stuff.

Run the CIS benchmark scan for master

```
docker run --pid=host -v /etc:/etc:ro -v /var:/var:ro -v  
$(which kubectl):/usr/local/mount-from-host/bin/kubectl  
-v ~/.kube:/.kube -e KUBECONFIG=/.kube/config -t  
aquasec/kube-bench:latest master
```

```
== Summary master ==  
45 checks PASS  
10 checks FAIL  
10 checks WARN
```

0 checks INFO

As you can see there are 10 Failed Checks. I would recommend here to read all the checks that are performed so that you have a general idea of what are the checks that it is doing and there can be questions in the exam related to any check where you might get asked to change the configurations wrt particular file.

After performing the kube-bench operations against the cluster you will not only get the results but also will get the **Remediations** that can be applied to PASS the failed checks.

Once you see the checks you will something like the following

```
[PASS] 1.2.28 Ensure that the --service-account-key-file argument is set as appropriate (Automated)
[PASS] 1.2.29 Ensure that the --etcd-certfile and --etcd-keyfile arguments are set as appropriate (Automated)
[PASS] 1.2.30 Ensure that the --tls-cert-file and --tls-private-key-file arguments are set as appropriate (Automated)
[PASS] 1.2.31 Ensure that the --client-ca-file argument is set as appropriate (Automated)
[PASS] 1.2.32 Ensure that the --etcd-cafile argument is set as appropriate (Automated)
[WARN] 1.2.33 Ensure that the --encryption-provider-config argument is set as appropriate (Manual)
[WARN] 1.2.34 Ensure that encryption providers are appropriately configured (Manual)
[WARN] 1.2.35 Ensure that the API Server only makes use of Strong Cryptographic Ciphers (Manual)
[INFO] 1.3 Controller Manager
[WARN] 1.3.1 Ensure that the --terminated-pod-gc-threshold argument is set as appropriate (Manual)
[FAIL] 1.3.2 Ensure that the --profiling argument is set to false (Automated)
[PASS] 1.3.3 Ensure that the --use-service-account-credentials argument is set to true (Automated)
[PASS] 1.3.4 Ensure that the --service-account-private-key-file argument is set as appropriate (Automated)
[PASS] 1.3.5 Ensure that the --root-ca-file argument is set as appropriate (Automated)
[PASS] 1.3.6 Ensure that the RotateKubeletServerCertificate argument is set to true (Automated)
[PASS] 1.3.7 Ensure that the --bind-address argument is set to 127.0.0.1 (Automated)
[INFO] 1.4 Scheduler
[FAIL] 1.4.1 Ensure that the --profiling argument is set to false (Automated)
[PASS] 1.4.2 Ensure that the --bind-address argument is set to 127.0.0.1 (Automated)
```

For PodSecurityPolicy you will see

```
[FAIL] 1.2.16 Ensure that the admission control plugin
PodSecurityPolicy is set (Automated)
```

For controller manager profiling you will see

```
[FAIL] 1.3.2 Ensure that the --profiling argument is set
to false (Automated)
```

Let's do the remediation steps



```
1.2.16 Follow the documentation and create Pod Security
Policy objects as per your environment.
Then, edit the API server pod specification file
/etc/kubernetes/manifests/kube-apiserver.yaml
on the master node and set the --enable-admission-plugins
parameter to a
value that includes PodSecurityPolicy:
--enable-admission-plugins=...,PodSecurityPolicy,...
Then restart the API Server.
1.3.2 Edit the Controller Manager pod specification file
/etc/kubernetes/manifests/kube-controller-manager.yaml
on the master node and set the below parameter.
--profiling=false
```

Run the Kube-bench test again to see if two issues got fixed.

```
1.2.16 Follow the documentation and create Pod Security
Policy objects as per your environment.
Then, edit the API server pod specification file
/etc/kubernetes/manifests/kube-apiserver.yaml
on the master node and set the --enable-admission-plugins
parameter to a
value that includes PodSecurityPolicy:
--enable-admission-plugins=...,PodSecurityPolicy,...
Then restart the API Server.
1.3.2 Edit the Controller Manager pod specification file
/etc/kubernetes/manifests/kube-controller-manager.yaml
on the master node and set the below parameter.
--profiling=false
```

```
[PASS] 1.2.16 Ensure that the admission control plugin PodSecurityPolicy is set (Automated)
```

```
[PASS] 1.3.2 Ensure that the --profiling argument is set to false (Automated)
```

Similar to this you can run it against the node as well that will check worker node configuration and kubelet.

```
docker run --pid=host -v /etc:/etc:ro -v /var:/var:ro -v  
$(which kubectl):/usr/local/mount-from-host/bin/kubectl  
-v ~/.kube:/.kube -e KUBECONFIG=/.kube/config -t  
aquasec/kube-bench:latest node  
  
== Summary node ==  
19 checks PASS  
1 checks FAIL  
3 checks WARN  
0 checks INFO
```

Again read through all the checks that the kube-bench is performing against the kube-apiserver, controller manager, scheduler and then try to fix them by editing the yaml files at /etc/kubernetes/manifests and check separately for the kubelet as well.

**Note - the tool installation (kube-bench) in this case is not something you need to remember. For this scenario you run kube-bench as a docker container or any other way you wish to.**

**The main goal is to see what configurations in the cluster is wrong and how can you fix that and comply with CIS Benchmark**

## Problem 9 - Container Runtimes

- Create a cluster with Containerd as the container Engine
- Create a new RuntimeClass named demo with runc as handler
- Create a pod test with image nginx and runtimeClass as demo
- Check if the pod is in running status

### Solution 9 -

I will be using the k3s cluster that by default comes with containerd. Now you can also check out the kubernetes documentation to see how you can set up the kubernetes cluster with containerd and then complete the remaining steps.

Edit the containerd configuration to use runc

```
[plugins.cri.containerd.runtimes.runc]
  runtime_type = "io.containerd.runc.v2"
```

Create a RuntimeClass

```
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: demo
handler: runc
```

Create the pod that uses RuntimeClass demo

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  runtimeClassName: demo
  containers:
    - name: test
      image: nginx
```

Check the running status of the pod

```
kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
test	1/1	Running	0	3s

You can run the same scenario on my Katacoda playground which is Kubernetes 1.22 + containerd and you do not have to configure anything for containerd, just create the RuntimeClass and then use that class.

Try here - <https://www.katacoda.com/pathaksaiyam/scenarios/kube122>

## Problem 10 : Falco

- Cluster Setup:
  - Create a cluster and install falco on the node
  - Create a rule for logging a shell spawned inside the container
  - Change the output format for “terminal shell in a container” to [timestamp][ container id] [container name] and store it in a file /tmp/falcologs

## Solution 10 :

Falco is an open source runtime security tool. From the kernel Falco parses the Linux sys calls at runtime and outputs the logs based on the rules defined. By default it comes with a lot of preloaded rules that can be changed/added. Falco alerts can be triggered as well when the rule is violated. Let's Install falco inside a kubernetes cluster. I am using my Katacoda Kubernetes playground for this scenario.

```
$ helm repo add falcosecurity https://falcosecurity.github.io/charts
"falcosecurity" has been added to your repositories
$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "falcosecurity" chart repository
Update Complete. ✨ Happy Helming! ✨
$ helm install falco falcosecurity/falco
NAME: falco
LAST DEPLOYED: Wed Jan  5 02:32:41 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Falco agents are spinning up on each node in your cluster. After a few
seconds, they are going to start monitoring your containers looking for
security issues.
No further action should be required.
Tip:
You can easily forward Falco events to Slack, Kafka, AWS Lambda and more
with falcosekick.
```

```
kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
falco-nsqqf   1/1     Running   0           4m33s

kubectl get cm
NAME          DATA   AGE
falco         5       5m45s
```

If falco is installed directly on the node and not via helm then you can see the configuration files at `/etc/falco` directory

To install falco on the node you can simply follow the docs and run below commands. I am installing this on the controlplane node.

```
curl -s https://falco.org/repo/falcosecurity-3672BA8F.asc
| apt-key add -

echo "deb https://download.falco.org/packages/deb stable
main" | tee -a /etc/apt/sources.list.d/falcosecurity.list

apt-get update -y
apt-get -y install linux-headers-$(uname -r)
apt-get install -y falco
cd /etc/falco/
falco_rules.local.yaml falco_rules.yaml falco.yaml
k8s_audit_rules.yaml rules.available rules.d
```

If you want to add a new rule or change the existing one or change the output format of any particular rule you will either change the configmap if it is deployed as a helm chart within kubernetes or you will go in the `/etc/falco` repository to make the changes.

For this scenario I will change the `falco.yaml` and `falco_rules.yaml` from the `/etc/falco/` directory.

```
ls
falco_rules.local.yaml falco_rules.yaml falco.yaml
k8s_audit_rules.yaml rules.available rules.d
```

Check where the output is from the falco.yaml

If you see this file you can see from where the rules are loaded

```
rules_file:
  - /etc/falco/falco_rules.yaml
  - /etc/falco/falco_rules.local.yaml
  - /etc/falco/k8s_audit_rules.yaml
  - /etc/falco/rules.d
```

Now check the output section where falco is dumping all the logs

```
stdout_output:
  enabled: true
```

Means falco is writing to **syslog**

Start falco service

```
service falco start
cat /var/log/syslog | grep falco
```

Spawn a shell inside a container

```
docker run --name ubuntu_bash --rm -i -t ubuntu bash
root@46c66499e2ec:/# exit
exit
```

Check the falco logs

```
cat /var/log/syslog | grep falco | grep ubuntu

Jan  5 02:45:50 host01 falco[16991]: 02:45:50.856451527:
Notice A shell was spawned in a container with an
attached terminal (user=root user_loginuid=-1 ubuntu_bash
(id=2f85cd33f927) shell=bash parent=<NA> cmdline=bash
terminal=34816 container_id=2f85cd33f927 image=ubuntu)
```

Let's change the output to just display containerid, timestamp and container name.

For this you can head over to falco docs

<https://falco.org/docs/rules/supported-fields/> and find all the fields that can be used as an output

So change the output of the rule in the `falco_rules.yaml` to :

```
output: "[%evt.time][%container.id] [%container.name]"
```

```
- rule: Terminal shell in container
  desc: A shell was used as the entrypoint/exec point into a container with an attached terminal.
  condition: >
    spawned_process and container
    and shell_procs and proc.tty != 0
    and container_entrypoint
    and not user_expected_terminal_shell_in_container_conditions
  output: "%evt.time %container.id %container.name"
  priority: NOTICE
  tags: [container, shell, mitre_execution]
```

Now run the container with shell again and check the output

```
docker run --name demo --rm -i -t ubuntu bash
root@46c66499e2ec:/# exit
exit

cat /var/log/syslog | grep falco | grep demo
Jan  5 03:02:58 host01 falco[20849]: 03:02:58.063345725:
Notice 03:02:58.063345725 ffce831596b4 demo
Jan  5 03:03:00 host01 falco[20849]: 03:03:00.115143290:
Warning Shell history had been deleted or renamed (user=root
user_loginuid=-1 type=openat command=bash
fd.name=/root/.bash_history name=/root/.bash_history
path=<NA> oldpath=<NA> demo (id=ffce831596b4))
```

You can see the highlighted output and it's changed according to the requirement. Now store inside a file.

```
$ echo "[%03:02:58.063345725][ffce831596b4] [demo]" >> /tmp/falcologs
$ cat /tmp/falcologs
[03:02:58.063345725][ffce831596b4] [demo]
```

This is how you can play around with falco rules and output.



## Problem 11 - Secrets

- Create a secret named database with data **username=sammy** and **password=demo123** in it
- Mount the secret into a pod as a volume
- Retrieve the secret from the pod and store in a file /tmp/secret
- Retrieve the service account token and store in a file /tmp/token

### Solution 11:

Secrets in Kubernetes is a way to store sensitive data. If you need a database password to be used by a pod then you can have that as secret in the form of volume or env and read inside the container.

For this scenario create a generic secret

```
$ kubectl create secret generic database
--from-literal=username=sammy
--from-literal=password=demo123
secret/database created
$ kubectl get secrets
NAME                                TYPE
DATA    AGE
database 2      5s
Opaque
```

Create a pod with redis image and then mount the secret as a volume

```
apiVersion: v1
kind: Pod
metadata:
  name: demo
spec:
  containers:
  - name: mypod
    image: redis
```

```
volumeMounts:
  - name: sec-vol
    mountPath: "/etc/sec"
    readOnly: true
volumes:
  - name: sec-vol
    secret:
      secretName: database
```

Check the secrets stored in /etc/sec/

```
kubectl exec demo -- ls /etc/sec/
password
username
$ kubectl exec demo -- cat /etc/sec/username
Sammy
$ kubectl exec demo -- cat /etc/sec/password
demo123
```

Say this secret was already created and you need to retrieve the secret and store the value inside the file. You can do in two ways:

- First is to directly get the secret as a yaml file and then decode the secret and store it
- Second is to describe the pod, see the volume mount for the secret and get the secret from inside the pod.

Let's do both approaches :

**First Way:**

```
$ kubectl get secret database -oyaml

apiVersion: v1
data:
  password: ZGVtbzEyMw==
  username: c2FtbXk=
kind: Secret
metadata:
  creationTimestamp: "2022-01-05T03:11:29Z"
```

```

name: database
namespace: default
resourceVersion: "3536"
uid: 53e805e7-a7cd-4af1-968a-973a6b9922e9
type: Opaque

$ echo "ZGVtbzEyMw==" | base64 -d
demo123

$ echo demo123 >> /tmp/sec
$ echo "c2FtbXk=" | base64 -d
sammy

$ echo sammy >> /tmp/sec

$ cat /tmp/sec
demo123
sammy

```

### Second Way:

See the volume mount inside the container to see which directory the secrets are located or check if they are as env.

In this case it's mounted as volume in `/etc/sec/` directory

Describe the pod and see the Mounts section :

```

Mounts:
  /etc/sec from sec-vol (ro)

```

From inside the pod get the secrets and store in a file

```

$ kubectl exec demo -- cat /etc/sec/username >> /tmp/sec
$ kubectl exec demo -- cat /etc/sec/password >> /tmp/sec
$ cat /tmp/sec
sammydemo123controlplane

```

Last part of this scenario is to get the service account token from the pod

```
echo "$(kubectl exec -it demo -- cat
/run/secrets/kubernetes.io/serviceaccount/token)" > /tmp/token
cat /tmp/token
eyJhbGciOiJSUzI1NiIsImtpZCI6Ikp1NmVJVU1VQzg5cUVyZWUtUXY5e1ZRdFNs
a0tfdVd3UjNXRXRleWh2UncifQ.eyJpc3MiOiJrdWJ1cm5ldGVzL3N1cnZpY2VhY
2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWN1YWNjb3VudC9uYW1lc3BhY2UiO
iJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWN1YWNjb3VudC9zZWNyZXQub
mFtZSI6ImRlZmF1bHQtdG9rZW4tenQ2dnYiLCJrdWJ1cm5ldGVzLmlvL3N1cnZpY
2VhY2NvdW50L3N1cnZpY2UtYWNjb3VudC5uYW1lIjoizGVmYXVsdCIsImt1YmVybm
V0ZXMuaW8vc2Vydm1jZWZjY291bnQvc2Vydm1jZS1hY2NvdW50LnVpZCI6Ij1hM
WU3ZDhhLTRiYWQtNDcyZS1iOWM0LTAzMmU1Nzg5ZWZkNyIsInN1YiI6InN5c3R1b
TpozZXJ2aWN1YWNjb3VudDpkZWZhdWx0OmRlZmF1bHQifQ.JwAqknjRkpsgDFM8v1
3gQ0cNfnqYhDhjPgIIievqevmu3MCdILDSxfSMUAmhmmHj7gs8xDQzQBBxrC-yc1
91UeKRaiXe8oRTDfcciVstx5bBrNMxhA4fUFP439iCTgtZwOCyOZmXEmq208G10f
LDpv_eHNy8a1P6ogqilRDCKxKLpk2lM8GNGNu6L3EQWA3f2sInMmQQWzmmrwQFrI
bcvvyC972KyGd5fFPI0jlVjEMdgqWd7kMhxn-mcNuJONBxYoLzlAVmsc1R96VEb4
mIOVAtoLx1b00aLm3o3PYLf8US0c18EypLvNkALyDodUV7UBmEJdmV4N4_V5mrZ7
vfkA
```

## Problem 12: PodSecurity Policy

- Enable Pod Security policy on the cluster
- Create a PodSecurityPolicy demo to prevent the creation of all privileged pods.
- Create a Role and Role binding to use this policy and assign it to a new service account sam. Do this in a new test namespace.

### Solution 12:

Pod Security Policies enable fine-grained authorization of pod creation and updates. PodSecurityPolicy is deprecated in v1.21+ and will be unavailable in v1.25+ but it is still part of CKS certification, so keeping this scenario as it is.

I will be using my katacoda Kubernetes playground for this example and will be using the same example as mentioned in the kubernetes documentation

First enable the PodSecurityPolicy plugin by changing the `/etc/kubernetes/manifests/kube-apiserver.yaml` file .

```
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=172.17.0.39
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction,PodSecurityPolicy
```

Create a PodSecurityPolicy to deny privileged pods

Note - Before enabling the admission plugin make sure to have a default allow policy so that pods are not prevented

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
```

```

  name: demo
spec:
  privileged: false # Don't allow privileged pods!
  # The rest fills in some required fields.
  selinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  volumes:
  - '*'

```

Create service account in test namespace

```

kubectl create ns test
namespace/test created

kubectl create sa sam -n test
serviceaccount/sam created

```

Create a Role and RoleBinding and bind it to SA sam

```

kubectl create role psp --verb=use
--resource=podsecuritypolicy --resource-name=demo -n test
role.rbac.authorization.k8s.io/psp created

kubectl create -n test rolebinding psp-binding --role=psp
--serviceaccount=test:sam
rolebinding.rbac.authorization.k8s.io/psp-binding created

```

No there can be scenarios where you can be asked to create a specific policy and you can control all of the below

Running of privileged containers	<code>privileged</code>
Usage of host namespaces	<code>hostPID</code> , <code>hostIPC</code>
Usage of host networking and ports	<code>hostNetwork</code> , <code>hostPorts</code>
Usage of volume types	<code>volumes</code>
Usage of the host filesystem	<code>allowedHostPaths</code>
Allow specific FlexVolume drivers	<code>allowedFlexVolumes</code>
Allocating an FSGroup that owns the pod's volumes	<code>fsGroup</code>
Requiring the use of a read only root file system	<code>readOnlyRootFilesystem</code>
The user and group IDs of the container	<code>runAsUser</code> , <code>runAsGroup</code> , <code>supplementalGroups</code>
Restricting escalation to root privileges	<code>allowPrivilegeEscalation</code> , <code>defaultAllowPrivilegeEscalation</code>
Linux capabilities	<code>defaultAddCapabilities</code> , <code>requiredDropCapabilities</code> , <code>allowedCapabilities</code>
The SELinux context of the container	<code>seLinux</code>
The Allowed Proc Mount types for the container	<code>allowedProcMountTypes</code>
The AppArmor profile used by containers	<code>annotations</code>
The seccomp profile used by containers	<code>annotations</code>
The sysctl profile used by containers	<code>forbiddenSysctls</code> , <code>allowedUnsafeSysctls</code>

Bookmark this page -

<https://kubernetes.io/docs/concepts/policy/pod-security-policy/>

You can find the examples here that you can use directly in the exam terminal.

**Example**, if you are asked to create a PodSecurityPolicy that allows hostPath volumes only for directory /foo then you can use the below snippet.

```
You can add the following to the PodSecurityPolicy
allowedHostPaths:
  # This allows "/foo", "/foo/", "/foo/bar" etc., but
  # disallows "/fool", "/etc/foo" etc.
  # "/foo/.." is never valid.
  - pathPrefix: "/foo"
    readOnly: true # only allow read-only mounts
```

This snippet is directly taken from the Kubernetes documentation so make sure you understand the concepts and bookmark the page.

## Problem 13: Security Context for a Pod or Container

- Cluster setup
- Create a pod nginx with image nginx
- Create a Security context to enable readonlyfilesystem
- Check the pod status is running after applying the security context

### Solution 13:

Kubernetes Security Context can be applied at pod level or container level to define the access level and permissions for a pod/container at runtime. You can configure a wide variety of security contexts like readonly root filesystem, whether pods can run as privileged and others.

For this scenario I am using my Katacoda Kubernetes playground(<https://www.katacoda.com/pathaksaiyam/scenarios/kube122>)

Create the pod with below yaml file

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  containers:
  - name: nginx
    image: nginx
    securityContext:
      readOnlyRootFilesystem: true
  volumeMounts:
  - name: run
    mountPath: /var/run
  - name: log
    mountPath: /var/log/nginx
  - name: cache
    mountPath: /var/cache/nginx
```



```
volumes:
- name: run
  emptyDir: {}
- name: log
  emptyDir: {}
- name: cache
  emptyDir: {}
```

If you see the above yaml carefully there is a security context added at the container level that makes the root filesystem read only, so even if the attacker gets access to the container they will not be able to make changes on the root filesystem. But sometimes there are some directories needed by the container to cache the files and write some data to during its complete runtime. In that case different directories can be made writable by using emptyDir as a Volume and mount to the specific paths in the container.

Let's apply the above yaml file and create the pod

```
kubectl apply -f demo.yaml
pod/security-context-demo created
kubectl get pods
NAME                                READY   STATUS
RESTARTS   AGE
security-context-demo 1/1     Running
3s
```

Exec into the pod and try to create a file on the

```
kubectl exec security-context-demo -- touch /etc/test
touch: cannot touch '/etc/test': Read-only file system
command terminated with exit code 1
```

## Problem14: Privileged Pods

Above question leads to another type of scenario where you are given a Pod and it has security context and other privileges and you are asked to

- Remove the pods that are running as privileged
- Remove pods that can write inside the container

### Solution 14:

There can be various ways in which the above can be configured and you need to know all the ways it can be done. One of the examples is

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 0
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: true
```

It is running as root user and privileged escalation is true so these ones need to be removed

## Problem 15: Dockerfile best practices and Deployment best practices

- Let's say you are given a Dockerfile and deployment file, you need to look at the instructions given in each of the files carefully, find the ones that are not right and remove the pods that use those.

### Solution 15:

There can be a lot of scenarios in this itself

- If you have a privileged user used in a deployment then you would want to remove it
- If you have secrets exposed in deployment you would want to remove it
- If you have a Dockerfile with secrets exposed or copied directly inside the Dockerfile, you might want to remove it.
- If you have bad security contexts for the pods, you might want to remove it.
- There can also be scenarios where you might be asked to edit the files to make it according to the security best practices.

```
FROM alpine
COPY demo.sh .
COPY secret .
USER root
CMD["demo.sh","./secret"]
```

Above you can see that the secret is directly copied into the Dockerfile which is not good.

[https://docs.docker.com/develop/develop-images/dockerfile\\_best-practices/](https://docs.docker.com/develop/develop-images/dockerfile_best-practices/)

## Problem 16: ImagePolicyWebhook

- Create ImagePolicyWebhook Admission-Controller-Plugin.
- You are required to enable ImagePolicyWebhook, create the configuration JSON for the webhook, set up the required files and create a demo pod to test.

### Solution 16 :

For this scenario I am using Katacoda Kubernetes playground

First clone the repository

```
Git clone https://github.com/saiyam1814/imagepolicy.git
```

Create a folder demo and copy contents

```
$ mkdir /etc/kubernetes/demo
$ cp -r imagepolicy/ /etc/kubernetes/demo
$ cd /etc/kubernetes/demo
$ ls
admission.json      api-client-key.pem  LICENSE
webhook.pem
api-client-cert.pem config              README.md
```

Now the admission.json file looks like below :

```
{
  "imagePolicy": {
    "kubeConfigFile": "/etc/kubernetes/demo/config",
    "allowTTL": 50,
    "denyTTL": 50,
    "retryBackoff": 500,
    "defaultAllow": false
  }
}
```

defaultAllow: false means that if the external service is not reachable then the pods won't be allowed to create.

The config file as below :

```
apiVersion: v1
kind: Config
clusters:
- cluster:
    certificate-authority:
/etc/kubernetes/demo/webhook.pem
    server: https://service-check:8888/check-image
    name: check-image
users:
- name: api-server
  user:
    client-certificate:
/etc/kubernetes/demo/api-client-cert.pem
    client-key: /etc/kubernetes/demo/api-client-key.pem
contexts:
- context:
    cluster: check-image
    user: api-server
    name: check-image
current-context: check-image
preferences: {}
```

Here the server is the Image checking service and rest are the CA for the service and the certs.

Now edit the file

```
vi /etc/kubernetes/manifests/kube-apiserver.yaml
```

Add the configuration (admission.json) and enable the plugin.

```
spec:
  containers:
  - command:
    - kube-apiserver
```

```
--admission-control-config-file=/etc/kubernetes/demo/admission.json
--enable-admission-plugins=NodeRestriction,ImagePolicyWebhook
```

Also add the hostpath volume and volume mount

```
volumeMounts:
- mountPath: /etc/kubernetes/demo
  name: admission
  readOnly: true

volumes:
- hostPath:
    path: /etc/kubernetes/demo
    name: admission
```

Once the apiserver is restarted you can create a pod and test that it's not getting created

```
kubect1 run nginx --image=nginx
Error from server (Forbidden): pods "nginx" is forbidden:
Post https://service-check:8888/check-image?timeout=30s: dial
tcp: lookup service-check on 8.8.8.8:53: no such host
```

**Note** - The CA and the certs used in this demo are randomly generated and you can use your own. Also the server I am pointing to is dummy so this will never be reachable and you won't be able to create the pod. The main aim is to see how to enable ImagePolicyWebhook, how to use it's configurations, and what all files are needed.

If you want to try out a real server check this ->

<https://github.com/flavio/kube-image-bouncer>

# From Where To Learn CKS Concepts and Bookmarks

**Killer.sh** CKS practice is like a real exam-like environment. + Their Udemey course as well  
**Walid Shaari GtiHub Repo** - Has lot of learning material  
<https://github.com/walidshaari/Certified-Kubernetes-Security-Specialist>

**My Bookmarks : These bookmarks are not mandatory but useful and all of them refer to a concept or an example.**

<https://kubernetes.io/docs/tasks/administer-cluster/declare-network-policy/>  
<https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/#restricting-network-access>  
<https://kubernetes.io/blog/2017/10/enforcing-network-policies-in-kubernetes/>  
<https://github.com/kubernetes/ingress-nginx/blob/master/docs/user-guide/tls.md>  
<https://kubernetes.io/docs/reference/generated/kubect/kubectl-commands#-em-secret-tls-em->  
<https://kubernetes.io/docs/tasks/administer-cluster/kubelet-config-file/>  
<https://kubernetes.io/docs/tutorials/clusters/apparmor/#pod-annotation>  
<https://kubernetes.io/docs/tutorials/clusters/seccomp/>  
<https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/>  
<https://kubernetes.io/docs/concepts/policy/pod-security-policy/>  
<https://falco.org/docs/rules/supported-fields/>  
<https://kubernetes.io/blog/2019/08/06/opa-gatekeeper-policy-and-governance-for-kubernetes>  
<https://kubernetes.io/docs/concepts/containers/runtime-class/>  
<https://kubernetes.io/docs/tasks/debug-application-cluster/audit/>  
<https://v1-19.docs.kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/>  
<https://docs.sysdig.com/>  
<https://github.com/aquasecurity/trivy>  
<https://gitlab.com/apparmor/apparmor/-/wikis/Documentation>  
<https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#imagepolicywebhook>  
<https://kubernetes.io/blog/2019/03/21/a-guide-to-kubernetes-admission-controllers/>  
<https://kubernetes.io/docs/reference/access-authn-authz/rbac/>  
<https://kubernetes.io/docs/tutorials/clusters/apparmor/#example>  
<https://kubernetes.io/docs/concepts/policy/pod-security-policy/#create-a-policy-and-a-pod>  
<https://kubernetes.io/docs/tasks/access-application-cluster/list-all-running-container-images/>  
<https://docs.sysdig.com/en/kubernetes-audit-logging.html>  
<https://kubernetes.io/docs/concepts/services-networking/network-policies/#default-deny-all-egress-traffic>

**Thank you for Purchasing the book and All the Best for your CKS Exam!!**