Published in Credera Engineering

Abdurahman Abukar Follow
Jan 23 · 15 min read · ✦ · Listen

A guide to passing the Kubernetes Trifecta in 40 days

Introduction

Having completed all three Kubernetes practical certifications, I have received a lot of questions asking me how I was able to pass all three hands-on exams in a matter of 5–6 weeks. It even came as a surprise to me, given that the only thing holding me back was the worry of being underprepared.

Before starting my Kubernetes journey, I looked around for advice and was bombarded with comments on how tough the exams were — to the point where it

almost made me consider learning an easier technology. However, I came across one comment in particular that surpassed all others and inspired me to take it on:

> *"An exam's difficulty is determined by how much time you spend preparing for it".*

It made sense to me. All I had to do was organise my time, account for how many hours the course would take me, and determine how long I would spend on the labs (including reattempting them).

So, how long does it take to complete the trifecta?

As mentioned in my previous article, my go-to instructor for Kubernetes is Mumshad Mannambeth. Based on my own experience, let me give you an overall understanding of how many hours you will be spending on the course based on the material.

- **CKA:** The CKA course was 18.5 hours in video time. Double this to include labs for each section, assuming that the lab takes you the same length of time as the video for it. Now add 20 hours of real-time (no breaks included) for the mock exams, attempts, and reattempts, until you get 100% (regardless of how many tries it takes you to get there). The mock labs are short — each at 1 hour long - and only include 8-9 questions. That is a total of 57 hours of study time
- **CKAD:** Having done CKA, you are now warmed up with enough hands-on experience. So all you really need to do at this point is complete the course and do the labs 1.5x faster than CKA. Crunching the numbers, that is 57/1.5 = 38 hours
- **CKS:** Now, CKS includes new software which needs to be covered. So this will be a repeat of CKA and no longer because you are still warmed up, and no shorter because the topics being covered require more time. As a rule of thumb, when covering new topics, the length it takes you to cover the videos is roughly the same as the time it will take to do the labs accompanying it. Total = 57 hours.
- To put things into perspective, I watched the videos at a speed of 1.5x the original — **Total hours required = 152 hours**
- So use this equation to figure out how long it will take you:

```
152 hours / hours (per week) = Number of weeks to get the Trifecta
```

- For example, if you have **20 hours** to study Kubernetes per week (~3 hours a day), you will become **CKA, CKAD, and CKS certified in 152 / 20 = ~ 8 weeks**

Pre-requisites

The figures above are applicable to those who are new to Kubernetes, but not for those who are new to DevOps altogether. It is necessary to have the below prerequisite knowledge before starting the journey to Trifecta:

- **Linux Fundamentals:** A basic understanding of Linux commands are a must. Commands like — ls (listing the current directory, or an external directory you pass), cd (change directory), grep, absolute path, .. (double-dot) for the parent directory, mkdir (creating directories), cat (viewing files), and other more basic commands.
- **VIM:** A text editor that you will be using often throughout the course of your Kubernetes studies.
- Awareness of **networking and services**
- Awareness of **apiserver in Kubernetes**
- **Docker fundamentals:** Basic docker commands, creating stand-alone containers using docker, etc.
- The further you go into the course, the more grounding you should have on the fundamentals of Kubernetes. For example, **CKS requires a solid grounding in Kubernetes fundamentals.**
- Good familiarity with writing **YAML files**, with the correct **indentation.**
- Although YAML is important to know, **efficiency is even more important** — so really simplifying the creation of resources using **imperative commands** and **dry-runs** saves you lots of time (and time is vital in the exam).

Understanding the essentials

Now that we have the prereqs out of the way, let's dive a little deeper into what is required for the exams. As previously mentioned, efficiency is key, so knowing all the tips and tricks to save time in the exam is of utmost importance:

- Using **k=kubectl**, instead of writing the whole command.

```
alias k=kubectl # To set up the alias
```

- Using — dry-run=client -o YAML to create a skeleton YAML file if you aren't able to pass all flags on the command line imperatively.

```
> export do="--dry-run=client -o yaml"

# To call on the export, use $do

> k run test --image nginx $do > test.yaml
> vi test.yaml
```

- When deleting object-like pods, Kubernetes allows for graceful termination when pods get deleted. This is by default set to 30 seconds, so we have to manually set this grace period to 0 when deleting the pods. The easiest way to do this is by using an export.

```
> export now="--force --grace-period=0"

# To call on this export, use $now

# Create the pod
> k run test --image nginx

# Delete the pod with 0 grace period
> k delete test $now
```

- As you go through Mumshad's course and labs, you will find yourself navigating the documentation — this is a process that can only be improved with repetition. Alternatively, having the pages you need to be bookmarked for easy access is a good approach.

- Do not spend too long on a question, especially if the percentage weight for the question is 4% or less. Too long = >5 minutes.
- Most objects have a shorthand notation, so practice using these beforehand:

```
po - PODs
rs - ReplicaSets
deploy - Deployments
svc - Services
ns - Namespaces
netpol - Network Policies
pv - Persistent Volumes
pvc - Persistent Volume Claims
sa - Service Accounts
```

CKA (Certified Kubernetes Administrator):

For a detailed walkthrough on getting the CKA, read my previous article on **"The Ultimate Guide to becoming Kubernetes Certified"**

The main focus of this exam is the configuration of objects, maintenance tasks like etcd backup and restore, upgrading the cluster to a newer version, marking nodes as un-schedulable (cordoning), and removing pods from nodes (draining). An administrator in the exam and Linux Foundation terms is someone who has:

> *"demonstrated proficiency in Application Lifecycle management, Installation, Configuration & Validation, Core concepts, Networking, Scheduling, Security, Cluster Maintenance, Logging / Monitoring, Storage, and Troubleshooting"*

Here are some tips specifically for questions in the exam:

- **Linux**: As you are required to work on the terminal, knowing key Linux commands and understanding services is a must. Being able to configure services and interacting with them (start, stop, disable, etc) is also vital.
- **Troubleshooting key components**: When asked to troubleshoot a worker node that is not working, you should know that it is the kubelet in the node that needs restarting. You should also understand that to restart the kubelet, you have to use the 'systemctl' tool to interact with it, and any other services, for that matter. Here's how:

```
> ssh node1
-------SSH message-------

node1 > systemctl restart kubelet
node1 > systemctl daemon-reload
node1 > exit

------Exit message-------

> k get nodes

# Node is schedulable again
```

- **ETCD Backup and Restore:** This is much more straightforward than it is made out to be. A thorough read of the ETCD documentation page is sufficient. As shown in the command below, you would just replace the generic files with the

```
> ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \\
  --cacert=<trusted-ca-file> --cert=<cert-file> --key=<key-file> \\
  snapshot save <backup-file-location>
```

- **Network Policies:** Lots of people tend to dread this topic. But it is as easy as copying the skeleton file from the **Network Policies documentation page,** then removing and replacing the file with what you need. Most of the correct indentation is done for you. Understanding them is important and repeats on all three of the exams. The question will give you all the detail that is needed, so understand which policyType and selector you will be using.

```
> vi np1.yaml
---------------------
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
```

```
  - from:
    - ipBlock:                          # delete
        cidr: 172.17.0.0/16             # delete
        except:                         # delete
        - 172.17.1.0/24                 # delete
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```


- If you have less than 15 hours a week to spend learning Kubernetes, it is likely you will forget things and have a tendency to go back to the course. As such, it may be beneficial for you to take notes — typing them is recommended as you can include screenshots of YAML files to refer to later on.
- **Persistent Volumes:** This is a topic that will likely appear in the exam. Understanding the flow of this question is key to remembering how to answer it. First, create a Persistent Volume as per the question requirements and refer to the **documentation for the definition file**. Then, create a Persistent Volume Claim (same page), dry run, and change storage to match the PV. Finally, configure the pod to use the PVC by adding a volume and volume mount as per the question.

```
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage                 # add
      persistentVolumeClaim:                # add
        claimName: task-pv-claim            # add
  containers:
    - name: task-pv-container
      image: nginx
```

```
    ports:
      - containerPort: 80
        name: "http-server"
    volumeMounts:                              # add
      - mountPath: "/usr/share/nginx/html". # add
        name: task-pv-storage                  # add
```

- **RBAC**: Configuring Role Base Access Control is key in the exam. Knowing how to configure RBAC for both service accounts and users is important. First, create the service account in the required namespace. Then, configure a role as per the question, and a role-binding that uses the role for the service account or user. The page you will find useful on creating these resources is the **reference page on the documentation, under create.**

- **Multi-container and Sidecar container pods:** This is a topic that usually confuses everyone. The key to answering this question is to dry-run a pod with a single container to begin with. Usually, this will be the container that has extra commands, which you can do imperatively. Then, edit the file to include the second container and the information for it. Don't forget to change the name of the pod — the name will be the same as the first container (given that you passed that first imperatively) but it should be different as per the question.

- **The rest:** The rest of the topics are covered well in Mumshad's course. Following the course and doing the labs is the best way to pass the exam…muscle memory!!

CKAD (Certified Kubernetes Application Developer):

Now that you have successfully completed CKA, it is time to start learning CKAD. Some people tend to skip this exam and move straight on to the CKS, but as this article is meant for all three, this section will cover tips on the exam for CKAD.

- There are a total of 19 questions to be completed in 2 hours.

- For details on the topics, check the **official Linux Foundation page**

- **Topics:** The topics for the CKAD are usually more straightforward than the CKA. This is because there is less need to create new objects and instead change currently existing objects. For example, liveness and readiness probes are just extra components that you pass into a file at the container level, and the documentation page for it is sufficient.

- **Time:** When there are more questions in the exam, there are more problems that you are likely to face if you have little time to solve them. There are 19 questions in the exam, so following the tips provided on time management above will help you here.

- **Dockerfile**: This comes up in the exam at a basic level, but knowing it well will mean less time spent on the question.

- **Jobs/Cronjobs:** This is a must-know and will be tested. Read up on the documentation for Jobs and Cronjobs.

- **Labels and selectors:** Labels and selectors are used heavily, especially for Network Policies, but the question could also simply ask you to label an object such as a node. To show the labels for objects, use the " — show-labels" flag on the command line. This will show the objects and their labels in key=value form. Labeling other components such as pods require you to edit the object:

```
> kubectl label node key1=value1
> kubectl get nodes --show-labels
> kubectl edit po test
-----Add label to pod-----
```

- **ConfigMap and secrets:** Ensuring that you know how to mount configmaps and secrets as either environment variables or volumes is a crucial aspect of successful DevOps practice.

```
> vi configmap-pod-volume.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
```

```
    - name: foo
      configMap:
        name: myconfigmap

> vi secret-pod-env.yaml

apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
  - name: mycontainer
    image: redis
    env:                                  # add
      - name: SECRET_USERNAME             # add
        valueFrom:                        # add
          secretKeyRef:                   # add
            name: mysecret                # add
            key: username                 # add
            optional: false               # add

      - name: SECRET_PASSWORD             # add
        valueFrom:                        # add
          secretKeyRef:                   # add
            name: mysecret                # add
            key: password                 # add
            optional: false               # add

  restartPolicy: Never
```

CKS (Certified Kubernetes Security Specialist)

- A pre-requisite to the CKS is the CKA
- The exam is 2 hours long and there are a total of 16 questions

In an era where security has become vital and containers are becoming popular, it comes as no surprise that CKS is a very respectable certification to have, and attractive to many DevOps employers. Doing the cert not only improves your security knowledge of Kubernetes, but dives deep into some Linux fundamentals. You get to learn more about Linux security modules, Dockerfile security best practices, how TLS communication is used, and other external tools that are utilised by Kubernetes, such as Trivy and Falco.

To discover all of the topics that materialise in the exam, visit the **official Linux Foundation page for CKS**.

While **Mumshad's KodeKloud course** is sufficient for learning all of these new tools, you maybe also benefit from some additional tips for the exam:

- The tools you are taught in CKS also have their own documentation — for example, when studying Falco, you will notice that you need filters to display the specific Falco logs. These are found on Falco's documentation page for supported fields. This is a page that can be accessed during the exam
- These third party tools are usually configured as services on the Kubernetes cluster, so you will need to know how to interact with these services:

```
> systemctl start falco
> systemctl list-units --type services | grep <service-name>
> systemctl stop <service-name>
> systemctl disable <service-name>
```

- You will come to notice that best security practices are obvious — for example, if asked to fix a docker file to follow security best practices, you will need to ensure that you run as a user instead of root to apply the principle of least privilege.
- The same applies to objects such as pods and deployments — containers should not run as root or be able to escalate their privileges. In the code below, runAsUser = 0 is running as the root user, so we change this to 1000 and also make the Security Context, allowPrivilegedEscalation = false, as by default it is set to true:

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000                                    # change
    runAsGroup: 3000
    fsGroup: 2000
  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:                                    # add
      allowPrivilegeEscalation: false                   # add
```

- **Security Contexts:** Knowing how to use 'Security Contexts' is vital, as this is used quite often in definition files — it is used when adding security practices such as privileged users and escalation privileges. It is also seen in third-party tools such as Seccomp profiles (which are used to restrict or allow a subset of syscalls — only the necessary syscalls required for a pod to run). We load it in the Security context shown below:

```
apiVersion: v1
kind: Pod
metadata:
  name: audit-pod
  labels:
    app: audit-pod
spec:
  securityContext:                                  # add
    seccompProfile:                                 # add
      type: Localhost                               # add
      localhostProfile: profiles/audit.json         # add
  containers:
  - name: test-container
    image: hashicorp/http-echo:0.2.3
    args:
    - "-text=just made some syscalls!"
    securityContext:
      allowPrivilegeEscalation: false
```

- **Api-server (or Kube-apiserver):** As you know by now, the kube-apiserver is the brain of the operation in Kubernetes, and it lives in the controlplane node. To give you a proper definition: the Kubernetes API server validates and configures data for the API objects which include pods, services, replication controllers, and others. So, a security specialist should be prepared to configure the kube-apiserver more than twice in the exam and on different clusters and master nodes. You will be configuring the main manifest file for kube-apiserver at

/etc/kubernetes/manifests/kube-apiserver.yaml. These static pod files create their objects without the need to run them directly

- **Kube-apiserver (cont..):** By changing the kube-apiserver, the API-server automatically restarts and applies your changes — if a change is incorrect, the cluster fails. This a common cause for panic in the CKS exam — but here's how to prevent that:

Make a copy of the kube-apiserver.yaml manifest file in your home directory, then proceed to make a change. Restart your cluster and observe if kubectl is working. If you don't know where you went wrong, and you are running out of time, you can just bring the copied file back into the manifest file. Follow the below steps:

```
> cd /etc/kubernetes/manifests
> cp kube-apiserver.yaml ~/kube1.yaml
> vi kube-apiserver.yaml

-----Changes-made--------

> k get po
---Connection to host failed message---
> rm kube-apiserver.yaml
> mv ~/kube1.yaml /etc/kubernetes/manifests/kube-apiserver.yaml

# When resetting api-server

> cd /etc/kubernetes/manifests
> vi kube-apiserver.yaml

-----Changes-made--------

> mv kube-apiserver.yaml ../
> crictl ps
> mv ../kube-apiserver.yaml ./
```

- In the above code, the first scenario is for the kube-apiserver crashing — to go back to a working version, we restore it from the manifest copy. In the second scenario, we are resetting the kube-apiserver after our changes. This is a good way to reset, as we are moving the file away from the manifests directory to the parent directory. This terminates the container associated with the kube-apiserver (we check using crictl to make sure it has successfully been removed), then brings it back to the manifest directory. By using the parent directory, we just pass in .. (double-dot).

- **Node components:** The kube-apiserver is the main component you will interact with on the master node and as part of it, you will use it for admission controller plugins such as ImagePolicyWebhook, PodSecurityPolicies, and Audit-logs. However, other components such as the kubelet, AppArmor, and seccomp profiles are only available to be used in the worker nodes.

- **SSH:** You will find yourself SSH-ing more than half the exam to interact with the correct node. You will often be told which node you need to SSH into, but it's good to know where these components are.

- **Apparmor:** Make sure to read the apparmor documentation well, as there are steps involved which can be easily forgotten. For example, adding the required annotation after loading the apparmor profile.

- **Audit-logs:** Setting up audit-logs means you have to add the relevant sections in the kube-apiserver. Take this question *very* slowly and carefully, as you will be adding two sets of volumes and volume mounts in the kube-apiserver.yaml file itself. This question has the most changes in the kube-apiserver. **The documentation for audit logs** explains the process well, so have a thorough read and keep it bookmarked. Here is a snippet of the volume hosts and mounts to be added, and the paths will be provided. However, you will need to create the policy — a skeleton file will be provided to you:

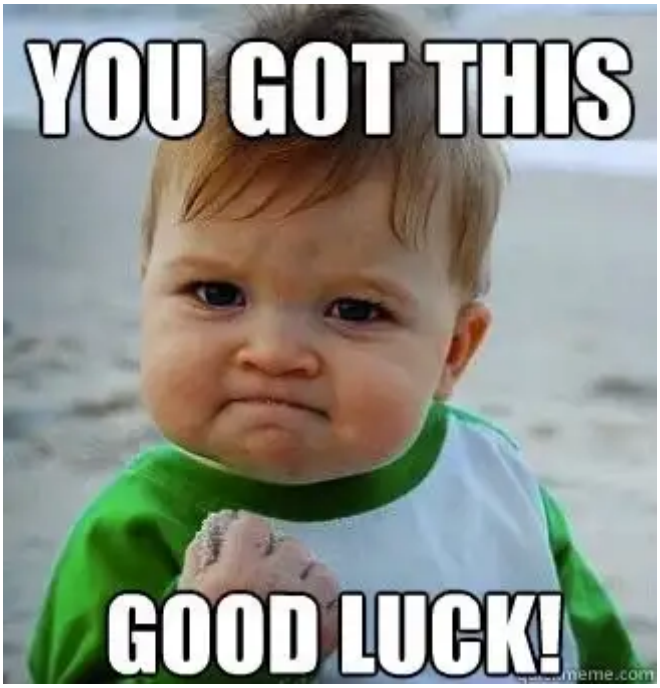
```
### VolumeMounts for auditing in kube-apiserver manifest file
...
volumeMounts:
- mountPath: /etc/kubernetes/audit-policy.yaml
  name: audit
  readOnly: true
- mountPath: /var/log/kubernetes/audit/
  name: audit-log
  readOnly: false

### Volumes for auditing in kube-apiserver manifest file
...
volumes:
- name: audit
  hostPath:
    path: /etc/kubernetes/audit-policy.yaml
    type: File
- name: audit-log
  hostPath:
```

```
    path: /var/log/kubernetes/audit/
    type: DirectoryOrCreate
```

- **OPA:** This topic introduces a new file — the rego file. It is not necessary to know how to write rego files, but it is important to be able to read them. The necessary thing to understand is that rego files can appear as configmaps, and these configmaps will tell you what type of 'images' the OPA policy accepts.
- **Killer.sh:** Finally, the Killer.sh simulator — is given to you when you purchase the exam. It's a no-brainer to do this as it will give you a feel for the exam. It is fairly difficult and covers content that you don't necessarily need to know, but I personally picked up a lot of Linux and K8s secrets (not the topic) from there, that I wouldn't have otherwise come across in a while.
- If you've come this far, then you have all of the tips and tricks for passing the exam (including the tips I didn't have, but wish I knew). Anything that I haven't already covered need not mentioning as it is covered well in both the course and the accompanying labs (so make sure to do them well!)

Parting words

Above is a detailed walkthrough of the process I took to pass the exam, but I found myself debugging and looking for answers even after I had finished Mumshad's course. This taught me that you need to be active in the learning process, as it will help you pick up on a lot along the way. Above all, good preparation prevails.



In summary

The learning process for Kubernetes is a really enjoyable one, and looking at it in this way makes it easier. By the time you obtain the trifecta, you will become an expert in not only Kubernetes but in docker and Linux too. The certifications are challenging, but every topic has its own flow, so understanding this will help you to achieve the best results.

If you have any questions on the topic discussed in this blog, you can reach out to me on LinkedIn: https://www.linkedin.com/in/a-abukar/

Interested in joining us?

Credera is currently hiring! View our open positions and apply here.

Got a question?

Please get in touch to speak to a member of our team.

Kubernetes Exam Preparation Kubernetes Administration Kubernetes Security

Kubernetes Development
