This document provides the curriculum outline of the Knowledge, Skills and Abilities that a Certified Kubernetes Administrator (CKA) can be expected to demonstrate.

CKA Curriculum

25% - Cluster Architecture, Installation & Configuration

- Manage role based access control (RBAC)
- Use Kubeadm to install a basic cluster
- Manage a highly-available Kubernetes cluster
- Provision underlying infrastructure to deploy a Kubernetes cluster
- Perform a version upgrade on a Kubernetes cluster using Kubeadm
- Implement etcd backup and restore

15% - Workloads & Scheduling

- Understand deployments and how to perform rolling update and rollbacks
- Use ConfigMaps and Secrets to configure applications
- Know how to scale applications
- Understand the primitives used to create robust, self-healing, application deployments
- Understand how resource limits can affect Pod scheduling
- Awareness of manifest management and common templating tools

20% - Services & Networking

- Understand host networking configuration on the cluster nodes
- Understand connectivity between Pods
- Understand ClusterIP, NodePort, LoadBalancer service types and endpoints
- Know how to use Ingress controllers and Ingress resources
- Know how to configure and use CoreDNS
- Choose an appropriate container network interface plugin

This document provides the curriculum outline of the Knowledge, Skills and Abilities that a Certified Kubernetes Administrator (CKA) can be expected to demonstrate.

CKA Curriculum

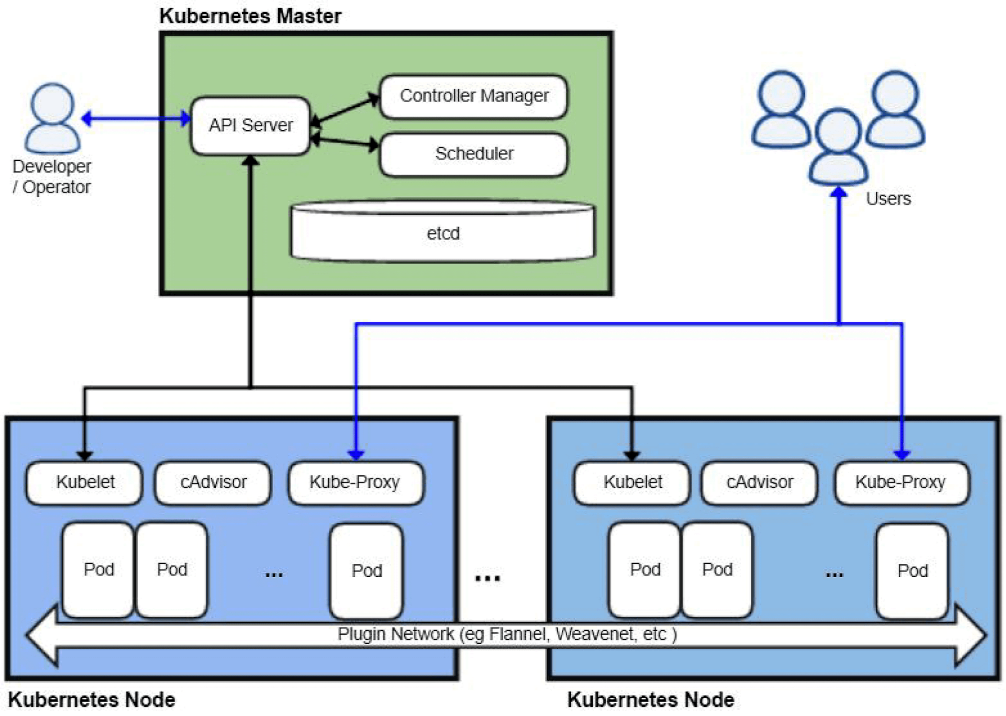
10% - Storage

- Understand storage classes, persistent volumes
- Understand volume mode, access modes and reclaim policies for volumes
- Understand persistent volume claims primitive
- Know how to configure applications with persistent storage

30% - Troubleshooting

- Evaluate cluster and node logging
- Understand how to monitor applications
- Manage container stdout & stderr logs
- Troubleshoot application failure
- Troubleshoot cluster component failure
- Troubleshoot networking

Kubernetes Architecture Diagram



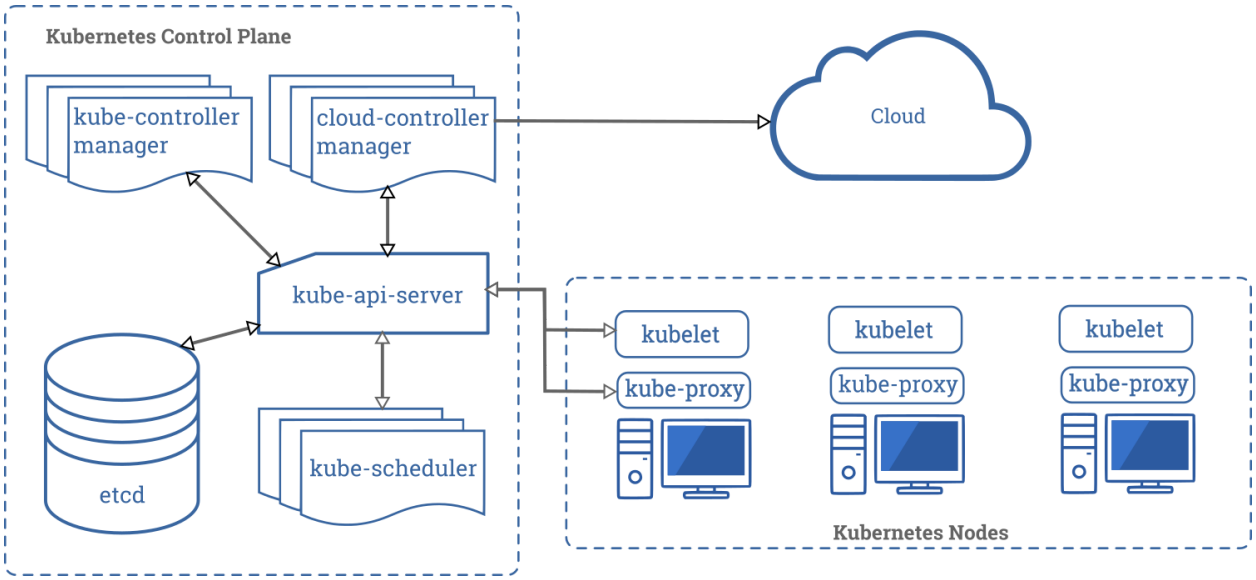
1) In the AWS Kubernetes architecture diagram above you can see, there is one or more master and multiple nodes. One or masters used to provide high-availability.

2) The Master node communicates with Worker nodes using Kube API-server to kubelet communication.

3) In the Worker node, there can be one or more pods and pods can contain one or more containers.

4) Containers can be deployed using the image also can be deployed externally by the user.

Kubernetes Architecture Components



Kubernetes Master Node

Master Node is a collection of components like Storage, Controller, Scheduler, API-server that makes up the control plan of the Kubernetes. When you interact with Kubernetes by using CLI you are communicating with the Kubernetes cluster's master node. All the processes run on a single node in the cluster, and this node is also referred to as the master.

Master Node Components:

1) Kube API-server performs all the administrative tasks on the master node. A user sends the rest commands as YAML/JSON format to the API server, then it processes and executes them. The Kube API-server is the front end of the Kubernetes control plane.

2) etcd is a distributed key-value store that is used to store the cluster state. Kubernetes stores the file in a database called the **etcd**. Besides storing the cluster state, etcd is also used to store the configuration details such as the subnets and the config maps.

3) Kube-scheduler is used to schedule the work to different worker nodes. It also manages the new requests coming from the API Server and assigns them to healthy nodes.

4) Kube Controller Manager task is to obtain the desired state from the API Server. If the desired state does not meet the current state of the object, then the corrective steps are taken by the control loop to bring the current state the same as the desired state.

There are different types of control manager in Kubernetes architecture:

- **Node Manager,** it manages the nodes. It creates new nodes if any node is unavailable or destroyed.
- **Replication Controller,** it manages if the desired number of containers is running in the replication group.
- **Endpoints controller,** it populates the endpoints object that is, joins Services & Pods.

Kubernetes Worker Node

The worker nodes in a cluster are the machines or physical servers that run your applications. The Kubernetes master controls each node. there are multiple nodes connected to the master node. On the node, there are multiple pods running and there are multiple containers running in pods.

Worker Node Components

1) Kubelet is an agent that runs on each worker node and communicates with the master node. It also makes sure that the containers which are part of the pods are always healthy. It watches for tasks sent from the API Server, executes the task like deploy or destroy the container, and then reports back to the Master.

2) Kube-proxy is used to communicate between the multiple worker nodes. It maintains network rules on nodes and also make sure there are necessary rules define on the worker node so the container can communicate to each in different nodes.

3) Kubernetes pod is a group of one or more containers that are deployed together on the same host. Pod is deployed with a shared storage/network, and a specification for how to run the containers. Containers can easily communicate with other containers in the same pod as though they were on the same machine.

4) Container Runtime is the software that is responsible for running containers. Kubernetes supports several container runtimes: Docker, containers.

kubectl - Cheat Sheet

Kubectl Autocomplete

BASH

```
source <(kubectl completion bash) # setup autocomplete in bash into the current shell, bash-completion package should be installed first.
echo "source <(kubectl completion bash)" >> ~/.bashrc # add autocomplete permanently to your bash shell.
```

You can also use a shorthand alias for `kubectl` that also works with completion:

```
alias k=kubectl
complete -F __start_kubectl k
```

ZSH

```
source <(kubectl completion zsh)  # setup autocomplete in zsh into the current shell
echo "[[ $commands[kubectl] ]] && source <(kubectl completion zsh)" >> ~/.zshrc # add autocomplete permanently to your zsh shell
```

Kubectl Context and Configuration

Set which Kubernetes cluster `kubectl` communicates with and modifies configuration information. See Authenticating Across Clusters with kubeconfig documentation for detailed config file information.

```
kubectl config view # Show Merged kubeconfig settings.

# use multiple kubeconfig files at the same time and view merged config
KUBECONFIG=~/.kube/config:~/.kube/kubconfig2

kubectl config view

# get the password for the e2e user
kubectl config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'
```

```
kubectl config view -o jsonpath='{.users[].name}'    # display the first
user
kubectl config view -o jsonpath='{.users[*].name}'   # get a list of users
kubectl config get-contexts                          # display list of
contexts
kubectl config current-context                       # display the
current-context
kubectl config use-context my-cluster-name           # set the default
context to my-cluster-name

# add a new user to your kubeconf that supports basic auth
kubectl config set-credentials kubeuser/foo.kubernetes.com
--username=kubeuser --password=kubepassword

# permanently save the namespace for all subsequent kubectl commands in
that context.
kubectl config set-context --current --namespace=ggckad-s2

# set a context utilizing a specific username and namespace.
kubectl config set-context gce --user=cluster-admin --namespace=foo \
  && kubectl config use-context gce

kubectl config unset users.foo                       # delete user foo
```

Apply

`apply` manages applications through files defining Kubernetes resources. It creates and updates resources in a cluster through running `kubectl apply`. This is the recommended way of managing Kubernetes applications on production. See Kubectl Book.

Creating Objects

Kubernetes manifests can be defined in YAML or JSON. The file extension `.yaml`, `.yml`, and `.json` can be used.

```
kubectl apply -f ./my-manifest.yaml            # create resource(s)
kubectl apply -f ./my1.yaml -f ./my2.yaml      # create from multiple
files
kubectl apply -f ./dir                         # create resource(s) in all
manifest files in dir
kubectl apply -f https://git.io/vPieo          # create resource(s) from
url
```

```
kubectl create deployment nginx --image=nginx  # start a single instance
of nginx
kubectl explain pods                           # get the documentation for
pod manifests

# Create multiple YAML objects from stdin
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "1000000"
---
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep-less
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "1000"
EOF

# Create a secret with several keys
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: $(echo -n "s33msi4" | base64 -w0)
  username: $(echo -n "jane" | base64 -w0)
EOF
```

Viewing, Finding Resources

```
# Get commands with basic output
kubectl get services                          # List all services in the
namespace
kubectl get pods --all-namespaces             # List all pods in all
namespaces
kubectl get pods -o wide                      # List all pods in the
current namespace, with more details
kubectl get deployment my-dep                 # List a particular
deployment
kubectl get pods                              # List all pods in the
namespace
kubectl get pod my-pod -o yaml                # Get a pod's YAML

# Describe commands with verbose output
kubectl describe nodes my-node
kubectl describe pods my-pod

# List Services Sorted by Name
kubectl get services --sort-by=.metadata.name

# List pods Sorted by Restart Count
kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'

# List PersistentVolumes sorted by capacity
kubectl get pv --sort-by=.spec.capacity.storage

# Get the version label of all pods with label app=cassandra
kubectl get pods --selector=app=cassandra -o \
  jsonpath='{.items[*].metadata.labels.version}'

# Get all worker nodes (use a selector to exclude results that have a
label
# named 'node-role.kubernetes.io/master')
kubectl get node --selector='!node-role.kubernetes.io/master'

# Get all running pods in the namespace
kubectl get pods --field-selector=status.phase=Running

# Get ExternalIPs of all nodes
kubectl get nodes -o
jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP")].address}'
```

```
# List Names of Pods that belong to Particular RC
# "jq" command useful for transformations that are too complex for
jsonpath, it can be found at https://stedolan.github.io/jq/
sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector |
to_entries | .[] | "\(.key)=\(.value),"')%?}
echo $(kubectl get pods --selector=$sel
--output=jsonpath={.items..metadata.name})

# Show labels for all pods (or any other Kubernetes object that supports
labelling)
kubectl get pods --show-labels

# Check which nodes are ready
JSONPATH='{range .items[*]}{@.metadata.name}:{range
@.status.conditions[*]}{@.type}={@.status};{end}{end}' \
 && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"

# List all Secrets currently in use by a pod
kubectl get pods -o json | jq
'.items[].spec.containers[].env[]?.valueFrom.secretKeyRef.name' | grep -v
null | sort | uniq

# List all containerIDs of initContainer of all pods
# Helpful when cleaning up stopped containers, while avoiding removal of
initContainers.
kubectl get pods --all-namespaces -o jsonpath='{range
.items[*].status.initContainerStatuses[*]}{.containerID}{"\n"}{end}' | cut
-d/ -f3

# List Events sorted by timestamp
kubectl get events --sort-by=.metadata.creationTimestamp

# Compares the current state of the cluster against the state that the
cluster would be in if the manifest was applied.
kubectl diff -f ./my-manifest.yaml
```

Updating Resources

```
kubectl set image deployment/frontend www=image:v2               # Rolling
update "www" containers of "frontend" deployment, updating the image
```

```
kubectl rollout history deployment/frontend                    # Check the history of deployments including the revision
kubectl rollout undo deployment/frontend                       # Rollback to the previous deployment
kubectl rollout undo deployment/frontend --to-revision=2       # Rollback to a specific revision
kubectl rollout status -w deployment/frontend                  # Watch rolling update status of "frontend" deployment until completion
kubectl rollout restart deployment/frontend                    # Rolling restart of the "frontend" deployment


cat pod.json | kubectl replace -f -                            # Replace a pod based on the JSON passed into std

# Force replace, delete and then re-create the resource. Will cause a service outage.
kubectl replace --force -f ./pod.json

# Create a service for a replicated nginx, which serves on port 80 and connects to the containers on port 8000
kubectl expose rc nginx --port=80 --target-port=8000

# Update a single-container pod's image version (tag) to v4
kubectl get pod mypod -o yaml | sed 's/\(image: myimage\):.*$/\1:v4/' |
kubectl replace -f -

kubectl label pods my-pod new-label=awesome                    # Add a Label
kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq     # Add an annotation
kubectl autoscale deployment foo --min=2 --max=10              # Auto scale a deployment "foo"
```

Patching Resources

```
# Partially update a node
kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'
```

```
# Update a container's image; spec.containers[*].name is required because it's a merge key
kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-hostname","image":"new image"}]}}'

# Update a container's image using a json patch with positional arrays
kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path": "/spec/containers/0/image", "value":"new image"}]'

# Disable a deployment livenessProbe using a json patch with positional arrays
kubectl patch deployment valid-deployment  --type json   -p='[{"op": "remove", "path": "/spec/template/spec/containers/0/livenessProbe"}]'

# Add a new element to a positional array
kubectl patch sa default --type='json' -p='[{"op": "add", "path": "/secrets/1", "value": {"name": "whatever" } }]'
```

Editing Resources

Edit any API resource in your preferred editor.

```
kubectl edit svc/docker-registry                      # Edit the service named docker-registry
KUBE_EDITOR="nano" kubectl edit svc/docker-registry   # Use an alternative editor
```

Scaling Resources

```
kubectl scale --replicas=3 rs/foo                                 # Scale a replicaset named 'foo' to 3
kubectl scale --replicas=3 -f foo.yaml                            # Scale a resource specified in "foo.yaml" to 3
kubectl scale --current-replicas=2 --replicas=3 deployment/mysql  # If the deployment named mysql's current size is 2, scale mysql to 3
kubectl scale --replicas=5 rc/foo rc/bar rc/baz                   # Scale multiple replication controllers
```

Deleting Resources

```
kubectl delete -f ./pod.json
# Delete a pod using the type and name specified in pod.json
kubectl delete pod,service baz foo
# Delete pods and services with same names "baz" and "foo"
kubectl delete pods,services -l name=myLabel
# Delete pods and services with label name=myLabel
kubectl -n my-ns delete pod,svc --all
# Delete all pods and services in namespace my-ns,
# Delete all pods matching the awk pattern1 or pattern2
kubectl get pods  -n mynamespace --no-headers=true | awk
'/pattern1|pattern2/{print $1}' | xargs  kubectl delete -n mynamespace pod
```

Interacting with running Pods

```
kubectl logs my-pod                                 # dump pod logs
(stdout)
kubectl logs -l name=myLabel                        # dump pod logs, with
label name=myLabel (stdout)
kubectl logs my-pod --previous                      # dump pod logs
(stdout) for a previous instantiation of a container
kubectl logs my-pod -c my-container                 # dump pod container
logs (stdout, multi-container case)
kubectl logs -l name=myLabel -c my-container        # dump pod logs, with
label name=myLabel (stdout)
kubectl logs my-pod -c my-container --previous      # dump pod container
logs (stdout, multi-container case) for a previous instantiation of a
container
kubectl logs -f my-pod                              # stream pod logs
(stdout)
kubectl logs -f my-pod -c my-container              # stream pod container
logs (stdout, multi-container case)
kubectl logs -f -l name=myLabel --all-containers    # stream all pods logs
with label name=myLabel (stdout)
kubectl run -i --tty busybox --image=busybox -- sh  # Run pod as
interactive shell
kubectl run nginx --image=nginx -n
mynamespace                                         # Run pod nginx in a
specific namespace
kubectl run nginx --image=nginx                     # Run pod nginx and
write its spec into a file called pod.yaml
```

```
--dry-run=client -o yaml > pod.yaml

kubectl attach my-pod -i                                # Attach to Running
Container
kubectl port-forward my-pod 5000:6000                   # Listen on port 5000
on the local machine and forward to port 6000 on my-pod
kubectl exec my-pod -- ls /                             # Run command in
existing pod (1 container case)
kubectl exec my-pod -c my-container -- ls /             # Run command in
existing pod (multi-container case)
kubectl top pod POD_NAME --containers                   # Show metrics for a
given pod and its containers
```

Interacting with Nodes and Cluster

```
kubectl cordon my-node                                                  #
Mark my-node as unschedulable
kubectl drain my-node                                                   #
Drain my-node in preparation for maintenance
kubectl uncordon my-node                                                #
Mark my-node as schedulable
kubectl top node my-node                                                #
Show metrics for a given node
kubectl cluster-info                                                    #
Display addresses of the master and services
kubectl cluster-info dump                                               #
Dump current cluster state to stdout
kubectl cluster-info dump --output-directory=/path/to/cluster-state     #
Dump current cluster state to /path/to/cluster-state

# If a taint with that key and effect already exists, its value is
replaced as specified.
kubectl taint nodes foo dedicated=special-user:NoSchedule
```

Resource types

List all supported resource types along with their shortnames, API group, whether they are namespaced, and Kind:

```
kubectl api-resources
```

Other operations for exploring API resources:

```
kubectl api-resources --namespaced=true      # All namespaced resources
kubectl api-resources --namespaced=false     # All non-namespaced
resources
kubectl api-resources -o name                # All resources with simple
output (just the resource name)
kubectl api-resources -o wide                # All resources with expanded
(aka "wide") output
kubectl api-resources --verbs=list,get       # All resources that support
the "list" and "get" request verbs
kubectl api-resources --api-group=extensions # All resources in the
"extensions" API group
```

Formatting output

To output details to your terminal window in a specific format, add the `-o` (or `--output`) flag to a supported `kubectl` command.

Output format	Description
`-o=custom-columns=<spec>`	Print a table using a comma separated list of custom columns
`-o=custom-columns-file=<filename>`	Print a table using the custom columns template in the `<filename>` file
`-o=json`	Output a JSON formatted API object
`-o=jsonpath=<template>`	Print the fields defined in a jsonpath expression
`-o=jsonpath-file=<filename>`	Print the fields defined by the jsonpath expression in the `<filename>` file
`-o=name`	Print only the resource name and nothing else

`-o=wide`	Output in the plain-text format with any additional information, and for pods, the node name is included
`-o=yaml`	Output a YAML formatted API object

Examples using `-o=custom-columns`:

```
# All images running in a cluster
kubectl get pods -A -o=custom-columns='DATA:spec.containers[*].image'

 # All images excluding "k8s.gcr.io/coredns:1.6.2"
kubectl get pods -A
-o=custom-columns='DATA:spec.containers[?(@.image!="k8s.gcr.io/coredns:1.6
.2")].image'

# All fields under metadata regardless of name
kubectl get pods -A -o=custom-columns='DATA:metadata.*'
```

Voucher code given below. This offer expires soon.

REGISTER AND SAVE $57 ON CKA EXAM TODAY

CKA Exam Voucher: Use coupon **DCUBEOFFER** at checkout

Here are some things to keep in mind regarding the exam.

1. The CKA exam is to be taken online and it is proctored remotely.
2. A score of 66% or above must be earned to pass.
3. CKA Certification is valid for 3 years.
4. After registration, you get one year to schedule the exam.
5. After registration, you get a maximum of 2 attempts to give the test. If you miss a scheduled exam for any reason – your second attempt gets nullified.

Note: You can always check the latest Kubernetes Certification Voucher Codes to save costs on the CKA, CKAD, and CKS certification registration

Certified Kubernetes Administrator – CKA Exam Preparation Guide

This section will go over resources and links that can help you prepare for the CKA exam better.

TABLE OF CONTENTS

CKA Exam Prerequisites

CKA does not require any candidate to have any other certification before they can appear for the CKA exam. The only thing required to clear the exam is a conceptual understanding of Kubernetes internal working and a lot of practice.

CKA Exam details

Exam Duration	2 hours
Pass Percentage	66%
Kubernetes Version	v1.21
CKA Validity	3 Years
Exam Cost	$375 USD

CKA Exam details

CKA exam is an open book exam i.e. you can use the following websites while you are taking the exam.

1. https://kubernetes.io/docs/
2. https://github.com/kubernetes/
3. https://kubernetes.io/blog/ and their subdomains. This includes all available language translations of these pages (e.g. https://kubernetes.io/zh/docs/)

K — KUBERNETES

CKS Exam Study Guide: Resources to Pass Certified Kubernetes Security Specialist

by **Bibin Wilson** · August 1, 2021

CKA Exam syllabus

The following are the domains and competencies part of the syllabus along with their respective weightage.

Topic	Weightage
Cluster Architecture, Installation & Configuration	25 %
Workloads & Scheduling	15 %
Services & Networking	20 %
Storage	10 %
Troubleshooting	30 %

CKA Exam Syllabus

CKA Preparation courses

Investing in a course will help you understand all the concepts for the exam in an easier manner. If you are a beginner, we strongly suggest you invest some time and money in a course of your choice.

We have the following recommendation.

1. CKA preparation course by Mumshad With Practice Tests [Udemy]: His course has a lot of quizzes and the quality is top-notch.

CKA Practice Labs

The best way to prepare is to get a clear understanding of the concepts involved and do a lot of hands-on practice! The below setups will give you a Kubernetes cluster where you can do all the required practice. The exam expects you to solve problems on a live cluster.

CKA does not have any MCQ-type format – so hands-on practice is a must.

> **Note:** When you purchase the subscription, you will get free access to https://killer.sh/cka exam simulator. You can make use of the simulator to give the CKA practice exam.

1. Katacoda
2. Minikube
3. Kubernetes Setup using Kubeadm [Detailed Guide]
4. Kubernetes Vagrant Setup using Kubeadm
5. GKE Cluster using free Google Cloud Credits
6. EKS Service on AWS using Free tier program
7. AKS service on Azure using free cloud credits
8. Kubernetes Cluster on Digital Ocean[Get $100 Digital Ocean Free Credits]

CKA Syllabus Wise Study resources

Here, we will be discussing the CKA syllabus-wise official and useful resources that

can be used to prepare for each topic of the CKA exam.

Cluster Architecture,Installation & Configuration [25%]

Manage role-based Access Control (RBAC)

Role-based access control is a method of managing access levels to applications or individual users. It's a handy tool in the hands of an administrator to give fine-grained controls to others.

RBAC essentials	Check RBAC Documentation

Use Kubeadm to Install a Basic Cluster

This section focuses on the setting up of a cluster using the kubeadm tool. The candidate must have a clear understanding of the underlying components of Kubernetes such as etcd, Kube API servers, SSL certificate management, etc.

> **Note:** Ensure swap is disabled on all nodes before deploying the Kubeadm cluster.

Learn About Kubeadm	Kubeadm Documentation
Tutorial	Kubeadm Cluster Setup Guide

Manage a Highly-available Kubernetes Cluster

One of the duties of a Kubernetes administrator is to ensure the high availability of the cluster. This involves proper maintenance tasks on the cluster as well as managing the worker nodes.

HA Kubernetes Cluster	HA cluster setup using Kubeadm

Most clusters today are created on AWS, Azure, or GCP where the respective cloud providers take responsibility for cluster availability. However, it is important to understand the logic behind the HA kubernetes cluster.

Provision Underlying Infrastructure to Deploy a Kubernetes cluster

A Kubernetes cluster needs a lot of components to work in sync. Therefore, a clear understanding of each component like etcd, Kube api-service, controllers, schedulers, kubelet, docker runtime are required.

Kubernetes Componentes	Kubernetes components overview

Perform a Version upgrade on a Kubernetes Cluster Using Kubeadm

Kubernetes is a continuously improving tool. Every now and then, a new version comes up with improvements and features. It is the duty of the administrator to take care of version upgrades.

Task	Kubernetes Version Upgrade Use Kubeadm

Implement etcd Backup and Restore

Etcd is a key-value store of the cluster. All the information regarding pods, services, etc are stored here in key-value format

Basic etcd commands to try:

```
etcdctl cluster-health

etcdctl member-list

etcdctl set k1 v1
```

```
etcdctl get k1
```

Reference	etcd Backup & Operations

Workloads & Scheduling [15%]

Understand Deployments and How to Perform Rolling Update and rollbacks

Kubernetes Deployment ensures a minimum no of replicas of an application is running at all times. In case a replica goes down, the Kubernetes API ensures that a new one is created within minutes.

> **Imperative commands:** These are commands which let you create Kubernetes objects via a CLI. Meaning they remove the need to write the whole YAML. Knowing imperative commands can help you save time in the exam. I highly recommend them!

Kubectl commands:

```
kubectl create deployment <name> --image=<name> //create deployment

kubectl create deployment <name> --image=<name> -- sleep 300 //with command arguments

kubectl scale deployment <name> --replicas=4 //scale up or down
```

Reference	Kubernetes Deployment Concepts

Sometimes, you may want to rollback a Deployment; for example, when the

Deployment is not stable, such as a crash loopback error, you can roll back the Deployment

Kubectl commands:

```
kubectl set image deployment <name of deployment> <name of container>=<new image name> // update image

kubectl rollout status deployment <name of deployment> //see status

kubectl rollout history deployment <name of deployment> //see history
```

Reference	Kubernetes Rolling Update

Use Config Maps and Secrets to Configure applications

Kubernetes Configmaps are useful to store non-critical data in key-value pair format. They can also be used to inject env vars into pods.

Reference	Kubernetes Configmap Concepts

Kubectl Commands for Configmaps:

```
kubectl create cm <name of configmap> --from-file=hello.txt
kubectl create cm <name of configmap> --from-literal=key1=value1
```

Secrets are useful to store sensitive data in key value pair format. They can also be used to inject env vars into pods.

Reference	Kubernetes Secrets Concepts

Kubectl commands for secrets:

```
kubectl create secrets <name of secret> --from-file=hello.txt
kubectl create secrets <name of secret> --from-literal=key1=value1
```

Know How to Scale Applications

Kubernetes provides a no. of ways to scale applications, you can use deployment objects and increase the no. of replicas of your application.

Horizontal Pod Autoscalers (HPAs) can also be used to increase the no. of replicas according to the application metrics

Task	Working With Horizontal Pod Autoscaler

Understand the Primitives Used to Create Robust, Self-healing, Application Deployments

This section is mostly conceptual, for any self-healing application you should use deployments or stateful sets so that whenever pods go down, Kubernetes recreates them instantly.

Deployments also give you the option to keep a track of all the changes you make. You can also roll back to a previous state very easily.

```
kubectl set image deployment <name of deployment> <name of container>=<new image name> // update image

kubectl rollout status deployment <name of deployment> //see status

kubectl rollout history deployment <name of deployment> //see history
```

Understand How Resource Limits Can Affect Pod Scheduling

Cluster management also involves the management of workloads, as an admin – you should ensure that each pod is able to get resources based upon its needs.

In kubernetes, each pod can be assigned a minimum and maximum CPU and memory usage.

Reference 01	Manage Container Resources
Reference 02	Pods with resoruce requests

Awareness of manifest management and common templating tools

This section expects you to be familiar with tools like kustomization, helm, etc.

Kubernetes Manifests	Managing Kubernetes Objects
Kustomization	Manage objects with Kustomize

Services & Networking [20%]

Understand host networking configuration on the cluster nodes

Kube-proxy is the component required on each worker node for the pods to communicate with each other. Networking between nodes involves Kube proxy participation as well.

Kubelet is how a worker node is in-network with the master node. All these concepts are required for understanding networking inside kubernetes.

Reference	Kubernetes Networking

Understand Connectivity Between Pods

Pods communicate with each other using services. Kube proxy is the component that makes this possible.

Reference	Understand Kube Proxy

Understand ClusterIP, NodePort, LoadBalancer service types and endpoints

Understanding each service type along with their use cases is very important. Special attention should be paid to understanding how pods can be added under a service.

Reference	Kubernetes Service Explained

Know how to use Ingress controllers and Ingress resources

Ingress resources are how external entities are assigned access to internal cluster services. Ingress controllers are load balancers that make it possible.

Reference 01	Kubernetes Ingress
Reference 02	Kubernetes Ingress Controller
Blog	Ingress Tutorial for Beginners

Know How to Configure and Use CoreDNS

CoreDNS is a flexible, extensible DNS server that can serve as the Kubernetes cluster DNS. Like Kubernetes, the CoreDNS project is hosted by the CNCF.

Task	Using CoreDNS for Service Discovery

Choose an appropriate container network interface plugin

CNI stands for Container Networking Interface and its goal is to create a generic plugin-based networking solution for containers.

There are a lot of solutions such as Flannel, Calico, etc. This section explores some of them.

Reference	Kubernetes Network Plugins

Storage [10%]

Understand Storage Classes, Persistent Volumes, Persistent Volume Claims

StorageClasses: StorageClass provides a way to describe the “classes” of storage available.

PersistentVolume: It is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using StorageClasses. They are created over StorageClasses.

PersistentVolumeClaim: It is a request for storage by a user. They are created over PersistentVolumes.

Reference	Kubernetes Persistent Volumes

Understand Volume Mode, Access Modes and Reclaim Policies for volumes

Volume modes: Kubernetes supports two types of volume modes: Filesystem & Block.

Access modes: Kubernetes supports three types of access modes: ReadWriteOnce, ReadOnlyMany & ReadWriteMany.

Reclaim Policy: Kubernetes supports three policies: Retain, Recycle & Delete

Reference	Kubernetes Volume Modes Kubernetes Volume Access Modes

Know How to Configure Applications With Persistent Storage

Application pods can use persistent storage by mounting a PVC.

Task Configure Kubernetes Volume in Pod

Troubleshooting [30%]

Evaluate Cluster and Node Logging & Managing Logs

Application logs can help in understanding the activities and status of the application. The logs are particularly useful for debugging problems and monitoring cluster activity.

Going through logs of kubernetes control plane components like etcd, scheduler can also be very helpful.

There are certain flags in kubectl commands which can help speed up your debugging cases, they are given below.

Kubectl Command To Check Logs:

```
kubectl logs deployment/<name of deployment>
kubectl logs deployment/<name of deployment> --tail=10
kubectl logs deployment/<name of deployment> --tail=10 -f
```

Reference Kubernetes Logging

Understand How to Monitor Applications

Monitoring applications can be done by storing logs and studying the application's metrics.

Tools like Prometheus & Grafana are popular as they make management of metrics very easy.

Very often, sidecar containers are used as metrics exporters of the main application container.

Troubleshoot Application Failure

Administrators are also required to help users debug applications that are deployed into Kubernetes and not behaving correctly.

Reference	Understanding Kubernetes Logging
Task	Debug Kubernetes Objects

Troubleshoot Cluster Component Failure

Cluster components need to be debugged and perform troubleshooting for failures when users are sure that their application has everything perfectly set up.

Task	Debug Kubernetes Cluster

Troubleshoot Networking

There can be scenarios where things are going wrong on the network end such as some incorrect configuration in ingress resources etc.

Some Unofficial Useful CKA Resources

1. Understand kubernetes SSL certificates
2. Simulator for hands-on practice
3. Vim shortcuts. This will help you save time on exams.
4. Hands on CKA practical question bank on Github

CKA Exam DO's

- While giving CKA practice exams, try to wrap up 15 minutes before the deadline – it will give you additional time to revise the solutions.
- Give a couple of practice exams, identify your weak topics and spend more time on those.
- On the exam day, keep an alternative internet source handy in case of Wi-Fi internet goes down. We don't want all our handwork to be wasted, do we?
- If any particular question will take more than 6-7 mins to solve, flag/mark it to solve for later and come back once you solve the rest.

CKA Exam DON'Ts

- Most people don't even use an alias. So no need to overwhelm yourself with an `alias` for everything.
- Don't give the exam on the last day. The idea is to give it in a pressure-free environment.
- At the time of the exam, you don't have anything on the table other than your workstation or laptop. Linux Foundation has strict rules on the CKA exam environment.

Conclusion

This **CKA exam study guide** will help you understand cluster components and their management in a much better way and help in your career progression.