Certification Tip!

Here's a tip!

As you might have seen already, it is a bit difficult to create and edit YAML files. Especially in the CLI. During the exam, you might find it difficult to copy and paste YAML files from browser to terminal. Using the kubectl run command can help in generating a YAML template. And sometimes, you can even get away with just the kubectl run command without having to create a YAML file at all. For example, if you were asked to create a pod or deployment with specific name and image you can simply run the kubectl run command.

Use the below set of commands and try the previous practice tests again, but this time try to use the below commands instead of YAML files. Try to use these as much as you can going forward in all exercises

Reference (Bookmark this page for exam. It will be very handy):

<https://kubernetes.io/docs/reference/kubectl/conventions/>

**Create an NGINX Pod**

kubectl run --generator=run-pod/v1 nginx --image=nginx

**Generate POD Manifest YAML file (-o yaml). Don't create it(--dry-run)**

kubectl run --generator=run-pod/v1 nginx --image=nginx --dry-run -o yaml

**Create a deployment**

kubectl run --generator=deployment/v1beta1 nginx --image=nginx

**Generate Deployment YAML file (-o yaml). Don't create it(--dry-run)**

kubectl run --generator=deployment/v1beta1 nginx --image=nginx --dry-run -o yaml

**Generate Deployment YAML file (-o yaml). Don't create it(--dry-run) with 4 Replicas (--replicas=4)**

kubectl run --generator=deployment/v1beta1 nginx --image=nginx --dry-run --replicas=4 -o yaml

**Save it to a file - (If you need to modify or add some other details before actually creating it)**

kubectl run --generator=deployment/v1beta1 nginx --image=nginx --dry-run --replicas=4 -o yaml > nginx-deployment.yaml

Certification Tips - Imperative Commands with Kubectl

While you would be working mostly the declarative way - using definition files, imperative commands can help in getting one time tasks done quickly, as well as generate a definition template easily. This would help save considerable amount of time during your exams.

Before we begin, familiarize with the two options that can come in handy while working with the below commands:

--dry-run: By default as soon as the command is run, the resource will be created. If you simply want to test your command , use the --dry-run option. This will not create the resource, instead, tell you whether the resource can be created and if your command is right.

-o yaml: This will output the resource definition in YAML format on screen.

Use the above two in combination to generate a resource definition file quickly, that you can then modify and create resources as required, instead of creating the files from scratch.

#### **POD**

**Create an NGINX Pod**

kubectl run --generator=run-pod/v1 nginx --image=nginx

**Generate POD Manifest YAML file (-o yaml). Don't create it(--dry-run)**

kubectl run --generator=run-pod/v1 nginx --image=nginx --dry-run -o yaml

#### **Deployment**

**Create a deployment**

kubectl run --generator=deployment/v1beta1 nginx --image=nginx

Or the newer recommended way:

kubectl create deployment --image=nginx nginx

**Generate Deployment YAML file (-o yaml). Don't create it(--dry-run)**

kubectl run --generator=deployment/v1beta1 nginx --image=nginx --dry-run -o yaml

Or

kubectl create deployment --image=nginx nginx --dry-run -o yaml

**Generate Deployment YAML file (-o yaml). Don't create it(--dry-run) with 4 Replicas (--replicas=4)**

kubectl run --generator=deployment/v1beta1 nginx --image=nginx --dry-run --replicas=4 -o yaml

kubectl create deployment does not have a --replicas option. You could first create it and then scale it using the kubectl scale command.

**Save it to a file - (If you need to modify or add some other details)**

kubectl run --generator=deployment/v1beta1 nginx --image=nginx --dry-run --replicas=4 -o yaml > nginx-deployment.yaml

#### **Service**

**Create a Service named redis-service of type ClusterIP to expose pod redis on port 6379**

kubectl expose pod redis --port=6379 --name redis-service --dry-run -o yaml

(This will automatically use the pod's labels as selectors)

Or

kubectl create service clusterip redis --tcp=6379:6379 --dry-run -o yaml  (This will not use the pods labels as selectors, instead it will assume selectors as **app=redis.**[You cannot pass in selectors as an option.](https://github.com/kubernetes/kubernetes/issues/46191)So it does not work very well if your pod has a different label set. So generate the file and modify the selectors before creating the service)

**Create a Service named nginx of type NodePort to expose pod nginx's port 80 on port 30080 on the nodes:**

kubectl expose pod nginx --port=80 --name nginx-service --dry-run -o yaml

(This will automatically use the pod's labels as selectors, [but you cannot specify the node port](https://github.com/kubernetes/kubernetes/issues/25478). You have to generate a definition file and then add the node port in manually before creating the service with the pod.)

Or

kubectl create service nodeport nginx --tcp=80:80 --node-port=30080 --dry-run -o yaml

(This will not use the pods labels as selectors)

Both the above commands have their own challenges. While one of it cannot accept a selector the other cannot accept a node port. I would recommend going with the `kubectl expose` command. If you need to specify a node port, generate a definition file using the same command and manually input the nodeport before creating the service.

**Reference:**

<https://kubernetes.io/docs/reference/kubectl/conventions/>

* [Overview](https://www.udemy.com/course/certified-kubernetes-administrator-with-practice-tests/learn/lecture/15018998#overview)
* [Q&A](https://www.udemy.com/course/certified-kubernetes-administrator-with-practice-tests/learn/lecture/15018998#questions)
* [Bookmarks](https://www.udemy.com/course/certified-kubernetes-administrator-with-practice-tests/learn/lecture/15018998#bookmarks)
* [Announcements](https://www.udemy.com/course/certified-kubernetes-administrator-with-practice-tests/learn/lecture/15018998#announcements)

A quick note on editing PODs and Deployments

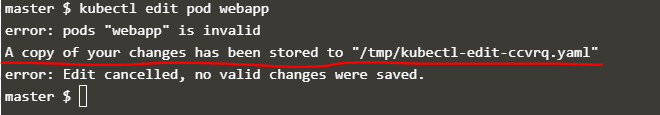
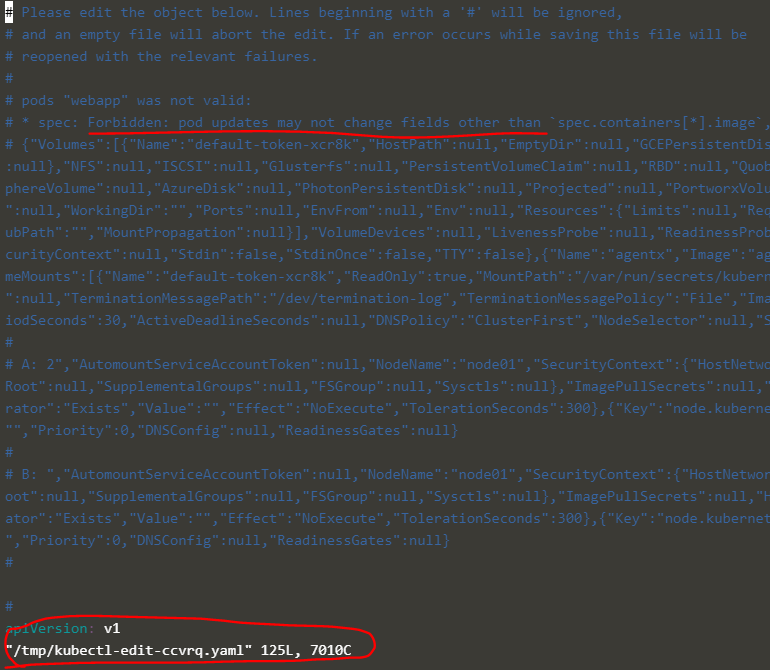
#### **Edit a POD**

Remember, you CANNOT edit specifications of an existing POD other than the below.

* spec.containers[\*].image
* spec.initContainers[\*].image
* spec.activeDeadlineSeconds
* spec.tolerations

For example you cannot edit the environment variables, service accounts, resource limits (all of which we will discuss later) of a running pod. But if you really want to, you have 2 options:

1. Run the kubectl edit pod <pod name> command.  This will open the pod specification in an editor (vi editor). Then edit the required properties. When you try to save it, you will be denied. This is because you are attempting to edit a field on the pod that is not editable.



A copy of the file with your changes is saved in a temporary location as shown above.

You can then delete the existing pod by running the command:

kubectl delete pod webapp

Then create a new pod with your changes using the temporary file

kubectl create -f /tmp/kubectl-edit-ccvrq.yaml

2. The second option is to extract the pod definition in YAML format to a file using the command

kubectl get pod webapp -o yaml > my-new-pod.yaml

Then make the changes to the exported file using an editor (vi editor). Save the changes

vi my-new-pod.yaml

Then delete the existing pod

kubectl delete pod webapp

Then create a new pod with the edited file

kubectl create -f my-new-pod.yaml

#### **Edit Deployments**

With Deployments you can easily edit any field/property of the POD template. Since the pod template is a child of the deployment specification,  with every change the deployment will automatically delete and create a new pod with the new changes. So if you are asked to edit a property of a POD part of a deployment you may do that simply by running the command

kubectl edit deployment my-deployment

* [Overview](https://www.udemy.com/course/certified-kubernetes-administrator-with-practice-tests/learn/lecture/14937592#overview)
* [Q&A](https://www.udemy.com/course/certified-kubernetes-administrator-with-practice-tests/learn/lecture/14937592#questions)
* [Bookmarks](https://www.udemy.com/course/certified-kubernetes-administrator-with-practice-tests/learn/lecture/14937592#bookmarks)
* [Announcements](https://www.udemy.com/course/certified-kubernetes-administrator-with-practice-tests/learn/lecture/14937592#announcements)

A note about Secrets!

Remember that secrets encode data in base64 format. Anyone with the base64 encoded secret can easily decode it. As such the secrets can be considered as not very safe.

The concept of safety of the Secrets is a bit confusing in Kubernetes. The [kubernetes documentation](https://kubernetes.io/docs/concepts/configuration/secret" \t "_blank) page and a lot of blogs out there refer to secrets as a "safer option" to store sensitive data. They are safer than storing in plain text as they reduce the risk of accidentally exposing passwords and other sensitive data. In my opinion it's not the secret itself that is safe, it is the practices around it.

Secrets are not encrypted, so it is not safer in that sense. However, some best practices around using secrets make it safer. As in best practices like:

* Not checking-in secret object definition files to source code repositories.
* [Enabling Encryption at Rest](https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/)for Secrets so they are stored encrypted in ETCD.

Also the way kubernetes handles secrets. Such as:

* A secret is only sent to a node if a pod on that node requires it.
* Kubelet stores the secret into a tmpfs so that the secret is not written to disk storage.
* Once the Pod that depends on the secret is deleted, kubelet will delete its local copy of the secret data as well.

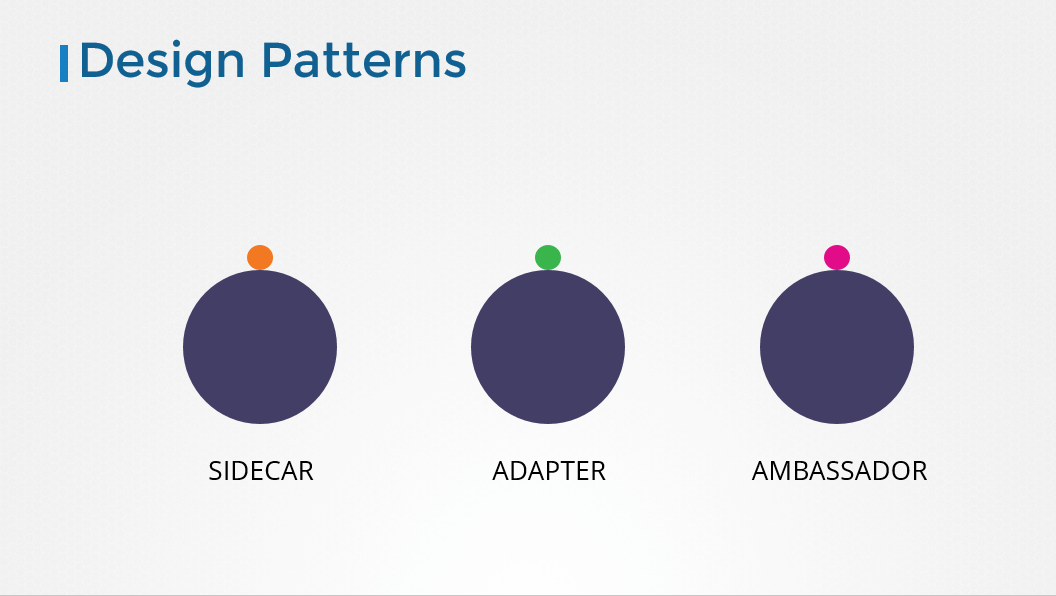
Read about the [protections](https://kubernetes.io/docs/concepts/configuration/secret/#protections)and [risks](https://kubernetes.io/docs/concepts/configuration/secret/#risks) of using secrets [here](https://kubernetes.io/docs/concepts/configuration/secret/#risks)

Having said that, there are other better ways of handling sensitive data like passwords in Kubernetes, such as using tools like Helm Secrets, [HashiCorp Vault](https://www.vaultproject.io/" \t "_blank). I hope to make a lecture on these in the future.

Multi-container PODs Design Patterns

There are 3 common patterns, when it comes to designing multi-container PODs. The first and what we just saw with the logging service example is known as a side car pattern. The others are the adapter and the ambassador pattern.

But these fall under the CKAD curriculum and are not required for the CKA exam. So we will be discuss these in more detail in the CKAD course.



InitContainers

In a multi-container pod, each container is expected to run a process that stays alive as long as the POD's lifecycle. For example in the multi-container pod that we talked about earlier that has a web application and logging agent, both the containers are expected to stay alive at all times. The process running in the log agent container is expected to stay alive as long as the web application is running. If any of them fails, the POD restarts.

But at times you may want to run a process that runs to completion in a container. For example a process that pulls a code or binary from a repository that will be used by the main web application. That is a task that will be run only  one time when the pod is first created. Or a process that waits  for an external service or database to be up before the actual application starts. That's where **initContainers**comes in.

An **initContainer**is configured in a pod like all other containers, except that it is specified inside a initContainers section,  like this:

1. apiVersion: v1
2. kind: Pod
3. metadata:
4. name: myapp-pod
5. labels:
6. app: myapp
7. spec:
8. containers:
9. - name: myapp-container
10. image: busybox:1.28
11. command: ['sh', '-c', 'echo The app is running! && sleep 3600']
12. initContainers:
13. - name: init-myservice
14. image: busybox
15. command: ['sh', '-c', 'git clone <some-repository-that-will-be-used-by-application> ; done;']

When a POD is first created the initContainer is run, and the process in the initContainer must run to a completion before the real container hosting the application starts.

You can configure multiple such initContainers as well, like how we did for multi-pod containers. In that case each init container is run **one at a time in sequential order**.

If any of the initContainers fail to complete, Kubernetes restarts the Pod repeatedly until the Init Container succeeds.

1. apiVersion: v1
2. kind: Pod
3. metadata:
4. name: myapp-pod
5. labels:
6. app: myapp
7. spec:
8. containers:
9. - name: myapp-container
10. image: busybox:1.28
11. command: ['sh', '-c', 'echo The app is running! && sleep 3600']
12. initContainers:
13. - name: init-myservice
14. image: busybox:1.28
15. command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservice; sleep 2; done;']
16. - name: init-mydb
17. image: busybox:1.28
18. command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sleep 2; done;']

Read more about initContainers here. And try out the upcoming practice test.

<https://kubernetes.io/docs/concepts/workloads/pods/init-containers/>

Self Healing Applications

Kubernetes supports self-healing applications through ReplicaSets and Replication Controllers. The replication controller helps in ensuring that a POD is re-created automatically when the application within the POD crashes. It helps in ensuring enough replicas of the application are running at all times.

Kubernetes provides additional support to check the health of applications running within PODs and take necessary actions through Liveness and Readiness Probes. However these are not required for the CKA exam and as such they are not covered here. These are topics for the Certified Kubernetes Application Developers (CKAD) exam and are covered in the CKAD course.

References

<https://kubernetes.io/docs/concepts/overview/kubernetes-api/>

Here is a link to kubernetes documentation if you want to learn more about this topic (You don't need it for the exam though):

<https://github.com/kubernetes/community/blob/master/contributors/devel/sig-architecture/api-conventions.md>

<https://github.com/kubernetes/community/blob/master/contributors/devel/sig-architecture/api_changes.md>

References

<https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/#backing-up-an-etcd-cluster>

<https://github.com/etcd-io/etcd/blob/master/Documentation/op-guide/recovery.md>

<https://www.youtube.com/watch?v=qRPNuT080Hk>