

Resource Creation Tips for the Kubernetes CKA / CKD Certification Exam



John Tucker

Jun 10, 2020 · 6 min read

In addition to having to understand the relevant Kubernetes concepts, one needs to be able to quickly create Kubernetes resources to be successful in the exam. Here are some tips that I have picked up as part of my preparations.



Apparently, there were a number of changes in the imperative resource creation commands between versions 1.17.X and 1.18.X; this article is written against version 1.18.3 and as such is aligned with the current exams.

Also, I cannot recommend enough the practice exams available at [CKA CKAD Simulator](#). Before using the simulator, I did not appreciate the need to be able to quickly create resources.

The first time-saver is simply creating an alias for the *kubectl* command at the start of the exam.

```
alias k="kubectl"
```

While this seemed unnecessary at first, after typing `kubectl` hundreds of times during a practice exam, I learned that this is indeed quite important.

Resources

Some (but not all) resources can be created using the *create* command; the full list of resource kinds that can be created can be seen using the following command:

```
k create --help
```

note: Some kinds of resources, e.g., DaemonSet will require copying a sample configuration from the Kubernetes documentation into a file, updating it, and applying it.

Then for each resource kind, one can get the specific options by using `--help`, e.g.,:

```
k create configmap --help
```

The bottom of the help output provides the required and common optional options (the top provides the complete list of options):

```
kubectl create configmap NAME [--from-file=[key=]source]
[--from-literal=key1=value1] [--dry-run=server|client|none]
[options]
```

note: In some cases, options can be repeated. For example, the ConfigMap `--from-literal=key1=value1` option can be repeated.

As not all resource configuration values can be set using the command line options, it is common to use the `--dry-run=client -o yaml` options with the *create* command to output the resource configuration into a file; where it can be then updated before applying it (with `k apply -f my-resource.yaml`).

Other benefits of using configuration files are:

- Resources can be deleted using the configuration file, e.g., with the `k delete -f my-resource.yaml` command
- Resources configuration files can also be later updated and re-applied

Another big time-saver is to set a shell variable at the start of the exam as follows:

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

and then use it as in the following example:

```
k create configmap example --from-literal=a=apple $do > my-resource.yaml
```

One important observation is that the generated configuration files do not include the `.metadata.namespace` key / value; even if you supply the `-n my-namespace` option. So, if you need to create resources in a Namespace other than *default*, you can do either:

- Add the `.metadata.namespace` key / value to the configuration file
- Supply the namespace when applying the configuration file, e.g., `k apply -f my-resource.yaml -n my-namespace`

note: There is also a way to set the namespace used for your current configuration context; `k config set-context --current --namespace=my-namespace`. But, I choose to not use this approach as I would undoubtedly forget what namespace is currently being used.

Pods

You may have noticed that you cannot create Pods using the *create* command and yet creating pods is a common activity. There is another command with numerous options to do this:

```
k run --help
```

The bottom of the help output provides the required and common optional options:

```
kubect1 run NAME --image=image [--env="key=value"] [--port=port]
[--dry-run=server|client] [--overrides=inline-json] [--command] --
[COMMAND]
[args...] [options]
```

note: The *env* option can be repeated but the *port* option cannot. The *overrides* option is more obscure (I have never used it).

The top of the help output provides the complete list of optional options; some of which are more commonly used:

- `--expose=true` : Will also create a Service exposing the Pod
- `--labels=a=apple,b=banana` : Create the Pod with the supplied label
- `--limits=cpu=200m,memory=512Mi` : Create the Pod with supplied limits
- `--requests=cpu=100m,memory=256Mi` : Create the Pod with supplied requests
- `--restart=Never` : Create the Pod that never restarts containers
- `--serviceaccount=my-sa` : Create the Pod with the supplied Service Account name

The following options are not to be used when performing a dry run but are useful in special cases:

- `-it` : A combination of the `-i` and `-t` options; can be used to attach to the container, e.g., displays its standard output
- `--rm` : Wait for the Pod to succeed / fail and then delete it

For example, one common scenario of creating a Pod without a dry run is to hit a URL from a web browser in a container, e.g.:

```
k run tmp --image=busybox --restart=Never -it --rm -- wget -O- -T 3
https://www.google.com
```

note: The `wget -O-` option redirects output to standard out; the `-T 3` option sets the timeout to three seconds.

As you likely observed, you will likely have to modify the generated Pod configuration file before applying it. Here are some common scenarios:

- Specify the container's *name*
- Create a Pod with multiple containers

Other scenarios require adding new keys; often by copying snippets from the Kubernetes documentation. There are a number of such cases, e.g., adding Pod Volumes, i.e., *spec.volumes*.

Pods can be deleted using the configuration file as we described earlier. However it is useful to use the `--force=true` option to avoid having to wait up to ten seconds for a graceful deletion (during the exam, every second counts). For example:

```
k delete -f my-resource.yaml --force=true
```

In general, Kubernetes does not allow one to update a Pod. There are some exceptions; as one can see from a representative error if one tries to incorrectly update a Pod:

```
The Pod "nginx" is invalid: spec: Forbidden: pod updates may not
change fields other than `spec.containers[*].image`,
`spec.initContainers[*].image`, `spec.activeDeadlineSeconds` or
`spec.tolerations` (only additions to existing tolerations)
```

To keep things simple, however, it is easier to simply delete the Pod and recreate it if it needs updating.

Deployments

Unlike many of the other *create* commands (and the *run* command for creating Pods), there are few options available when creating Deployments as you can see by running:

```
k create deployment --help
```

The bottom of the help output provides the required and pretty much only common option:

```
kubectl create deployment NAME --image=image [--dry-run=server|client|none]
[options]
```

As such, you will likely have to modify the generated Deployment configuration file before applying it. Here are some common scenarios:

- Replicas, i.e., *spec.replicas*
- Pod template, i.e., *spec.template*

One trick for modifying the Deployment's template is to also generate a Pod configuration file (see *Pods* above) and reading it into the Deployment configuration file.

note: You will need to update the imported Pod's labels to match the Deployment's selector. You will also need to delete its name.

Services

While one can create Services using the *create* command, it is much quicker to use the *expose* command in conjunction with running Pod or Deployment. This is because this command will automatically create the Service label selector based on the Pod or Deployment.

note: As you may recall, we can also create a Service exposing a Pod at Pod creation time with the `--expose=true` option.

Running the following command gives us the command options:

```
k expose --help
```

The bottom of the help output provides the required and common optional options (the top provides the complete list of options):

```
kubectl expose (-f FILENAME | TYPE NAME) [--port=port]
[--protocol=TCP|UDP|SCTP] [--target-port=number-or-name] [--
name=name]
[--external-ip=external-ip-of-service] [--type=type] [options]
```

For example, to create a NodePort service called *my-service* on port 80 targeting a deployment called *nginx* running an *nginx* container (exposes port 80) we can execute:

```
k expose deployment nginx --port=80 --target=port=80 --name=my-
service --type=NodePort
```

Wrap-Up

Hopefully you found this helpful. Good luck on your exams.

[Kubernetes](#) [Cka](#) [Ckad](#) [Exam](#)

[About](#) [Help](#) [Legal](#)

Get the Medium app

