Certified Kubernetes Administrator (CKA) — Tips and Tricks — Part 1

[](https://medium.com/@imarunrk?source=post_page-----2e98e9b31de4----------------------)

[Arun Ramakani](https://medium.com/@imarunrk?source=post_page-----2e98e9b31de4----------------------)



Certified Kubernetes Administrator is a challenging exam by CNCF. Unlike many other certifications, it’s a practical lab. It took two attempts for me to get through. Thought sharing my experience will help you all in passing with flying colors. There are many aspects to clearing the exam with ease, one of the aspects is “Time Management”. We will look into time management today and look at the other aspects in an upcoming article.

You are given 24 questions within 3 hours, some of the questions may be tricky and eat a lot of your time. You have to score 76 and above to clear the exam. This means you are supposed to attend almost all the questions, with limited mistakes. Let’s see, how to get your time management correct.

**Tip 1: Create Aliases**

kubectl is the most-used tool throughout the exam. You will be spending a lot of time in typing kubectl command. It is worth to create aliases to use time more efficient and effective. On my first attempt, I answered only 20 out of 24 questions due to inefficient time management. That atleast saved me 15 min on my second attempt by creating following alias.

# Get resources

alias k=”kubectl”  
alias kn=”kubectl get nodes -o wide”  
alias kp=”kubectl get pods -o wide”  
alias kd=”kubectl get deployment -o wide”  
alias ks=”kubectl get svc -o wide”

# Describe K8S resources

alias kdp=”kubectl describe pod”  
alias kdd=”kubectl describe deployment”  
alias kds=”kubectl describe service”  
alias kdn=”kubectl describe node”

**Tip 2: Finish the easy once first**

On my first attempt, I got stuck with a tricky question on static pods. ( we will talk about it in detail in the next blog) . It took almost 30 min on that question still, I got it wrong at the end. You can’t waste your time like this. If you are struck more than 10 min on a question, jump to the next one. You can always come back and re-attempt at the end. Attempting all the low hanging fruits will help gain quick confidence.

**Tip 3: Don’t fight with YAML’s**

Kubernetes is all about YAML’s and typing it all yourself is a hell. It’s very easy to forget some attributes in YAML or get an alignment issue. This is not a good way to spend your time. Always use YAML generators to get your YAML generated.

Generate pod yaml with below command

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

Generate deployment yaml with below command

kubectl create deploy nginx — image=nginx — dry-run -o yaml > nginx-ds.yaml

Generate service yaml with below command

kubectl expose pod hello-world — type=NodePort — name=example-service

kubectl expose deployment hello-world — type=NodePort — name=example-service

You can also generate YAML from existing resources from the cluster and then edit as you need nor your work. For example, if you are trying to create a deployment and there is one already running, try using:

*kubectl get deployment “deployment name” -n “namespace” -o yaml > “new-deployment.yaml*

This will give you a working deployment yaml that you can edit and apply to make sure you’ve got one working.

**Tip 4: Reuse YAML’s**

You will need to create pod and deployment resources again and again. Once yaml is generated for a pod or deployment, we can easily reuse the same yaml for different questions with small modifications.

*cp pod1.ymal pod2.yaml*

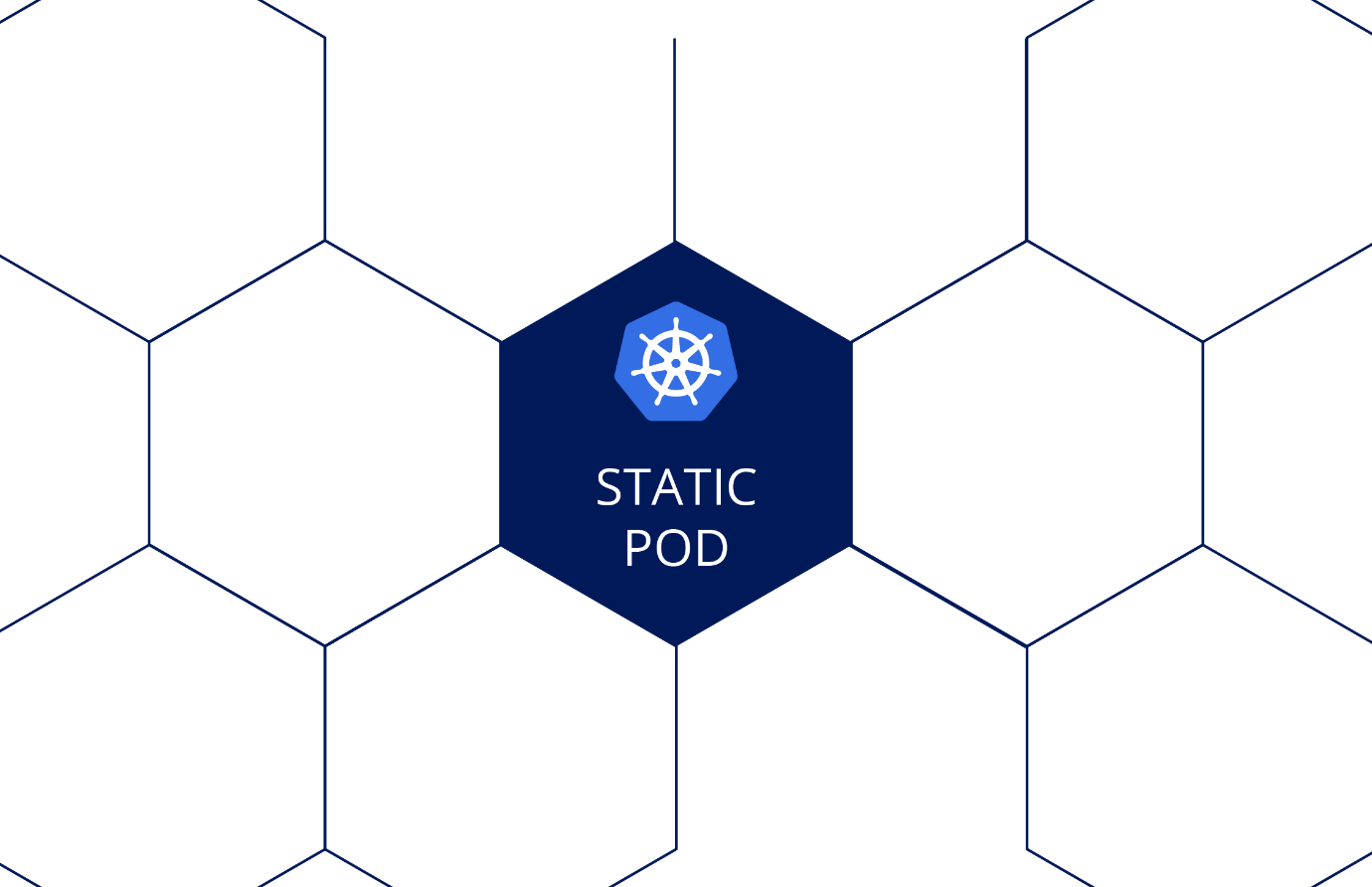
Some may feel using the Tip 3 is better than tip 4. Try and determine which is faster for you.

Certified Kubernetes Administrator (CKA) — Tips and Tricks — Part 2

Today let’s see about the static pod. There is a good weight for static pods and there is a high chance of it being a part of your exam. I did not get it correct on the first go and hence writing this blog.

**Challenge 1: Recognizing the Question**

The first challenge is to recognize the question as a static pod question. There may not be an explicit mention of the static pod, you will be asked to create a pod in a specific node by placing the pod definition file in a particular folder “/etc/kubernetes/manifests”. This is the folder where generally static pod definitions are kept. With these hints, you should recognize the question is all about static pods.



**Challenge 2: Getting your YMAL Generator working**

If you SSH into the node and tried using YAML generator as below

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

It will not work as we are within a cluster on a specific node, kubeconfig on the specific node can’t connect to the cluster. The trick is to generate YAML before we SSH to the specific node, then copy the YAML into the exam notepad to use it after SSH.

**Challenge 3: Make the node pick up the YAML**

Once we keep the YAML definition into the folder “/etc/kubernetes/manifests” I expected it to work automatically, it did not work. The reason is static pod configuration is not done on the specific node kubelet by default. To make the static pod working, kubelet configuration file should have “staticPodPath”. Following steps will help to get the static pod up and running

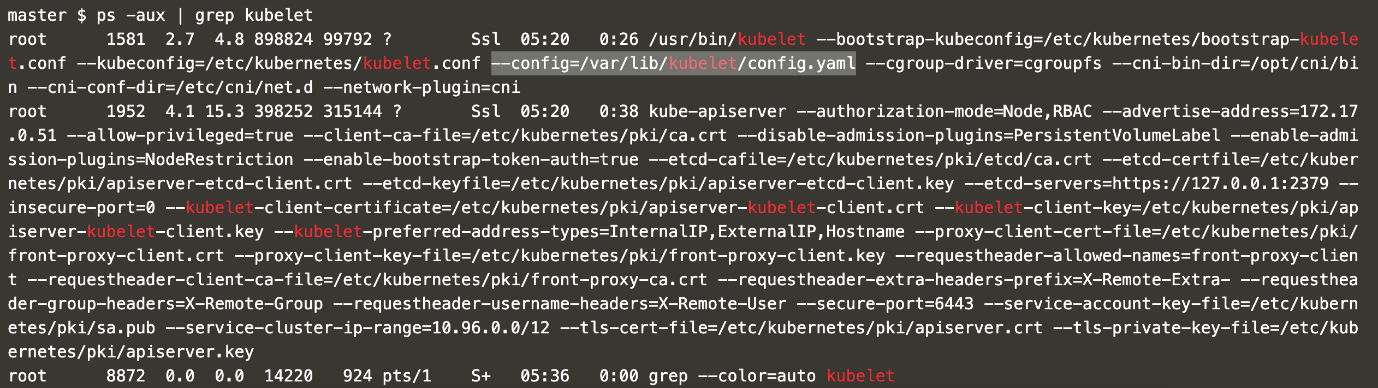
SSH to the node: “ssh my-node1”

Gain admin privileges to the node: “sudo -i”

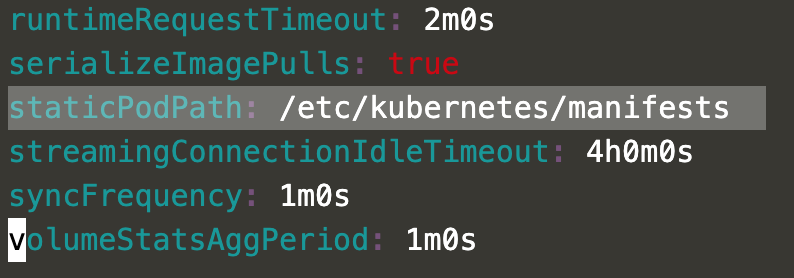
Move to the manifest-path “cd /etc/kubernetes/manifests”

Place the generated YAML into the folder “vi nginx.yaml”

Find the kubelet config file path “ps -aux | grep kubelet” . This will output information on kubelet process. Locate the kubelet config file location as highlighted below



6. Edit the config file “vi /var/lib/kubelet/config.yaml” to add staticPodPath as highlighted below



7. Finally, restart the kubelet “systemctl restart kubelet”

You are all done. The static pod will be running on the specific node.

Certified Kubernetes Administrator (CKA) — Tips and Tricks — Part 3

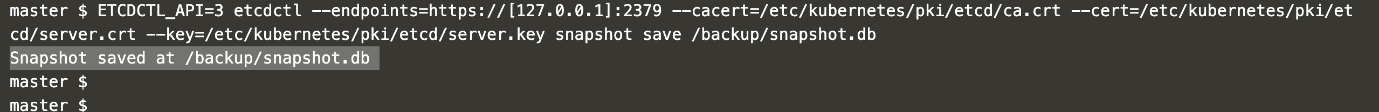
Today let’s look into ETCD backup. If you get this question it’s a jackpot. You can score the full mark in less than a minute, if you know how to do it. This will save some time for other questions. ETCD back up is a one-line command, but you need to collect a few pieces of information needed to execute the command.

**Tip 1: Parts of the**ETCD**Backup Command**

To back up the cluster, we should use the below command

ETCDCTL\_API=3 etcdctl — endpoints=[ENDPOINT] — cacert=[CA CERT] — cert=[ETCD SERVER CERT] — key=[ETCD SERVER KEY] snapshot save [BACKUP FILE NAME]

Executing the command will immediately give feedback if the backup is taken correctly. In case if you have not got the command correctly, you will have immediate feedback of failure.



The above instruction has 6 important parts to it

Command to take a backup — See **Tip 2**on how to escape memorizing

ENDPOINT — See **Tip 3**on how to get this value

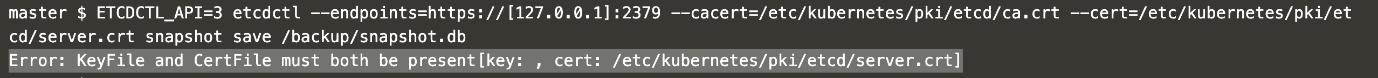
CA CERT — See **Tip 3**on how to get this value

ETCD SERVER CERT — See **Tip 3**on how to get this value

ETCD SERVER KEY — See **Tip 3**on how to get this value

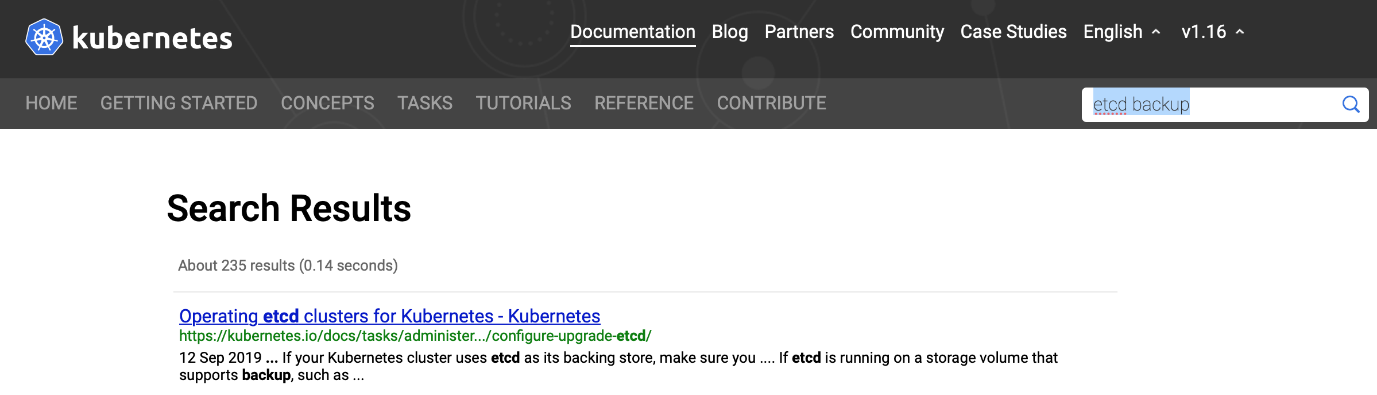
BACKUP FILE NAME — This will be given as a part of question itself

Any missing options will throw an error

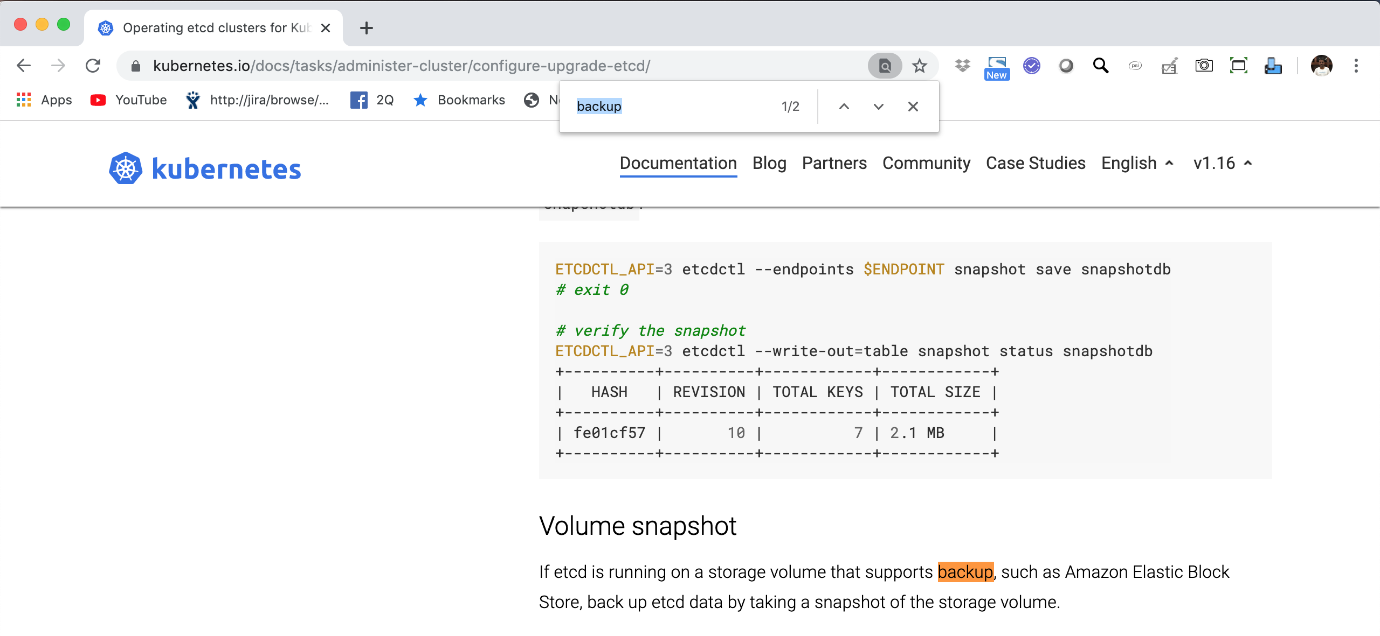


**Tips 2: No Need to Memorize the Command**

You don’t need to memorize the command for backing up ETCD. You will be allowed to refer to the Kubernetes documentation page during the exam. From the Kubernetes documentation page ([doc page](https://kubernetes.io/docs/home/) ) search for “etcd backup”, then from the results click the first link “Operating etcd clusters …”.

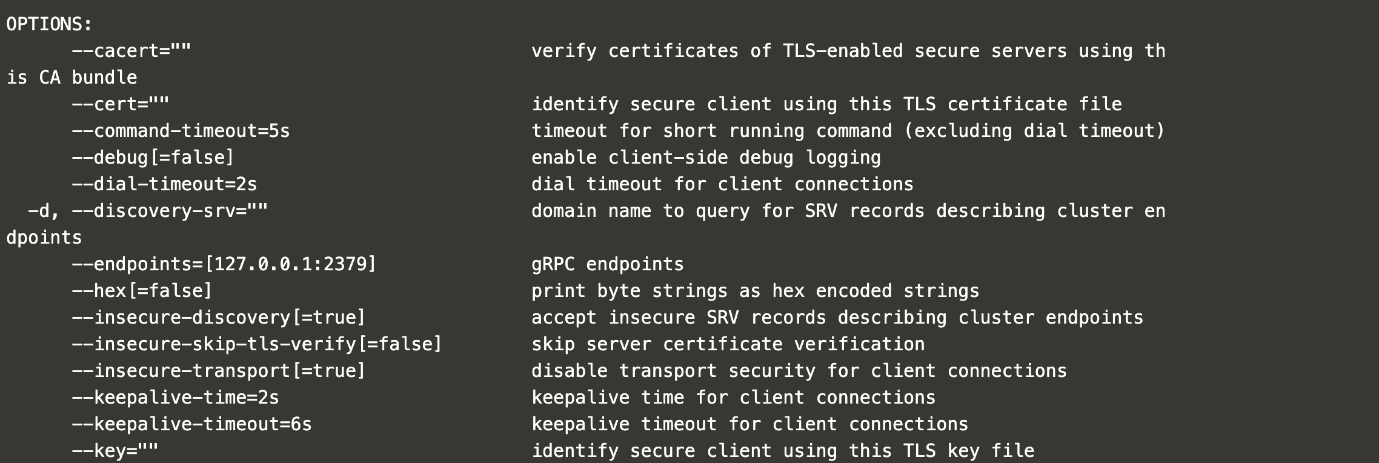


Look for the word “backup” in the resulting page, you will be able to locate the command for the backup.



ETCDCTL\_API=3 etcdctl — endpoints $ENDPOINT snapshot save snapshotdb

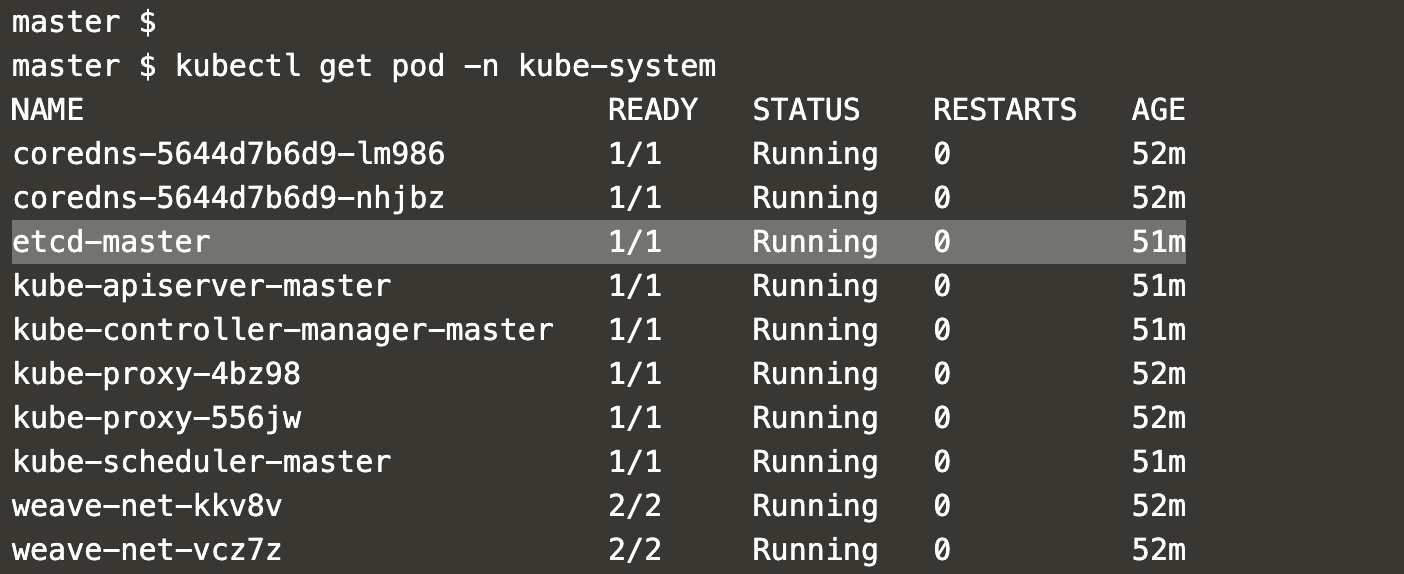
Now wait, this is not the full command that we saw in the beginning. There are some missing parts. Should I memorize the rest? No, run “ETCDCTL\_API=3 etcdctl help” you will see all the options, you can recognize the missed options here.



**Tips 3: Finding the Values**

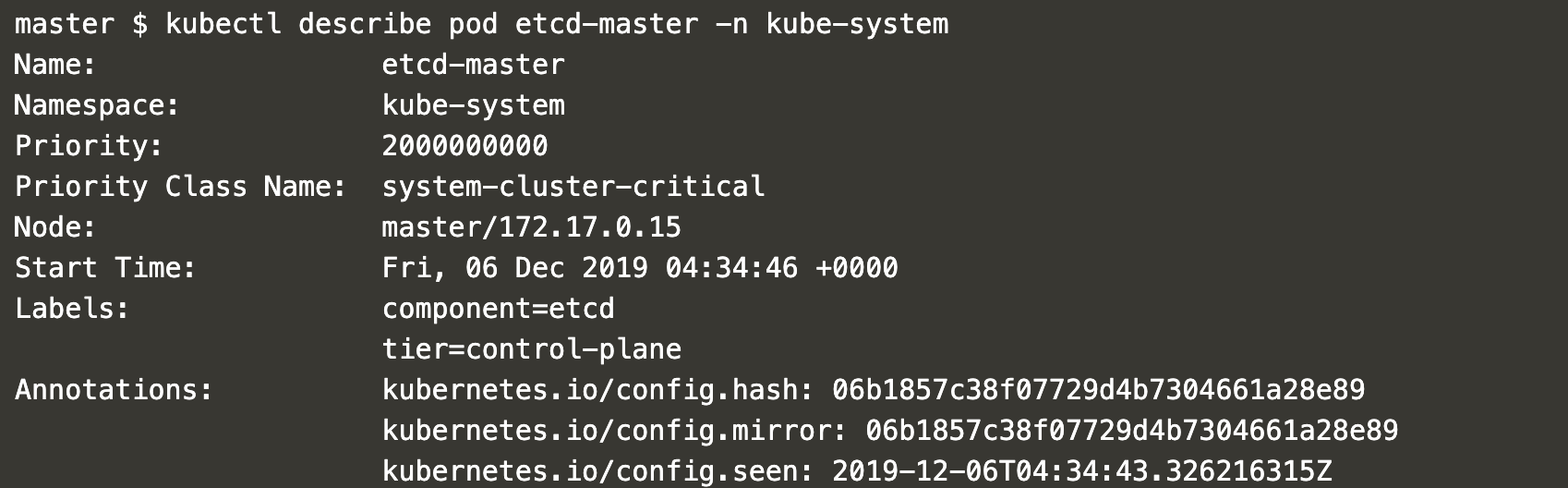
Exam cluster setup is done with kubeadm, this means ETCD used by the kubernetes cluster is coming from static pod. Confirm this by looking into pods in kube-system namespace.

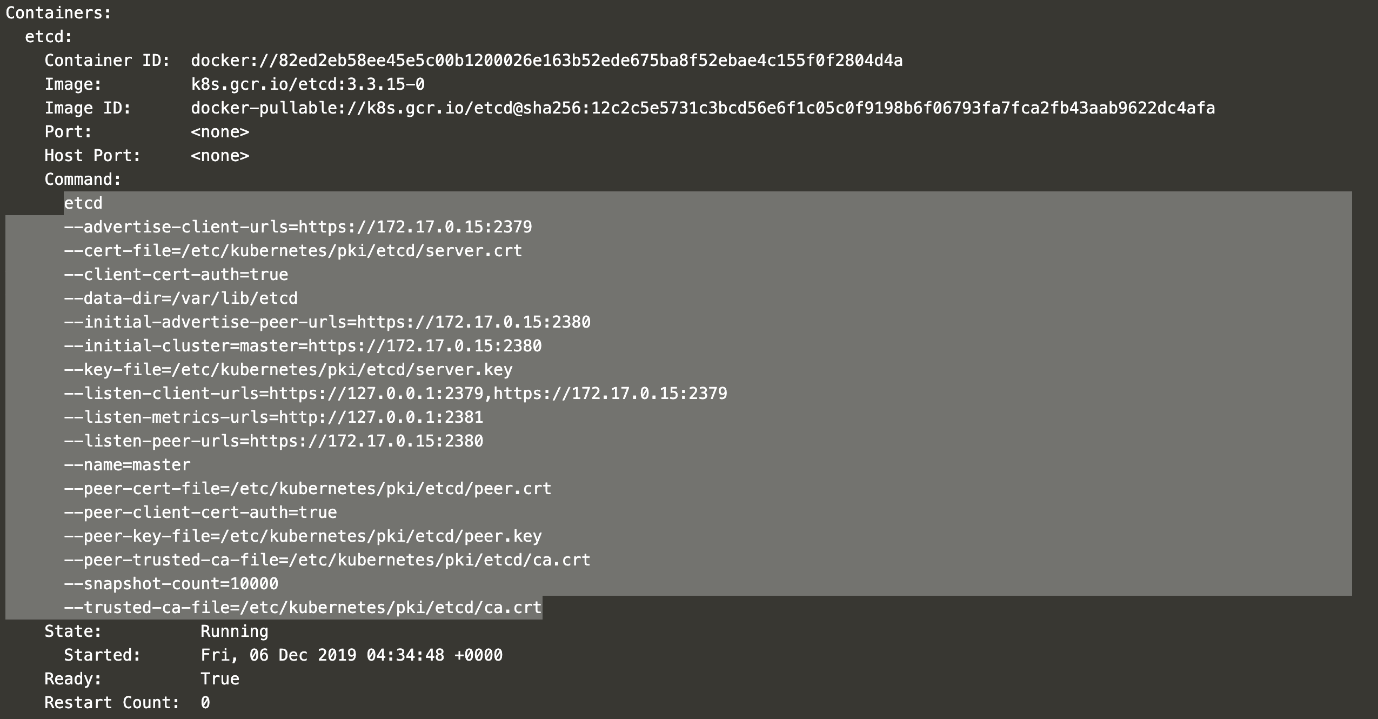
kubectl get pod -n kube-system



2. Once you recognize the pod in kube-system namespace, just describe the pod to see command line options from container section.

kubectl describe pod etcd-master -n kube-system





You can locate the information on

**endpoint:**— advertise-client-urls=[https://172.17.0.15:2379](https://172.17.0.15:2379/)

**ca** **certificate:**— trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt

**server certificate :** — cert-file=/etc/kubernetes/pki/etcd/server.crt

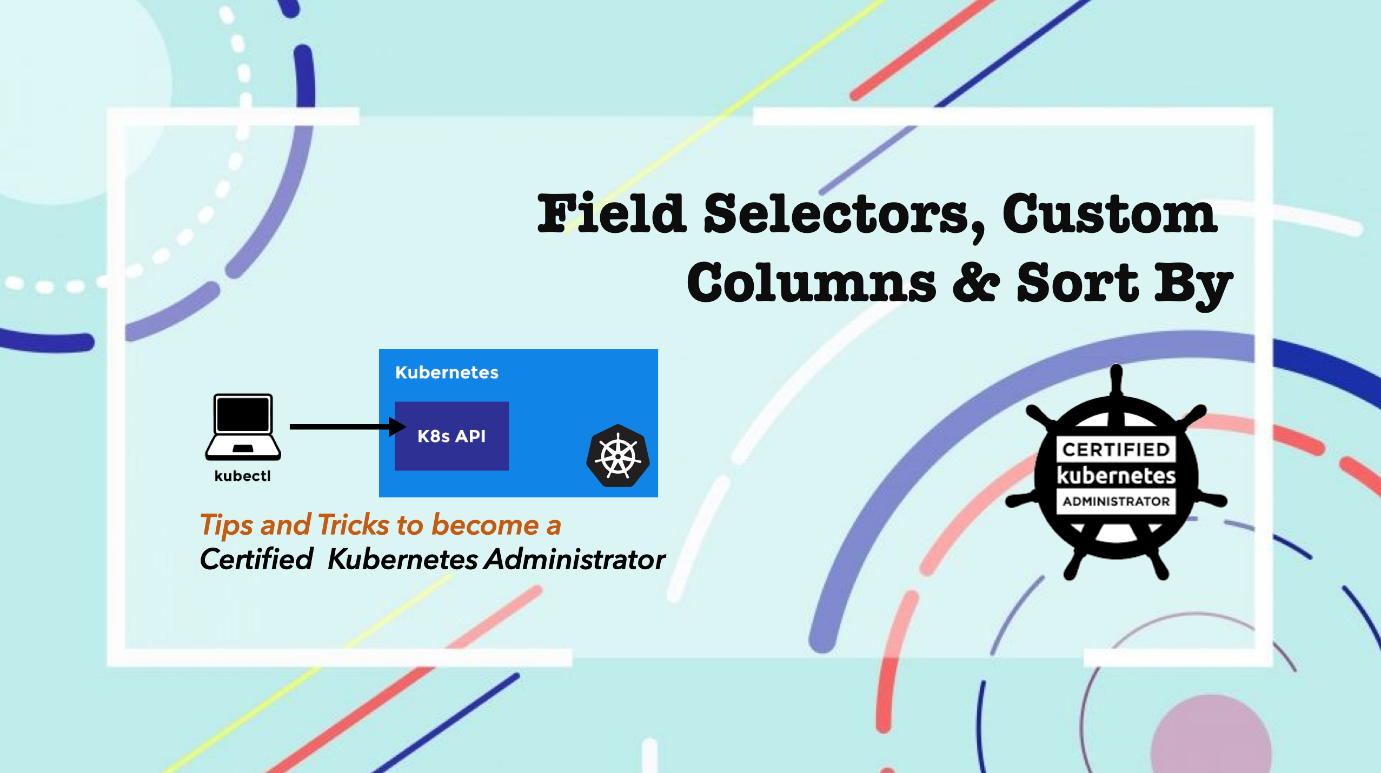
**key:**— key-file=/etc/kubernetes/pki/etcd/server.key

**Tips 4: Difference in Option Names [IMP]**

Please note that the command option name, you get from pod describes and actual “ETCDCTL\_API=3 etcdctl” are different.

You are all done. The ETCD back will be in the specified location.

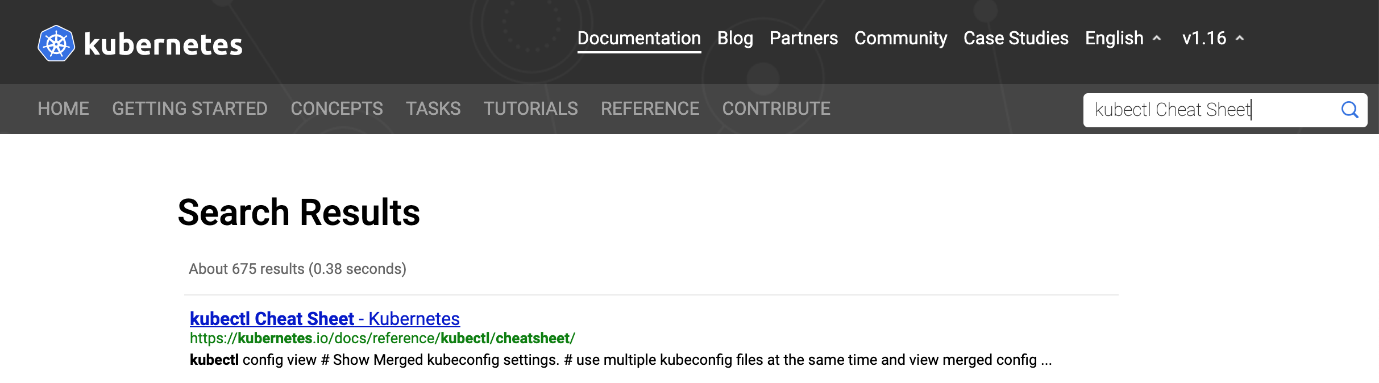
**Certified Kubernetes Administrator (CKA) — Tips and Tricks — Part 4**



Mastering “Field Selectors”, “Custom Column” and “Sort By” will help us in a few questions at the CKA exam. In this blog, we will look into a framework that will help us to construct any kubectl “Field Selectors”, “Custom Column” and “Sort By” commands without memorizing much. If you are lucky you will get two questions from the below examples.

**Tip 1: Reach Documentation And Escape Memorizing**

You will be allowed to refer the Kubernetes documentation page during the exam. From the Kubernetes documentation page ([*doc page*](https://kubernetes.io/docs/home/)) search for “kubectl Cheat Sheet” & “Custom Column” respectively, then from the results click the first link.





Scan through the resulting pages to get hold of the Field Selectors, Custom columns and Sort By commands.

kubectl get nodes -o jsonpath=’{.items[\*].status.addresses[?(@.type==”ExternalIP”)].address}’

*kubectl get services — sort-by=.metadata.name*

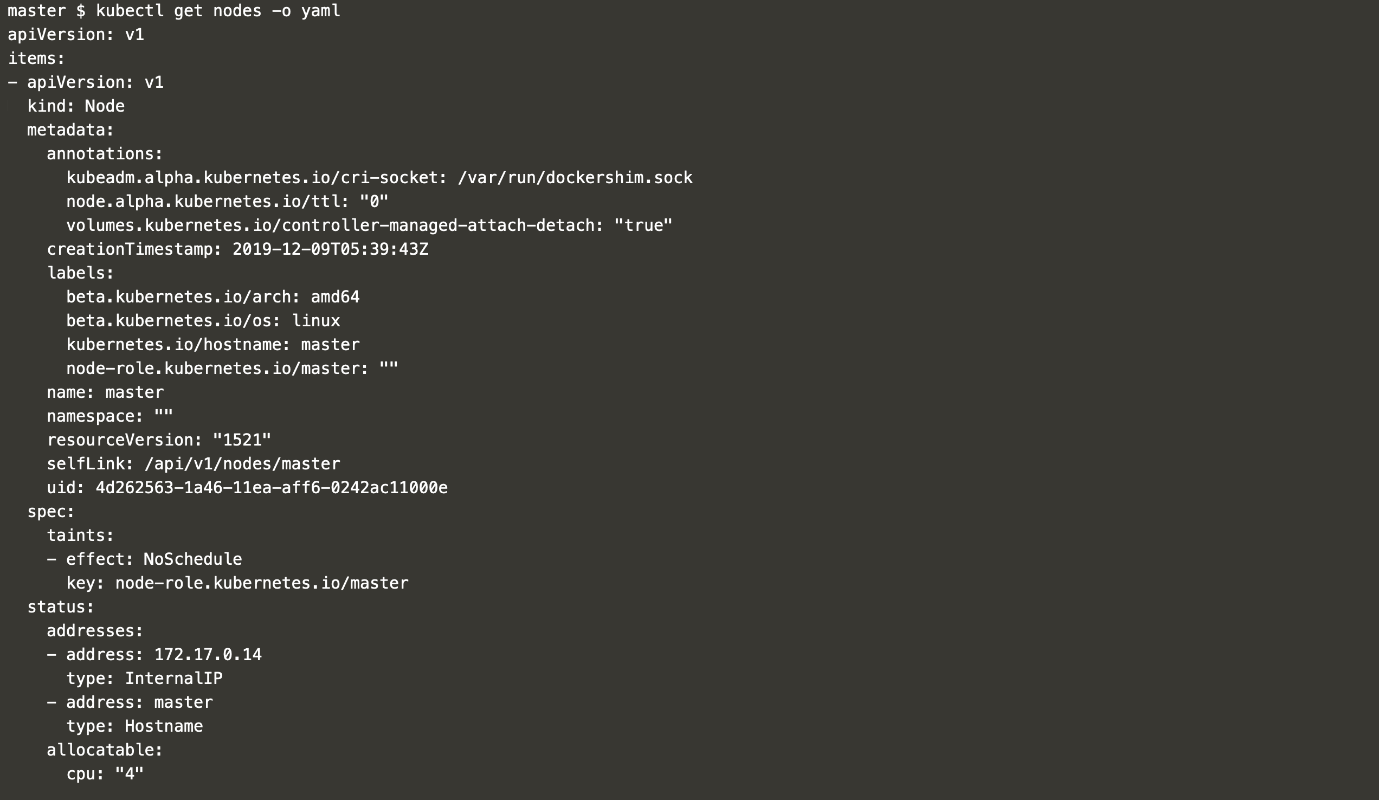
*kubectl get pods <pod-name> -o custom-columns=NAME:.metadata.name,RSRC:.metadata.resourceVersion*

**Tip 2: Looking at the output structure**

To start using the JSON path in all these commands, we should know the output structure from kubectl commands. You can either use “-o yaml” or “-o json” on any kubectl command to understand the structure. I prefer “-o yaml”, because the output from “-o json” is less readable than the output of “-o yaml”. You can choose the option which looks better readable for you.

*kubectl get node -o yaml*

*kubectl get node -o json*

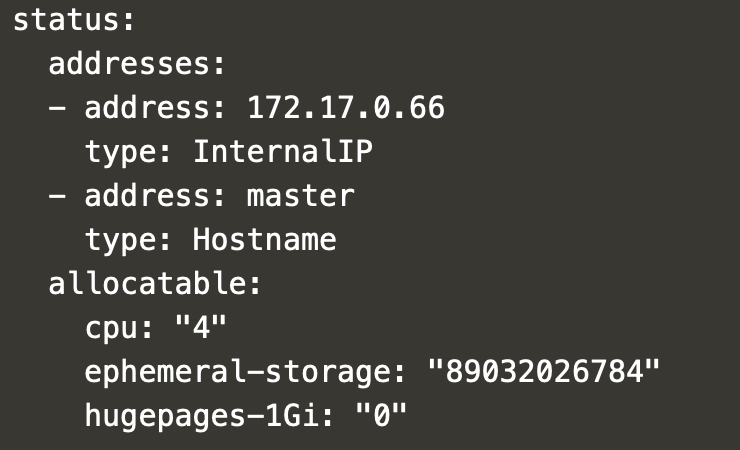


**Tip 3: Find Arrays and Map**

The next step is to know, how to find arrays and maps from the YAML structure. Concerning YAML, it’s the same as the YAML’s that we use to create any Kubernetes resource. The one that starts with “-” is an array. If you look at the below image

“addresses” is an Array

“allocatable” is a Map



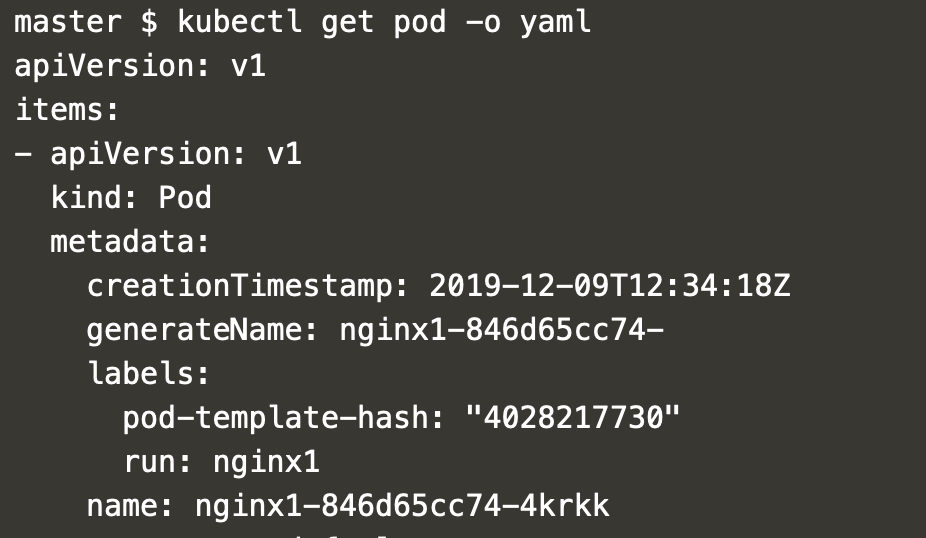
**Tip 4: Field Selectors**

Now we know the base command, how to look and interpret the output structure, the next step is to learn how to construct the given requirement into a kubectl command. We can look into two different examples moving from simple to complex scenarios. so, that we will be able to answer any exam question.

**Challenge 1:**List all pods name, lastProbeTime from status where the type is ready.

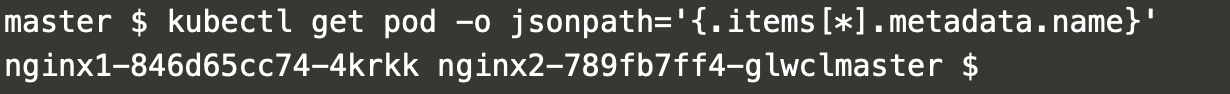
Execute “kubectl get pod -o yaml” and look for the needed fields in the YAML output.

“pods name:: items : Array → metadata : Map → name”



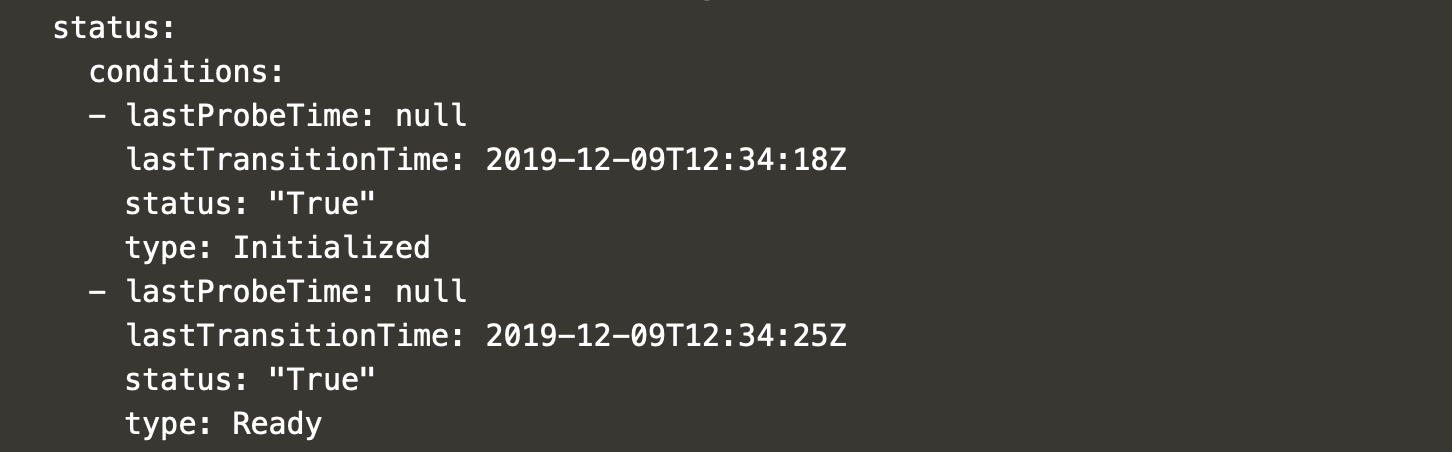
*Command below will give you all the pod names.*

*kubectl get pod -o jsonpath=’{.items[\*].metadata.name}’*



**Learning 1:**Use *[\*]*for getting all array items.

*lastTransitionTime:: items : Array → status : map → conditions : Array → lastProbeTime*



*Below command will give all the pod names and lastTransitionTime*

*kubectl get pod -o jsonpath=’{.items[\*].metadata.name}{.items[\*].status.conditions[?(@.type==”Ready”)].lastTransitionTime}’*



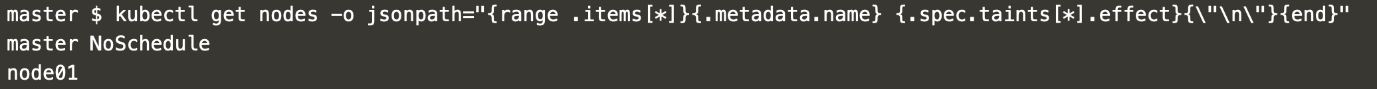
**Learning 2:**Use *[?(@.type==”Ready”)]*to apply “where condition” with the array.

“?” represents an if condition.

“@” represents the current element in the array.

**Challenge 2:**Get all schedulable nodes. Each node should be displayed in a new row.

*kubectl get nodes -o jsonpath=”{range .items[\*]}{.metadata.name} {.spec.taints[\*].effect}{\”\n\”}{end}”*



**Learning 3:**To display every node in an individual row, we have to use a loop to process node by node. Use *{range .items[\*]} …… {end}*to loop through the list of items.

**Learning 4:**Within the loop, directly refer the items from the looping node. For example, use *{.metadata.name} under {range .items[\*]} for*referring*name*. Should not use *{.items[\*].metadata.name}*.

**Learning 5:**At the end of every iteration use *{\”\n\”}*for new line.

*kubectl get nodes -o jsonpath=”{range .items[\*]}{.metadata.name} {.spec.taints[\*].effect}{\”\n\”}{end}” | grep -v NoSchedule*



**Learning 6:**Use “*grep -v NoSchedule” to exclude node with taint ‘*NoSchedule’.

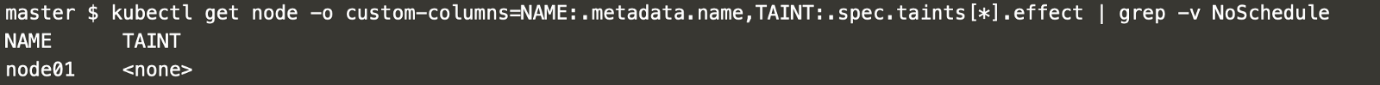
Note: “*grep -v” is inverse grep.*

**Tip 5: Custom Columns**

We can achieve the same requirement of challenge 2 with Custom Columns.

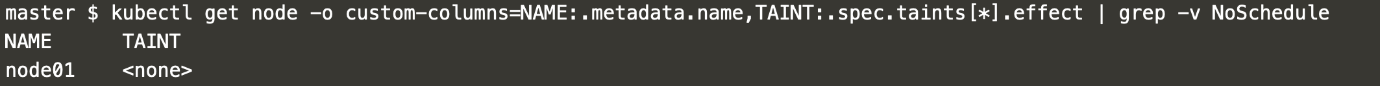
*Get all schedulable nodes. Each node should be displayed in a new row.*

*kubectl get node -o custom-columns=NAME:.metadata.name,TAINT:.spec.taints[\*].effect | grep -v NoSchedule*



**Learning 7:**For**“**custom columns” and “sort by”we will use the same yaml structure, excluding the outer “items array” (i.e exclude — .items[\*] )

**Learning 8:**Custom columns can have titles. NAME, TAINT in the above example are titles.



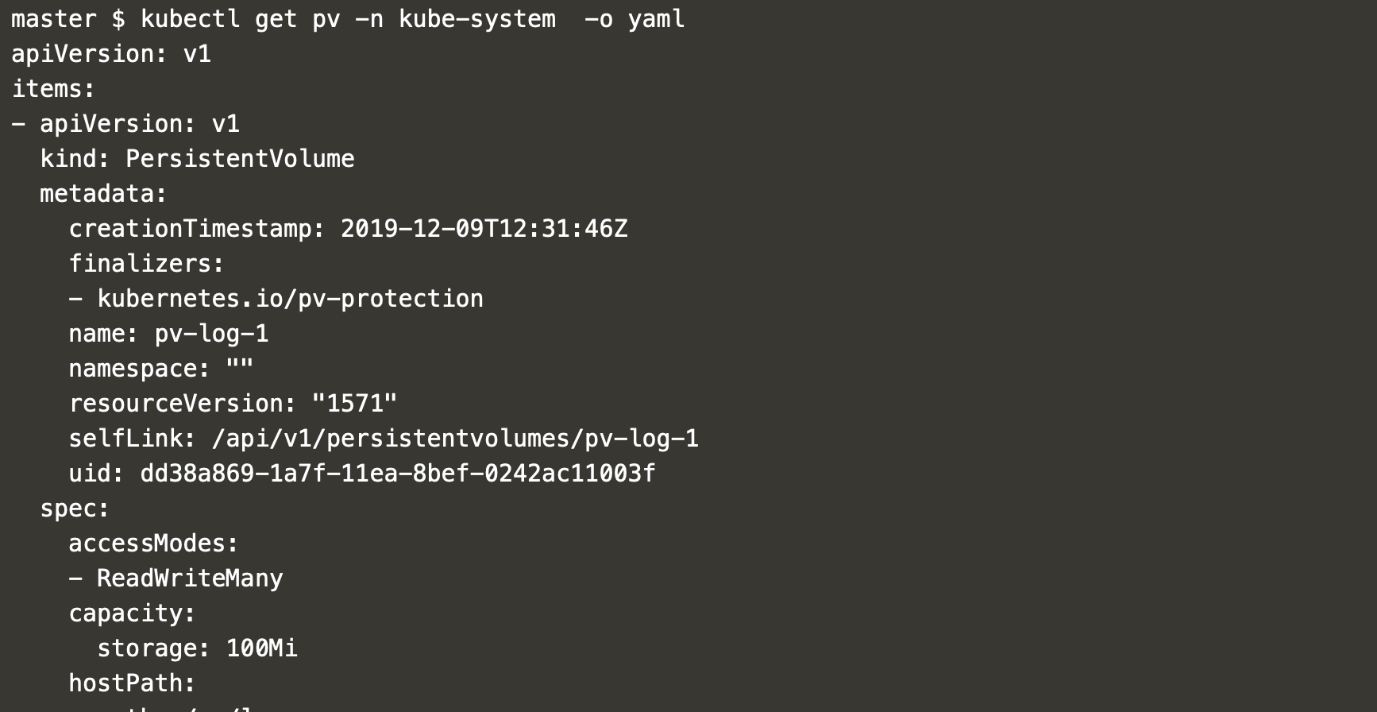
**Tip 6: Sort By**

Sort by will help us to order the output based on an attribute.

**Challenge 3:**Get all persistence volume from kube-system namespace ordered with capacity.

*Look at the json structure to locate the capacity.*

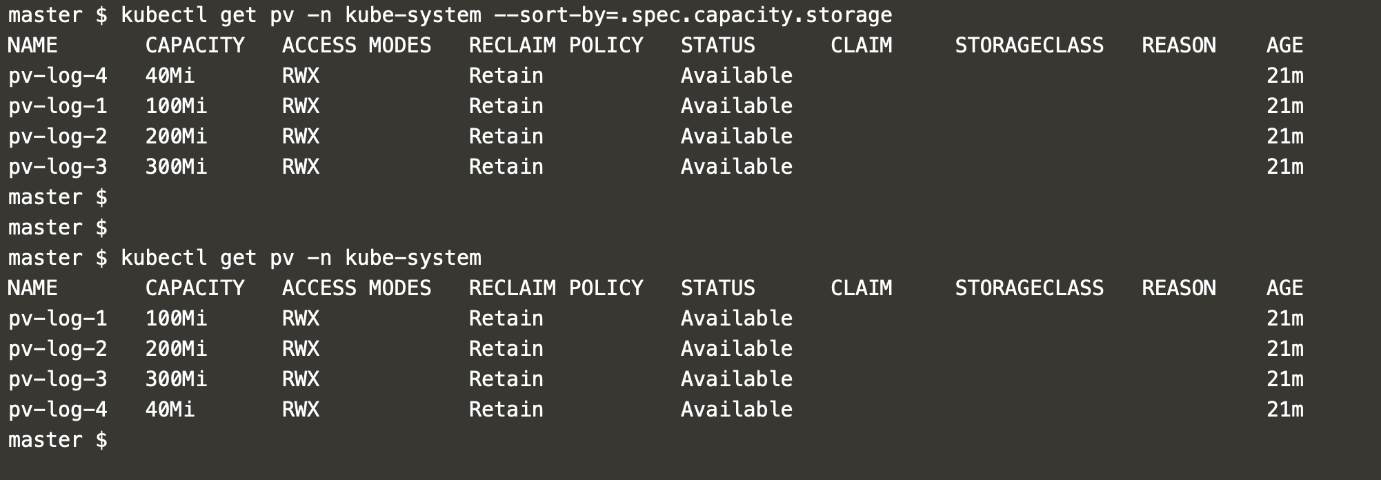
“*capacity*:: items : Array → spec : Map → capacity : storage”



*The command for both sorted and unsorted output,*

*kubectl get pv -n kube-system — sort-by=.spec.capacity.storage*

*kubectl get pv -n kube-system*



With this you should be able to handle any “Field Selectors”, “Custom Column” and “Sort By” related qestion.

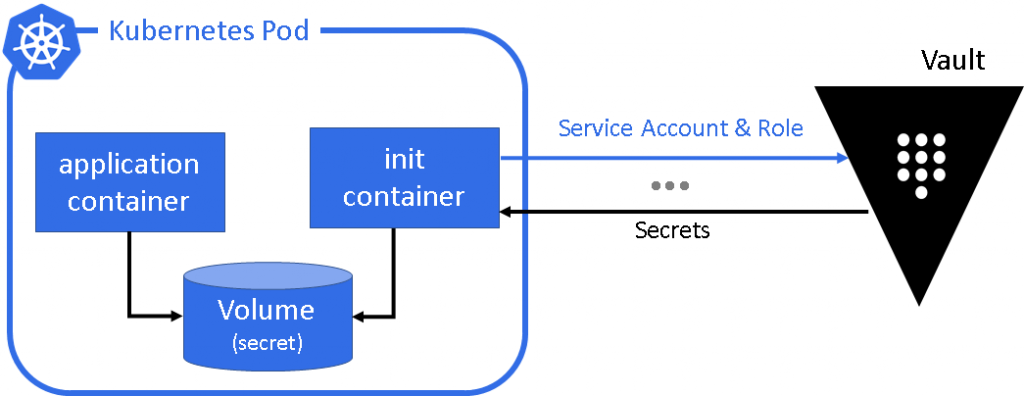
Certified Kubernetes Administrator (CKA) — Tips and Tricks — Part 5

**Init Container is the way to do some setup task before the actual container starts 🏄‍♀**

Init container is an important concept for the exam. There is a very high chance that this is one of your 24 questions. This blog will attempt to make you aware of the traps that you may get into and answer any form of init container question.

Init container containers are specialized containers that run before the normal containers in a Pod. Init containers generally contain setup scripts, that we are not able to make it as a part of our standard container. So when do we use init container? let’s look at a real-time example.

Init containers can be used to delay app container startup until a set of preconditions are met. Say we have to download a security key from the key vault, which we do not wish to make it as the past of our regular container for security reasons, then init container is the best choice. The below picture represents the above-mentioned scenarios.



The secret is downloaded and made available in a volume, for wich the main container will have access.

Some Facts

* Below are some of the facts that we should know about init container.
* We can have more than one init container in a pod
* Init Containers always run to completion
* Init Container executes in the order they are specified within the YAML definition
* Each Init Container must complete successfully before the next one starts
* If an init container fails, Kubernetes repeatedly restarts the Pod until the init container succeeds
* Init containers do not support readiness probes because they must run to completion before the Pod can be ready.

**Init Container YAML**

Init Container YAML config looks very similar to standard container. I will have a name, image name with optional command and volume mount details. Init container documentations which can be accessed during the exam are available [here](https://kubernetes.io/docs/concepts/workloads/pods/init-containers/)

**Init Container Image**

The best way to use init containers is, creating an image and adding the necessary script to the docker file to do your task. But you don’t have an exam environment to do this task during the exam.

The way we should do in the exam is to use a dummy image name and then add a command attribute with all the necessary script representing the task that we wish to achieve. So which dummy image I use? “busybox:1.28” come to the rescue. The above code snippet represents a simple task of init container, enabling sleep for a specified time, blocking the main container to start.

**Shared Volume**

For most of the requirements with Init Container, a shared volume between init container and application container is a key. In the exam, you may get a similar question involving shared volume. Let’s take a look at an example

*Example: We should use Init Container to create a file named “sharedfile.txt” under the “work” directory and the application container should check if the file exists and sleep for a while. If the file does not exist the application container should exit.*

Let have a full YAML view first and

**emptyDir: {} — For Shared Volume**

In the above example we are using “emptyDir: {}” for sharing volume between init container and application container. What is that ❓Documentation for the same is [here at the Kubernetes documentation page which](https://kubernetes.io/docs/concepts/storage/volumes/) you can use during the exam. As mentioned in the Kubernetes documentation

“An emptyDir volume is first created when a Pod is assigned to a Node, and exists as long as that Pod is running on that node. As the name says, it is initially empty. Containers in the Pod can all read and write the same files in the emptyDir volume, though that volume can be mounted at the same or different paths in each Container. When a Pod is removed from a node for any reason, the data in the emptyDir is deleted forever.”

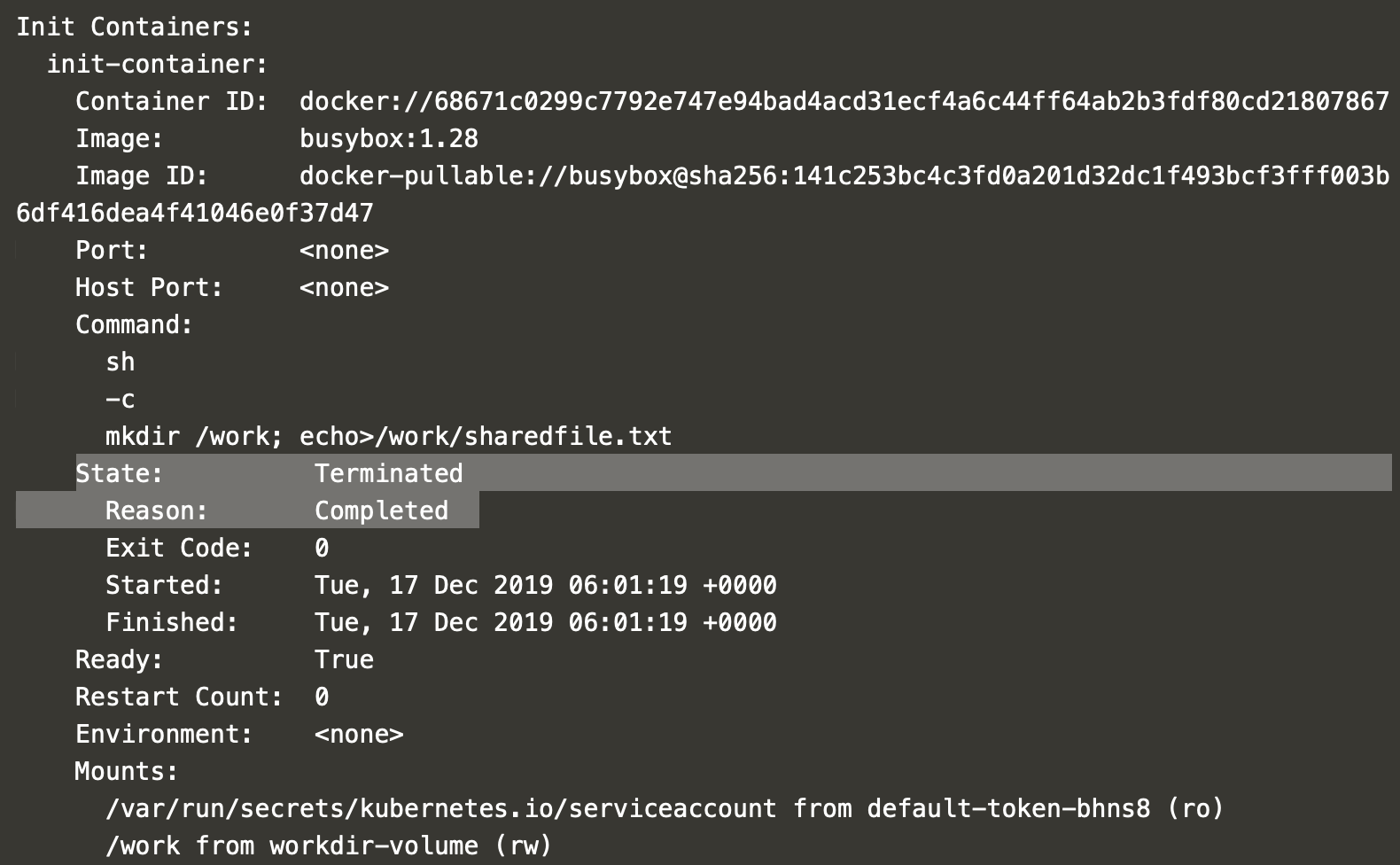
**Validation**

Once you can create the Pod, we should validate if the pod is running and init container completed its task

*Run “kubectl apply -f initpod.yaml”*

*then “kubectl describe pod init-container-test”*

You can see the status of init container, you will see “Terminated” and reason as “Completed”. This shows that the init container completed its job successfully. You will also be able to see that the volume mount is created.

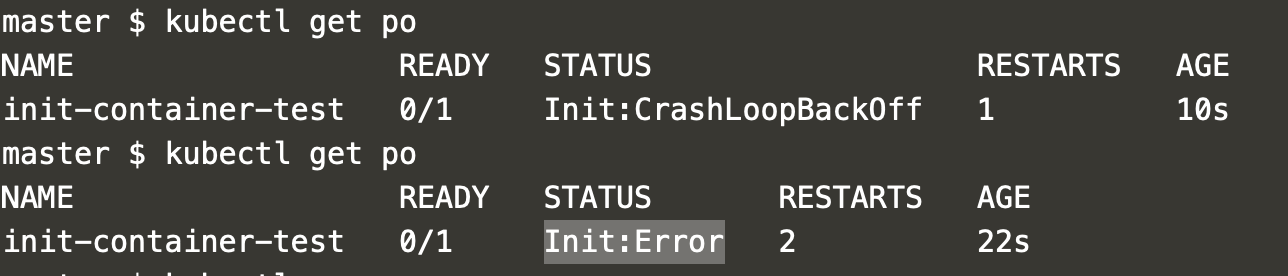


**Debugging Errors**

You may end up in errors, let’s see how to debug that. Update the in YAML @ init container section with the below line

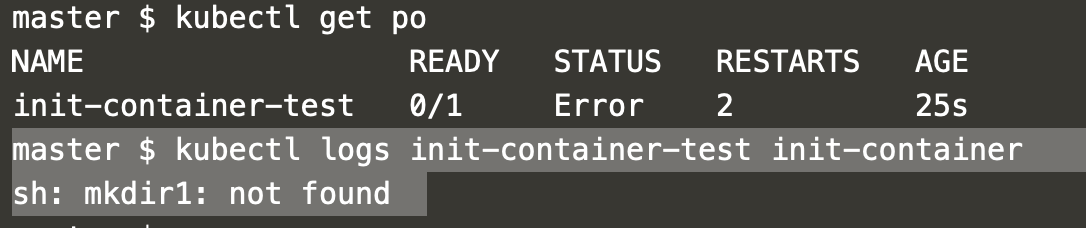
command: ['sh', '-c', 'mkdir1 /work; echo>/work/sharedfile1.txt']

This will fail the init container as we trying to use “mkdir1” a command which does not exist. Note the restart count indication.



Effectively use logs to identify the error. Look at init container error with

*kubectl logs init-container-test init-container*



That is all you need to know about init-container, we are done.

Certified Kubernetes Administrator (CKA) — Tips and Tricks — Part 6

**How to store sensitive information into the cluster with kubernetes secrets**

Kubernetes secrets are one of the high-value questions in the CKA exam. I don’t think that there is another question in the exam where the concept is simple but the marks are maximum. One cannot afford to get this question wrong. Let’s quickly look into the basics of Kubernetes secrets and various ways in which you may face an exam question.

**What are Kubernetes secrets?**

Do you have a small amount of sensitive data that cannot be exposed in a pod specification or ConfigMaps, then Kubernetes secrets is the way to store them in the cluster. Let’s quickly see some facts about Kubernetes secrets.

Kubernetes secrets are namespaced object and lives in the context of the namespace.

Having a separate Kubernetes resource type for secrets enables us a precise RBAC control.

Secrets can be made available to a pod via environment variable or volume.

The limit for the size of data in secrets is 1MB

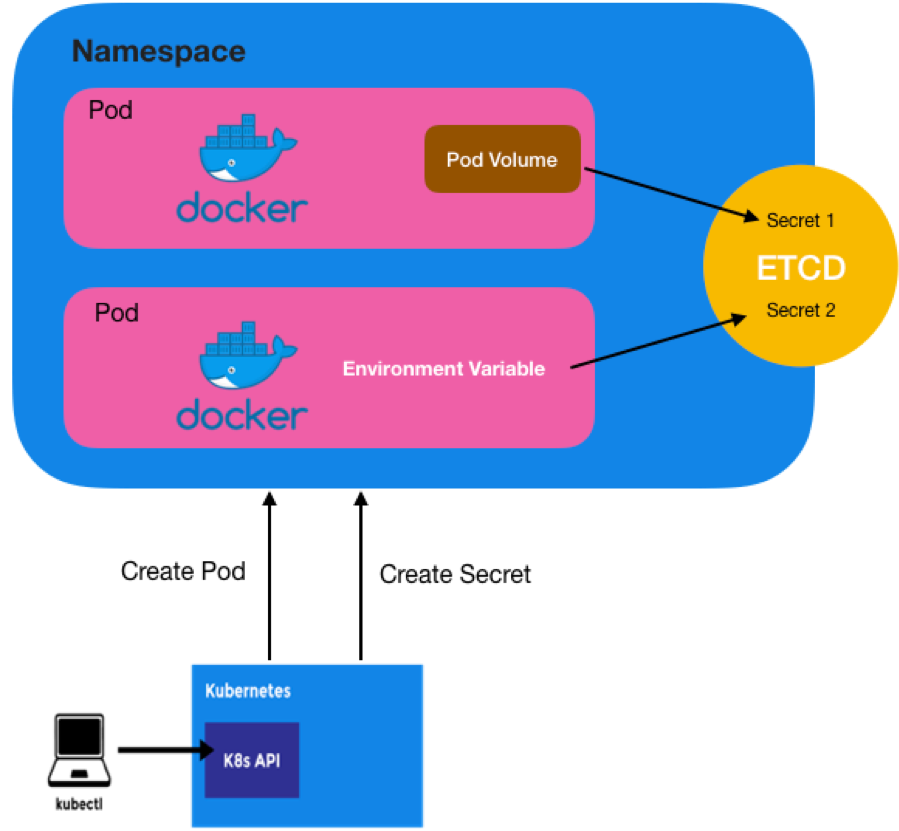
Securing ETCD access is very critical, as secrets are stored as Base64 encoding in ETCD. This is the same as storing in a plaintext.

If you are running some serious production workloads, try sealed secrets or a vault solution which is more production hardened. We will not look into this as it’s out of the scope for the exam.

Of all the above facts, the fact below is very critical for the exam

*Secrets can be made available to a pod via environment variable or volume*

Let’s understand this in a quick picture



**How should I create secret ?**

*kubectl create secret [TYPE] [NAME] [DATA]*

**[TYPE]**

**generic** To create a secret from a local file, directory, or literal value.

**docker-registry** This will create a **dockercfg**secret file to authenticate against any Docker registries.

**tls** Used when you have a tls public/private key pair to be stored.

**[NAME]** — Name of the secret, will be used when you refer inside a POD

**[DATA]**

**— from-file** Used when you load the data from a file. (the value of the Secret is the entire content of the file)

**— from-env-file**Used when you want to load multiple secrets key-value pair from a file.

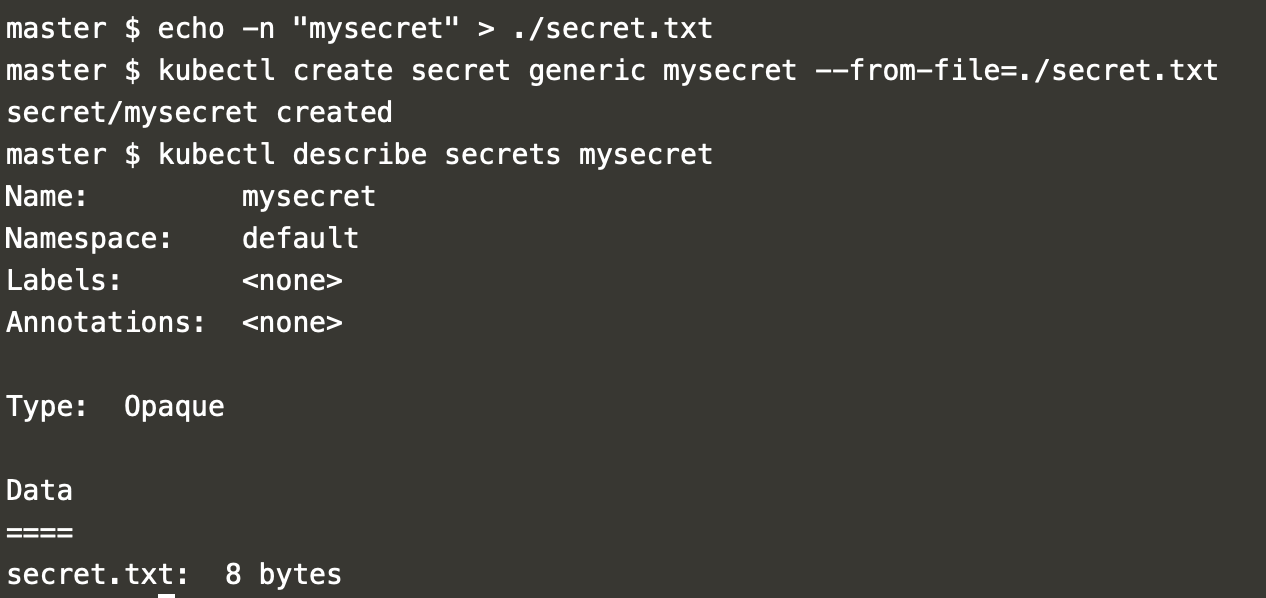
**— from-literal** Used when you load the secret data from key-value pair.

**Creating secret will full file content**

*echo -n “mysecret” > ./secret.txt*

*kubectl create secret generic mysecret --from-file=./secret.txt*

*kubectl describe secrets mysecret*

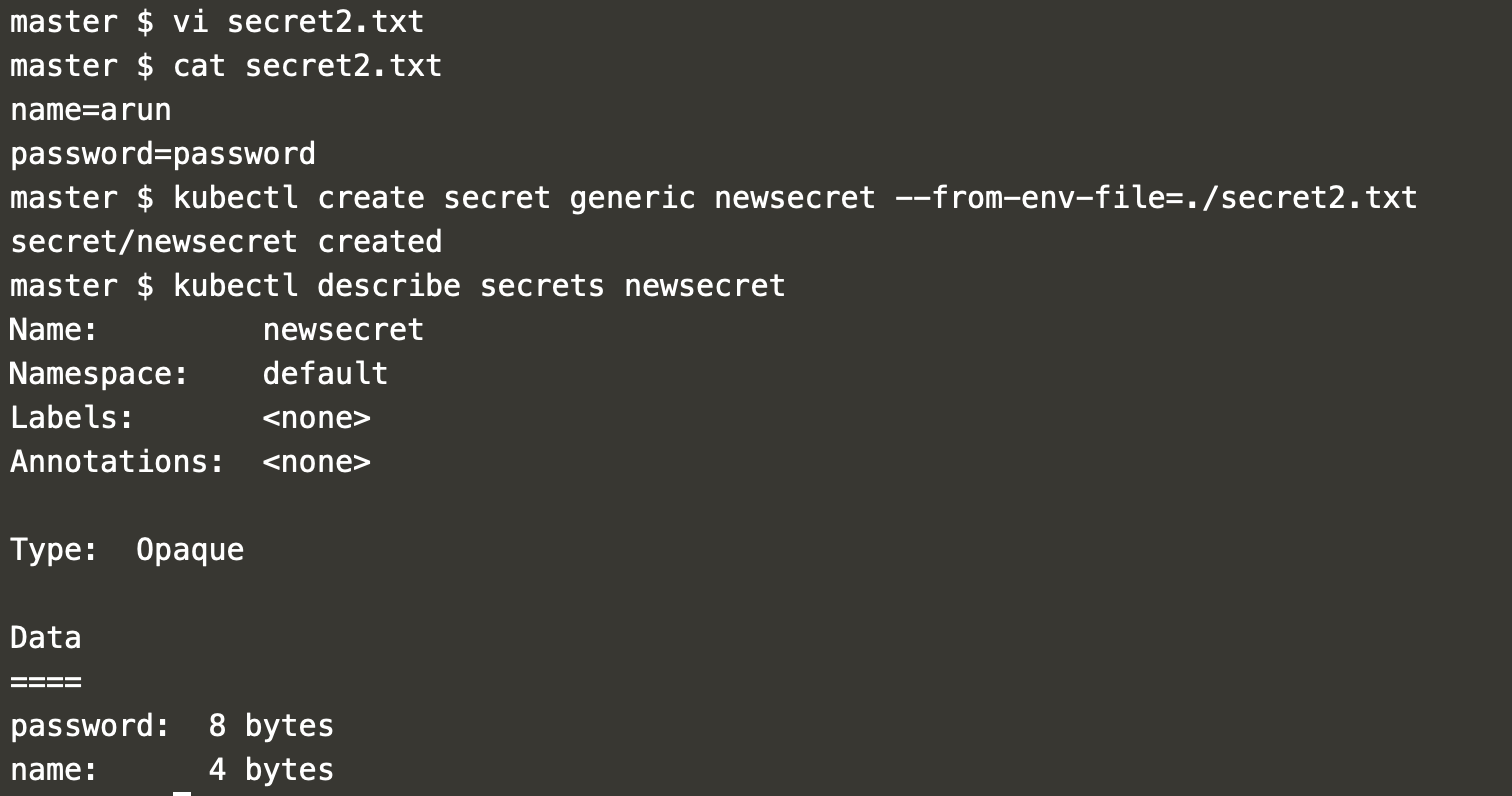


**Creating secret from a file with key-values**

*vi secret2.txt, then add your key values each in a new line*

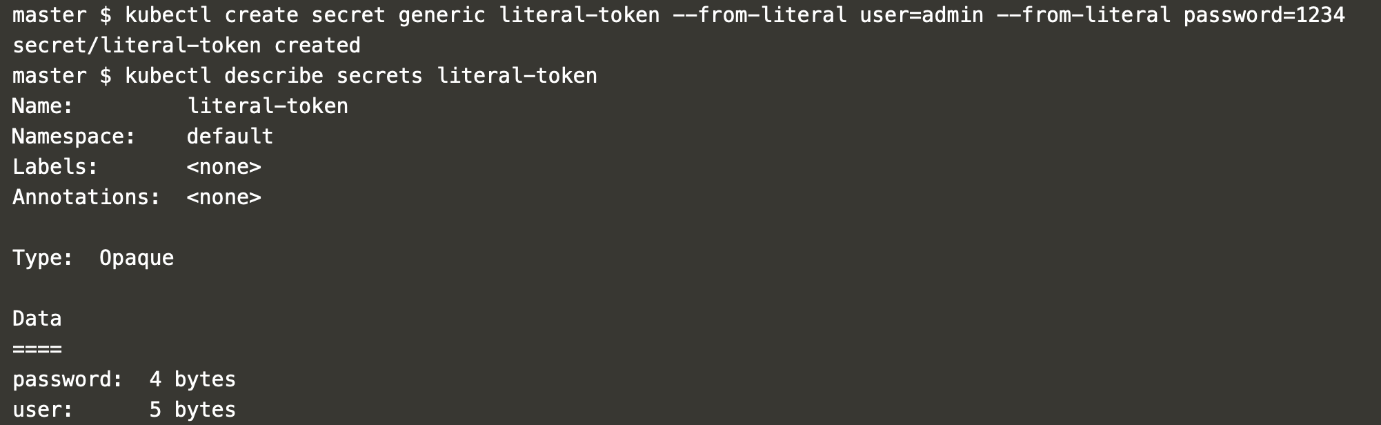
*kubectl create secret generic newsecret --from-env-file=./secret2.txt*

*kubectl describe secrets newsecret*



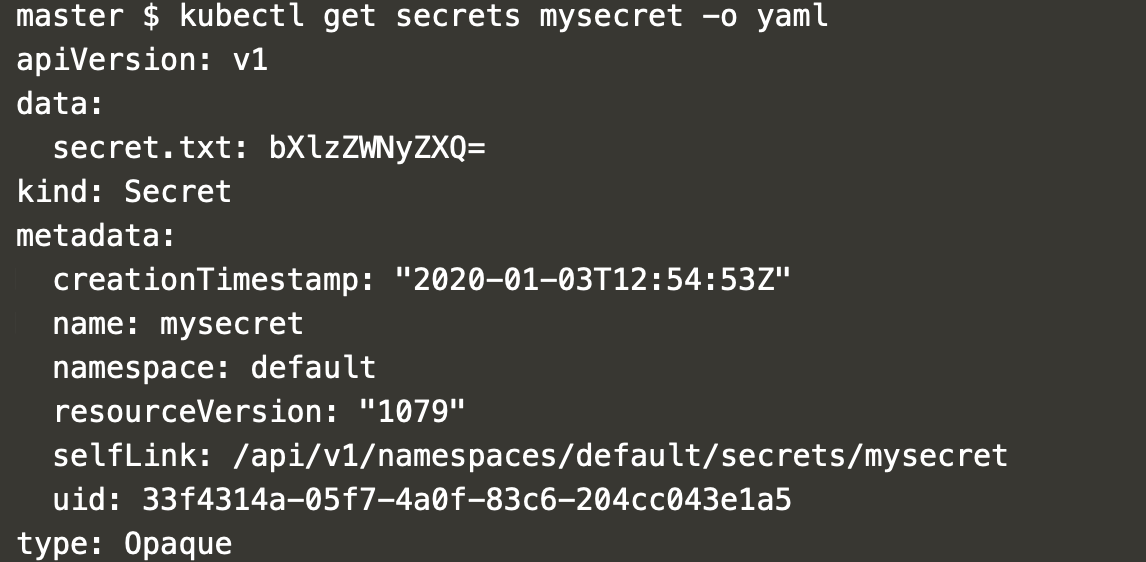
**Creating secret from literal**

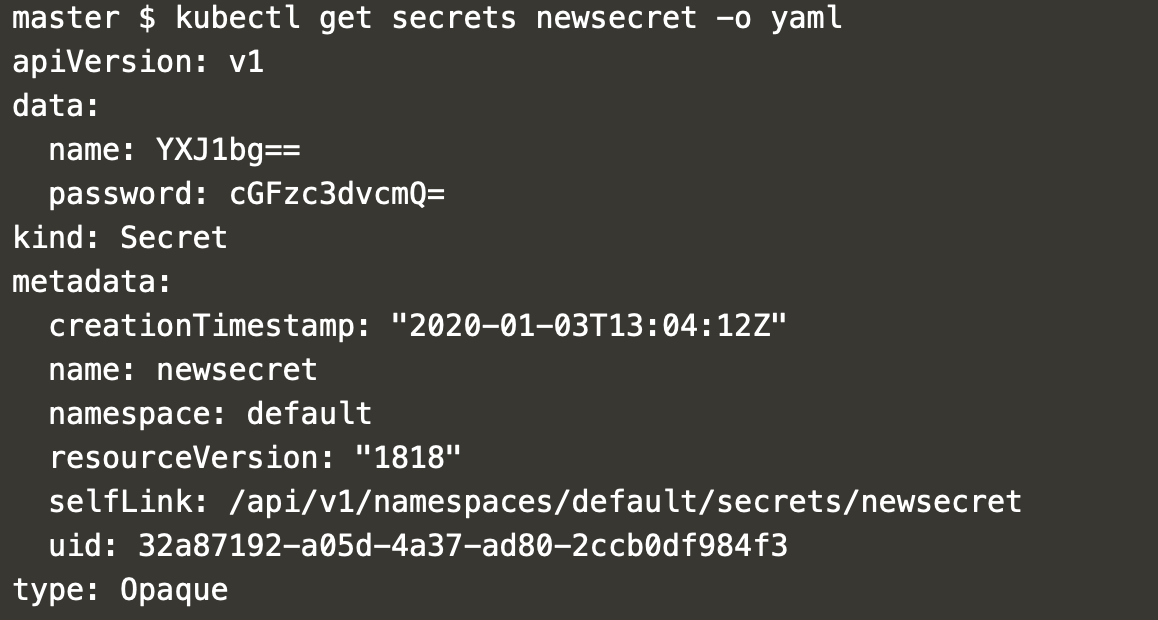
*kubectl create secret generic literal-token --from-literal user=admin --from-literal password=1234*

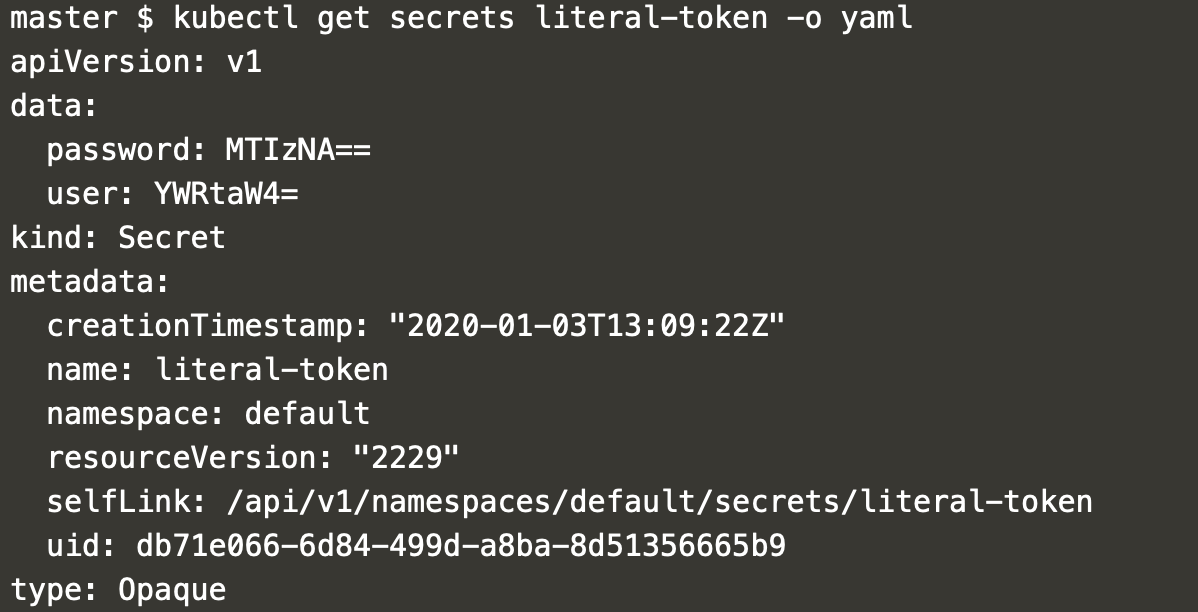


**Creating secret from YAML**

There is an important fact that you have to know, before looking into the YAML way of creating a secret. All the secrets will be Base64 encoded automatically in all the above ways of creating a secret. See the attached YAML outputs.







Now lets see how YAML way of creating is different

If you look at the YAML above, we are manually encoding the value of the secret into YAML we create.

To encode any text use

*echo -n ‘admin’ | base64*

*Then run*

*kubectl apply -f secret.yaml*

**Exam Tips**

For the exam, we should be able to do the below

Map the created secret as an environment variable in the pod specification.

Map the created secret to a volume and map it to the pod.

**Environment Variable**

When you write your pod specification use “valueFrom” into the environment variable with “secretKeyRef” and then use the name of the secret with the key of the secret.

*In the below example pod, we pick up two environment variable “SECRET\_USERNAME”, “SECRET\_PASSWORD” and fill them with value from the secret “mysecret” with the keys “username” and “password” respectively*

**Pod Volume Mapping**

First create a pod volume from the secret.

*volumes → name: [Volume Name] → secret → secretName: [Secret Name]*

Then mount the volume to the pod

*volumeMounts → name: [Volume Name] → mountPath: “[Mount Path]” → readOnly: true*

You can use “readOnly” attribute optionally, if you wish that the secret file in the volume should not be modified.

A final important note: If you are not able to memorize any of the above commands, no worries.

[Kubernetes documentation page for secrets is there for the rescue during the exam.](https://kubernetes.io/docs/concepts/configuration/secret/)

That is all you need to know about secrets, we are done.