How Kubectl Apply works

There are 3 types of file.

1) local file - developed by SDE, stored in Github
2) last applied Configuration file - which will have annotations in json format
3) Live object Configuration file. (stored in etcd)

Local file

→ It is the file created by SDE to design the infrastructure.

→ Stored in Version Control System

→ YAML file.

Last applied Configuration file

→ Initially, if we create / edit an object using declarative approach {YAML manifest file}, a copy of the local file will be created in ~~metadata~~, in K8s object metadata, which is called last applied Configuration file

⇒ It in JSON representation.

→ Once you've created an object using declarative way (i.e) using manifest file. For eg: here it is "pod creation"

Once the pod got created, Execute

```
kubectl get pod nginx <pod-name> -o yaml
```

→ Output "pod" in "YAML"

Then look for

```
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: {...}
```

To view the last applied config from the file, use

```
kubectl apply view-last-applied -f <deployment.yaml>
```

we can't create the annotations manually. kubectl apply will create by default under metadata

when last-applied-configuration file will be removed?

> If you want to delete annotation manually

```
kubectl annotate pod <pod-name> kubectl.k8s.io/last-applied-configurations-
```

use "kubectl annotate" to delete annotation which will delete any resources (pods, deployments, services, etc.,)

7. > If you use kubectl replace (or) kubectl create instead of apply it doesnot make any change to the annotation whereas kubectl apply make changes to the annotation.

Live object configuration file - stored in etcd

> This file has the exact configuration of the object in cluster

> Additionally, it has "status of the object"

⇒ So, if you're making any changes to the k8s object then k8s will compare all 3 files before applying the changes

⇒ If you've created an object using declarative way then "last applied configuration" file will be created in k8s object metadata, then post "kubectl apply" command the object will be created in cluster, then the "configuration for the object" will be stored in k8s memory

⇒ If you use Imperative command to create an object then "last applied configuration file" will not be created

> But if you want "live object last applied Configuration file" for the object created using imperative command, we can do this. The "live-object configuration file" is stored in "etcd"

1) Export the live object to a file

```
kubectl get pod nginx -o yaml > nginx.yaml
```

2) Remove unwanted fields like status, metadata.uid

3) Apply declaratively

```
kubectl apply -f nginx.yaml
```

Initially, if you've created objects using declarative way it has the same configuration in all 3 files.

And if you make any changes / edit the object using "Imperative command", then the object will get changed in the cluster, and as per the object in the cluster. the "live object configuration file" will get automatically but the changes were not recorded in "last applied configuration file" nor in "local file"

So then in future if you want to make any change to the same object, there will be no records for the changes made through imperative command & you would have also forgotten as what if some of the other team mate have to make change

For eg:- Initially the object was created with the configuration of (replicas = 3) & (nginx = 1.18), then later changes made through imperative command for (replica=5) now the "live object configuration file" has (replicas=5) & (nginx = 1.18) in cluster, but in "local" & "last applied configuration file" has ("replicas=3") & ("nginx = 1.18")

Now if we make changes as "nginx:1.19" & apply the changes, now the cluster will have "nginx:1.19" & "replicas:3" as the "last applied configuration file" doesnot have the changes made through imperative command.

⇒ So, the best practice here is while creating an object / editing an object, always use the declarative way of approach, so that all the 3 files will be insync.

→ Imperative approach is for one-time deployment & you no longer read this object.

kubectl apply vs kubectl create vs kubectl replace

command	Effect on last-applied-configuration
kubectl apply -f file.yaml	Updates the annotation with the new config (default behaviour)
kubectl create -f file.yaml	Does not add the annotation (only apply does)
kubectl replace -f file.yaml	Overwrites the resource without respecting the annotation & forces the new state, which means the "last applied-configuration file" annotations will be removed immediately after "kubectl replace"

Use "kubectl replace -f file.yaml --force" ⇒ use this when the object got corrupted

Points to Remember:-

1) To delete the annotation:-

Kubectl annotate <resource> <annotation key>-

Kubectl annotate deployment nginx-deployment [example.com/owner]-

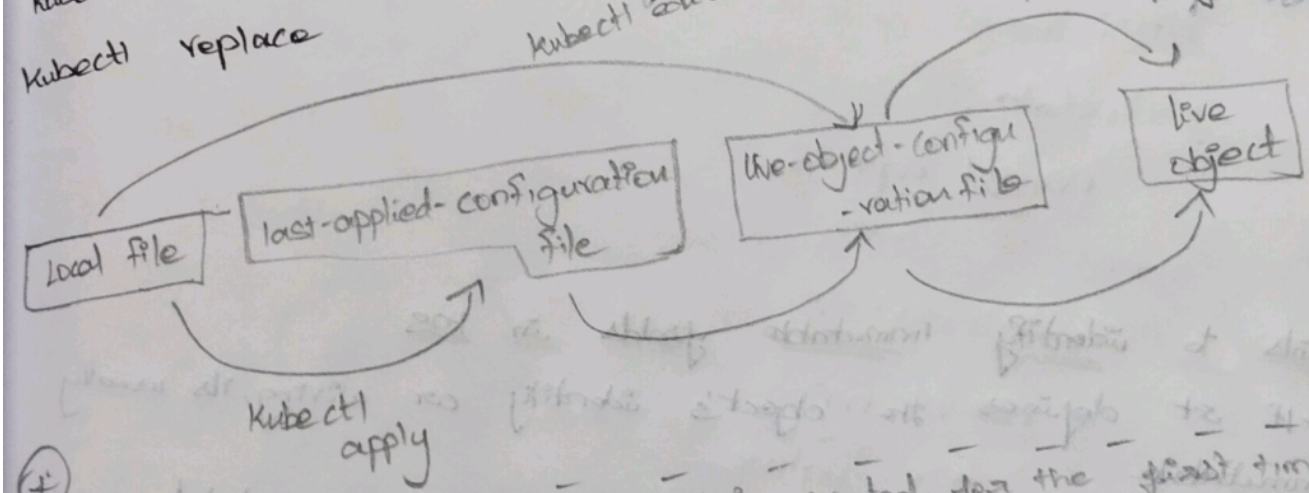
Google what if 35 we remove this hyphen -

annotation key

2) To prevent future annotation:-

⇒ When we say "prevent future annotations", we mean avoiding the automatic addition or update of the kubectl.kubernetes.io/last-applied-configuration annotation to a resource.

⇒ This annotation is only added/modified by kubectl apply, not by other commands like kubectl create, kubectl replace



Kubectl apply:- If the object is getting created for the first time "kubectl apply" will create "annotations" in "last-applied configuration file" & if you make any changes in local file & execute using "kubectl apply" then annotations will get updated. Hence it is always recommended to use "kubectl apply" as it will be helpful for tracking.

Kubectl create:- This will ignore the "annotations", if a change has been made in local file & executed using "kubectl create", then the annotations will not get updates & if the "object" exists already then it will say "object already exists & does not make any changes to the annotations in "last-applied-configuration file"

Kubectl replace:- This will delete/remove the annotation in "last-applied-configuration" of the object configuration is broken

To view the last-applied-configuration file.

`Kubectl get <object> <object-name> -o yaml`

⇒ The above cmd will show the "last-applied configuration object" with "annotation" in "YAML" format

* If you want to make any change/edit in "last-applied configuration" file always use Kubectl apply command as it preserves annotations with changes made

⇒ Always use `kubectl apply` whenever you are making changes in "last-applied-configuration file"

⇒ `Kubectl replace` deletes annotation, use it as last option which is used for quick fixes.

Hints to identify Immutable fields in k8s

⇒ If it defines the object's identity or wiring, its usually immutable

Field type	
Selectors (spec.selectors)	Defines which pods the controller manages - Changing it breaks ownership
Service wiring [ClusterIP, nodeport, serviceName]	Controls n/w identity - changing breaks routing.
PVC access & storage class [accessModes, storageClassName]	Affects how storage is provisioned - can't change once bound
Pod Volume setup [VolumeMounts, volumes]	Tied to controller file system - can't swap after start
Names (metadata.name, container names)	Used as keys or indexes internally - renaming breaks things