Kubectl apply

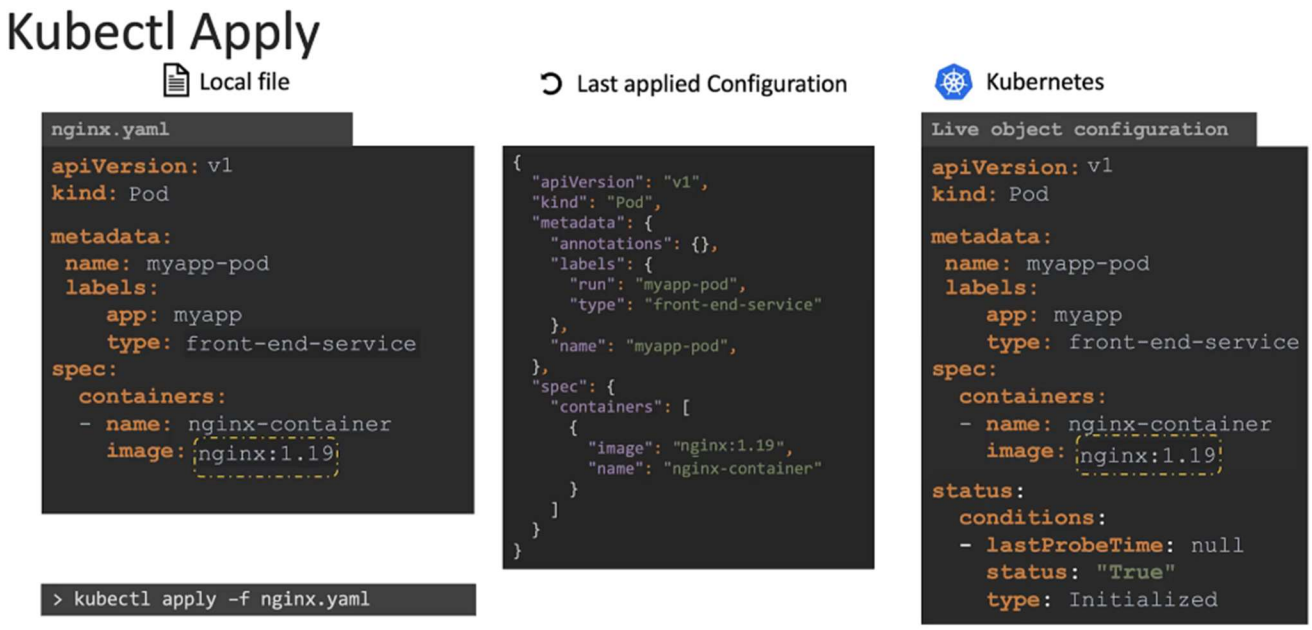
In this lecture, we'll understand more about how the "kubectl apply" command works. In the previous lecture, we saw how a "kubectl apply" command can be used to manage objects in a declarative way. In this lecture, we will see a bit more about how the command works internally.

The apply command takes into consideration the local configuration file, the live object definition on Kubernetes, and the last applied configuration before making a decision on what changes are to be made.

So when you run the apply command, if the object does not already exist, the object is created. When the object is created, an object configuration, similar to what we created locally, is created within Kubernetes but with additional fields to store the status of the object. This is the live configuration of the object on the Kubernetes cluster. This is how Kubernetes internally stores information about an object, no matter what approach you use to create the object.

But when you use the "kubectl apply" command to create an object, it does something a bit more. The YAML version of the local object configuration file we wrote is converted to a JSON format, and it is then stored as the last applied configuration.

Going forward, for any updates to the object, all three are compared to identify what changes are to be made on the live object. For example, say when the nginX image is updated from 1.18 to 1.19 in our local file and we are on the "kubectl apply" command, this value is compared with the value in the live configuration. And if there is a difference, the live configuration is updated with the new value. After any change, the last applied JSON format is always updated to the latest so that it's always up to date.



So, why do we then really need the last applied configuration, right? So if a field was deleted, say, for example, the type label was deleted, and now when we run the "kubectl apply" command, we see that the last applied configuration had a label but it's not present in the local configuration. This means that the field needs to be removed from the live configuration.

So if a field was present in the live configuration and not present in the local or the last applied configuration, then it will be left as is.

But if a field is missing from the local file and it is present in the last applied configuration, so that means that in the previous step, or whenever the last time we ran the "kubectl apply" command, that particular field was there and it is now being removed.

So the last applied configuration helps us figure out what fields have been removed from the local file, right? So that field is then removed from the actual, the live configuration.

Okay, so we saw the three sets of files, and we know that the local file is what's stored on our local system. The live object configuration is in the Kubernetes memory. But where is this JSON file that has the last applied configuration stored? Well, it's stored on the live object configuration on the Kubernetes cluster itself as an annotation named "last applied configuration." So remember that this is only done when you use the apply command. The "kubectl create" or "replace" command does not store the last applied configuration like this. So you must bear in mind not to mix the imperative and declarative approaches while managing the Kubernetes objects.

Once you use the apply command, going forward, whenever a change is made, the apply command compares all three sections: the local part definition file, the live object configuration, and the last applied configuration stored within the live object configuration file for deciding what changes are to be made to the live configuration, similar to what we saw in the previous slide.