Imperative versus declarative commands

Until now, our examples were focused on quick and imperative commands by using `kubectl <command>` i.e. to create deployments and services running on a one-node cluster with `minikube`. This is convenient for something quick, but does not easily expose the full flexibility of the API. To leverage all the options available via Kubernetes, it is often more effective to manage files that describe the deployment you want.

When using these files, you can use `kubectl <command>` along with the `-f option` to specify the file to use. Kubernetes offers a declarative mechanism as well, leveraging the `kubectl apply` command, which takes in files, reviews the current state, and manages the updates—creating, removing, and so on—as needed, while also keeping a simple audit log of the changes.

It's recommended using the declarative mechanism for anything more complex than running a single process, which will likely be most of your developed services. You may not need the audit trails in development. You probably would in a staging/canary environment or in production, so being familiar and comfortable with them is advantageous to understand them.

Best of all, you can include declarative files in source control, treating them like code. This gives you a consistent means of sharing that application structure among your team members, all of which can use it to provide a consistent environment.

The `kubectl apply command` has an -R option as well that will recursively descend directories if you are establishing a complex deployment.

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/en/modul/m321_aws/topics/08?rev=1758895150

Last update: **2025/09/26 15:59**