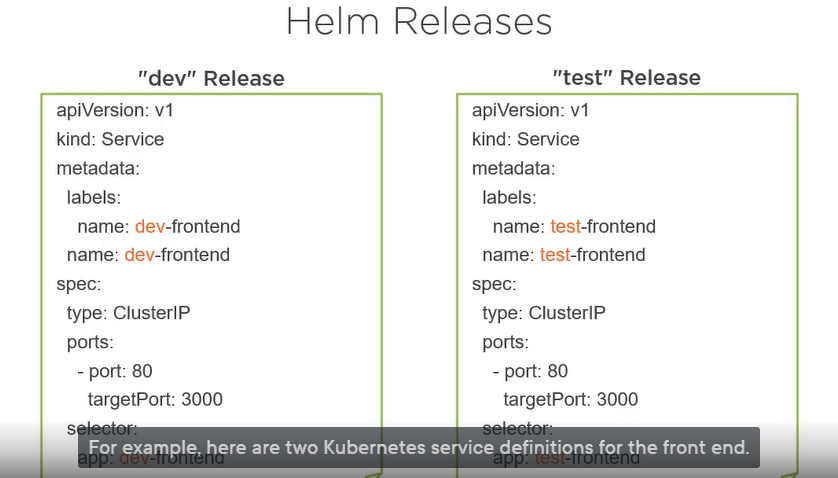
1. **Why Helm templates**

* To release a new version of the application we changed the image version in frontend.yaml file above by hand . Instead of changing values by hand they should be externalized and automatically replaced when we call the helm install command. That’s exactly what using a helm template aims for.
* The second reason why we need helm templates is:

Remember that we should be able to install two releases of the same chart on different clusters, on the same cluster, or even in the same namespace. But in the same namespace, the name of all the k8s objects must be unique. So, if you want to install 2 releases of the same application, we need a way to generate unique names for each of the k8s objects.

The solution is to generate the name of the k8s objects based on the helm release name.

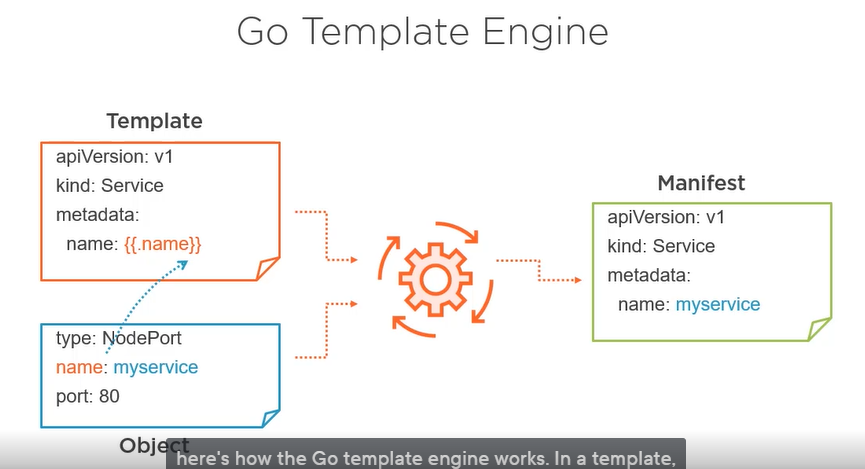
For eg. Consider below



The service name is prefixed with the name of the release, one for the dev release, and one for the test release. To generate those names based on the release name, we need a tool. That tool is the helm template engine.

1. **What is Helm template Engine?**

Helm templates are processed by your template engine. Template Engines are like place holder whose values get replaced at run time. Like below



Template engine is actually based on the Go template engine. The difference is that the values used to replace the directives can come from different sources. Some values are defined in values.yaml file, and some are predefined data that are, for example , in the chart definition or part of the release runtime metadata.

But where and when does the helm template engine run? It runs on the client side. When you launch the helm install or helm upgrade command, before sending the file definition to the kubernetes API, helm first processes your template with the template engine which executes the directives or replace them with values to create a manifest. Then helm sends the result to the k8s API.