Choosing What to Include in a Build Pack

This lesson discusses the reasoning behind choosing what to include in a custom build back and what are we going to add in the one that we will create in this chapter.

We'll cover the following

We might be tempted to create build packs that contemplate all the variations present in our applications. That is often a bad idea. Build packs should provide everything we need, *within reason*. Or, to be more precise, they should provide the things that are repeatable across projects. Still, they should not contemplate so many combinations that build packs themselves would become hard to maintain, and complicated to use.

> ***Simplicity is the key, without sacrificing the fulfillment of our needs.***

When in doubt, it is often better to create a new build pack than to extend an existing one by adding endless `if/else` statements.

What are the requirements of our build pack?

If we take the *go-demo-6* application as an example, we can assume that other projects are written in Go, and that use MongoDB. Even if that's not the case right now, that is such a common combination that we can guess that someone will create a similar application in the future. Even if that is not true within our organization, surely there are many other teams doing something similar.

The popularity of Go is on the constant rise, and MongoDB is one of the most popular databases. There must be many using that combination. All in all, a build pack for an application written in Go with MongoDB as a backend is potentially an excellent candidate both for the internal use within our organization, as well as a contribution to the Jenkins X community.

MongoDB was not the only thing that we had to add when we imported *go-demo-6* based on the `go` template. We also had to change the `probePath` value from `/` to `/demo/hello?health=true`.

Should we add that to the build pack? The answer is no. It is highly unlikely that a similar application from a different team will use the same path for health checks. We should leave that part outside of the soon-to-create build pack and let it continue having root as the default path. We'll let the developers, those that will use our build pack, modify the `values.yaml` file after importing their project to Jenkins X. It will be up to them to design their applications to use the root path for health checks or to choose to change it by modifying `values.yaml` after they import their projects.

All in all, we'll keep the `probePath` value intact, even though our *go-demo-6* application will have to change it. That part of the app is unique, and others are not likely to have the same value.

Next, let's discuss the detailed process step by step.