Why Did We Do All That?

This lesson explains the reasoning behind the exercise that we just performed.

We'll cover the following

- The reasoning behind this exercise

You might not be using MongoDB, and your application might be responding to `/` for health checks. If that's the case, you might be wondering why I took you through the exercise of fixing those things in *go-demo-6*. Was that useful to you?

The reasoning behind this exercise

Here's the secret. Jenkins X does not do magic. It does some very clever work in the background and it helps a lot by simplifying things, by installing and configuring tools, and by joining them all into a single easy to use process. It does a lot of heavy lifting for us, and a part of that is by importing projects into its own brace.

When we import a project, **Jenkins X** creates a lot of things. They are not a blind copy of files from a repository. Instead, it creates solutions tailored to our needs. The real question is how does Jenkins X know what we need? The answer is hugely related to our work. It evaluates what we did so far by scanning files in a repository where our application resides. In our example, it figured out that we are using Go, and it created a bunch of files that provide the definitions for various tools (e.g., **Helm**, **Skaffold**, **Docker**, etc.). Yet, Jenkins X could not read our minds. It failed to figure out that we needed a MongoDB and it did not know that our application required a *"special"* address for Kubernetes health checks. We had to reconfigure some of the files created through the import process.

The real question is whether a person who imports a project has the skills to discover the cause of an issue created by missing entries in configuration files generated during the import process. If the answer is:

- *"Yes, he knows how to find out what's missing in Kubernetes definitions."*
- *"Yes, he uses **Helm**."*

- … and many other yeses.

Then, that person should have an easy time figuring out things similar to what we did together. Nevertheless, Jenkins X is not made to serve only people who are Kubernetes ninjas, who know **Helm**, **Skaffold**, **Docker**, **Knative Build**, and other tools Jenkins X packages. It is designed to ease the pain caused by the complexity of today's solutions. As such, it is silly to expect that everyone will go through the same process as we did. Someone must, but the end result should not be what we just did. Instead, the solution lies in the creation of build packs which we'll explore in the next chapter. We'll try to create a solution that will allow anyone with an application written in Go and dependent on MongoDB to import it into Jenkins X and expect everything to work from the first attempt.

The application we imported is on one end of the spectrum. It had nothing but Go code. There was no `Dockerfile`, no `jenkins-x.yml`, no `Makefile`, no `skaffold.yaml`, and certainly no **Helm** chart. Your applications might not be in such a state. Maybe you already created a `Dockerfile`, but you're missing the rest. Or, perhaps you have all those except `skaffold.yaml`.

> When your project has some of the files, but not all, the import process will generate only those that are missing. It does not matter much what you do and doesn't have, Jenkins X will fill in the missing pieces.

In the next lesson, we will wrap up this discussion and remove any used resources.