Getting Rid of Repetition

This lesson discusses the downsides of Jenkinsfile based pipeline model.

We'll cover the following

Copying and pasting code is a major sin among developers. One of the first things we learn as software engineers is that duplicated code is:

- Hard to maintain.
- Prone to errors.

That's why we are creating libraries. We do not want to repeat ourselves, so we even came up with a commonly used acronym **DRY** (don't repeat yourself).

With that in mind, all I can say is that Jenkins users are *sinful*.

Identical `Jenkinsfile` in projects

When we create pipelines through Jenkins (or almost any other similar tool), every project gets a `Jenkinsfile` based on the pipeline residing in the build pack we chose. If we have ten projects, there will be ten identical copies of the same `Jenkinsfil`e. Over time, we'll modify those `Jenkinsfile`s to where they might not all be precisely the same. Even in those cases, most of the `Jenkinsfile` contents will remain untouched. It does not matter whether 100% of `Jenkinsfile` is repeated across projects or it that number drops to 25%. There is a high level of repetition.

Groovy libraries to remove redundancy

In the past, we fought such repetition through shared libraries. We would encapsulate repeated lines into Groovy libraries and invoke then from any pipeline that needed to use those features. But we abandoned that approach with Jenkins X since Jenkins shared libraries have quite a few deficiencies. They can be written only in Groovy, and that might not be a language everyone wants to learn. They cannot be easily tested in isolation as we'd need to run a Jenkins pipeline that invokes the library as a way of testing it. Finally, we could not easily run them locally without Jenkins.

Executables to remove redundancy

While shared libraries can be used with static Jenkins X, we probably should not go down that route. Instead, I believe that a much better way to encapsulate features is by writing executables. Instead of being limited to Groovy, we can write an executable in Bash, Go, Python, or any other language that allows us to execute code. Such executables (which are usually scripts) can be easily tested locally, they can be used by developers with Jenkins X, and can be executed from inside pipelines. If you take another look at any Jenkins X pipeline, you'll see that there are no plugins, and there are no shared libraries. It's mostly a series of `sh` commands that execute Shell commands (e.g., `cat`, `jx`, etc.). Such pipelines are easy to understand, easy to run with or without Jenkins X (e.g., on a laptop), and easy to maintain. Both plugins and shared libraries are dead.

Downside of plugins

Why do I believe that plugins are dead? To answer that question we need to take a look at the reasons for their existence. Most plugins are created either to isolate users from even basic commands or because the applications they integrate with don't have a decent API.

Isolating anyone from basic commands is just silly. For example, using a plugin that will build a Docker image instead of merely executing `docker image build` is beyond comprehension. On the other hand, if we do need to integrate with an application that does not have an API and CLI, we are better off throwing that application in the trash. It's not 1999 anymore. Every application has a good API, and most have a CLI. Those that don't are unworthy of our attention.

So, there are no more plugins, and we shouldn't use shared libraries, and repetitive features should be in executables (e.g., script). With that in mind, do we still have repetition? We do. Even if all the features (e.g., deployment to

production) are in scripts reused across pipelines, we are still bound to repeat a lot of orchestration code.

The solution to this problem

The serverless flavor of Jenkins X solves the problem of unnecessary repetition through the **pipeline extension model**. We'll see it in action soon. For now, we need a cluster with Jenkins X up-and-running.

In the next lesson, we will create a cluster with Jenkins X.