This is a continuation of the **Four Factors of an Effective Software Development Pipeline**, where we will focus on factor four: **Legos! Not Assembly Lines.** Check out Factor 3: Pipeline Maintenance Should be Free if you missed it.

Factor #4: Legos! Not Assembly Lines

The primary concept of this factor is that builds and pipelines should be both complex *and*

Does your web client have a combination of multiple javascript frameworks? You probably need to support multiple types of linters then, right? Are you using typescript but not everywhere? Is your API written in Spring but you forked a major component for and so you need separate testing around that component? Maybe you deliberately forked your own library to support backwards compatibility of an old browser client for a specific customer because they pay gobsmacks and so it just is that way; you need to support those builds too!

All that to say there's no reason we cannot support that *and*

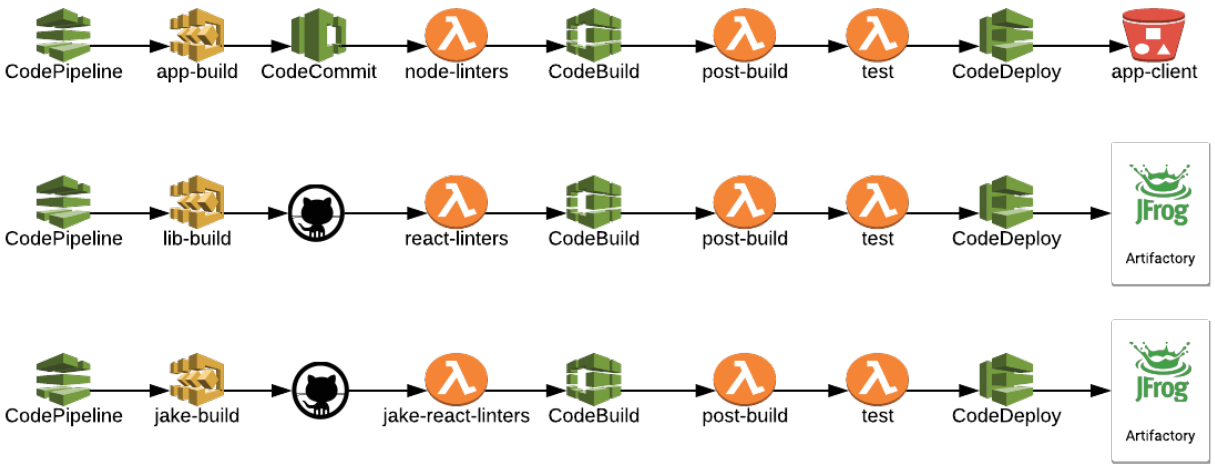
For example, let's presume we have three components of our business application:

1. Our primary web based `app` written in Javascript.
2. A javascript component shared amongst our app and a partner of ours named `lib`
3. Finally, we have an internal only application `jakeJS` that we use for team-building and some office-fun.

A pretty standard approach is to build a Jenkins build for each of these and then we'll worry about the rest later, right? No. Please no!

Instead envision we leverage the previous three factors and we've built three separate pipelines in Amazon CodePipeline. At least we're better off than Jenkins.

This would look something like this:

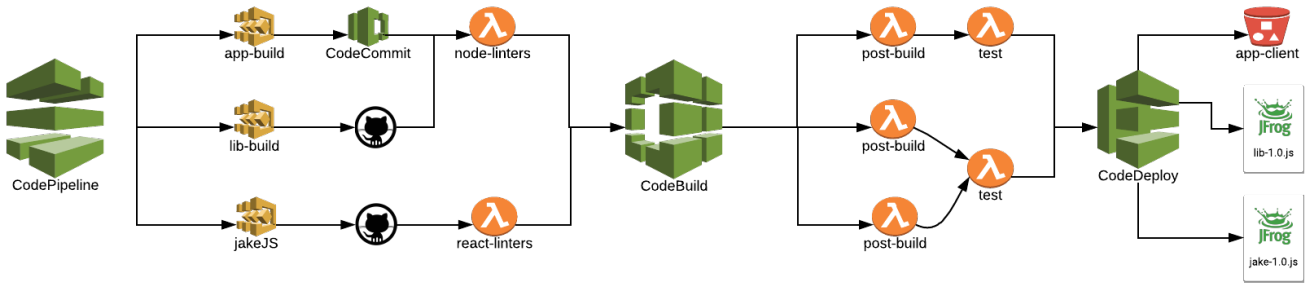


Each “thing” has a unique pipeline start-to-finish.

You’re thinking; fine what’s the big deal? I can get that setup by myself in less than a day and be done with it, and it will hardly ever change so what’s the harm. And when it does change it’s just a silly internal thing so it’s no big deal if it breaks for a little bit. Or any other rationalization you can conjure to move on to the next thing. In the last ~15 years I have heard (and been guilty of myself) them all.

Over time however, this will become a nightmare; trust me. You will end up with dozens or even hundreds of builds, and they will be split across multiple Jenkins build servers which will have different plugins, and your builds will break and you will have no idea how or why, and then you’ll recreate it somewhere else and not delete the old one, and then there will be two builds for someone else to figure out; starting with which of the two actually builds the damn code!

Let’s save our future selves that headache! Think of your pipelines as constructed lego masterpieces. These masterpieces have many of the same lego pieces that we can reuse across more than one pipeline.



Combined overlapping steps to reuse components.

For example, we're using CodePipeline, so let's standardize how we setup builds all in the same CodePipeline region and environment. Now everything is all in one place; an obvious but hugely important first step.

Next, we notice that our react javascript apps/libraries use the same linters as part of our pre-build step to validate syntax and format. We also notice that the testing script is identical so we can consolidate that to a single Lambda we can reuse.

The point is we identify many similar reusable components, and so we reuse them, all-the-while maintaining our uniqueness across pipelines. If we later determine the testing step needs to be separated, the change process is very straight forward and as simple as cloning the original and updating the new Lambda.

Summarizing

In the end, the complexity of software and business in general is going to largely dictate the technical solutions your teams implement. I simply hope this post and its three related factors articulate a generalist approach that sheds light into some best practices that will minimize the pain and suffering later for you and your teams.

Recapping

Previously we discussed the second factor, which is to Pipeline Maintenance Should be Free.