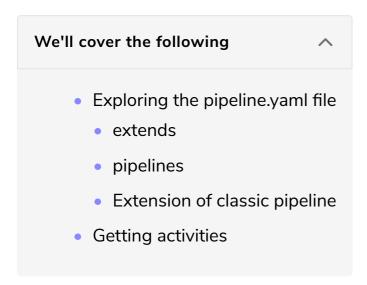
Exploring Build Pack Pipelines

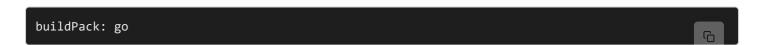
In this lesson we will explore the "pipeline.yaml" file and understand its various components.



As a reminder, we'll take another quick look at the jenkins-x.yml file.



The output is as follows.



The pipeline is as short as it can be. It tells Jenkins that it should use the pipeline from the build pack go.

We already explored build packs and learned how we could extend them to fit out specific needs. But, we skipped discussing one of the key files in build packs. I intentionally avoided talking about the <code>pipeline.yaml</code> file because it uses the new format introduced in serverless Jenkins X. Given that at the time we explored build packs we did not yet know about the existence of serverless Jenkins X, it would have been too early to discuss new pipelines. Now that you are getting familiar with the serverless flavor, we can go back and explore <code>pipeline.yaml</code> located in each of the build packs.

Open *packs* followed with the *go* directory. We can see that *pipeline.yaml* is one of the build pack files, so let's take a closer look.

```
curl "https://raw.githubusercontent.com/jenkins-x-buildpacks/jenkins-x-kubernetes/master/packs/go/
```

That pipeline is in the same format used for pipelines in serverless Jenkins X. Pipelines of our projects are referencing that one in the <code>jenkins-x.yml</code> file with the single line <code>buildPack</code>: go that tells the system to use the pipeline from the build pack, instead of copying it into the application repository.

Exploring the pipeline.yaml file

Let us quickly digest the key sections of the pipeline.yaml file.

extends

It starts with the extends instruction that is as follows.

```
extends:
import: classic
file: go/pipeline.yaml
```

The extends section is similar to how we extend libraries in most programming languages. It tells the system that it should use the build pack <code>go/pipeline.yaml</code> from the <code>classic</code> mode. That's the one we'd use if we choose <code>Library Workloads:</code> <code>CI+Release but no CD</code> when we installed Jenkins X. Those pipelines are meant for the flows that do not involve deployments. Since there is an overlap between <code>Library</code> and <code>Kubernetes</code> and we do not want to repeat ourselves, the <code>Kubernetes</code> one extends the <code>Library</code> (the name <code>classic</code> might be misleading).

pipelines

Further down is the collection of pipelines. The output, limited to the pipeline names is as follows.

```
pipelines:
pullRequest:
...
release:
...
```

There are three types of pipelines:

1 nullRequest

- release
- 3. feature

In the definition we see in front of us there are only the first two since feature pipelines are not very common.

It should be evident that the pullRequest pipeline is executed when we create PR. The release is run when we push or merge something into the master branch. Finally, feature is used with long term feature branches. Since trunk-based or short-term branches are the recommended model, feature is not included in build pack pipelines. You can add it yourself if you do prefer the model of using long-term branches

Each pipeline can have one of the following lifecycles:

- 1. setup
- 2. setversion
- 3. prebuild
- 4. build
- 5. postbuild
- 6. promote.

If we go back to the definition in front of us, the output, limited to the lifecycles of the pullRequest pipeline, is as follows.

```
pipelines:
pullRequest:
build:
...
postBuild:
...
promote:
...
```

In the YAML we're exploring, the pullRequest pipeline has lifecycles build, postbuild, and promote, while release has only build and promote.

Inside a lifecycle is any number of steps containing sh (the command) and an optional name, comment, and a few other instructions.

As an example, the full definition of the build lifecycle of the pullRequest pipeline

is as follows.

```
pipelines:
  pullRequest:
  build:
    steps:
    - sh: export VERSION=$PREVIEW_VERSION && skaffold build -f skaffold.yaml
    name: container-build
...
```

Don't worry if the new pipeline format is confusing. It is very straightforward, and we'll explore it in more depth soon.

Extension of classic pipeline

Before we move on, if you were wondering where the extension of the classic pipeline comes from, it is in the <code>jenkins-x-buildpacks/jenkins-x-classic</code> repository. The one used with Go can be retrieved through the command that follows.

```
curl "https://raw.githubusercontent.com/jenkins-x-buildpacks/jenkins-x-classic/master/packs/go/pip
```

The classic pipeline that serves as the base follows the same logic with pipelines, lifecycle, and steps. The only significant difference is that it introduces a few additional instructions like comment and when. I'm sure you can get what the former does. The latter (when) is a conditional that defines whether the step should be executed in static (!prow) or in serverless (prow) Jenkins X.

As you can see, even though the new format for pipelines is used directly only in serverless Jenkins X, it is vital to understand it even if you're using the static flavor. Pipelines defined in build packs are using the new format and translating it into <code>Jenkinsfile</code> used with static Jenkins X, or extending them if we're using the serverless flavor. If you decide to extend community-maintained buildpacks, you will need to know how the new <code>YAML</code> -based format works, even if the end result will be <code>Jenkinsfile</code>.

Getting activities

By now, the first activity of the newly imported *go-demo-6* project should have finished. Let's take a look at the result.

```
--filter go-demo-6 \
--watch
```

The output is as follows.

```
STEP
                                              STARTED AGO DURATION STATUS
                                                    6m57s
                                                               5m2s Succeeded Version: 1.0.56
vfarcic/go-demo-6/master #1
  from build pack
                                                     6m57s
                                                               5m2s Succeeded
    Credential Initializer Pmvfj
                                                     6m57s
                                                                0s Succeeded
   Git Source Vfarcic Go Demo 6 Master Nck5w
                                                     6m55s
                                                                 Os Succeeded https://github.com/vfa
   Place Tools
                                                     6m52s
                                                                 0s Succeeded
   Git Merge
                                                     5m41s
                                                                 0s Succeeded
   Setup Jx Git Credentials
                                                     3m51s
                                                                 1s Succeeded
   Build Make Build
                                                     3m49s
                                                                16s Succeeded
   Build Container Build
                                                                 4s Succeeded
                                                     3m30s
    Build Post Build
                                                     3m24s
                                                                 0s Succeeded
   Promote Changelog
                                                     3m23s
                                                                 6s Succeeded
   Promote Helm Release
                                                                 9s Succeeded
                                                     3m16s
   Promote Jx Promote
                                                             1m11s Succeeded
                                                      3m6s
 Promote: staging
                                                              1m7s Succeeded
                                                      3m2s
    PullRequest
                                                      3m2s
                                                               1m7s Succeeded PullRequest: https:/
   Update
                                                     1m55s
                                                                 0s Succeeded
```

We've seen similar outputs many times before and you might be wondering where does Jenkins X get the names of the activity steps. They are a combination of the lifecycle and the name of a step from the pipeline which, in this case, is defined in the build pack. That's the same pipeline we explored previously. We can see all that from the output of the activity. It states that the pipeline is from build pack. Further down are the steps.

The first few steps are not defined in any pipeline. No matter what we specify, Jenkins X will perform a few setup tasks. The rest of the steps in the activity output is a combination of the lifecycle (e.g., Promote) and the step name (e.g., jx-promote) > Jx Promote).

Now that we have demystified the meaning of buildPack: go in our pipeline, you are probably wondering how to extend it. We will do that in the next lesson.