Importing a Project

This lesson gives you step by step instructions to import a project into Jenkins X.

We'll cover the following

We'll import the application stored in the vfarcic/go-demo-6 repository. We'll use it as a guinea pig to test the import process and to flesh out potential problems we might encounter.

Fork the repository

But before we import the repository, you'll have to fork the code. Otherwise, you won't be able to push changes since you are not yet a collaborator on that specific repository.

```
open "https://github.com/vfarcic/go-demo-6"
```

Please make sure that you are logged in and click the *Fork* button located in the top-right corner. Follow the on-screen instructions.

Clone the repository

Next, we need to clone the repository you just forked.

⚠ Please replace `[...]` with your GitHub user before executing the commands that follow.

```
GH_USER=[...]

git clone \
  https://github.com/$GH_USER/go-demo-6.git

cd go-demo-6
```

Replace the `master` branch with the `orig` branch

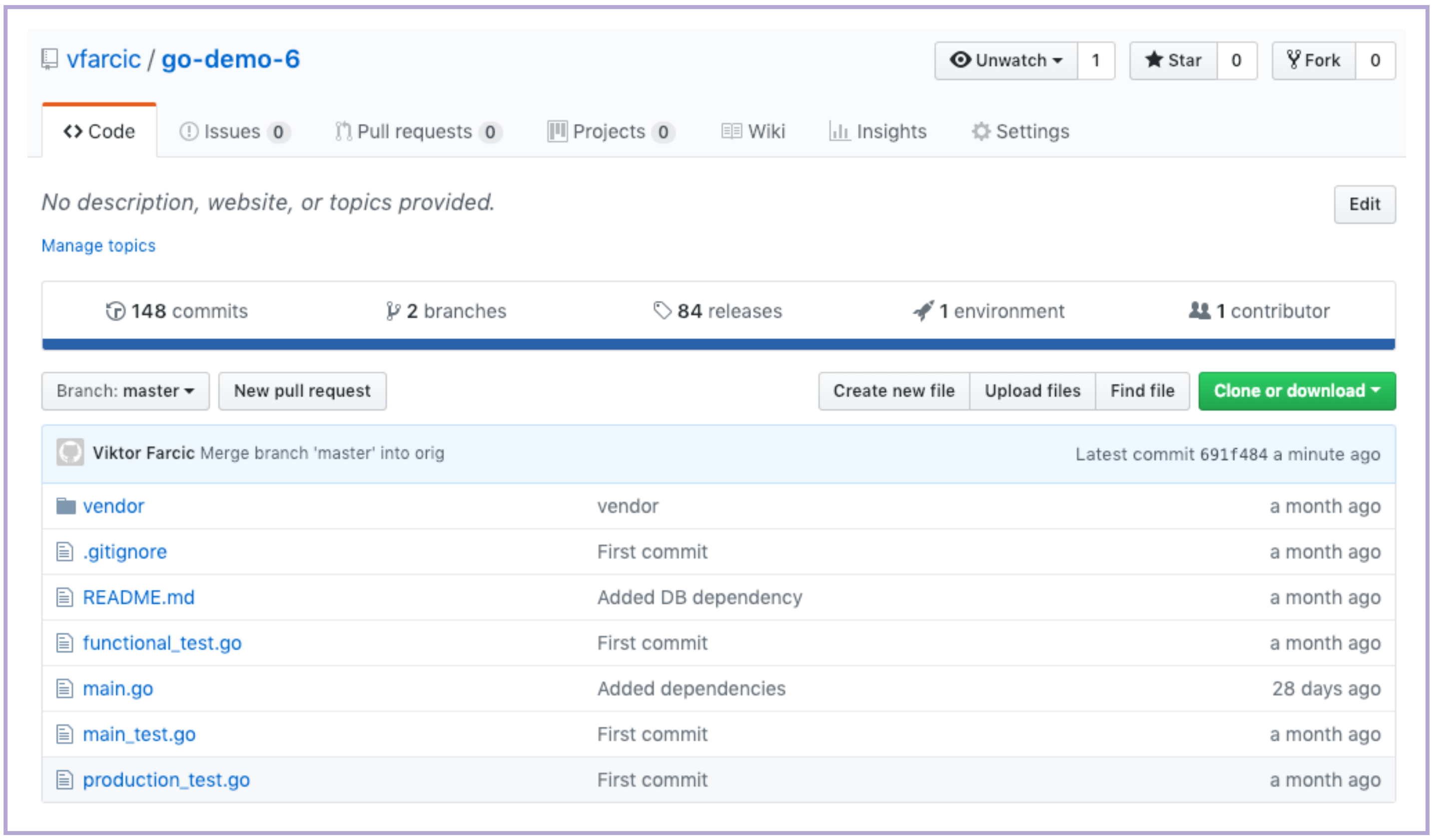
Chances are that I did not leave the master branch in the state I intended it to be or that I'm doing something with it while you're reading this. I might be testing some recently released Jenkins X feature, or maybe I'm trying to fix a problem from one of the examples. To be sure that you have the correct version of the code, we'll replace the `master` branch with the `orig` branch, and push it back to GitHub.

```
git pull

git checkout orig

git merge -s ours master --no-edit

git checkout master

git merge orig

rm -rf charts

git push
```

Inspect the repository

Now you should have the intended code in the master branch of the repository you forked. Feel free to take a look at what we have by opening the repository in a browser. Fortunately, there is a `jx` command that does just that.

```
jx repo --batch-mode
```


vfarcic / go-demo-6

No description, website, or topics provided.

148 commits · 2 branches · 84 releases · 1 environment · 1 contributor

Branch: master

Viktor Farcic Merge branch 'master' into orig — Latest commit 691f484 a minute ago

vendor	vendor	a month ago
.gitignore	First commit	a month ago
README.md	Added DB dependency	a month ago
functional_test.go	First commit	a month ago
main.go	Added dependencies	28 days ago
main_test.go	First commit	a month ago
production_test.go	First commit	a month ago

Exploring the project files

Let's quickly explore the files of the project before we import it into Jenkins X.

```
ls -1
```

The output is as follows:

```
README.md
functional_test.go
main.go
main_test.go
production_test.go
vendor
```

What do we see in the project directory?

As you can see, there's (almost) nothing in that repository but Go code (`*.go`) and libraries (`vendor`).

That project is one extreme of the possible spectrum of projects we might want to import to Jenkins X. It only has the code of the application, there is no `Dockerfile` and there is no **Helm** chart. Heck, there isn't even a script for building a binary, nor is there a mechanism to run tests. And there definitely isn't a `jenkins-x.yml` that defines a continuous delivery pipeline for the application. There's only code and almost nothing else.

> Such a situation might not be your case. Maybe you do have scripts for running tests or building the code. Or perhaps you do have Dockerfile, and maybe you are already a heavy Kubernetes user, and you do have a **Helm** chart. You might have other files as well. We'll discuss those situations later. **For now, we'll work on the case when there is nothing but a code of an application.**

Importing repository into Jenkins X

Let's see what happens when we try to import that repository into Jenkins X.

```
jx import --batch-mode
```

The output is as follows:

```
No username defined for the current Git server!
performing pack detection in folder /Users/vfarcic/code/go-demo-6
--> Draft detected Go (100.000000%)
selected pack: /Users/vfarcic/.jx/draft/packs/github.com/jenkins-x-buildpacks/jenkins-x-kubernetes,
replacing placeholders in directory /Users/vfarcic/code/go-demo-6
app name: go-demo-6, git server: github.com, org: vfarcic, Docker registry org: vfarcic
skipping directory "/Users/vfarcic/code/go-demo-6/.git"
Created Jenkins Project: http://jenkins.jx.35.229.111.85.nip.io/job/vfarcic/job/go-demo-6/

Watch pipeline activity via:    jx get activity -f go-demo-6 -w
Browse the pipeline log via:    jx get build logs vfarcic/go-demo-6/master
Open the Jenkins console via    jx console
You can list the pipelines via: jx get pipelines
When the pipeline is complete:  jx get applications

For more help on available commands see: https://jenkins-x.io/developing/browsing/

Note that your first pipeline may take a few minutes to start while the necessary images get downl

Creating GitHub webhook for vfarcic/go-demo-6 for url http://jenkins.jx.35.229.111.85.nip.io/githul
```

We can see from the output that Jenkins X detected that the project is `100.000000%` written in Go, so it selected the `go` build pack.

- It applied it to the local repository and pushed the changes to GitHub.
- It created a Jenkins project as well as a GitHub webhook that will trigger builds whenever we push changes to one of the selected branches. Those branches are by default:

  - `master`

  - `develop`
  - `PR-.*`
  - `feature.*`

We could have changed the pattern by adding the `--branches` flag. But, for our purposes, and many others, those branches are just what we need.

Inspect the local copy of the repository

Now, let's take another look at the files in the local copy of the repository.

```
ls -1
```

The output is as follows:

```
Dockerfile
Jenkinsfile
Makefile
OWNERS
OWNERS_ALIASES
README.md
charts
functional_test.go
go.mod
jenkins-x.yml
main.go
main_test.go
production_test.go
skaffold.yaml
vendor
watch.sh
```

We can see that quite a few new files were added to the project through the import process.

- We acquired `Dockerfile` that will be used to build container images
- We acquired `jenkins-x.yml` that defines all the steps of our pipeline.
- Further on, `Makefile` is new as well. It, among others, defines targets to build, test, and install the application.

- There is the `charts` directory that contains files in **Helm** format. We'll use it to package, install, and upgrade our application.
- Then, there is `skaffold.yaml` that contains instructions on how to build container images.
- Finally, we acquired `watch.sh`. It allows us to compile the binary and create a container image every time we change a file in that project. It is a handy script for local development. There are a few other new files (e.g., `OWNERS`) added to the mix.

Do not think that is the only explanation you'll get about those files. We'll explore these files in much more detail later on in the course. For now, what matters is that we imported our project into Jenkins X and that it should contain everything the project needs for both local development and continuous delivery pipeline.

Now that the project is in Jenkins X, we should see it as one of the activities and observe the first build in action. You already know that we can limit the retrieval of Jenkins X activities to a specific project and that we can use `--watch` to watch the progress.

```
jx get activities \
    --filter go-demo-6 \
    --watch
```

By the time the build is finished, the output, without the entries repeated due to changes in status, should be as follows.

```
STEP                                            STARTED AGO DURATION STATUS
vfarcic/go-demo-6/master #1                           3m46s     3m6s Succeeded Version: 1.0.420
  meta pipeline                                       3m46s      30s Succeeded
    Credential Initializer Dkm8m                      3m46s       0s Succeeded
    Working Dir Initializer Gbv2k                     3m46s       1s Succeeded
    Place Tools                                       3m45s       1s Succeeded
    Git Source Meta Vfarcic Go Demo 6 Master ...      3m44s      15s Succeeded https://github.com/
    Git Merge                                         3m29s       1s Succeeded
    Merge Pull Refs                                   3m28s       1s Succeeded
    Create Effective Pipeline                         3m27s       2s Succeeded
    Create Tekton Crds                                3m25s       9s Succeeded
  from build pack                                     3m15s    2m35s Succeeded
    Credential Initializer D5fc9                      3m15s       0s Succeeded
    Working Dir Initializer 7rfg7                     3m15s       1s Succeeded
    Place Tools                                       3m14s       1s Succeeded
    Git Source Vfarcic Go Demo 6 Master ...           3m13s      25s Succeeded https://github.com/
    Git Merge                                          2m48s        1s Succeeded
    Setup Jx Git Credentials                           2m47s        0s Succeeded
    Build Make Build                                   2m47s       20s Succeeded
    Build Container Build                              2m27s        4s Succeeded
    Build Post Build                                   2m23s        1s Succeeded
    Promote Changelog                                  2m22s        6s Succeeded
    Promote Helm Release                               2m16s        5s Succeeded
    Promote Jx Promote                                 2m11s     1m31s Succeeded
  Promote: staging                                      2m6s     1m26s Succeeded
    PullRequest                                         2m6s     1m26s Succeeded  PullRequest: https
    Update                                               40s        0s Succeeded
```

> **Please stop watching the activities by pressing *ctrl+c*.**

So far, the end result looks similar to the one we got when we created a quickstart. Jenkins X created the files it needs, it created a GitHub webhook, it created a pipeline, and it pushed changes to GitHub. As a result, we got our first build, and by the look of it, it was successful.

Check for potential problems

Since I have a paranoid nature, we'll double-check that everything indeed looks ok.

> **Please open the `PullRequest` link from the activity output.**

So far, so good. The *go-demo-6* job created a pull request to the *environment-jx-rocks-staging* repository. As a result, the webhook from that repository should have initiated a pipeline activity, and the result should be a new release of the application in the staging environment. We won't go through that part of the process just yet. For now, just note that the application should be running, and we'll check that soon.

The information we need to confirm that the application is indeed running is in the list of the `applications` running in the `staging` environment. We'll explore the environments later. For now, just run the command that follows.

```
jx get applications
```

The output is as follows:

```
APPLICATION STAGING PODS URL
```

```
go-demo-6   1.0.420      http://go-demo-6.jx-staging.34.206.148.101.nip.io
```

We can see the address through which our application should be accessible in the `URL` column. Please copy it and use it instead of `[...]` in the command that follows.

```
STAGING_ADDR=[...]

curl "$STAGING_ADDR/demo/hello"
```

The output is as follows:

```
<html>
<head><title>503 Service Temporarily Unavailable</title></head>
<body>
<center><h1>503 Service Temporarily Unavailable</h1></center>
<hr><center>nginx/1.15.6</center>
</body>
</html>
```

Now that was unexpected! Everything looks ok from Jenkins X's perspective, but the application is not accessible. *Did we fail to do something, or did Jenkins X fail to do the right thing?*

When in doubt, we can always take a look at the logs.

```
kubectl --namespace jx-staging logs \
    -l app=jx-go-demo-6
```

The output is as follows:

```
2019/01/24 23:16:52 Starting the application
2019/01/24 23:16:52 Configuring DB localhost
panic: no reachable servers
...
```

The problem is in the database; it's missing. The application tried to connect to MongoDB but couldn't find it.

In retrospect, that makes sense. While Jenkins X does the right thing most of the time, it could not know that we need a database, specifically MongoDB. There is no such information in the repository, except inside the Go code. Excluding the possibility of scanning the whole code and figuring out that MongoDB is needed based on imported libraries, it's normal that Jenkins X did not add it to the **Helm** chart it generated when we imported the project.

The only sensible thing we can do is to modify the **Helm** chart and add the `YAML` files Kubernetes will need to spin up a database together with the application.

> Please note that we'll explore how to fix the issues that occurred from importing a specific application into Jenkins X. Your problems will likely be different. Nevertheless, I believe that the exercises that follow will give you an insight into what to expect when importing projects. Watch out for auto-generated **Helm** charts as they are the most common culprit.

In the next lesson, we will fix the auto-generated **Helm** chart.