Testing the New Build Pack

This lesson will test the newly made build pack by importing a project that uses it.

We'll cover the following

Let's check whether our new build pack works as expected.

```
cd ..

cd go-demo-6
```

We entered into the local copy of the `go-demo-6` repository.

Remove go-demo-6 project

If you are reusing Jenkins X installation from the previous chapter, you'll need to remove the *go-demo-6* application as well as the activities so that we can restart the process of importing it.

> ⚠ Please execute the commands that follow only if you did not destroy the cluster from the previous chapter and if you still have the *go-demo-6* project inside Jenkins X. The first command will delete the application, while the second will remove all the Jenkins X activities related to *go-demo-6*.

```
jx delete application \
    $GH_USER/go-demo-6 \
    --batch-mode

kubectl --namespace jx delete act \
    --selector owner=$GH_USER \
    --selector sourcerepository=go-demo-6
```

Undo commits made to `master`

To make sure that our new build pack is indeed working as expected, we'll undo all the commits we made to the `master` branch in the previous chapter and start over.

```
git pull

git checkout orig

git merge -s ours master --no-edit

git checkout master

git merge orig

rm -rf charts

git push
```

We replaced the `master` branch with `orig` and pushed the changes to GitHub.

Import project using `go-mongo` pack

Now we're ready to import the project using the newly created `go-mongo` pack.

```
jx import --pack go-mongo --batch-mode
```

The output should be almost the same as the one we saw when we imported the project based on the `go` pack. The only significant difference is that this time we can see in the output that it used the pack `go-mongo`.

Before it imported the *go-demo-6* project, Jenkins X cloned the build packs repository locally to the `.jx` directory. The next time we import a project or create a new one based on a quickstart, it will pull the latest version of the repository, thus keeping it always in sync with what we have in GitHub.

Listing local files

We can confirm that the repository was indeed cloned to `.jx` and that `go-mongo` is there, by listing the local files.

```
ls -1 ~/.jx/draft/packs/github.com/$GH_USER/jenkins-x-kubernetes/packs
```

The output, limited to the relevant entries, is as follows.

```
...
go
go-mongo
...
```

We can see that the `go-mongo` pack is indeed there.

Inspecting the activities

Let’s take a quick look at the activities and check whether everything works as expected.

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

Once the build is finished, you should see the address of the *go-demo-6* application deployed to the staging environment from the `Promoted` entry (the last one).

> Remember to stop watching the activities by pressing *ctrl+c* when all the steps are executed.

Inspecting the pods

Let’s take a look at the pods that were created for us.

```
kubectl --namespace jx-staging get pods
```

The output is as follows:

```
NAME                          READY STATUS  RESTARTS AGE
jx-go-demo-6-...              0/1   Running 2        2m
jx-go-demo-6-db-arbiter-0     1/1   Running 0        2m
jx-go-demo-6-db-primary-0     1/1   Running 0        2m
jx-go-demo-6-db-secondary-0 1/1   Running 0        2m
```

The database Pods seem to be running correctly, so the new pack was indeed applied. However, the application Pod is restarting. From past experience, you probably already know what the issue is. But in case you forgot, let’s take a look at it again.

Diagnosing the issue

Please execute the command that follows.

```
kubectl --namespace jx-staging \
    describe pod \
    --selector app=jx-go-demo-6
```

We can see from the events that the probes are failing. That was to be expected since we decided that hard-coding `probePath` to `/demo/hello?health=true` is likely not going to be useful to anyone except *go-demo-6* application. So, we left it as `/` in our `go-mongo` build pack. Owners of the applications that will use our new build pack should change it if needed. Therefore, we’ll need to modify the application to accommodate the “special” probe path.

Inspecting the `values.yaml` file

As a refresher, let’s take another look at the `values.yaml` file.

```
cat charts/go-demo-6/values.yaml
```

The relevant parts of the output are as follows:

```
...
probePath: /
...
```

Modifying the `probePath` entry

As with the rest of the changes we did in this chapter, we'll use `sed` to change the value. I won't hold it against you if you prefer making changes in your favorite editor instead.

```
cat charts/go-demo-6/values.yaml \
    | sed -e \
    's@probePath: /@probePath: /demo/hello?health=true@g' \
    | tee charts/go-demo-6/values.yaml
```

Adding `probePath` to the `preview`

Just as charts added as dependencies do not take into account their dependencies (dependencies of the dependencies), they ignore custom values as well. We'll need to add `probePath` to the `preview` as well.

```
echo '
  probePath: /demo/hello?health=true' \
    | tee -a charts/preview/values.yaml
```

> It would be much easier if we could specify the values when importing an application, instead of modifying files afterward. At the time of this writing (February 2019), there is an open issue that requests just that. Feel free to follow the issue 2928.

Push the updates to the application

All that's left is to push the changes and wait until Jenkins updates the application.

```
git add .

git commit \
    --message "Fixed the probe"

git push

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

> Press *ctrl+c* when the new build is finished.

Check the application

All that's left is to check whether or not the application is now running correctly.

> ⚠ Make sure to replace `[...]` with the address from the `URL` column of the `jx get application` command.

```
kubectl --namespace jx-staging get pods

jx get applications

STAGING_ADDR=[...]

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

The first command should show that all the Pods are now running, while the last should output the familiar "hello, world" message.

Revert to the `go` build pack

Before we proceed, we'll go back to the `go` build pack. That way, we won't depend on it in the upcoming chapters.

```
cat jenkins-x.yml \
    | sed -e \
    's@buildPack: go-mongo@buildPack: go@g' \
    | tee jenkins-x.yml

git add .

git commit -m "Reverted to the go buildpack"

git push

cd ..
```

In the next lesson, we will discuss how to contribute to the community.