Creating a Build Pack for Go Applications with MongoDB Datastore

This lesson explains the step by step process to create a custom build pack for Go with MongoDB.

We'll cover the following

- Fork the repository
- Clone the forked repository
- Exploring the packs directory
- Exploring the go sub-directory
- Copy the go build pack
- Add DB environment variable
- Customize MongoDB chart
- Exploring the preview directory
- Inspect and modify the requirements.yaml file
- Push the changes to the repository
- Change the location of kubernetes-workloads packs

We are going to create a build pack that will facilitate the development and delivery of applications written in Go and with MongoDB as the datastore. Given that there is already a pack for applications written in Go (without the DB), the easiest way to create what we need is by extending it. We'll make a copy of the `go` build pack and add the things we're missing.

The community-based packs are located in *~/.jx/draft/packs/github.com/jenkins-x-buildpacks/jenkins-x-kubernetes*. That's where those used with *Kubernetes Workloads* types are stored. Should we make a copy of the local `packs/go` directory? If we did that, our new pack would be available only on our laptop, and we would need to zip it and send it to others if they are to benefit from it. Since we are engineers, we should know better. All code goes to Git and build packs are not an exception.

> If right now you are muttering to yourself something like *"I don't use Go, I don't care"*, just remember that the same principles apply if you use a different build pack as the base that will be extended to suit your needs. Think of this as a learning experience that can be applied to any build pack.

Fork the repository

We'll fork the repository with community build packs. That way, we'll store our new pack safely to our repo. If we choose to, we'll be able to make a pull request back to where we forked it from, and we'll be able to tell Jenkins X to add that repository as the source of build packs. For now, we'll concentrate on forking the repository.

```
open "https://github.com/jenkins-x-buildpacks/jenkins-x-kubernetes"
```

Please fork the repository by clicking the *Fork* button located in the top right corner of the screen and follow the on screen instructions.

Clone the forked repository

Next, we'll clone the newly forked repo.

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

```
GH_USER=[...]

git clone https://github.com/$GH_USER/jenkins-x-kubernetes

cd jenkins-x-kubernetes
```

We cloned the newly forked repository and entered inside it.

Exploring the `packs` directory

Let's see what we got inside the `packs` directory.

```
ls -1 packs
```

The output is as follows:

```
D
appserver
csharp
dropwizard
environment
go
gradle
imports.yaml
javascript
liberty
maven
maven-java11
php
python
ruby
rust
scala
swift
typescript
```

As you can see, those directories reflect the same choices as those presented to us when creating a Jenkins X quickstart or when importing existing projects.

> 🔍 If you see `go-mongodb` in the list of directories, the pull request I made a while ago was accepted and merged to the main repository. Since we are practicing, using it would be cheating; therefore, ignore its existence. I made sure that the name of the directory we'll use (`go-mongo`) is different from the one I submitted in the PR (`go-mongodb`); that way, there will be no conflicts.

Exploring the `go` sub-directory

Let's take a quick look at the `go` directory.

```
ls -1 packs/go
```

The output is as follows:

```
Dockerfile
Makefile
charts
pipeline.yaml
preview
skaffold.yaml
```

```
skaffold.yaml
watch.sh
```

Those are the files Jenkins X uses to configure all the tools involved in the process that ultimately results in the deployment of a new release. We won't dive into them just now. Instead, we'll concentrate on the `charts` directory that contains the **Helm chart** that defines everything related to installation and updates of an application. I'll let you explore it on your own. If you're familiar with **Helm**, it should take you only a few minutes to understand the files it contains.

Copy the `go` build pack

Since we'll use the `go` build pack as our baseline, our next step is to copy it.

```
cp -R packs/go packs/go-mongo
```

Add `DB` environment variable

The first thing we'll do is to add the environment variable `DB` to the `charts/templates/deployment.yaml` file. Its purpose is to provide our application with the address of the database. That might not be your preferred way of retrieving the address so you might come up with a different solution for your applications. Nevertheless, it's my application we're using for this exercise, and that's what it needs.

I won't tell you to open your favorite editor and insert the changes. Instead, we'll accomplish the same result with a bit of `sed` magic.

```
cat packs/go-mongo/charts/templates/deployment.yaml \
    | sed -e \
    's@ports:@env:\
        - name: DB\
          value: {{ template "fullname" . }}-db\
        ports:@g' \
    | tee packs/go-mongo/charts/templates/deployment.yaml
```

The command we just executed added the `env` section right above `ports`. The modified output was used to replace the existing content of `deployment.yaml`.

The next file we have to change is the `requirements.yaml` file. That's where we'll add `mongodb` as a dependency of the **Helm chart**.

```
echo "dependencies:
- name: mongodb
```

```
  alias: REPLACE_ME_APP_NAME-db
  version: 5.3.0
  repository:  https://kubernetes-charts.storage.googleapis.com
  condition: db.enabled
" | tee packs/go-mongo/charts/requirements.yaml
```

> Please note the usage of the `REPLACE_ME_APP_NAME` string. Today (February 2019), that is still one of the features that are not documented. When the build pack is applied, it'll replace that string with the actual name of the application. After all, it would be silly to hard-code the name of the application since this pack should be reusable across many.

Customize MongoDB chart

Now that we created the `mongodb` dependency, we should add the values that will customize MongoDB chart so that the database is deployed as a MongoDB replica set (a Kubernetes `StatefulSet` with two or more replicas). The place where we change variables used with a chart is `values.yaml`. But, since we want to redefine the values of dependency, we need to add it inside the name or, in our case, the alias of that dependency.

```
echo "REPLACE_ME_APP_NAME-db:
  replicaSet:
    enabled: true
" | tee -a packs/go-mongo/charts/values.yaml
```

Just as with `requirements.yaml`, we used the "magic" string `REPLACE_ME_APP_NAME` that will be replaced with the name of the application during the import or the quickstart process. The `replicaSet.enabled` entry will make sure that the database is deployed as a multi-replica `StatefulSet`.

> If you're interested in all the values available in the `mongodb` chart, please visit the project README.

You might think that we are finished with the changes, but that is not true. I wouldn't blame you for that if you hadn't yet used Jenkins X with a pull request (PR). I'll leave the explanation of how PRs work in Jenkins X for later. For now, it should be enough to know that the `preview` directory contains the template of the

Helm chart that will be installed whenever we make a pull request and that we need to add `mongodb` there as well. The rest is on a need-to-know basis and reserved for the discussion of the flow of a Jenkins X PRs.

Exploring the `preview` directory

Let's take a quick look at what we have in the `preview` directory.

```
ls -1 packs/go-mongo/preview
```

The output is as follows:

```
Chart.yaml
Makefile
requirements.yaml
values.yaml
```

As you can see, that is not a full-fledged **Helm chart** like the one we have in the `charts` directory. Instead, it relies on dependencies in `requirements.yaml`.

Inspect and modify the `requirements.yaml` file

```
cat packs/go-mongo/preview/requirements.yaml
```

The output is as follows:

```
# !! File must end with empty line !!
dependencies:
- alias: expose
  name: exposecontroller
  repository: http://chartmuseum.jenkins-x.io
  version: 2.3.56
- alias: cleanup
  name: exposecontroller
  repository: http://chartmuseum.jenkins-x.io
  version: 2.3.56

  # !! "alias: preview" must be last entry in dependencies array !!
  # !! Place custom dependencies above !!
- alias: preview
  name: REPLACE_ME_APP_NAME
  repository: file://../code
```

If we exclude the `exposecontroller` which we will ignore for now (it creates **Ingress** for our applications), the only dependency is the one aliased `preview`. It points to the directory where the application chart is located. As a result,

whenever we create a preview (through a pull request), it'll deploy the associated

application. However, it will not install dependencies of that dependency, so we'll need to add MongoDB there as well.

Just as before, the `preview` uses `REPLACE_ME_APP_NAME` tag instead of a hard-coded name of the application.

If you take a look at the comments, you'll see that the file must end with an empty line. More importantly, the `preview` must be the last entry. That means that we need to add `mongodb` somewhere above it.

```
cat packs/go-mongo/preview/requirements.yaml \
    | sed -e \
    's@  # !! "alias@- name: mongodb\
  alias: preview-db\
  version: 5.3.0\
  repository:  https://kubernetes-charts.storage.googleapis.com\
\
  # !! "alias@g' \
    | tee packs/go-mongo/preview/requirements.yaml

echo '
' | tee -a packs/go-mongo/preview/requirements.yaml
```

We performed a bit more `sed` of magic to add the `mongodb` dependency above the comment that starts with `# !! "alias`. Also, to be on the safe side, we added an empty line at the bottom as well.

Push the changes to the repository

Now we can push our changes to the forked repository.

```
git add .

git commit \
    --message "Added go-mongo build pack"

git push
```

With the new build pack safely stored, we should let Jenkins X know that we want to use the forked repository.

Change the location of `kubernetes-workloads` packs

We can use `jx edit buildpack` to change the location of our `kubernetes-workloads` packs. However, at the time of this writing (February 2019), there is a bug that prevents us from doing that (issue 2955). The good news is that there is a workaround. If we omit the name (`-n` or `--name`), Jenkins X will add the new packs location, instead of editing the one dedicated to `kubernetes-workloads` packs.

```
jx edit buildpack \
    --url https://github.com/$GH_USER/jenkins-x-kubernetes \
    --ref master \
    --batch-mode
```

From now on, whenever we create a new quickstart or to import a project, Jenkins X will use the packs from the forked repository `jenkins-x-kubernetes`.

Go ahead and try it out if you have a Go application with MongoDB at hand.

In the next lesson, we will test the new build pack.