Promoting a Release to the Production Environment

This lesson shows how we can promote an application running in the staging environment to the production environment.

We'll cover the following

- Getting applications running in production
- Getting applications running in staging
- Promoting go-demo-6 to production
- Confirming that the release was deployed

Now that we feel our new release is production-ready, we can promote it to production. Before we do that, we'll check whether we already have something running in production.

Getting applications running in `production`

```
jx get applications --env production
```

The output states that `no applications` were `found in environments production`.

Getting applications running in `staging`

How about staging? We must have the release of our *go-demo-6* application running there. Let's double-check.

```
jx get applications --env staging
```

The output is as follows:

```
APPLICATION STAGING PODS URL
go-demo-6   0.0.184 1/1  http://go-demo-6.jx-staging.35.237.161.58.nip.io
```

For what we're trying to do, the important piece of the information is the version displayed in the `STAGING` column.

> ⚠ Before executing the command that follows, please make sure to replace `[...]` with the version from the `STAGING` column from the output of the previous command.

```
VERSION=[...]
```

Promoting go-demo-6 to production

Now we can promote the specific version of *go-demo-6* to the production environment.

```
jx promote go-demo-6 \
    --version $VERSION \
    --env production \
    --batch-mode
```

It'll take a minute or two until the promotion process is finished.

The command we just executed will create a new branch in the production environment (*environment-jx-rocks-production*). Further on, it'll follow the same practice based on pull requests as the one employed in anything else we did so far. It'll create a PR and wait until a Jenkins X build is finished and successful. You might see errors stating that it `failed to query the Pull Request`. That's normal. The process is asynchronous, and `jx` is periodically querying the system until it receives the information that confirms that the pull request was processed successfully.

Once the pull request is processed, it'll be merged to the master branch, and that will initiate another Jenkins X build. It'll run all the steps we defined in the repository's `jenkins-x.yaml`. By default, those steps are only deploying the release to production, but we could have added additional validations in the form of integration or other types of tests. Once the build initiated by the merge to the master branch is finished, we'll have the release running in production and the final output will state that `merge status checks all passed so the promotion worked!`

The process of manual promotion (e.g., production) is the same as the one we

experienced through automated promotions (e.g., staging). The only difference is who executes promotions. Those that are automated are initiated by application pipelines pushing changes to Git, while manual promotions are triggered by us (humans).

Confirming that the release was deployed

Next, we'll confirm that the release was indeed deployed to production by retrieving all the applications in that environment.

```
jx get applications --env  production
```

The output is as follows:

```
APPLICATION PRODUCTION PODS URL
go-demo-6   0.0.184    1/1  http://go-demo-6.jx-production.35.237.161.58.nip.io
```

In my case, the output states that there is only one application (`go-demo-6`) running in production and that the version is `0.0.184`.

To be on the safe side, we'll send a request to the release of our application running in production.

> ⚠ Before executing the commands that follow, please make sure to replace `[...]` with the `URL` column from the output of the previous command.

```
PROD_ADDR=[...]

curl "$PROD_ADDR/demo/hello"
```

The output should be the familiar message `hello, PR!`. We confirmed that promotion to production works as expected.

Before we proceed, we'll go out of the `go-demo-6` directory.

```
cd ..
```

Now, let's wrap up this discussion and remove the used resources.