Creating Pull Requests

This lesson shows how we can create a pull request using the Jenkins X CLI.

We'll cover the following

What is the first thing a developer should do when starting to work on a new feature? If that developer is used to the “old processes”, they might wait until someone creates a set of branches, a few environments, infinite approvals, and a bunch of other silly things no one should need today. We already saw that creating a development environment is very easy and fast with DevPods. I’m going to ignore talking about the approvals, so the only thing left is branches.

Creating a branch

Today, no self-respecting developer needs others to create branches. Since we are not developing against a common “development” branch any more but using feature branches, we can create one ourselves and start developing a new feature.

```
git checkout -b my-pr
```

We created a branch called `my-pr`.

We’ll skip the steps to create a personal project-specific development environment with DevPods. Instead, we’ll use the following commands to make a few simple changes. Our goal here is not to develop something complex, but rather to make just enough changes to help us distinguish the new from the old release while moving it through the process of creating a pull request and a preview environment.

Modifying the code

So, we'll change our `hello` message to something else. I'll save you from opening your favorite IDE and changing the code yourself by providing a few simple `cat`, `sed`, and `tee` commands that will make the changes for us.

```
cat main.go | sed -e \
    "s@hello, devpod with tests@hello, PR@g" \
    | tee main.go

cat main_test.go | sed -e \
    "s@hello, devpod with tests@hello, PR@g" \
    | tee main_test.go
```

We changed the code of the application and the tests so that the output message is `hello, PR` instead of `hello, devpod with tests`. Those changes are not much different than the changes we did in the previous chapters. The only notable difference is that this time we're not working with the `master`, but rather with a feature branch called `my-pr`.

There's one more change we'll make.

Disabling one of the databases

Right now, MongoDB is defined in `charts/go-demo-6/requirements.yaml`, and it will run as a replica set. We do not need that for previews. A single replica DB should be more than enough. We already added a non-replicated MongoDB as a dependency of the preview (`charts/preview/requirements.yaml`) when we created a custom build pack. Since we have the DB defined twice (once in the app chart, and once in the preview), we'd end up with two DBs installed. We don't need that, so we'll disable the one defined in the application chart and keep only the one from the preview chart (the single replica DB).

To speed things up, we'll also disable the persistence of the DB. Since previews are temporary and used mostly for testing and manual validations, we don't need to waste time creating a persistent volume.

Please execute the command that follows

Please execute the command that follows.

```
echo "

db:
  enabled: false

preview-db:
  persistence:
    enabled: false" \
  | tee -a charts/preview/values.yaml
```

Committing and pushing the changes

The next steps should be no different than what you may be doing every day at your job. We'll commit and push the changes to the upstream.

```
git add .

git commit \
    --message "This is a PR"

git push --set-upstream origin my-pr
```

What comes next should be something you're doing all the time. We'll create a pull request, but we might do it slightly differently than what you're used to.

Creating a PR with CLI using `jx`

Typically, you would open GitHub UI and click a few buttons that would create a pull request. If you prefer UIs and the "common" way of creating PRs, please do so. On the other hand, if you'd like to do it using CLI, you'll be glad to know that `jx` allows us to create PRs as well. If a terminal is your weapon of choice, please execute the command that follows to create a PR.

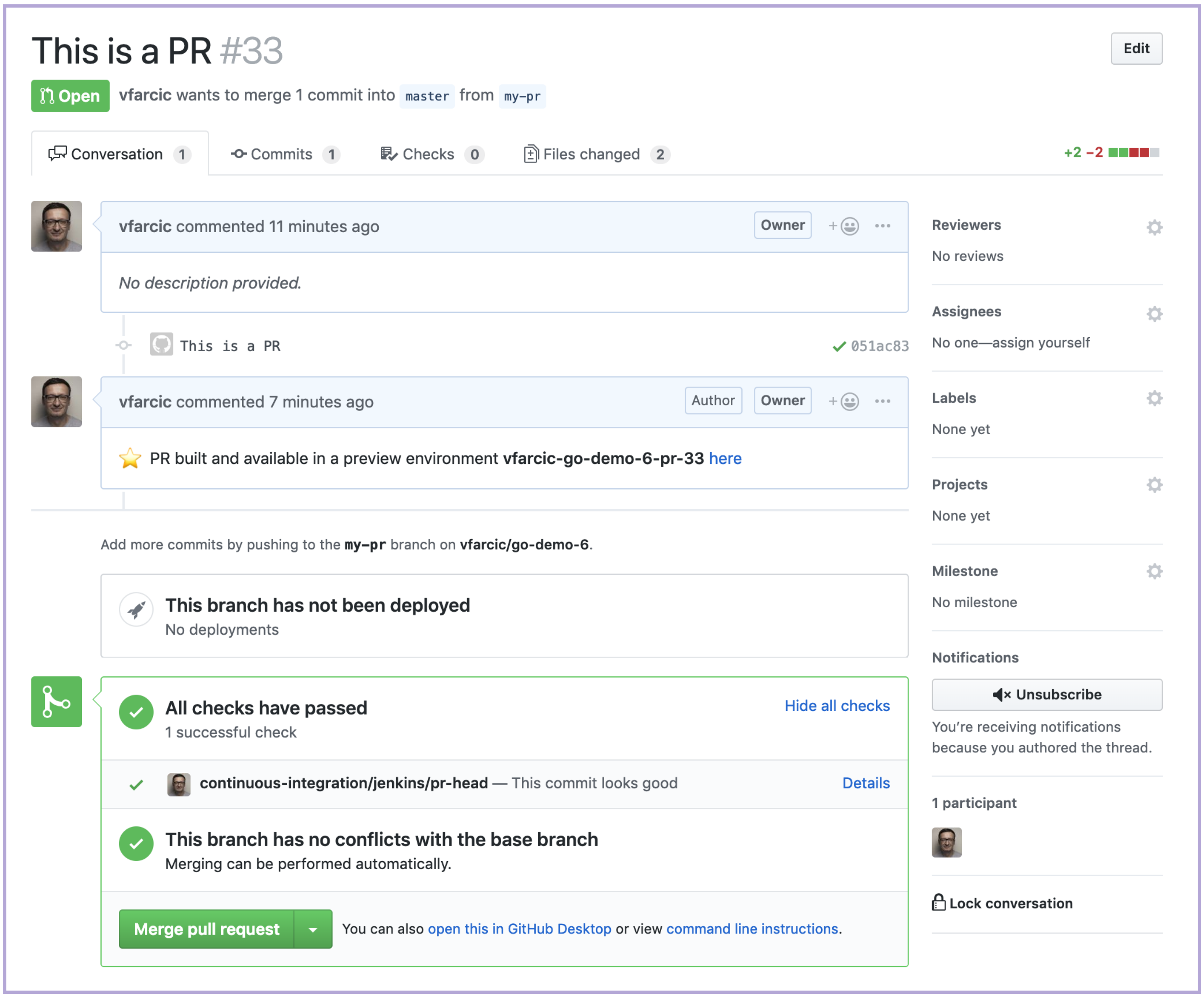
```
jx create pullrequest \
    --title "My PR" \
    --body "This is the text that describes the PR
and it can span multiple lines" \
    --batch-mode
```

The output should show the ID of the pull request as well as a link to see the pull request in a browser. Please open the link.

When the build started, Jenkins X notified GitHub that the validation is in progress, and you should be able to confirm that from the GitHub UI. If you do not see any progress, the build probably hasn't started yet, and you might need to refresh your screen after a while. When the build is finished and if it is successful, we should see a new comment stating that the PR is built and available in a preview environment [USER]-go-demo-6-pr-[PR_ID]. Keep in mind that you might need to refresh your browser to see the change of the pull request status.

What that comment tells us is that Jenkins X created a new environment (Namespace) dedicated to that PR, and it will keep it up-to-date. If we push additional changes to the pull request, the process will repeat, and it will end with new deployment to the same environment.

The end of the comment is a link (*here*). You can use it to open your application. Feel free to click the link and do not be confused if you see *503* or a similar error message. Remember that the *go-demo-6* application does not respond to requests to the root and that you need to add */demo/hello* to the address opened through the *here* link.



This is a PR #33

Open vfarcic wants to merge 1 commit into master from my-pr

vfarcic commented 11 minutes ago

No description provided.

This is a PR 051ac83

vfarcic commented 7 minutes ago

PR built and available in a preview environment **vfarcic-go-demo-6-pr-33** here

Add more commits by pushing to the **my-pr** branch on **vfarcic/go-demo-6**.

This branch has not been deployed
No deployments

All checks have passed
1 successful check

continuous-integration/jenkins/pr-head — This commit looks good

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

Reviewers: No reviews
Assignees: No one—assign yourself
Labels: None yet
Projects: None yet
Milestone: No milestone
Notifications: You're receiving notifications because you authored the thread.
1 participant

A pull request with checks completed and a PR built and available

If you are like me and you prefer CLIs, you might be wondering how to retrieve similar information from your terminal. How can we see the previews we have in the cluster? What are the associated pull requests? In which Namespaces are they running? What are the addresses we can use to access the previews?

Retrieving resources with CLI using `jx get`

Just like the `kubectl get` command allows you to retrieve any Kubernetes resource, `jx get` does the same but limited to the resources related to Jenkins X. We just need to know the name of the resource. The one we're looking for should be easy to guess.

```
jx get previews
```

The output is as follows.

```
PULL REQUEST                                NAMESPACE                APPLICATION
https://github.com/vfarcic/go-demo-6/pull/33 jx-vfarcic-go-demo-6-pr-33 http://go-demo-6.jx-vfarci
```

We can see from the output the address of the pull request in GitHub, the Namespace where the application and its dependencies were deployed, and the address through which we can access the application (through auto-generated **Ingress**).

Confirming the installed previews

Next, we'll confirm that the preview was indeed installed and that it is working by sending a simple request to `/demo/hello`.

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

```
PR_ADDR=[...]

curl "$PR_ADDR/demo/hello"
```

The output should reflect the changes to the code that we made before creating the pull request. It should display `hello, PR!`

The critical thing to understand is that every pull request will be deployed to its own environment unless we change the default behavior.

I will let you “play” a bit with `jx` CLI. Explore the logs of the pull request, output the activities, and run the other commands we learned so far.