Exploring Jenkins X Environments

This lesson discusses Jenkins X environments and their types in detail.

We'll cover the following

- NAME column
  - dev environment
  - production environment
- KIND column
  - Development environments
  - Permanent environments
- PROMOTE column
  - Never
  - Auto
  - Manual
- NAMESPACE column
- SOURCE column
- Applying filters on environments

We'll continue using the *go-demo-6* application. This time, we'll dive deeper into the role of the `staging` environment and how it relates to the process executed when we push a change to an application.

Let's take a look at the environments we currently have.

```
jx get env
```

The output is as follows.

```
NAME       LABEL       KIND        PROMOTE NAMESPACE     ORDER CLUSTER SOURCE
dev        Development Development Never   jx            0
staging    Staging     Permanent   Auto    jx-staging    100           https://github.com/vfarcic/
production Production  Permanent   Manual  jx-production 200           https://github.com/vfarcic/
```

`NAME` column

We already experienced the usage of the `staging` environment, while the other two might be new.

`dev` environment

The `dev` environment is where Jenkins X and all the other applications that are involved in continuous delivery are running. That's also where agent Pods are created and live during the duration of builds. Even if we were not aware of it, we already used that environment or, to be more precise, the applications running there.

`production` environment

The `production` environment is still unused, and it will remain like that for a while longer. That's where we'll deploy our production releases. But, before we do that, we need to learn how Jenkins X treats pull requests.

`KIND` column

Aside from the name of an environment, you'll notice a few other potentially important pieces of information in that output.

Our current environments are split between the `Development` and `Permanent` kinds.

`Development` environments

This is where the action (building, testing, etc.) is happening.

`Permanent` environments

On the other hand, this is where our releases should run indefinitely.

`PROMOTE` column

Typically, we don't remove applications from those environments, but rather upgrade them to newer releases. The `staging` environment is where we install (or upgrade) new releases for the final round of testing. The current setup will automatically deploy an application there every time we push a change to the master branch. We can see that through the `PROMOTE` column.

`Never`

The `dev` environment is set `Never` to receive promotions. New releases of our applications will not run there.

`Auto`

The `staging` environment, on the other hand, is set to `Auto` promotion. That means a new release will be deployed to that environment, and to all the others with promotion set to `Auto`.

`Manual`

The `production` environment has the promotion set to `Manual`. As a result, new releases will not be deployed there through a pipeline. Instead, we'll need to make a decision on which release will be deployed to production and when that should happen.

> We'll explore how promotions work soon. For now, we're focusing only on the purpose of the environments, not the mechanism that allows us to promote a release.

`NAMESPACE` column

We can also see the relationship between an environment and a Kubernetes Namespace. Each environment is a Namespace.

The `production` environment, for example, is mapped to Kubernetes Namespace `jx-production`.

`SOURCE` column

Finally, the `SOURCE` column tells us which Git repository is related to an environment. Those repositories contain all the details of an environment and only a push to one of those will result in new deployments. We'll explore them soon.

Needless to say, we can change the behavior of any of the environments, and we can create new ones.

> 📝 We have not yet explored the `preview` environments simply because we did not yet create a PR that would trigger the creation of such an environment. We'll dive into pull requests soon. For now, we'll focus on the environments we have so far.

Applying filters on environments

We have only three environments. With such a low number, we probably do not need to use filters when listing them. But, that number can soon increase. Depending on how we're organized, we might give each team a separate environment. Jenkins X implements a concept called teams that we'll explore later. The critical thing to note is that we can expect the number of environments to increase and that might create a need to filter the output.

> 🔍 When running the commands that follow, please imagine that the size of our operations is much bigger and that we have tens or even hundreds of environments.

Let's see which environments are configured to receive promotions automatically.

```
jx get env --promote Auto
```

`Auto` in the command given above is case sensitive. The output should show that we have only one environment (`staging`) with automatic promotion.

Similarly, we could have used `Manual` or `Never` as the filters applied to the `promote` field (`--promote`).

Before we move further, we'll have to go through a rapid discussion about the type of tests we might need to run. That will set the scene for the changes we'll apply to one of our environments.