What is Jenkins X Boot?

This lesson introduces us to the concept of Jenkins X Boot and explains how it tackles the previous issues.

We'll cover the following

- The current shortcomings
- How does Jenkins X boot handle these issues?
- The issue with Jenkins X boot
- The solution to the problem

The current shortcomings

What's wrong with `jx create cluster` and `jx install` commands? Why do we need a different way to install, manage, and upgrade Jenkins X? Those are ad-hoc commands that do not follow GitOpts principles.

- They are not *idempotent* (you cannot run them multiple times and expect the same result).
- They are not *stored in Git*, at least not in a form that the system can interpret and consume in an attempt to converge the desired into the actual state.
- They are not *declarative*.

We know all those reasons. They are the same issues we're trying to get rid of with other applications by implementing pipelines and declarative configurations and storing everything in Git. But, there is one more reason that we did not discuss yet.

> The `jx` CLI contains a lot of custom-made code written specifically for Jenkins X.

Part of it, however, consists of wrappers. The `jx create pullrequest` command is a wrapper around `hub`. Similarly, `jx create cluster` is a wrapper around `az` (AKS), `eksctl` (EKS), `gcloud` (GKE), and a few others. That's all great because `jx` allows us

to have a single user-facing CLI that enables us to do (almost) everything we need, at least with the tasks related to the application lifecycle. But, in my opinion, Jenkins X tried to do more than it should. Its scope went overboard by attempting to enter into the configuration management sphere and trying to provide ways to create Kubernetes clusters. Truth be told, that was done mostly so that project contributors could spin up clusters easily.

Nevertheless, the feature became popular, and many started using it. That begs the question. *If many are adopting something, isn't that a sign that it is useful?*

Over time, the number of requests to add additional features (arguments) to `jx create cluster` kept increasing. People were missing things that are already available in the arguments of the commands it was wrapping, so the community kept adding them. At one point, the command became too confusing with too many arguments, and it went far beyond the initial idea to quickly spin up a cluster, mostly for demo purposes.

How does Jenkins X boot handle these issues?

Jenkins X boot is a result of two primary needs;

- to create infrastructure,
- to install and maintain Jenkins X and other system-level components.

It, in a way, puts the creation of the cluster back to the commands like `gcloud`, `az`, and `eksctl` (to name a few). At the same time, it urges people not to use them and to opt for tools like **Terraform** to create a Kubernetes cluster. On the other hand, it transforms the `jx install` process from being a command into a GitOps-based process. As you will see soon, `jx boot` entirely depends on a set of declarative files residing in a Git repository. It embraces GitOps principles, as it should have been done from the start.

So, given that having a declarative GitOps-based process is better than one-shot commands, you might be asking, *"why did it take the community so long to create it?"* Jenkins X Boot was released sometime around August 2019, over a year and a half after Jenkins X project started. **What took us so long?**

The issue with Jenkins X boot

Jenkins X Boot had the *"chicken and egg"* type of a problem. Jenkins X assumes that

everything is a pipeline triggered by a Git webhook. We make a change to Git, Git triggers a webhook, and that webhook notifies the cluster that there is a change

that should trigger a pipeline that will ultimately converge the actual into the desired state.

The process that performs that convergence is executed by Jenkins X running in that cluster. Now, if we say that the installation of Jenkins X should follow the same process, *who will run those steps?* **How can we make Jenkins X install itself?** If it doesn't exist, it cannot run the process that will install it. When there is nothing in a cluster, nothing *cannot* create something. That would be magic.

The solution to the problem

The solution to the problem was to allow `jx` CLI to run a pipeline as well. After all, until we install Jenkins X inside a Kubernetes cluster, the only thing we have is the CLI. So, the community made that happen. It created the necessary changes in the CLI that allows it to run pipelines locally. After that, all that was needed was to create a pipeline definition that we can use together with a bunch of configuration files that define all aspects of Jenkins X installation.

As a result, now we can use Jenkins X Boot to run the first build of a pipeline specifically defined for installation and upgrades of Jenkins X and all its dependencies. To make things even more interesting, the same pipeline can (and should) be executed by Jenkins X inside a Kubernetes cluster. Finally, we can have the whole platform defined in a Git repository, let a pipeline run the first time locally and install Jenkins X, as well as configure webhooks in that repo so that each consecutive change to it triggers new builds that will change the actual state of the platform into the desired one. There is no need to run the pipeline locally ever again after the initial run. From there on, it'll be handled by Jenkins X inside a Kubernetes cluster just like any other pipeline. Finally, we can treat Jenkins X just like any other application (platform) controlled by pipelines and defined as declarative files.

We could summarize all that by saying that ad-hoc commands are bad, that GitOps principles backed by declarative files are good, and that Jenkins X can be installed and managed by pipelines, no matter whether they are run by a local CLI or through Jenkins X running inside a Kubernetes cluster. Running a pipeline from a local CLI allows us to install Jenkins X for the first time (when it's still not running inside the cluster). Once it's installed, we can let Jenkins X maintain itself by

changing the content of the associated repository. As an added bonus, given that a pipeline can run both locally and inside a cluster, if there's something wrong with Jenkins X in Kubernetes, we can always repair it locally by re-running the pipeline. *Neat, isn't it?*

Still, confused? Let's see all that in practice in the next lesson.