


# Getting Started with Applying GitOps Principles in Jenkins X

In this lesson, we look back on the exercises that we have done and determine if we have been applying GitOps principles.

## We'll cover the following

- Are we applying GitOps principles?
  - Why not use a single one-off command instead?
- Summary

 At the time of this writing (February 2020), the examples in this chapter are validated only with **serverless Jenkins X** in **GKE** and **EKS**. Jenkins X Boot is currently verified by the community to work only there, even though it likely works in AKS and other Kubernetes flavors. Over time, the community will be adding support for all Kubernetes distributions, and, with a slight delay, I will be updating this chapter to reflect that. Still, there will be an inevitable delay between the progress of that support and me incorporating it into this book, so I strongly advise you to check the official documentation to see whether your Kubernetes flavor is added.

If there is a common theme in Jenkins X, that is *GitOps*. The platform was designed and built around the idea that:

- **Everything is defined as code.**
- **Everything is stored in Git.**
- **Every change to the system is initiated by a Git webhook.**

It's a great concept, and we should ask ourselves whether we were applying it in all our examples. *What do you think?*

## Are we applying GitOps principles?

Using a single `jx create cluster` command to create a whole cluster and install all the tools that comprise Jenkins X is excellent. *Isn't it?*

The community behind Jenkins X thought that providing a single command that will create a cluster and install Jenkins X is a great idea. But we were wrong. What we did not envision initially was that there is an almost infinite number of permutations we might need. There are too many different hosting providers, too many different Kubernetes flavors, and too many components we might want to have inside our clusters. As a result, the number of arguments in `jx create cluster` and `jx install` commands was growing continuously. What started as a simple yet handy feature ended up as a big mess. Some were confused with too many arguments, while others thought that too many are missing. In my opinion, Jenkins X, in this specific context, tried to solve problems that were already addressed by other tools. We should not use Jenkins X to create a cluster. Instead, we should use the tool specialized for that. My favorite is **Terraform**, but almost any other (e.g., **CloudFormation**, **Ansible**, **Puppet**, **Chef**, etc.) would do a better job. What that means is that Jenkins X should assume that we already have a fully operational Kubernetes cluster. We should have an equivalent of the `jx install` command, while `jx create cluster` should serve particular use-cases, mostly related to demos and playgrounds, not a real production setting.

## Why not use a single one-off command instead? #

Now, you might just as well say:

- "If I'm not going to use `jx create cluster`, why should we use configuration management tools like **Terraform**?"
- "Isn't it easier to simply run `gcloud`, `eksctl`, or `az` CLI?"

It is indeed tempting to come up with a single one-off command to do things. But, that leads us to the same pitfalls as when using UIs. We could just as easily replace the command with a button. But that results in a non-reproducible, non-documented, and not omnipotent way of doing things. Instead, we want to modify some (preferably declarative) files, push them to Git, and let that initiate the process that will converge the actual with the desired state. Or, it can be vice versa as well. We could run a process based on some files and push them later. As long as a Git repository always contains the desired state, and the system's actual state

matches those desires, we have a reproducible, documented, and (mostly) automated way to do things. Now, that does not mean that commands and buttons are always a bad thing. Running a command or pressing a button in a UI works well only when that results in changes of the definition of the desired state being pushed to a Git repository so that we can keep track of changes. Otherwise, we're just performing arbitrary actions that are not documented and not reproducible. What matters is that Git should always contain the current state of our system. How we get to store definitions in Git is of lesser importance.

## Summary #

I'm rambling how bad it is to run arbitrary commands and pushing buttons because that's what we were doing so far. To be fair, we did follow GitOps principles most of the time, but one crucial part of our system kept being created somehow arbitrarily. So far, we were creating a Kubernetes cluster and installing Jenkins X with a single `jx create cluster` command, unless you are running it in an existing cluster. Even if that's the case, I taught you to use the `jx install` command. No matter which of the two commands you used, the fact is that we do not have a place that defines all the components that constitute our Jenkins X setup. That's bad, and it breaks the GitOps principle that states that everything is defined as code and stored in a Git repository, and that Git is the only one that initiates actions aimed at converging the actual into the desired state. We'll change that next by adding the last piece of the puzzle that prevented us from having the full system created by following the GitOps principles.

---

We'll learn how *Jenkins X Boot* works. But before we do that, in the next lesson, we'll take a quick look at a cardinal sin we were committing over and over again.