Discussing the Cardinal Sin

In this lesson, we will discuss the approach that we were previously using, what was wrong with it and how we are going to rectify the situation.

We'll cover the following

How did we specify dependencies in Git?

Our applications and their direct third-party dependencies are managed by Jenkins X pipelines. A process is initiated with a push to a Git repository, thus adhering to GitOps principles. If our applications have direct third-party dependencies, we'll specify them in `requirements.yaml`. We've been doing that for a while, and there is probably no need to revisit the process that works well.

If we move higher into system-level third-party applications, we had mixed results so far. We could specify them as dependencies in environment repositories. If, for example, we need **Prometheus** in production, all we have to do is define it as yet another entry in `requirements.yaml` in the repository associated with it. Similarly, if we'd like to deploy it to the staging environment for testing, we could specify it in the repository related to staging. So far, so good. We were following GitOps principles. But, there was one crucial application in this group that we chose to deploy ignoring all the best practices we established so far. The one that eluded us is Jenkins X itself.

Where did we go wrong?

I was very vocal that we should enforce GitOps principles as much as we can. I insisted that everything should be defined as code, that everything should be stored in Git, and that only Git can initiate actions that will change the state of the cluster. Yet, I failed to apply that same logic to the creation of a cluster and installation of Jenkins X. Running commands like `jx create cluster` and `jx install` goes against everything I preached (at least in this course). Those commands were not stored in Git, and they are not idempotent. Other people working with us would not be able to reproduce our actions. They might not even know what was done and when.

Why did we use this approach?

So, why did I tell you to do things that go against my own beliefs?

Running `jx create cluster` command was the easiest way to get you up to speed with Jenkins X quickly. It is the fastest way to create a cluster and install Jenkins X. At the same time, it does not need much of an explanation. *Run a command, wait for a few minutes, and start exploring the examples that explain how Jenkins X works.* For demos, for workshops, and for examples in a course, `jx create cluster` is probably the best way to get up-and-running fast. But, the time has come to say that such an approach is not good enough for any serious usage of Jenkins X. We explored most of the essential features Jenkins X offers, and I feel that you are ready to start using it in "real world". For that to happen, we need to be able to set up Jenkins X using the same principles we applied to other use-cases. We should store its definition in Git, and we should allow webhooks to notify the system if any aspect of Jenkins X should change. We should have a pipeline that drives Jenkins X through different stages so that we can validate that it works correctly. In other words, Jenkins X should be treated the same way as other applications. Or, to be more precise, maybe not in exactly the same way but, at least, with a process based on the same principles. We'll accomplish that with `jx boot`. But, before we explore it, there is one more level we should discuss.

What are we going to change now?

We used `jx create cluster` both to create a cluster and to install Jenkins X. The latter part will, from now on, be done differently using `jx boot`. We still need to figure out how to create a cluster using code stored in Git. But, for better or worse, I won't teach you that part. That's outside the scope of this course. The only thing I will say is that you should define your whole infrastructure as code using one of the tools designed for that. My personal preference is **Terraform by HashiCorp**. However, since infrastructure-as-code is not the subject, I won't argue why I prefer Terraform over other tools. You can use anything you like; **CloudFormation**,

Ansible, **Puppet**, **Chef**, **SaltStack**, etc.

All in all, we'll continue deploying our application and associated third-party services as well as system-level third-party apps using Jenkins X pipelines. We'll explore `jx boot` as a much better way to install and manage Jenkins X. As for infrastructure, it'll be up to you to choose what you prefer. From now on, I will provide commands to create a Kubernetes cluster. Do NOT take those commands as a recommendation. They are only the representation of my unwillingness to dive into the subject of *"infrastructure as code"* in this course. So, from now on, whenever you see `gcloud`, `eksctl`, or `az` commands, remember that is not what you should do with the production system.

Now that we got that out of the way, let's create a Kubernetes cluster and start "playing" with `jx boot` in the next lesson.