Exploring the Changes Done By the Boot

In this lesson, we will take a look at all the files that were changed because of Jenkins X Boot.

We'll cover the following

Now, let's take a look at the changes Jenkins X Boot did to the local copy of the repository.

```
jx repo --batch-mode
```

Please click the *commits* tab and open the last one. You'll see a long list of the files that changed.

That's a long list, isn't it? Jenkins X Boot changed a few files. Some of those changes are based on our answers, while others are specific to the Kubernetes cluster and the vendor we're using.

We won't comment on all the changes that were done, but rather on the important ones, especially those that we might choose to modify in the future.

The `env/parameters.yaml` file

If you take a closer look at the output, you'll see that it created the `env/parameters.yaml` file. Let's take a closer look at it.

```
cat env/parameters.yaml
```

The output, in my case, is as follows.

```
adminUser:
  password: vault:jx-boot/adminUser:password
  username: admin
enableDocker: false
pipelineUser:
  email: viktor@farcic.com
  token: vault:jx-boot/pipelineUser:token
  username: vfarcic
prow:
  hmacToken: vault:jx-boot/prow:hmacToken
```

The `parameters.yaml` file contains the data required for Jenkins X to operate correctly. There are the administrative `username` and `password` and the information that Docker is not enabled (`enableDocker`). The latter is not really Docker but rather that the system is not using an internal Docker Registry but rather an external service. Further on, we can see the `email`, the `token`, and the `username` pipeline needs for accessing our GitHub account. Finally, we can see the `hmacToken` required for the handshake between GitHub webhooks and `prow`.

As you can see, the confidential information is not available in the plain-text format. Since we are using Vault to store secrets, some values are references to the Vault storage. That way, it is safe for us to keep that file in a Git repository without fear that sensitive information will be revealed to prying eyes.

We can also see from the `git diff` output that the `parameters.yaml` did not exist before. Unlike `requirements.yml` that existed from the start (and we modified it), that one was created by `jx boot`.

Which other file did the Boot process create or modify?

The `jenkins-x.yml` file

We can see that it made some modifications to the `jenkins-x.yml` file. We did not explore it just yet, so let's take a look at what we have right now.

```
cat jenkins-x.yml
```

The output is too long to be presented in a book, and you can see it on your screen anyway.

That is a `jenkins-x.yml` file like any other Jenkins X pipeline. The format is the same, but the content is tailor-made for the Boot process. It contains everything it needs to install or, later on, upgrade the platform.

What makes this pipeline unique, when compared with those we explored earlier, is that the `buildPack` is set to `none`. Instead of relying on a buildpack, the whole pipeline is defined in that `jenkins-x.yml` file. Whether that's a good thing or bad depends on the context. On the one hand, we will not benefit from future changes the community might make to the Boot process. On the other hand, those changes are likely going to require quite a few other supporting files. So, the community decided not to use the buildpack. Instead, if you'd like to update the Boot process, you'll have to merge the repository you forked with the upstream.

Now, let's get to business and take a closer look at what's inside `jenkins-x.yml`.

We can see that it is split into two pipelines:

1. **`pullRequest`**
2. **`release`**

The `pullRequest` pipeline

The `pullRequest` is simple, and it consists of a single `stage` with only one `step`. It executes `make build`. If you take a look at `env/Makefile` you'll see that it builds the charts in the `kubeProviders` directory, and afterward, it lints it. The real purpose of the `pullRequest` pipeline is only to validate that the formatting of the charts involved in the Boot process is correct. It is very similar to what's being done with pull requests to the repositories associated with permanent environments like staging and production.

The "real" action is happening in the `release` pipeline, which, as you already know, is triggered when we make changes to the master branch.

The `release` pipeline

The `release` pipeline contains a single stage with the same name. What makes it special is that there are quite a few steps inside it, and we might already be familiar with them from the output of the `jx boot` command.

We'll go through the steps very briefly. All but one of those are based on `jx` commands, which you can explore in more depth on your own.

The list of the steps, sorted by order of execution, is as follows.

Step	Command	Description
`validate-git`	`jx step git validate`	Makes sure that the `.gitconfig` file is configured correctly so that Jenkins X can interact with our repositories
`verify-preinstall`	`jx step verify preinstall`	Validates that our infrastructure is set up correctly before the process installs or upgrades Jenkins X
`install-jx-crds`	`jx upgrade crd`	Installs or upgrades Custom Resource Definitions required by Jenkins X
`install-velero`	`jx step helm apply`	Installs or upgrades **Velero** used for creating backups of the system
`install-velero-backups`	`jx step helm apply`	Defines the schedule for Velero backups
`install-nginx-controller`	`jx step helm apply`	Installs nginx Ingress
`create-install-values`	`jx step create install values`	Adds missing values (if there are any) to the

	`values`	there are any) to the `cluster/values.yaml` file used to install cluster-specific charts
`install-external-dns`	`jx step helm apply`	Installs or upgrades the support for external DNSes
`install-cert-manager-crds`	`kubectl apply`	Installs or upgrades **CertManager** CRDs
`install-cert-manager`	`jx step helm apply`	Installs or upgrades **CertManager** in charge of creating **Let’s Encrypt** certificates
`install-acme-issuer...`	`jx step helm apply`	Installs or upgrades **CertManager** issuer
`install-vault`	`jx step boot vault`	Installs or upgrades **HashiCorp** Vault
`create-helm-values`	`jx step create values`	Creates or updates the `values.yaml` file used by Charts specific to the selected Kubernetes provider
`install-jenkins-x`	`jx step helm apply`	Installs Jenkins X
`verify-jenkins-x-env...`	`jx step verify`	Verifies the Jenkins X environment
`install-repositories`	`jx step helm apply`	Makes changes to the repositories associated with environments

		(e.g., webhooks)
`install-pipelines`	`jx step scheduler`	Creates Jenkins X pipelines in charge of environments
`update-webhooks`	`jx update webhooks`	Updates webhooks for all repositories associated with applications managed by Jenkins X
`verify-installation`	`jx step verify install`	Validates Jenkins X setup

Please note that some of the components (e.g., Vault) are installed, upgraded, or deleted depending on whether they are enabled or disabled in `jx-requirements.yml`.

As you can see, the process consists of the following major groups of steps.

- Validate requirements
- Define values used to apply **Helm** charts
- Apply **Helm** charts
- Validate the setup

The **Helm** charts used to set up the system are stored in `systems` and `env` directories. They do not contain templates, but rather only `values.yaml` and `requirements.yaml` files. If you open any of those, you'll see that the `requirements.yaml` file is referencing one or more Charts stored in remote **Helm** registries.

That's all we should know about the `jenkins-x.yml` file, at least for now. The only thing left to say is that you might choose to extend it by adding steps specific to your setup, or you might even choose to add those unrelated with Jenkins X. For now, I will caution against such actions. If you do decide to modify the pipeline, you might have a hard time merging it with upstream. That, by itself, shouldn't be a big deal, but there is a more important reason to exercise caution. I did not yet a big deal, but there is a more important reason to exercise caution. I did not yet explain everything there is to know about Jenkins X Boot. Specifically, we are yet to

explore Jenkins X Apps (not to be confused with Applications). They allow us to add additional capabilities (components, applications) to our cluster and manage them through Jenkins X Boot. We'll get there in due time. For now, we'll move to yet another file that was modified by the process.

The `jx-requirements.yml` file

Another file that changed is `jx-requirements.yml`. Part of the changes are those we entered, like `clusterName`, `environmentGitOwner`, and quite a few others. But, some of the modifications were done by `jx boot` as well. Let's take a closer look at what we got.

```
cat jx-requirements.yml
```

My output is as follows (yours will likely differ).

```
autoUpdate:
  enabled: false
  schedule: ""
bootConfigURL: https://github.com/vfarcic/jenkins-x-boot-config
cluster:
  clusterName: jx-boot
  environmentGitOwner: vfarcic
  gitKind: github
  gitName: github
  gitServer: https://github.com
  namespace: jx
  project: devops-26
  provider: gke
  registry: gcr.io
  zone: us-east1
environments:
- ingress:
    domain: 34.74.196.229.nip.io
    externalDNS: false
    namespaceSubDomain: -jx.
    tls:
      email: ""
      enabled: false
      production: false
  key: dev
- ingress:
    domain: ""
    externalDNS: false
    namespaceSubDomain: ""
    tls:
      email: ""
      enabled: false
      production: false
  key: staging
```

```
- ingress:
    domain: ""
    externalDNS: false
    namespaceSubDomain: ""
    tls:
      email: ""
      enabled: false
      production: false
  key: production
gitops: true
ingress:
  domain: 34.74.196.229.nip.io
  externalDNS: false
  namespaceSubDomain: -jx.
  tls:
    email: ""
    enabled: false
    production: false
kaniko: true
repository: nexus
secretStorage: vault
storage:
  backup:
    enabled: false
    url: ""
  logs:
    enabled: true
    url: gs://jx-boot-logs-0976f0fd-80d0-4c02-b694-3896337e6c15
  reports:
    enabled: true
    url: gs://jx-boot-reports-8c2a58cc-6bcd-47aa-95c6-8868af848c9e
  repository:
    enabled: true
    url: gs://jx-boot-repository-2b84c8d7-b13e-444e-aa40-cce739e77028
vault: {}
velero: {}
versionStream:
  ref: v1.0.214
  url: https://github.com/jenkins-x/jenkins-x-versions.git
webhook: prow
```

As you can see both from `git diff` and directly from `jx-requirements.yml`, the file does not contain only the changes we made initially. The `jx boot` command modified it as well by adding some additional information. I'll assume that you do remember which changes you made, so we'll comment only on those done by Jenkins X Boot.

> My output might not be the same as the one you see on the screen. Jenkins X Boot might have features that did not exist at the time I wrote this chapter (October 2019). If you notice changes that I am not commenting on, you'll have to consult the documentation to get more information.

The `autoUpdate` section

At the very top, we can see that `jx boot` added the `autoUpdate` section, which did not even exist in the repository we forked. That's normal since that repo does not necessarily contain all the entries of the currently available Boot schema. So, some might be added even if they are empty.

We won't go into `autoUpdate` just yet. For now, we're interested only in installation. Updates and upgrades are coming later.

The `cluster` section

Inside the `cluster` section, we can see that it set `gitKind` and `gitName` to `github` and `gitServer` to `https://github.com`. Jenkins X's default assumption is that you're using GitHub. If that's not the case, you can change it to some other Git provider. As I already mentioned, we'll explore that together with Lighthouse later. It also set the `namespace` to `jx`, which, if you remember, is the default Namespace for Jenkins X.

The `environments` section

The `environments` section is new, as well. We can use one of its sub-entries to change Ingress used for a specific environment (e.g., `production`). Otherwise, the `ingress` entries (at the root) are applied to the whole cluster (to all the environments).

Speaking of `ingress`, you'll notice that the `domain` entry inside it was auto-generated. Since we did not specify a domain ourselves, it used `.nip.io` in the same way we were using it so far. Ingress, both on the cluster and on the environment level, is a separate topic that we'll explore later.

Finally, we can see that it added values to `url` entries in `storage`. We could have created storage and set those values ourselves. But we didn't. We let Jenkins X Boot create it for us and update those entries with the URLs. We'll explore in more detail what we can do (if anything) with those storages. For now, just note that the Boot created them for us.

That's it. Those are all the files that were changed.

As you already saw, Jenkins X Boot ran a pipeline that installed the whole platform. Given that we did not have Jenkins X running inside the cluster, that pipeline was executed locally. But, we are not expected to keep maintaining our setup by running pipelines ourselves. If, for example, we'd like to upgrade some aspect of the platform, we should be able to push changes to the repository and let the system run the pipeline for us, even if that pipeline will affect Jenkins X itself. The first step towards confirming that is to check which pipelines we currently have in the cluster.

In the next lesson, we will verify the Jenkins X Boot installation.