Install Jenkins X in an Existing Kubernetes Cluster

In this lesson, we will discuss the steps to install Jenkins X in an existing Kubernetes cluster.

We'll cover the following

I will assume that your cluster passed compliance tests, or if you did not execute them, that you are confident that it works according to Kubernetes specifications and best practices. Also, I'm assuming that it is accessible from the outside world. To be more precise, the cluster needs to be reachable from GitHub, so that it can send webhook notifications whenever we push code changes. Please note that the accessibility requirement is valid only for the purpose of the exercises. In a real-world situation you might use a Git server that is inside your network (e.g., GitLab, BitBucket, GitHub Enterprise, etc.).

Installing Jenkins X in an existing Kubernetes cluster

Now that we have run the compliance tests that showed our cluster complies with Kubernetes, we can install Jenkins X.

IP address

We'll need a few pieces of information before we install the tools we need. The first in line is the IP address.

Typically, your cluster should be accessible through an external load balancer. Assuming that we can guarantee its availability, an external load balancer provides a stable entry point (IP address) to the cluster. Its job is to ensure that the requests are forwarded to one of the healthy nodes of the cluster. That is, more or less, all we need an external load balancer for.

If you do have an external LB, please get its IP address. If you don't, you can use the IP address of one of the worker nodes of the cluster, as long as you understand that a failure of that node would make everything inaccessible.

Please replace `[...]` with the IP of your load balancer or one of the worker nodes before you execute the command that follows.

```
LB_IP=[...]
```

Domain

Next, we need a domain. If you already have one, make sure that its DNS records are pointing to the cluster. Please replace `[...]` with the domain before you execute the command that follows.

```
DOMAIN=[...]
```

If you do NOT have a domain, we can combine the IP with the nip.io service to create a fully qualified domain. If that's the case, please execute the command that follows.

```
DOMAIN=$LB_IP.nip.io
```

Select provider

We need to figure out which provider to use. The available providers are the same as those you saw through the `jx create cluster help` command. For your convenience, we'll list them again through the `jx install` command.

```
jx install --help | grep "provider="
```

The output is as follows.

```
--provider='': Cloud service providing the Kubernetes cluster.  Supported providers: aks, aws, eks
```

As you can see, we can install Jenkins in **AKS, AWS (created with kops), EKS, GKE, ICP, IKS, Minikube, Minishift, OKE, OpenShift, and PKS**. The two I skipped from that list are `jx-infra` and `kubernetes`. The former is used mostly internally by the maintainers of the projects, while the latter (`kubernetes`) is a kind of a wildcard provider. We can use it if our Kubernetes cluster does not match any of the available providers (e.g., Rancher, Digital Ocean, etc.).

All in all, if your Kubernetes is among one of the supported providers, use it. Otherwise, choose the `kubernetes` provider. There are two exceptions though; Minikube and Minishift run locally and are not accessible from GitHub. Please avoid them since some of the features will not be available. The main features missing are GitHub webhook notifications. While that might sound like a minor issue, they are a crucial element of the system we're trying to build. Jenkins X relies heavily on GitOps which assumes that any change is stored in Git and that every push could potentially initiate some processes (e.g., deployment to the staging environment).

Please replace `[...]` with the selected provider in the command that follows.

```
PROVIDER=[...]
```

NGINX Ingress controller

Do you have a NGINX Ingress controller running in your cluster? If you don't, `jx` will install it for you. In that case, feel free to skip the commands that declare the `INGRESS_*` variables. Also, when we come to the `jx install` command, remove the arguments `--ingress-namespace` and `--ingress-deployment`.

On the other hand, if you do have a NGINX Ingress controller, we need to find out which Namespace you installed it in. Let's list them and see which one hosts

Ingress.

```
kubectl get ns
```

The output is as follows.

```
NAME          STATUS AGE
default       Active 10m
ingress-nginx Active 6m
kube-public   Active 10m
kube-system   Active 10m
```

In my case, it’s `ingress-nginx`. In yours, it might be something else. Or, it might be inside the `kube-system` Namespace. If that’s the case, list the Pods with `kubectl --namespace kube-system get pods` to confirm that it’s there.

Before executing the command that follows, please replace `[...]` with the Namespace where Ingress resides.

```
INGRESS_NS=[...]
```

Next, we need to find out the name of the Ingress Deployment.

```
kubectl --namespace $INGRESS_NS get deployments
```

The output for my cluster is as follows, yours might differ.

```
NAME                     DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
nginx-ingress-controller 1       1       1          1         7m
```

In my case, the Deployment is called `nginx-ingress-controller`. Yours is likely named the same. If it isn’t, please modify the command that follows accordingly.

```
INGRESS_DEP=nginx-ingress-controller
```

Jenkins X installation command

Now we are finally ready to install Jenkins X into your existing Kubernetes cluster. Please make sure to remove `--ingress-*` arguments if you do not have an NGINX Ingress controller in your cluster and you want `jx` to install it.

```
jx install \
    --provider $PROVIDER \
    --external-ip $LB_IP \
    --domain $DOMAIN \
    --default-admin-password admin \
    --ingress-namespace $INGRESS_NS \
    --ingress-deployment $INGRESS_DEP \
    --default-environment-prefix jx-rocks
```

If, by any chance, you followed the instructions for GKE, EKS, or AKS, you'll notice that `jx install` executes the same steps as those performed by `jx cluster create`, except that the latter creates a cluster first. You can think of `jx install` as a subset of `jx cluster create`.

What kind do we want to install? Static or serverless?

Please answer with `Serverless Jenkins X Pipelines with Tekton`.

The process will create a `jx` Namespace. It will also modify your local `kubectl` context.

Git credentials

At this point, `jx` will try to deduce your Git name and email. If it fails to do so, it'll ask you for that info.

Ingress installation

The next in line is Ingress. The process will try to find it inside the `kube-system` Namespace and install it if it's not there. The process installs it through a Helm chart.

Long-term logs

You might be asked *to enable long term logs storage*. Make sure to answer with *n*. We will not need it for our exercises and, at the time of this writing, it is still in the "experimental" phase.

Git and Github settings

Next, we'll be asked a few questions related to Git and GitHub. You should be able to answer those. In most cases, all you have to do is confirm the suggested answer by pressing the enter key. As a result, `jx` will store the credentials internally so that it can continue interacting with GitHub on our behalf. It will also install the software necessary for the correct functioning of those environments (Namespaces) inside our cluster.

Jenkins X and other installations

Finally, the installation of Jenkins X and a few other applications (e.g., ChartMuseum for storing Helm charts) will start. The exact list of apps that will be installed depends on the Kubernetes flavor, the type of the setup, and the hosting vendor. But, before it proceeds, it will ask us a few more questions.

Organization

We're almost done. Only one question is pending. `Select the organization where you want to create the environment repository?` Choose one from the list.

GitOps

The process will create two GitHub repositories; `environment-jx-rocks-staging` that describes the staging environment and `environment-jx-rocks-production` for production. Those repositories will hold the definitions of those environments. For example, when you decide to promote a release to production, your pipelines will not install anything directly. Instead, they will push a change to `environment-jx-rocks-production` which, in turn, will trigger another job that will comply with the updated definition of the environment.

That's GitOps

Nothing is done without recording a change in Git. Of course, for that process to work, we need new jobs in Jenkins, so the process created two jobs that correspond to those repositories. We'll discuss the environments in greater detail later.

`kubectl` context change

Finally, the `kubectl` context was changed to point to the `jx` Namespace, instead of `default`.

Now you're ready to use Jenkins X.