ToiletPaper #88

Integrating cdk8s with Flux

Author: Maximilian Brenner / DevOps Engineer / Office Leipzig

✗ Intro

After I got used to cdk8s, I was curious how well it integrates with some current continuous delivery tools for Kubernetes. Therefore, I sat down for a quick session for integrating it with Flux. I will give you a short introduction for both tools to make sure you understand everything when we put them together. If you are already familiar with cdk8s and Flux you can probably skip the next two sections.

? Flux

Flux is a tool for deploying Kubernetes manifests to your cluster. It does so by following the GitOps paradigm which essentially says that one or more Git repositories are the single source of truth for the configuration of your system (in this case being Kubernetes). So in its simplest form imagine a Git repository with a single YAML file that contains a basic Kubernetes manifest (e.g. Deployment + Service). Flux will take care of applying this to your cluster. If you push more changes into this repository Flux will take care of keeping your cluster in the desired state.

Now, one could say: “Alright, so what’s the benefit over executing ~~kubectl apply -f *.yml~~ on every code change through my CI/CD system?” The answer is very straightforward… Flux simply doesn’t need such a system. Instead of running on any infrastructure that you or someone else has to manage Flux lives in the Kubernetes cluster itself. If the cluster is down, Flux can and will be down as well. If the cluster is up, Flux should be running.

? cdk8s

AWS Labs’ cdk8s is a framework that allows you to define Kubernetes deployments with object-oriented programming languages like TypeScript or Python. It provides you with predefined classes, so called structs, for each Kubernetes resource. Below you can find a part of some example code defining a single deployment:

```
k8s.Deployment(self, 'deployment',
               spec=k8s.DeploymentSpec(
                 selector=k8s.LabelSelector(match_labels="my-app"),
                 template=k8s.PodTemplateSpec(
                   metadata=k8s.ObjectMeta(labels="my-app"),
                   spec=k8s.PodSpec(containers=[
                     k8s.Container(
                       name='hello-kubernetes',
                       image='paulbouwer/hello-kubernetes:1.7',
                       ports=[k8s.ContainerPort(container_port=8080)])]))))
```

Out of this code cdk8s is able to generate valid manifest file(s).

If you wanna get a more detailed introduction feel free to check out my last toiletpaper.

✓ Putting them together

After getting to know the purpose of our two tools, one could ask how to put them together. How can I deploy my K8s manifests that I define and generate with cdk8s using Flux? As Flux only understands plain manifest files the main step that we have to automate is their generation. I came up with two possible solutions that you will find below. If you know of any better or different ways feel free to let me know.

→ The obvious way

What’s the first thing to do after celebrating that we don’t need a CI system when using Flux? Correct, introducing a CI system. ;) The idea is to put our cdk8s code into a repository and setting up a CI job that generates the Kubernetes manifests and pushes them to a separate branch. Afterwards we setup Flux to use this branch as its configuration source.

The bash script for our CI job could look like this:

```
cdk8s synth # `cdk8s synth` generates the K8s manifests and by default puts them in a folder called dist
git checkout --orphan manifests || git checkout manifests
git add dist
git commit -m "Update K8s manifests"
git push origin manifests
```

Be aware that the above script assumes that a valid cdk8s installation is already available. After putting this script into a CI pipeline that is being triggered on every code change the only thing that is left to do is correctly configuring Flux. We'll use the following **fluxctl** command:

```
fluxctl install \
  --git-user=${USER} \
  --git-email=${USER}@mycompany.com \
  --git-url=git@git.mycompany.com:${USER}/k8s-deployments \ # the URL of the Git 
repository containing the cdk8s code and the generated K8s manifests
  --git-branch=manifests \ # the branch that contains the K8s manifests
  --git-path=dist \ # the folder that contains the K8s manifests
  --namespace=flux | kubectl apply -f -
```

~~fluxctl~~ works very similar to other CLI tools in the Kubernetes ecosystem. It generates K8s manifests that you can directly pipe into ~~kubectl apply -f -~~ and that take care of setting up the Flux controller with the desired parameters.

After having configured everything correctly the complete flow will look like this:

1. some developer makes changes to the cdk8s code, e.g. in the *master* branch
2. the CI pipeline is triggered by the new commit
3. the bash script is being executed within the context of the *master* branch
4. the K8s *manifests* are being generated
5. the output is being pushed to the *manifests* branch
6. Flux syncs its copy of the *manifests* branch
7. Flux detects changes and applies them to the cluster

➔ The (not yet) elegant way

If you are someone that is familiar with Flux, I already hear you shouting at me: "Max, just use Flux's generators!" and that's exactly what my second solution is based on. Initially I thought this approach is better than the first one due to the fact that we don't need a CI system. Though while implementing it a big issue came up. But first things first, what is a Flux generator?

Generators allow you to produce new or modify existing Kubernetes manifests on the fly before applying them on your cluster. So instead of your manifests you place a simple ~~.flux.yaml~~ file along with your cdk8s code into your repository that could look like this:

```
version: 1
commandUpdated:
  generators:
  - command: cdk8s synth > /dev/null && cat dist/*
```

Flux will take of care of executing the command(s), capture the output, which should be valid K8s manifests and apply them on the cluster. Be aware that you have to mute every command that does generate output not containing K8s manifests. Otherwise you're gonna run into problems.

To enable the generators feature we have to pass the ~~--manifest-generation=true~~ parameters when setting up Flux:

```
fluxctl install \
--git-user=${USER} \
--git-email=${USER}@mycompany.com \
--git-url=git@git.mycompany.com:${GHUSER}/k8s-deployments \ # the URL of the Git 
repository containing the cdk8s code and the .flux.yaml file
--manifest-generation=true \ # enable generators
--namespace=flux | kubectl apply -f -
```

So afterwards we simply put the above ~~.flux.yaml~~ file next to our cdk8s code, push everything to a repo and we should be good to go, right? Of course not as here comes the big bummer. The execution of generators happen within the fluxd, the controller of Flux, container. This essentially means that we need to install the whole node/yarn- and Python/pip-stack into it to be able to execute cdk8s. For me, blowing up a container that needs to have full control over at least one K8s namespace to this extent is not acceptable. Anyway I checked if this approach is actually realizable so below you can find the working ~~.flux.yaml~~.

```
version: 1
commandUpdated:
  generators:
```

```
  - command: (/sbin/apk add yarn python3 npm && pip3 install pipenv && pipenv run
pip install constructs cdk8s && yarn global add cdk8s-cli && cdk8s import && cdk8s
synth) > /dev/null && cat dist/*
```

According to Flux’s documentation the developers are planning to allow executing generators in a separate container but for now this is not possible.

✚ Conclusion

So there you have it. Two options of setting up a GitOps process using cdk8s and Flux. For a productive setup I’d accept the additional effort of setting up / managing a CI system (which one already has in most cases anyway) and prefer the first option for now. As soon as Flux supports the execution of generators in a separate container the second option becomes more viable in my opinion.

All in all, I was surprised how well both tools work together. As cdk8s becomes more mature I’ll definitely take the two into consideration for setting up my next K8s delivery pipeline.