ToiletPaper #88

cdk8s mittels Flux integrieren

Autor: Maximilian Brenner / DevOps Engineer / Standort Leipzig

✕ Einführung

Nachdem ich sicher im Umgang mit cdk8s war, habe ich mich gefragt, wie gut es sich mit einigen aktuellen Continuous-Delivery-Tools für Kubernetes integrieren lässt. Daher habe ich mich kurzerhand mit der Integration mit Flux befasst. Folgend liefere ich eine kurze Einführung für beide Tools, um sicherzustellen, dass die Zusammenführung beider verständlich ist. Wer sich bereits mit cdk8s und Flux auskennt, kann die nächsten beiden Abschnitte wahrscheinlich überspringen.

➔ Flux

Flux ist ein Tool zum Deployment von Kubernetes-Manifesten auf eurem Cluster. Dabei folgt es dem GitOps-Paradigma, das im Wesentlichen besagt, dass ein oder mehrere Git-Repositorys die „Single Source of Truth“ für die Konfiguration eures Systems sind (in diesem Fall Kubernetes). Man stellt sich also ein Git-Repository in seiner einfachsten Form mit einer einzigen YAML-Datei vor, die ein einfaches Kubernetes-Manifest enthält (z.B. Deployment + Service). Flux wird dieses auf euer Cluster anwenden. Wenn man dann weitere Änderungen in dieses Repository lädt, wird Flux dafür sorgen, dass euer Cluster im gewünschten Zustand bleibt.

Nun könnte man sagen: „Na ja, aber was ist der Vorteil gegenüber der Ausführung von `kubectl apply -f *.yml` bei jeder Code-Änderung durch mein CI/CD-System?“ Die Antwort ist sehr einfach ... Flux braucht ein solches System einfach nicht. Anstatt auf irgendeiner Infrastruktur zu laufen, die ihr oder jemand anderes verwalten müssen, existiert Flux im Kubernetes-Cluster selbst. Wenn der Cluster ausfällt, wird auch Flux ausfallen. Wenn der Cluster läuft, sollte auch Flux laufen.

➔ cdk8s

Das cdk8s von AWS Labs ist ein Framework, das es ermöglicht, Kubernetes-Deployments mit objektorientierten Programmiersprachen wie TypeScript oder Python zu definieren. Es stellt vordefinierte Klassen, so genannte Structs, für jede Kubernetes-Ressource zur Verfügung. Nachfolgend findet ihr Teile eines Beispiel-Codes, der ein einzelnes Deployment definiert.

```
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)])])))))
```

cdk8s kann aus diesem Code eine gültige Manifestdatei(en) erzeugen.

Für eine detailliertere Einführung, schaut euch mein letztes ToiletPaper an.

✔ Beide zusammenführen

Nachdem man jetzt die Funktionsweise beider Tools kennt, könnte man sich fragen, wie man beide zusammenführen kann. Wie kann ich meine K8s-Manifeste, die ich mit cdk8s definiere und generiere, mit Flux deployen? Da Flux nur einfache Manifestdateien versteht, ist der wichtigste Schritt, den wir automatisieren müssen, ihre Generierung. Ich habe mir zwei mögliche Lösungen überlegt (s. unten). Wenn euch eine bessere oder andere Möglichkeit einfällt, kontaktiert mich gern.

✔ Der offensichtliche Weg

Was sollte man Erstes tun, nachdem man sich darüber gefreut hat, dass bei der Verwendung von Flux kein CI-System benötigt wird? Richtig, die Einführung eines CI-Systems ;) Die Idee ist, unseren cdk8s-Code in ein Repository zu stellen und einen CI-Job einzurichten, der die Kubernetes-Manifeste generiert und sie in einen separaten Branch schiebt. Danach richten wir Flux so ein, dass dieser Zweig als Konfigurationsquelle verwendet wird.

Das Bash-Skript für unseren CI-Job könnte wie folgt aussehen:

```
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
```

Zu beachten ist, dass das obige Skript davon ausgeht, dass bereits eine gültige cdk8s-Installation vorhanden ist. Nach dem Ablegen dieses Skripts in eine CI-Pipeline, die bei jeder Codeänderung ausgelöst wird, bleibt nur noch die korrekte Konfiguration von Flux. Wir werden dabei den folgenden **fluxctl**-Befehl verwenden:

```
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` funktioniert sehr ähnlich wie andere CLI-Tools im Kubernetes-Ökosystem. Es erzeugt K8s-Manifeste, die man direkt in `kubectl apply -f -` pipen kann. Diese sorgen dann dafür, dass der Flux-Controller mit den gewünschten Parametern eingerichtet wird.

Nach der korrekten Konfiguration sieht der komplette Flow wie folgt aus:

1. einige Entwickler nehmen Änderungen am cdk8s Code vor, z.B. im master Branch
2. die CI-Pipeline wird durch den neuen Commit ausgelöst
3. das Bash-Skript wird im Kontext des master Branches ausgeführt
4. die K8s-*Manifeste* werden erstellt
5. die Ausgabe wird an den manifest Branch weitergeleitet
6. Flux synchronisiert seine Kopie des manifest Branches
7. Flux erkennt Änderungen und wendet sie auf dem Cluster an

✔ Der (noch nicht ganz) elegante Weg

Wer sich mit Flux auskennt, den höre ich meckern: „Max, nimm doch einfach Flux-Generatoren“, und genau darauf basiert meine zweite Lösung. Anfangs dachte ich, dieser Ansatz sei besser als der erste, weil wir kein CI-System brauchen. Doch während der Implementierung bin ich auf ein großes Problem gestoßen. Aber zunächst einmal: Was ist ein Flux-Generator?

Generatoren ermöglichen es, neue Kubernetes-Manifeste zu erstellen oder bestehende zu modifizieren, bevor man sie auf dem Cluster anwendet. Anstelle der Manifeste legt man also eine einfache `.flux.yaml`-Datei zusammen mit dem cdk8s-Code in ein Repository, die so aussehen könnte:

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

Flux kümmert sich um die Ausführung der Befehle, erfasst die Ausgabe (es sollten gültige K8s-Manifeste herauskommen) und wendet sie auf dem Cluster an. Es gilt zu beachten, dass jeder Befehl, der eine Ausgabe erzeugt und kein K8s-Manifest enthält, stumm geschaltet werden muss. Andernfalls stößt man auf Probleme.
Um die Generatoren-Funktion zu aktivieren, müssen beim Einrichten von Flux die Parameter `--manifest-generation=true` übergeben werden.

```
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 -
```

Danach setzt man einfach obige `.flux.yaml`-Datei neben den cdk8s-Code, lädt alles in ein Repo und schon sollte es losgehen oder? Natürlich nicht, denn jetzt kommt die große Enttäuschung. Die Ausführung der Generatoren geschieht innerhalb des fluxd-Containers, dem Controller von Flux. Das bedeutet im Wesentlichen, dass der gesamte node/yarn- und Python/pip-stack darin installiert werden muss, um cdk8s ausführen zu können. Für mich ist es inakzeptabel, einen Container, der die volle Kontrolle über mindestens einen K8s-Namespace haben muss, so weit aufzublasen. Dennoch habe ich geprüft, ob dieser Ansatz tatsächlich realisierbar ist und folgend ist die funktionierende `.flux.yaml` zu finden:

```
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/*
```

Laut der Dokumentation von Flux planen die Entwickler, die Ausführung von Generatoren in einem separaten Container zu ermöglichen. Im Moment ist das aber noch nicht möglich.

✚ Fazit

Da haben wir sie also – zwei Optionen zum Einrichten eines GitOps-Prozesses mit cdk8s und Flux. Für ein produktives Setup würde ich den zusätzlichen Aufwand der Einrichtung/Verwaltung eines CI-Systems (, das man in den meisten Fällen ohnehin schon hat) in Kauf nehmen und die erste Option für den Moment bevorzugen. Sobald Flux die Ausführung von Generatoren in einem separaten Container unterstützt, wird die zweite Option meiner Meinung nach praktikabler.

Alles in allem war ich überrascht, wie gut beide Tools zusammenarbeiten. Sobald cdk8s ausgereifter ist, werde ich die beiden auf jeden Fall bei der Einrichtung meiner nächsten K8s-Delivery-Pipeline in Betracht ziehen.