ToiletPaper #85

cdk8s mittels Argo CD integrieren

Autor: Maximilian Brenner / DevOps Engineer / Standort Leipzig

Nachdem wir uns im ToiletPaper #88 damit beschäftigt hatten, wie man cdk8s mit Hilfe von Flux integriert, kam die Frage nach einer Integration mittels Argo CD auf. Los geht's!

✗ Argo CD

Aber zunächst sollte erst einmal geklärt werden, was Argo CD überhaupt ist. Ähnlich wie Flux ist es ein Tool zum Deployment von Kubernetes-Manifesten und folgt dabei dem GitOps-Paradigma. Dieses besagt, dass der gewünschte Systemzustand in einem oder mehreren Git-Repositories verwaltet werden sollte. Tools wie Argo CD sorgen dafür, dass System (in diesem Fall Kubernetes) und Zustand synchron bleiben. In seiner einfachsten Form stellt man sich ein Repository vor, das ein einziges Kubernetes-Manifest enthält und Deployment und Service definiert. Argo CD stellt sicher, dass diese im Cluster deployed werden und setzt diesen Vorgang bei jeder neuen Änderung fort.
Zur Steuerung von Argo CD wird es mit einem CLI-Tool und einer netten Web-Benutzeroberfläche geliefert, die es ermöglicht, den Status der Deployments in Echtzeit zu überprüfen. Beide ermöglichen die Erstellung neuer Anwendungen oder die Errichtung neuer Cluster-Verbindungen. Durch letztere kann die Multi-Cluster-Funktionalität zur Orchestrierung von Deployments über mehrere Cluster hinweg mit einer einzigen Instanz von Argo CD genutzt werden. Das Setup einer solchen Instanz wäre nun der nächste Schritt.

✓ Vorbereitung

Zuerst benötigt man eine Argo CD-Instanz. Gern kann dabei auf mein Snippet zurückgegriffen werden, um eine solche Instanz innerhalb von 5 Minuten einzurichten. Andernfalls kann man sich natürlich auch mit dem offiziellen Getting Started Guide von Argo CD behelfen.
Wer sich mit cdk8s noch nicht auskennt, sollte sich vorher unbedingt in meinem Blog einen ersten oder etwas detaillierteren Überblick verschaffen. Und dann kann's losgehen!
Zunächst brauchen wir ein wenig cdk8s-Code, der später deployed wird. Wir werden das argocd-example-apps-Repository als Ausgangspunkt nutzen. Es enthält die gleiche Konfiguration in verschiedenen Formaten, wie z. B. einfache K8s-Manifeste, ein Helm-Chart oder kustomize-Dateien, die alle eine einfache *Guestbook*-Anwendung bestehend aus einem Deployment und einem Service bereitstellen. Unseren Code legen wir in einen neuen Ordner namens ~~`cdk8s-guestbook`~~.

Wie bei jeder cdk8s-Anwendung beginnen wir mit der Ausführung von ~~`cdk8s init python-app`~~. Ich werde dafür Python verwenden, man kann aber auch mit TypeScript arbeiten. Danach definieren wir die Deployment- und Service-Objekte in der *main.py*.

```
...
  label = {"app": "guestbook-ui"}

  k8s.Service(self, 'service',
              spec=k8s.ServiceSpec(
                type='LoadBalancer',
                ports=[k8s.ServicePort(port=80,
target_port=k8s.IntOrString.from_number(80))],
                selector=label))

  k8s.Deployment(self, 'deployment',
                 spec=k8s.DeploymentSpec(
                   replicas=1,
                   selector=k8s.LabelSelector(match_labels=label),
                   template=k8s.PodTemplateSpec(
                     metadata=k8s.ObjectMeta(labels=label),
                     spec=k8s.PodSpec(containers=[
                       k8s.Container(
                         name='guestbook-ui',
                         image='gcr.io/heptio-images/ks-guestbook-demo:0.2',
                         ports=[k8s.ContainerPort(container_port=80)])]))))
...
```

Wenn das erledigt ist, bringen wir den Code in ein Repo, das vorzugsweise öffentlich zugänglich ist. Andernfalls müssen wir Argo CD später die Zugangsdaten für das Repo übergeben. Danach können wir mit der Integration fortfahren.

➔ Integration

Derzeit wird cdk8s von Argo CD nicht *out-of-the-box* unterstützt. Um es nutzen zu können, müssen wir cdk8s als ein benutzerdefiniertes Konfigurationsverwaltungs-Plugin registrieren. Dies funktioniert einfach durch Erstellung/Aktualisierung der *argocd-cm* Kubernetes ConfigMap z. B. wie folgt:

```
# config.yml
data:
  configManagementPlugins: |
    - name: cdk8s # the name of the plugin that we'll later use to reference it
      init: # some optional preprocessing commands
        command: ["bash"]
        args: ["-c", "pipenv install && cdk8s import -l python && cdk8s synth"] # making sure
everything is installed and generating the K8s manifest(s)
      generate: # the output of this command will be deployed onto the target cluster
        command: ["bash"]
        args: ["-c", "cat dist/*"] # printing the generated Kubernetes manifests
kind: ConfigMap
apiVersion: v1
metadata:
  annotations:
  labels:
    app.kubernetes.io/name: argocd-cm
    app.kubernetes.io/part-of: argocd
  name: argocd-cm
  namespace: argocd
  selfLink: /api/v1/namespaces/argocd/configmaps/argocd-cm
```

Mit ~~`kubectl apply -f config.yml`~~ angewendet, ist unser Plugin nun einsatzbereit.

Also lasst es uns ausprobieren. Ich verwende dafür das Argo CD CLI-Tool, aber das Erstellen der Anwendung mit der Web-Benutzeroberfläche wird auch funktionieren.
Nach der Ausgabe dieses Befehls …

```
$ argocd app create guestbook \ # creating an application called guestbook
      --repo https://github.com/brennerm/argocd-example-apps.git \ # the URL of our repo
      --path cdk8s-guestbook \ # the path to the folder containing our config
      --dest-server https://kubernetes.default.svc \ # the cluster we want to deploy to
      --dest-namespace default \ # the namespace we want to deploy to
      --sync-policy automated \ # enabling automatic sync of changes in the repo
      --config-management-plugin cdk8s # make sure to use our cdk8s plugin
```

… erhalten wir folgende Fehlermeldung:

> *FATA[0006] rpc error: code = InvalidArgument desc = application spec is invalid: InvalidSpecError: Unable to generate manifests in cdk8s-guestbook: rpc error: code = Unknown desc = 'bash -c pipenv install && cdk8s import -l python && cdk8s synth' failed exit status 127: bash: pipenv: command not found*

Ein gewisses Hintergrundwissen über Argo CD macht diese Meldung einigermaßen vorhersehbar. Jedes Konfigurationsverwaltungs-Plugin wird in einer Komponente namens *argocd-repo-server* ausgeführt. Damit unser benutzerdefiniertes Plugin funktioniert, müssen wir auch sicherstellen, dass die von uns verwendeten Tools in dieser Umgebung verfügbar sind. In unserem Fall sind das *pipenv* und *cdk8s*. Lösungen können sein:

- Verwendung von Volume-Mounts, die die erforderlichen Binärdateien enthalten
- Bereitstellung eines benutzerdefinierten Images für den *argocd-repo-server*

Wer meinen letzten Beitrag gelesen hat, weiß, dass Flux genau dasselbe Problem hat und dass ich von diesen Optionen ziemlich enttäuscht bin. Aber es nützt ja nichts … Da mir die Lösung mit dem benutzerdefinierten Bild einfacher erschien, probiere ich es damit. Folgend sieht man meine Dockerfile mit den hinzufügten fehlenden Binärdateien:

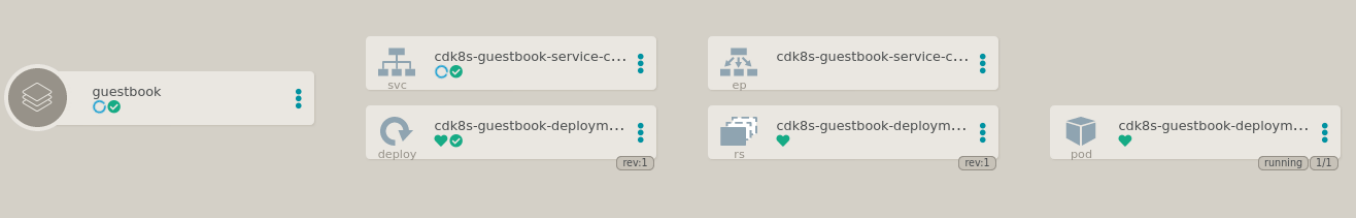
```
FROM argoproj/argocd:latest

USER root

RUN apt-get update && \
    apt-get install -y \
        curl \
        python3-pip && \
    apt-get clean && \
    pip3 install pipenv

RUN curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | apt-key add -
RUN echo "deb https://dl.yarnpkg.com/debian/ stable main" | tee /etc/apt/sources.list.d/yarn.list
RUN apt-get update && apt-get install -y yarn
RUN yarn global add npm cdk8s-cli

USER argocd
```

Nachdem diese Dockerdatei erstellt und das resultierende Bild in eine Docker-Registry eurer Wahl verschoben wurde (Profi-Tipp: bei kind gibt es ein wirklich tolles Lade-Feature), müssen wir den *argocd-repo-server* Kubernetes Deployment updaten, um das neue Bild zu verwenden, z. B. mit `kubectl edit -n argocd deployments.apps argocd-repo-server`.

Wenn wir sichergestellt haben, dass der *argocd-repo-server*-Pod mit dem neuen Bild erstellt wurde, können wir noch einmal mit der Einrichtung der Anwendung beginnen. Diesmal sollte alles funktionieren und am Ende wird die *Guestbook-Pod* gestartet:

```
$ kubectl get pods
NAME                                                READY   STATUS              RESTARTS   AGE
cdk8s-guestbook-deployment-967cec91-65b878495d-jcczj   0/1     ContainerCreating   0          16s
```

Um sicherzustellen, dass Argo CD die Änderungen korrekt synchronisiert, setzen wir den *replicaCount* in der *main.py* auf 2, laden diese Änderung hoch und siehe da:



Live-Update der Änderung des *replicaCount* in der Web-UI von Argo CD

+ Fazit

Die wichtigsten Schritte einer voll funktionsfähigen Integration von cdk8s mittels Argo CD sind also:

1. Registrierung des cdk8s Konfigurationsmanagement-Plugins
2. Bereitstellung der notwendigen Tools auf dem *argocd-repo-server*
3. Verwendung des cdk8s-Plugins bei der Erstellung der Argo CD-Anwendung

So wirklich glücklich macht mich die benutzerdefinierte Anpassung interner Dienste nicht, damit alles funktioniert, aber derzeit gibt es meines Wissens keinen anderen Weg. Sollte jemand andere Infos oder Verbesserungsvorschläge haben oder einen besseren Weg kennen, gebt mir gern Bescheid.