ToiletPaper #87

cdk8s als Zukunft für Kubernetes Applikations-Deployments?

Autor: Maximilian Brenner / DevOps Engineer / Standort Leipzig

✕ Einführung

cdk8s (wahrscheinlich als Cloud-Development-Kit für Kubernetes gedacht) ist ein neues Framework, das von AWS Labs (der Open-Source-Organisation von AWS) veröffentlicht wurde und in TypeScript geschrieben ist. Es ermöglicht die Definition von Kubernetes-Manifesten unter Verwendung moderner objektorientierter Programmiersprachen. Auf diese Weise kann man eine sehr flexible Lösung entwickeln, die komplexe Operationen zur Vorbereitung des Deployments ermöglicht.

Um dies zu erreichen, stellt cdk8s so genannte **constructs** zur Verfügung – Abstraktionen von Kubernetes-Ressourcen (Deployment, Service, Ingress, ...). Eine logische Sammlung davon wird als **Chart** bezeichnet (ähnlich einem Helm Chart). Schließlich wird eine **App** durch ein oder mehrere Charts definiert. Später wird es noch ein Beispiel geben, das diese Einzelteile und ihre Beziehung im Detail darstellt.
Jetzt könnte man sich fragen, wie man die Programmiersprache seiner Wahl verwenden soll, wenn cdk8s selbst in TypeScript geschrieben ist. Die Antwort heißt jsii. jsii von AWS ist ein Tool zur Erzeugung von Bindings für mehrere Programmiersprachen (derzeit Python, Java und C#), die es ermöglichen, mit Type-/JavaScript-Klassen zu interagieren. Es wird bereits vom AWS CDK verwendet (cdk8s, aber für CloudFormation-Dateien) und wird wahrscheinlich in Zukunft weitere Sprachen unterstützen.

✔ Nutzung

Die Installation von cdk8s erfordert, dass man den Standard-Node.js, yarn/npm-Stack, auf seinem Rechner hat. Um das Bootstrapping und die Generierung von **constructs** für eure spezielle Kubernetes-Version zu unterstützen, wird das Kit mit einem CLI-Tool geliefert. Um eine detaillierte Einführung zu erhalten, einfach den Abschnitt Getting Started lesen.
Nach dieser, ist man bereit, Code zu schreiben. Unten findet ihr ein Snippet, das eine einzelne cdk8s-Anwendung definiert, die aus einem einzigen Chart besteht, das einen Service und ein Deployment enthält.

```
#!/usr/bin/env python
from constructs import Construct # importing base class for type hinting
from cdk8s import App, Chart

from imports import k8s # importing resource bindings for your particular Kubernetes version, previously
generated by "cdk8s import"

class MyChart(Chart):
  def __init__(self,
               scope: Construct, # our app instance
               ns: str, **kwargs): # careful! this is not the K8s namespace but just a prefix for our
resources
    super().__init__(scope, ns, **kwargs)

    # defining some common variables
    label = {"app": "hello-kubernetes"}
    container_port = 8080

    # defining a deployment with one container and two replicas
    k8s.Deployment(self, 'deployment',
                   spec=k8s.DeploymentSpec(
                     replicas=2,
                     selector=k8s.LabelSelector(match_labels=label),
                     template=k8s.PodTemplateSpec(
                       metadata=k8s.ObjectMeta(labels=label),
                       spec=k8s.PodSpec(containers=[
                         k8s.Container(
                           name='hello-kubernetes',
                           image='paulbouwer/hello-kubernetes:1.7',
                           ports=[k8s.ContainerPort(container_port=container_port)])]))))

    # defining a service for pods created by the deployment above
    k8s.Service(self, 'service',
                spec=k8s.ServiceSpec(
                  type='LoadBalancer',
                  ports=[k8s.ServicePort(port=80,
target port=k8s.IntOrString.from number(container port))],
                  selector=label))


app = App() # creating an App instance
MyChart(app, "hello") # installing our chart in the app under a certain namespace

app.synth() # this method call takes care of generating the K8s manifests
```

Die Ausführung des obigen Codes oder die Ausführung von `cdk8s synth` führt zu folgendem Manifest:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deployment-c51e9e6b
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hello-kubernetes
  template:
    metadata:
      labels:
        app: hello-kubernetes
    spec:
      containers:
        - image: paulbouwer/hello-kubernetes:1.7
          name: hello-kubernetes
          ports:
            - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: hello-service-9878228b
spec:
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: hello-kubernetes
  type: LoadBalancer
```

Wie man in diesem einfachen Beispiel sehen kann, kümmert sich cdk8s um die Erstellung einer Eins-zu-eins-Übersetzung der Objekte in ein Kubernetes-Manifest. Es gilt aber zu beachten, dass die Strukturen von cdk8s sehr explizit sind. Sie bieten keinerlei Defaults (z. B. für die Anzahl der Pod-Instanzen) oder erkennen keinerlei Beziehungen zwischen Objekten (z. B. das Erkennen des Containerports und dessen Verwendung im Service). Hier kommt ihr ins Spiel und entwickelt Charts, die eure gewünschte Logik umsetzen. Einerseits könnte dies ein generisches Chart sein, das ein Deployment- und ein Service-Manifest für ein beliebiges Container-Image erstellt. Andererseits könnte es ein Chart sein, das eine ConfigMap erstellt, die ihre Werte dynamisch aus einem Key-Value-Store holt. Hier könnt ihr erreichen, was auch immer die von euch gewählte Programmiersprache so hergibt.

➔ Im Vergleich zu kustomize und Helm

An dieser Stelle denkt man vielleicht, warum soll ich nicht einfach weiter Tools wie kustomize oder Helm verwenden und in den meisten Fällen hat man damit wahrscheinlich recht.

Je nach eurem Background sind kustomize und Helm vielleicht leichter zu erlernen. Sie erfordern nicht, dass man tatsächlichen Code schreiben muss, sondern verwenden eine Templating-Sprache. Mit diesen kann man sich an einfachen Operationen wie Variablenersetzung, verzweigte Blöcke oder For-Loops bedienen. Aber sobald man komplexere Funktionen benötigt oder mit externen Diensten kommunizieren muss, wird man an die Grenzen dieser Tools stoßen.

Meiner Meinung nach ist cdk8s einer der nächsten offensichtlichen Schritte in der DevOps-Bewegung, der die Ops näher an die Entwicklung heranführt und es den Entwicklern ermöglicht, Ops-Aufgaben zu übernehmen. Da die Infrastruktur immer dynamischer und komplexer wird, steigt der Bedarf an leistungsfähigere Möglichkeiten zur Definierung von Applikations-Deployments für Systeme, wie z.B. Kubernetes.

✚ Persönliche Einschätzung

Würde ich Stand heute cdk8s in einem produktiven System einsetzen? Nein, da es noch keine stabile Version gibt und es derzeit sehr aktiv weiterentwickelt wird, so dass es unvermeidlich ist, das Setup von Zeit zu Zeit zu brechen. Stattdessen werde ich den Fortschritt weiter im Auge behalten und es für einige meiner Nebenprojekte verwenden. Sobald cdk8s etwas ausgereifter ist (wahrscheinlich als erste Hauptversion), werde ich definitiv noch mal darüber nachdenken, es für neue Projekte einzusetzen.