ToiletPaper #87

cdk8s – the future of Kubernetes application deployments?

Author: Maximilian Brenner / DevOps Engineer / Office Leipzig

✕ Intro

cdk8s (probably cloud development kit for Kubernetes) is a new framework released by AWS Labs (AWS's open source organization) that is written in TypeScript. It allows you to define Kubernetes manifests using modern object-oriented programming languages. This enables you to develop a very flexible solution that is capable of complex operations for establishing your application deployment process to Kubernetes.
To achieve this, cdk8s provides so called **constructs**, which are abstractions of Kubernetes resources (Deployment, Service, Ingress, …). A logical collection of these is called a **chart** (similar to a Helm chart). Finally, an **app** is defined by one or more charts. Later, you will find an example that will showcase these units and their relation in more detail.
Now you may wonder how you can use your programming language of choice if cdk8s itself is written in TypeScript. The answer is called jsii. AWS' jsii is a tool to generate bindings for several programming languages (currently Python, Java and C#) that allow you to interact with Type-/JavaScript classes. It is already in use by the AWS CDK (cdk8s but for CloudFormation files) and will probably get support for more languages in the future.

✔ Usage

Installing cdk8s requires you to have the standard Node.js, yarn/npm stack available on your machine. To assist with bootstrapping and generating constructs for your particular Kubernetes version, the kit comes with a CLI tool. Follow the Getting Started section to get a detailed introduction.
After going through the introduction, we are ready to write our code. Below you find a snippet that defines a single cdk8s app consisting of a single chart containing one service and one deployment.

```
#!/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
```

Executing the above code or running `cdk8s synth` results in the following manifest.

```yaml
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
```

As you can see in this simple example, cdk8s takes care of creating a one to one translation of your objects into a Kubernetes manifest. Take into consideration that cdk8s' structs are very explicit. They do not provide any kind of defaults (e.g. for the replica count) or detect any relations between objects (e.g. detecting the container port and using it for the service). This is where you come into play and develop charts that implement your desired logic. On the one hand, this could be a generic chart that creates a Deployment and a Service manifest for an arbitrary container image. On the other hand, it could be a chart creating a ConfigMap that dynamically pulls its values out of your key value store. You can achieve whatever your chosen programming language is capable of.

➔ Comparison with kustomize and Helm

After reading to this point, you may think why you should not just keep using tools like kustomize or Helm and in most cases you are probably right in doing so.

Depending on your background, kustomize and Helm may be easier to learn. They do not require you to write actual code but use a templating language. This allows you to do simple operations, like variable replacement, conditional blocks, or for-loops. But as soon as you require more complex functionality or need to communicate with external services, you are going to run into the limits of these tools.
In my opinion, cdk8s is one of the next obvious steps in the DevOps movement, bringing Ops people closer to the development and enabling developers to take over Ops tasks. As infrastructure gets more and more dynamic and complex, the need for powerful application deployment orchestration rises.

✚ Personal Opinion

Would I use cdk8s in production today? No, there is no stable version yet and the development is very active, making it unavoidable to break your setup from time to time. Instead I am going to keep an eye on its progress and use it for some of my side projects. After cdk8s reaches a certain level of maturity (probably the first major version), I will definitely consider using it for new projects.