Spring into Kubernetes

Paul Czarkowski

pczarkowski@pivotal.io

Twitter: @pczarkowski
Spring into Kubernetes

Paul Czarkowski

pczarkowski@pivotal.io

Twitter: @pczarkowski
"Kubernetes is named after the greek god of spending money on cloud services" - @QuinnyPig
Traditional Ticket Based Human Toil

More Control
Less Efficiency

Build App Artifact
Config Management
Infrastructure As Code

Infrastructure Platform

Hardware Platform

IaaS API
PXE boot?

Build App Container(s)
Deployment Manifest

Container Platform

K8s API

Application Platform

App → to the Platform

CF API

???

Less Control
More Efficiency

?????

Function Platform

??? API
Containers
Containers Isolate and Abstract Resources

- Apps run in a container
- Containers allow multiple apps to run on a single virtual machine (VM)
- Maximizes utilization without OS overhead
“What are the reasons for opting to use container technologies? (Such as Docker, rkt, CoreOS)?”

- Faster/easier deployment: 80%
- Flexibility in deployment: 60%
- Better isolation: 55%
- Architectural reasons (microservices): 50%
- Cost saving: 30%
- Other: 5%

FROM maven:3.6-jdk-11-slim as BUILD
COPY . /src
WORKDIR /src
RUN mvn install -DskipTests

FROM openjdk:11.0.1-jre-slim-stretch
EXPOSE 8080
WORKDIR /app
ARG JAR=hello-0.0.1-SNAPSHOT.jar

COPY --from=BUILD /src/target/$JAR /app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
$ docker build -t paulczar/hello .

$ docker push paulczar/hello

$ docker pull paulczar/hello

$ docker run -d -p 8080:8080 paulczar/hello
<plugin>
   <groupId>com.google.cloud.tools</groupId>
   <artifactId>jib-maven-plugin</artifactId>
   <version>1.6.1</version>
   <configuration>
      <to>
         <image>myimage</image>
      </to>
   </configuration>
</plugin>
Kubernetes
Desired State

Actual State
Unix Philosophy:
Do one thing. Do it well.
$ kubectl
Imperative

$ kubectl run hello \
   --image=paulczar/go-hello-world

$ kubectl scale hello \
   --replicas=3

$ kubectl create service clusterip \n   hello --tcp=80:80
Declarative

$ kubectl apply -f hello-world.yaml
Declarative
Vs
Imperative
apiVersion: v1
kind: Pod
metadata:
  name: hello
spec:
  containers:
  - image: paulczar/go-hello-world
    imagePullPolicy: Always
    name: hello
Resources
• Pods
• Services
• Volumes
one or more containers that share a network and storage
the minimum scalable unit of your application
$ kubectl
$ kubectl create deployment hello \
    --image=paulczar/hello
• **kubectl** run created a *deployment* “deployments.apps/hello”

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>UP-TO-DATE</th>
<th>AVAILABLE</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment.apps/hello</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1m</td>
</tr>
</tbody>
</table>

• The **deployment** created a *replicaset* “replicaset.apps/hello-64f6bf9dd4”

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>READY</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>replicaset.apps/hello-64f6bf9dd4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1m</td>
</tr>
</tbody>
</table>

• Which created a *pod* “pod/hello-64f6bf9dd4-tq5dq”

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>pod/hello-64f6bf9dd4-tq5dq</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>2s</td>
</tr>
</tbody>
</table>
$ kubectl scale --replicas=3 deployment/hello
$ kubectl scale --replicas=3 deployment/hello
deployment.extensions/hello scaled

$ kubectl get all

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>pod/hello-64f6bf9dd4-2bndq</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>15m</td>
</tr>
<tr>
<td>pod/hello-64f6bf9dd4-4kq9l</td>
<td>0/1</td>
<td>ContainerCreating</td>
<td>0</td>
<td>2s</td>
</tr>
<tr>
<td>pod/hello-64f6bf9dd4-8lkcs</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>5s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>UP-TO-DATE</th>
<th>AVAILABLE</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment.apps/hello</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>16m</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>READY</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>replicaset.apps/hello-64f6bf9dd4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>16m</td>
</tr>
</tbody>
</table>
$ kubectl set env deployment/hello \
--env "MESSAGE=Hello Krakow"
$ kubectl get all

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>pod/hello-5c75b546c7-4lwnn</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>1m</td>
</tr>
<tr>
<td>pod/hello-5c75b546c7-bwxxq</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>1m</td>
</tr>
<tr>
<td>pod/hello-5c75b546c7-sl2pg</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>1m</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>UP-TO-DATE</th>
<th>AVAILABLE</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment.apps/hello</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>23m</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>READY</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>replicaset.apps/hello-5c75b546c7</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1m</td>
</tr>
<tr>
<td>replicaset.apps/hello-64f6bf9dd4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>23m</td>
</tr>
</tbody>
</table>
$ kubectl get deployment hello \
-o yaml
$ kubectl port-forward deployment/hello 8080
Forwarding from 127.0.0.1:8080 -> 8080

$ curl localhost:8080
<html><head><title>HELLO I LOVE YOU!!!!</title></head><body>HELLO I LOVE YOU!!!!</body></html>
Service
$ kubectl expose deployment hello --type=LoadBalancer --port 80 --target-port 8080
kubectl expose deployment hello

- creates a service with a ClusterIP that acts as an internal loadbalancer to all pods in the “hello” deployment

   --type=LoadBalancer

- Creates a NodePort
- Configures a LoadBalancer to access the pods via the NodePort

$ kubectl get services

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td>LoadBalancer</td>
<td>10.39.248.123</td>
<td>35.184.17.129</td>
<td>80:30468/TCP</td>
<td>5m</td>
</tr>
</tbody>
</table>

$ curl 35.184.17.129

<html><head><title>HELLO I LOVE YOU!!!!</title></head><body>HELLO I LOVE YOU!!!!!!</body></html>
Service track Pods based on metadata and provides connectivity and service discovery (DNS, Env variables) for them.

**Type**

*ClusterIP* (default) exposes service on a *cluster-internal IP*. 
Service

track Pods based on metadata and provides connectivity and service discovery (DNS, Env variables) for them.

Type

NodePort extends ClusterIP to expose services on each node’s IP via a static port.
Service

track Pods based on metadata and provides connectivity and service discovery (DNS, Env variables) for them.

Type

LoadBalancer extends NodePort to configure a cloud provider’s load balancer using the cloud-controller-manager.
Ingress

a controller that manages an external entity to provide load balancing, SSL termination and name-based virtual hosting to services based on a set of rules.
Volume
Volume

Is [effectively] a Directory, possibly with data in it, available to all containers in a Pod.

Usually Shares lifecycle of a Pod (Created when Pod is created, destroyed when Pod is destroyed).

Persistent Volumes outlive Pods.

Can be mounted from local disk, or from a network storage device such as a EBS volume, iscsi, NFS, etc.
$ kubectl create configmap hello \
    --from-literal='message=Hello S1T'
kubectl create configmap hello --from-file=index.html

- creates a configmap called “hello” containing the contents index.html

$ kubectl get configmap hello -o yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: hello
data:
  index.html: "<html>
  <head>
  <title>Hello to my friends</title>
  </head>
  <body>
  Hello to my friends
  </body>
  </html>"
kubectl create secret generic hello --from-file=index.html

- creates a secret called “hello” containing a base64 hash of contents index.html

$ kubectl get secret hello -o yaml

apiVersion: v1
kind: Secret
metadata:
  name: hello
data:
  index.html:
    PGh0bWw+CjxoZWFkPgoJPHHRpdGxlPkh1bGxvIHRvIG15IGZyaWVuZHM8L3RpdGxlPgo8L2hlYWQ+Cjxib2R5
    PGoJSGVsbG8gdG8gbXkgZnJpZW5kcwo8L2JvZHk+CjwvaHRtbD4KCg==
Provides **key-value pairs** to be injected into a **pod** much like user-data is injected into a Virtual Machine in the cloud.

Allows you to do **last minute configuration** of applications running on Kubernetes such as setting a database host, or a admin password.

**ConfigMaps** store values as **strings**, **Secrets** store them as **byte arrays** (serialized as base64 encoded strings).

**Secrets** are [currently] **not encrypted** by default. This is likely to **change**.

Can be injected as files in a Volume, or as Environment Variables.
Spring Cloud Kubernetes
● Discovery
● Ribbon Discovery
● Dynamic Config
● Profile Autoconfiguration
● Istio Awareness
● Pod Health Indicators
Service Discovery

```java
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-kubernetes</artifactId>
</dependency>

---

@SpringBootApplication
@EnableDiscoveryClient
public class Application {
  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }
}

---

@Autowired
private DiscoveryClient discoveryClient;
```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-kubernetes-ribbon</artifactId>
</dependency>
```java
@SpringBootApplication
@EnableDiscoveryClient
@EnableWebFlux
public class EdgeServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(EdgeServiceApplication.class, args);
    }

    @Bean
    @LoadBalanced
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```
Native Service Discovery
<dependencies>
    <!-- needed for spring kubernetes config reloads -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <!-- spring cloud kubernetes config server -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-kubernetes-config</artifactId>
    </dependency>
</dependencies>
spring-cloud-kubernetes-config

spring:
  application:
    name: cloud-k8s-app
cloud:
  kubernetes:
    config:
      name: default-name
      namespace: default-namespace
      sources:
        - name: c1
        - namespace: n2
        - namespace: n3
        - name: c3
oh and automatic prometheus metrics

<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>