Java EE Application using WildFly and MySQL

The following document describes the deployment of a Java EE application using WildFly application server and MySQL database server on Kubernetes. The sample application source code is at: https://github.com/javaee-samples/javaee7-simple-sample.

Prerequisites

https://github.com/kubernetes/kubernetes/blob/master/docs/user-guide/prereqs.md

Start MySQL Pod

In Kubernetes a *Pod* is the smallest deployable unit that can be created, scheduled, and managed. It's a collocated group of containers that share an IP and storage volume.

Here is the config for MySQL pod: mysql-pod.yaml

<!-- BEGIN MUNGE: mysql-pod.yaml -->
<!-- END MUNGE: EXAMPLE -->

Create the MySQL pod:

```
kubectl create -f examples/javaee/mysql-pod.yaml
```

Check status of the pod:

```
kubectl get -w po
NAME        READY     STATUS    RESTARTS   AGE
mysql-pod   0/1       Pending   0          4s
NAME        READY     STATUS    RESTARTS   AGE
mysql-pod   0/1       Running   0          44s
mysql-pod   1/1       Running   0         44s
```

Wait for the status to `1/1` and `Running`.

Start MySQL Service

We are creating a *Service* to expose the TCP port of the MySQL server. A Service distributes traffic across a set of Pods. The order of Service and the targeted Pods does not matter. However Service needs to be started before any other Pods consuming the Service are started.

In this application, we will use a Kubernetes Service to provide a discoverable endpoints for the MySQL endpoint in the cluster. MySQL service target pods with the labels `name: mysql-pod` and `context: docker-k8s-lab`.

Here is definition of the MySQL service: mysql-service.yaml

<!-- BEGIN MUNGE: mysql-service.yaml -->

<!-- END MUNGE: EXAMPLE -->

Create this service:

```
kubectl create -f examples/javaee/mysql-service.yaml
```

Get status of the service:

```
kubectl get -w svc
NAME            LABELS
 SELECTOR                                  IP(S)          P
ORT(S)
kubernetes      component=apiserver,provider=kubernetes
 <none>                                    10.247.0.1     4
43/TCP
mysql-service   context=docker-k8s-lab,name=mysql-pod
 context=docker-k8s-lab,name=mysql-pod   10.247.63.43   3
306/TCP
```

If multiple services are running, then it can be narrowed by specifying labels:

```
kubectl get -w po -l context=docker-k8s-lab,name=mysql-po
d
NAME        READY     STATUS    RESTARTS   AGE
mysql-pod   1/1       Running   0          4m
```

This is also the selector label used by service to target pods.

When a Service is run on a node, the kubelet adds a set of environment variables for each active Service. It supports both Docker links compatible variables and simpler `{SVCNAME}_SERVICE_HOST` and `{SVCNAME}_SERVICE_PORT` variables, where the Service name is upper-cased and dashes are converted to underscores.

Our service name is `mysql-service'' and so` MYSQL_SERVICE_SERVICE_HOST'' and ``MYSQL_SERVICE_SERVICE_PORT'' variables are available to other pods. This host and port variables are then used to create the JDBC resource in WildFly.

Start WildFly Replication Controller

WildFly is a lightweight Java EE 7 compliant application server. It is wrapped in a Replication Controller and used as the Java EE runtime.

In Kubernetes a *Replication Controller* is responsible for replicating sets of identical pods. Like a *Service* it has a selector query which identifies the members of it's set. Unlike a service it also has a desired number of replicas, and it will create or delete pods to ensure that the number of pods matches up with it's desired state.

Here is definition of the MySQL service: wildfly-rc.yaml.

<!-- BEGIN MUNGE: wildfly-rc.yaml -->

<!-- END MUNGE: EXAMPLE -->

Create this controller:

```
kubectl create -f examples/javaee/wildfly-rc.yaml
```

Check status of the pod inside replication controller:

```
kubectl get po
NAME                READY     STATUS    RESTARTS   AGE
mysql-pod           1/1       Running   0          1h
wildfly-rc-w2kk5    1/1       Running   0          6m
```

Access the application

Get IP address of the pod:

```
kubectl get -o template po wildfly-rc-w2kk5 --template={{
.status.podIP}}
10.246.1.23
```

Log in to node and access the application:

```
vagrant ssh node-1
Last login: Thu Jul 16 00:24:36 2015 from 10.0.2.2
```

```
[vagrant@kubernetes-node-1 ~]$ curl http://10.246.1.23:8080/employees/resources/employees/
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><collection><employee><id>1</id><name>Penny</name></employee><employee><id>2</id><name>Sheldon</name></employee><employee><id>3</id><name>Amy</name></employee><employee><id>4</id><name>Leonard</name></employee><employee><id>5</id><name>Bernadette</name></employee><employee><id>6</id><name>Raj</name></employee><employee><id>7</id><name>Howard</name></employee><employee><id>8</id><name>Priya</name></employee></collection>
```

Delete resources

All resources created in this application can be deleted:

```
kubectl delete -f examples/javaee/mysql-pod.yaml
kubectl delete -f examples/javaee/mysql-service.yaml
kubectl delete -f examples/javaee/wildfly-rc.yaml
```

<!-- BEGIN MUNGE: GENERATED_ANALYTICS -->

<!-- END MUNGE: GENERATED_ANALYTICS -->