Deploying a Stateless Application in the Kubernetes Cluster

26m 57s left

100% **Open Cloud Environment**
Setup completed
Average setup time: 2m 31s

100% **Open Code Environment**
Setup completed

Credentials

Account ID
781848569 Copy

Username
student Copy

Password
Ca1_bcnVp Copy

Region
US West 2 Copy

PEM Download PPK Download

Bastion Public Ip
35.89.147 Copy

Cluster SSH
ssh ubunt

Introduction

With a Kubernetes cluster up and running, you can deploy an application in the cluster. You will use a Docker image of a game for the application. The application runs in a client web browser and doesn't store any state across sessions. The deployment you will perform in this lab step is effective for such stateless applications.

Instructions

1. SSH to the control plane node with the following command:

Copy code

```
ssh control-plane -oStrictHostKeyChecking=no
```

2. Enter the following command to create a deployment resource file:

Copy code

```
cat <<EOF > deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: game-deployment
spec:
  # tells deployment to run 1 pods matching the template
  replicas: 1
  # create pods using the pod definition in this template
  selector:
    matchLabels:
      app: game
  template:
    metadata:
      # name is automatically generated based on the
deployment.name
      labels:
        app: game
    spec:
```

The deployment declares a single replica and specifies a single Docker container. The container image is hosted on Docker Hub.

3. Create the deployment resource specified in the deployment.yml file:

Copy code

```
kubectl create -f deployment.yml
```

4. Get the deployment resource:

Copy code

```
kubectl get deployment game-deployment
```

The **DESIRED** column shows how many replicas are desired.
The **CURRENT** column shows how many pod replicas are currently running.
Both values are **1**:

```
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
game-deployment   1/1     1            1           9s
```

5. Show more details about the deployment:

Copy code

```
kubectl describe deployment game-deployment
```

In the event of failed deployments, you can use this command to understand the reason the deployment failed. Under the **Conditions** section, only the **Available** condition is present:

```
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
```

It has a **Status** of **True** because the minimum number of replicas for the deployment are available. There would be other conditions if the deployment failed.

Lab Steps

```
kubectl get events | more
```

You can understand the process taken for deploying the application by inspecting the **Message** column:

```
Pulling image "lrakai/tetris:latest"
Successfully pulled image "lrakai/tetris:latest" in 2.455871233s
Created container tetris
Started container tetris
```

7. Get the pod resource created by the deployment:

Copy code

```
kubectl get pods
```

The **READY** column indicates **1** of **1** containers are running:

```
NAME                               READY   STATUS    RESTARTS   AGE
game-deployment-776f4f94c-7p9xc    1/1     Running   0          40s
```

8. Describe the pod that is running the container in the deployment:

Copy code

```
kubectl describe pods
```

You can see a variety of details including **Containers**, and **Events** for the pod. You can append a specific pod name to the end of the command to describe a specific pod. In this case, there is only one pod, so there is no need to specify the name.

9. Describe all the pods with the label `app=game`:

Copy code

```
kubectl get pods -l app=game
```

Recall that the `app: game` label is specified in the template metadata in the deployment yml file. Each pod created by the deployment is assigned this

Copy code

```
cat <<EOF > service.yml
apiVersion: v1
kind: Service
metadata:
  name: game
  labels:
    app: game
spec:
  selector:
    # Use labels to select the pods to route traffic to
    app: game
  ports:
  - protocol: TCP
    port: 80
  # Allocate a port on each node in the cluster
  type: NodePort
EOF
```

A service is required to access the application from outside of the cluster.

11. Create the service:

Copy code

```
kubectl create -f service.yml
```

12. Describe the service and record what **NodePort** was allocated:

Copy code

```
kubectl describe services game
```

```
Name:                     game
Namespace:                default
Labels:                   app=game
Annotations:              <none>
Selector:                 app=game
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.98.113.146
IPs:                      10.98.113.146
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30044/TCP
```


erved for services.

service from your browser:

Copy code

```
ssh worker1 -o StrictHostKeyChecking=no "curl -s ifconfig.me; echo"
```

14. Open a web browser and enter the node's public IP address followed by a *:* and the NodePort you recorded. The full address will resemble the address in the image below:

```
34.218.59.223:30044
```

The network security group has been pre-configured to allow traffic to the ports that the service can map to.

15. Return to your browser terminal and scale the application by changing the deployment `replicas` count to 2:

Copy code

```
sed -i 's/\(replicas: \).*/\12/' deployment.yml
```

16. Apply the new resource:

Copy code

```
kubectl apply -f deployment.yml
```

17. Verify that the number of pods increases to 2:

Copy code

```
kubectl get pods -l app=game
```

It takes a few seconds for the new pod to reach the **Running** state:

```
NAME                               READY   STATUS    RESTARTS   AGE
game-deployment-776f4f94c-7p9xc    1/1     Running   0          4m7s
                                   1/1     Running   0          64s
```

In this lab step, you deployed a stateless web application in the Kubernetes cluster. You used a deployment resource to launch the pods on the worker node and used a service resource to map external ports to the application running in the container.

VALIDATION CHECKS

1 **Checks** Start check

Created Pod named game-deployment that has two replicas

Created a Kubernetes Deployment with a Pod named game-deployment which has two replicas

Kubernetes

Did you like this step?