Pod Deployment Patterns - Sidecar

Lexique

Deployment file

Task Details

1. Launching Lab Environment.
2. SSH into the EC2 instance through the Whiz terminal.
3. Create the YML declarative file for the init container example.
4. Record the provisioning of the resources with kubectl command.
5. Delete the resources.

Points clés

- Illustration de l'utilité des conteneurs d'initialisation pour faciliter la gestion et l'isolation des tâches dans Kubernetes.

Lab Steps

Task 1: Launching Lab Environment

Launch the lab environment by clicking on the button.

Please wait until the lab environment is provisioned. It will take around 2 minutes to provision the lab environment which contains the single node Kubernetes cluster. The unique Username, Public IP address, and the Password are displayed on the right corner of the screen.

Alongside the EC2 instance, the Whizlabs terminal is also being provisioned.

Note: If you face any issues, please go through FAQs and Troubleshooting for Labs.

Task 2: SSH into the EC2 Instance via the Whiz Terminal:
Copy the IP address and the Password for SSH to the EC2 instance configured with the Kubernetes cluster.

Run the following command to SSH into the system:

```
bashCopy code
ssh username@ip_address
```

Replace the unique username and the IP address with the details of your own lab environment.

Type "yes" if prompted for the SSH permission for the first time. Then, insert the password in the prompt as it is.

Now switch the user to root with the command below:

```
bashCopy code
sudo su root
```

This EC2 instance is configured with Kubernetes packages and all prerequisites are fulfilled. This is a single node cluster and we will use the same node as the client node to communicate and control the containers.

Task 3: Create the YAML Declarative File for the Init Container Example.

In this step, we will create the manifest file for a sample nginx container wherein we

will define an init container and mount a shared volume to both the init container and the main application container. Also, we will put the webpage content in the init container which will be eventually mounted with the main nginx container serving the content to the end users.

So, without going inside the nginx container, we will inject the web server data through a shared volume mounted with the init container. This is how init containers can be useful in performing supportive tasks and enhancing isolation for the processes in the main application container.

Create the manifest file using the YAML approach.

1. To observe real-time changes, let's first create a script that will run in the background and collect the output of the `kubectl get pods` command and store the same in a sample text file. With this, we can notice the behind-the-scenes init container processes and get a deeper understanding of the actual workflow.

```
bashCopy code
vim output.sh
```

2. Now paste the following block of code in the script file:

```
bashCopy code
while :
do
    kubectl get pods --watch > pods
done
```

Now, change the permission of this script file and make it executable using the command below:

```
bashCopy code
chmod +x output.sh
```

The next step is to keep running this script in the background so that the watch command will append the behind-the-scenes process to print the pod provisioning in a sequential manner:

```
bashCopy code
./output.sh &
```

Now create a file in which we will paste the configurations of the deployment containing two container descriptions.

```
bashCopy code
vim init.yaml
```

3. The next step is to create a deployment declaration file with the init container function declared. Here we will use the keyword `initcontainer` in the manifest file and configure it.

```
yamlCopy code
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: nginx
  name: nginx-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      run: nginx
  template:
    metadata:
      labels:
        run: nginx
    spec:
      volumes:
      - name: init-volume
        emptyDir: {}
      initContainers:
      - name: busybox
        image: busybox
```

```
        volumeMounts:
        - name: init-volume
          mountPath: /nginx-data
        command: ["/bin/sh"]
        args: ["-c", "echo '<h1>Hello Kubernetes Whizlabs</h1
      containers:
      - image: nginx
        name: nginx
        volumeMounts:
        - name: init-volume
          mountPath: /usr/share/nginx/html
```

As you can see in the YAML file, we have used the busybox container image for the init container and embedded one command which will append the content in the index file of the web server, and the same directory is mounted with the document root of the nginx pod. We have kept only 1 replica for this demo.

4. Now, let’s run the declared manifest file and apply the changes which will run a deployment with two containers: one for the nginx application and the other for the init container.

```
bashCopy code
kubectl apply -f init.yaml
```

Till here, we have deployed the declaration file with the init container and the main nginx application container, and with the script file, we have stored the behind-the-scenes processes considering the init containers.

Task 4: Record the Provisioning of the Resources with kubectl Command

Run the command below to verify the service resource is applied and working properly.

```
bashCopy code
kubectl get pods
```

To get a better idea about the init container, let’s check the output stored inside the pod file using the command below:

```
bashCopy code
cat pods
```

Here, we can see that the intermediate pods with “init” status are created and destroyed as soon as the execution of the mentioned argument is completed. And sequentially, the main application pod for the nginx will start running.

Let’s now check from the browser the working of the nginx deployment. Run the command below and fetch the port number for the exposed service:

```
bashCopy code
kubectl get svc
```

Now, note the port number for the NodePort service and pick the public IP address from the Whizlabs web console and paste the IP:port binding in the browser to see the web content. The port number should be in the range of 31000.

Task 5: How to Delete the Service?

By running the command below, we can delete the launched deployment, which will eventually clean up the application pod and init container declarations.

```
bashCopy code
kubectl delete deploy nginx-deploy
```

Completion and Conclusion

You have successfully created a deployment file with an init container declaration.

You have successfully described the created resources and recorded the output for the whole procedure.

You have verified the output from the web console and deleted the created resource.

End Lab

Exit from the terminal SSH session.

You have successfully completed the lab.

Once you have completed the steps, click on the button from your Whizlabs dashboard.