VASANTDADA PATIL PRATISHTHAN'S COLLEGE OF ENGINEERING AND VISUAL ARTS

ISO 9001:2015 Certified Institute

Department of Information Technology

NBA Accredited Course (Dated 01/07/2024 to 30/06/2027)

EXPERIMENT - 4

Aim: To install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy Your First Kubernetes Application.

Theory:

After successfully creating the Kubernetes cluster, you can now deploy your application to it using the kubectl.

You need to create manifest’s for your project.

- Deployment.yaml : You actual app deploying

- service.yaml : To access your app outside the network in public.

Before moving forward you can install kubectl in your system using package manager:

```
□ ~ □ sudo pacman -S kubectl
```

Some common terms in manifest:

In Kubernetes manifests, you’ll encounter several common terms and concepts. Here’s a brief overview of some key ones:

- kind: Should be Deployment.
- apiVersion: Specifies the API version, e.g., apps/v1.
- metadata: Contains the name, namespace, labels, and annotations of the deployment.
- spec: The specification for the deployment.
- replicas: Number of pod replicas to run.
- selector: Defines how to select pods managed by the deployment.
  - matchLabels: Key-value pairs for selecting pods.
- template: The pod template that describes the pods.
  - metadata: Labels for the pods.
  - spec: Defines the containers and their configurations.
    - containers: List of container specs.
      - name: Name of the container.

- image: Container image to use.
- ports: List of ports to expose.
- env: Environment variables for the container.
- volumeMounts: Specifies where to mount volumes in the container.
-

Service manifest:

- apiVersion: Typically v1.
- kind: Should be Service.
- metadata: Contains the name, namespace, labels, and annotations of the service.
- spec: The specification for the service.
-
- selector: Defines which pods the service will target based on labels.
- ports: List of ports exposed by the service.
  - port: Port that the service will expose.
  - targetPort: Port on the container to forward traffic to.
- type: Defines the service type (e.g., ClusterIP, NodePort, LoadBalancer).

Here’s mine:

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name:  node-deployment
 namespace: thrifty
 labels:
      app: node-api
spec:
 replicas: 2
 selector:
      matchLabels:
         app: node-api
 template:
       metadata:
            labels:
            app: node-api
      spec:
         containers:
         - name: node-api
            image: darkkernel/node-api
            ports:
            -  containerPort:  8080
```

Service.yaml

```
apiVersion: v1
kind: Service
metadata:
 name:  external-svc
 namespace: thrifty
 labels:
      app: external-svc
spec:
 type: LoadBalancer
 ports:
      - port: 80
         targetPort: 8080
         protocol: TCP
 selector:
      app:  node-api
```

1. Now let’s start with deployment, check if the cluster is ready.

```
□ ~ □ kubectl get nodes
NAME
ip-192-168-26-47.ec2.internal          STATUS    ROLES     AGE      VERSION    v1.30.2-eks-
ip-192-168-33-0.ec2.internal           Ready     <none>    5m32s    1552ad0    v1.30.2-eks-
                                       Ready     <none>    5m43s    1552ad0
```

2. Get your project in your system.

```
> git clone https://github.com/Dark-Kernel/node-api.git
Cloning into 'node-api'//.
remote: Enumerating objects: 1513, done.
remote: Counting objects: 100% (337/337), done.
remote: Compressing objects: 100% (194/194), done.
remote: Total 1513 (delta 126), reused 337 (delta 126), pack-reused 1176 (from 1)
Receiving objects: 100% (1513/1513), 2.16 MiB | 5.71 MiB/s, done.
Resolving deltas: 100% (357/357), done.
>
> cd  node-api/
> ls Kubernetes/
□  deployment.yaml  □  service.yaml
>
```

3. Create Namespace if required

```
> kubectl create namespace thrifty
namespace/thrifty created
```

4. Now, create the deployment using the kubectl command.

```
> kubectl apply -f Kubernetes/deployment.yaml
deployment.apps/node-deployment   created
```

5. You can check if it is applied.

```
> kubectl get deployments -n thrifty
```

```
NAME              READY   UP-TO-DATE   AVAILABLE   AG
node-deployment   2/2     2            2           E
>                                                  27s
```

6. Then deploy the service

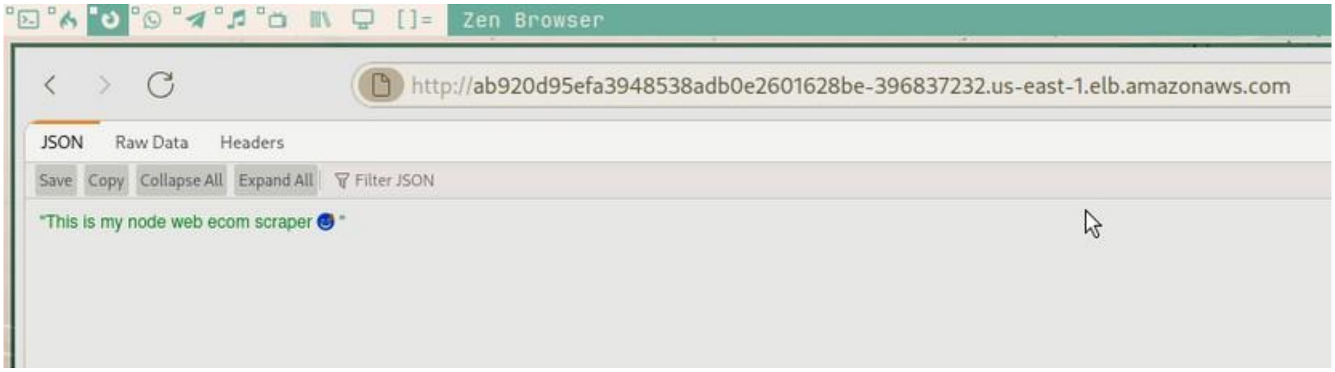
```
> kubectl apply -f Kubernetes/external-svc.yaml
service/external-svc created
```

7. Now, you can check your service if it is done.

```
> kubectl get svc -n thrifty
```

```
NAME           TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)        AGE            10.100.232.57    ab920d95efa3948538adb0e2601628be-396837232.us-
external-svc   LoadBalancer
east-1.elb.amazonaws.com        80:31713/TCP     107s
```

Now using the external IP, which is a subdomain actually you can access your application.
And our application deployment is successful



http://ab920d95efa3948538adb0e2601628be-396837232.us-east-1.elb.amazonaws.com

"This is my node web ecom scraper 😃 "

Conclusion: Thus, we have successfully installed Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deployed our First Kubernetes Application.