

**EXPERIMENT – 4**

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

**Theory:**

 **Installing kubectl and Deploying a Kubernetes Application**

1. **Purpose of kubectl:**
   * kubectl is the command-line interface for interacting with Kubernetes clusters. It facilitates the management of resources, configuration, and deployment of applications.

# Installation Process:

## Dependencies:

* + - Requires a compatible operating system (Linux, macOS, or Windows) and internet access for downloading.

## Downloading and Setting Up:

* + - The tool can be installed from the official Kubernetes release repositories, using package managers or direct downloads. The installation involves making the binary executable

and placing it in the system’s PATH for accessibility.

# Verification:

* + After installation, kubectl can be verified using the version command, ensuring that it was installed correctly and is functional.

**Theory: Deploying a Kubernetes Application**

# Containerization:

* + Before deploying to Kubernetes, applications must be packaged into containers using technologies like Docker. This involves creating a Docker image that contains the application and its dependencies.

# Kubernetes Resources:

* + Applications in Kubernetes are defined through resource configurations written in YAML. Key resources include:
    - **Deployments:** Manage the desired state of application pods, including scaling and updating.
    - **Services:** Abstract network access to a set of pods, defining how to expose applications to users or other services.

# YAML Configuration:

* + Each resource is defined in a structured YAML file, outlining specifications such as the number of replicas, container images, ports, and selectors.

# Application Deployment:

* + The kubectl apply command is used to create or update the specified resources in the Kubernetes cluster based on the YAML definitions. This command communicates with the Kubernetes API server to manage the state of the resources.

# Monitoring and Accessing the Application:

* + After deployment, the state of pods and services can be monitored with kubectl get commands. Users can access the application through defined service types, which may include NodePort or LoadBalancer, depending on how external access is configured.

**Output :**

**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

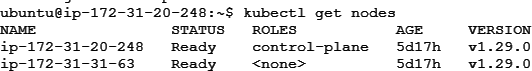
targetPort: 8080 protocol: TCP

selector:

app: node-api

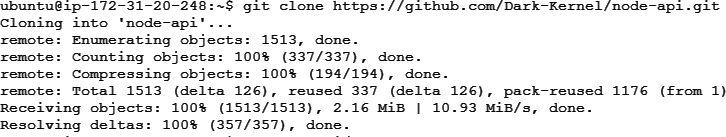
1. **Now let’s start with deployment, check if the cluster is ready.**

~❯ kubectl get nodes



1. **Getyour project in your system.**

❯ git clone <https://github.com/Dark-Kernel/node-api.git>



❯ cd node-api/

❯ ls Kubernetes/



**Create Namespace if required**

❯ kubectl create namespace thrifty



1. **Now, create the deployment using the kubectl command.**

❯ kubectl apply –f Kubernetes/deployment.yaml



1. **You can check if it is applied.**

❯ kubectl get deployments -n thrifty



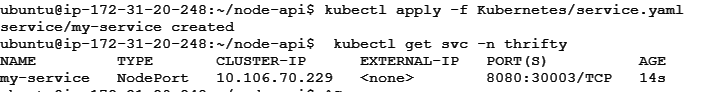
1. **Then deploy the service**

❯ kubectl apply -f Kubernetes/external-svc.yaml

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



❯ kubectl get svc -n thrifty



http://3.6.91.36:30813/

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

**Conclusion:**

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