**Deploying the Voting Application Using Kubernetes**

**Kubernetes**

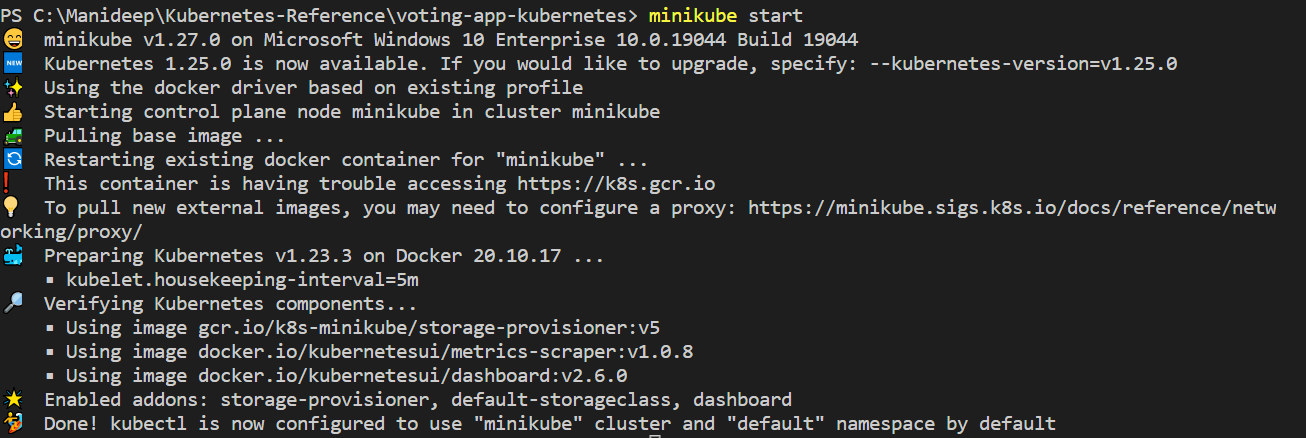
Kubernetes, also known as K8s. It is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

**Minikube**

Minikube is local Kubernetes, focusing on making it easy to learn and develop for Kubernetes.

All you need is a Docker (or similarly compatible) container or a Virtual Machine environment, and Kubernetes is a single command away after Installation of Minikube on the system:

1. minikube start (Docker should be running in the background)



**Deployment**

Deployment in Kubernetes is an upgraded and higher version of Replication controller. They manage the deployment of a replica set which is also an upgraded version of the replication controller.

* Deployment in Kubernetes allows you to describe an application life cycle, such as which images to use for the app, the number of pods, and the way in which they should be updated.
* In Kubernetes, Deployment has the capability to update the replica set and is also capable of rolling back to the previous version. They provide many updated features of match labels and selectors.
* What we do is we describe a desired state in a deployment, and we have a controller called deployment controller which makes it happen. Deployment controller changes the actual state to the desired state.
* Deployments are declarative, which means that they have what to achieve not how to achieve. To achieve this desired state, deployment uses Replica Sets, which further maintains the required set of pods.

**Service**

In Kubernetes, services are a group of pods on the cluster, it is a logical, abstract layer, and here all the pods perform the same function.

Types of service:

1) ExternalName: This service type helps us map the service to a predefined externalName filed.

2) Cluster Ip: This type of service helps us to expose a service that can only be accessible within the cluster.

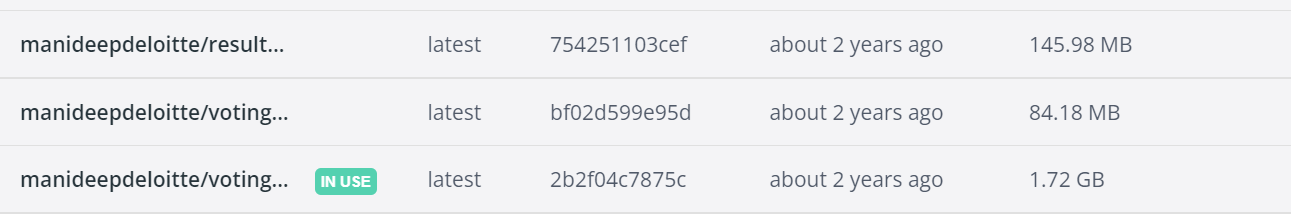
3) Load Balancer: this type of service helps us to expose the service with the help of a cloud provider.

4) NodePort: This type of service helps us to expose the service with the help of a static port present on each node IP.

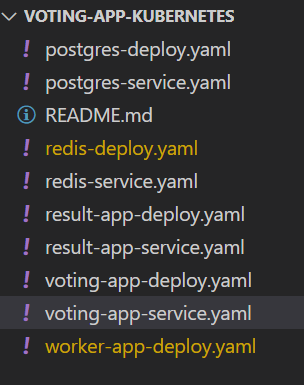
**About Web Application**

We are going to deploy a web application which has two Graphical user Interface nodes, messaging node with Redis, Data Management node with Postgres and worker node.

I have created docker images for Voter Webpage, Result Webpage, and worker node

****

**Project’s Structure**

****

voting-app-deploy.yaml

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: voting-app-deploy

  labels:

    name: voting-app-deploy

    app: demo-voting-app

spec:

  replicas: 1

  selector:

    matchLabels:

      name: voting-app-pod

      app: demo-voting-app

  template:

    metadata:

      name: voting-app-pod

      labels:

        name: voting-app-pod

        app: demo-voting-app

    spec:

      containers:

      - name: voting-app

        image: manideepdeloitte/voting-app

        ports:

        - containerPort: 80

Voting-app-service.yaml

---

apiVersion: v1

kind: Service

metadata:

  name: voting-service

  labels:

    name: voting-service

    app: demo-voting-app

spec:

  type: NodePort

  ports:

  - port: 80

    targetPort: 80

    nodePort: 30004

  selector:

    name: voting-app-pod

    app: demo-voting-app

redis-deploy.yaml

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: redis-deploy

  labels:

    name: redis-deploy

    app: demo-voting-app

spec:

  replicas: 1

  selector:

    matchLabels:

      name: redis-pod

      app: demo-voting-app

  template:

    metadata:

      name: redis-pod

      labels:

        name: redis-pod

        app: demo-voting-app

    spec:

      containers:

        - name: redis

          image: redis

          ports:

            - containerPort: 6379

redis-service.yaml

---

apiVersion: v1

kind: Service

metadata:

  name: redis

  labels:

    name: redis-service

    app: demo-voting-app

spec:

  ports:

  - port: 6379

    targetPort: 6379

  selector:

    name: redis-pod

    app: demo-voting-app

postgres-deploy.yaml

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: postgres-deploy

  labels:

    name: postgres-deploy

    app: demo-voting-app

spec:

  replicas: 1

  selector:

    matchLabels:

      name: postgres-pod

      app: demo-voting-app

  template:

    metadata:

      name: postgres-pod

      labels:

        name: postgres-pod

        app: demo-voting-app

    spec:

      containers:

      - name: postgres

        image: postgres

        ports:

        - containerPort: 5432

        env:

          - name: POSTGRES\_USER

            value: "postgres"

          - name: POSTGRES\_PASSWORD

            value: "postgres"

          - name: POSTGRES\_HOST\_AUTH\_METHOD

            value: trust

postgres-service.yaml

---

apiVersion: v1

kind: Service

metadata:

  name: db

  labels:

    name: postgres-service

    app: demo-voting-app

spec:

  ports:

  - port: 5432

    targetPort: 5432

  selector:

    name: postgres-pod

    app: demo-voting-app

result-app-deploy.yaml

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: result-app-deploy

  labels:

    name: result-app-deploy

    app: demo-voting-app

spec:

  replicas: 1

  selector:

    matchLabels:

      name: result-app-pod

      app: demo-voting-app

  template:

    metadata:

      name: result-app-pod

      labels:

        name: result-app-pod

        app: demo-voting-app

    spec:

      containers:

      - name: result-app

        image: manideepdeloitte/result-app

        ports:

        - containerPort: 80

result-app-service.yaml

---

apiVersion: v1

kind: Service

metadata:

  name: result-service

  labels:

    name: result-service

    app: demo-voting-app

spec:

  type: NodePort

  ports:

  - port: 80

    targetPort: 80

    nodePort: 30005

  selector:

    name: result-app-pod

    app: demo-voting-app

worker-app-deploy.yaml

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: worker-app-deploy

  labels:

    name: worker-app-deploy

    app: demo-voting-app

spec:

  replicas: 1

  selector:

    matchLabels:

      name: worker-app-pod

      app: demo-voting-app

  template:

    metadata:

      name: worker-app-pod

      labels:

        name: worker-app-pod

        app: demo-voting-app

    spec:

      containers:

      - name: worker-app

        image: manideepdeloitte/voting-app-worker-node

Create the Deployment for the Voting Page

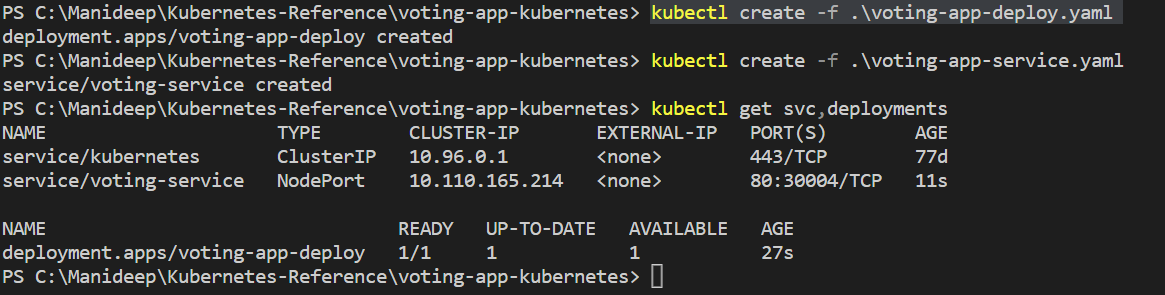
kubectl create -f voting-app-deploy.yaml

Create the service for the same

Kubectl create -f voting-app-service.yaml

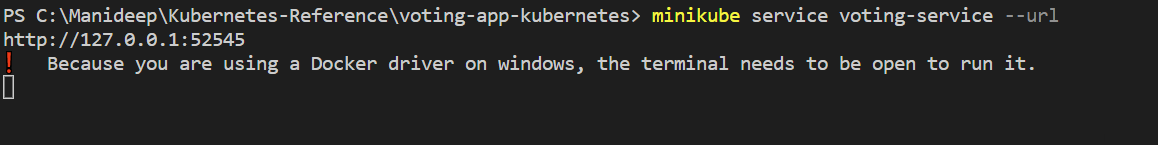
Check whether both Deployment and Service are created or not

Kubectl get service, deployments



Get the IP address for the Voter’s service

minikube service voting-service --url



Open the address in the browser

Graphical user interface, application, website

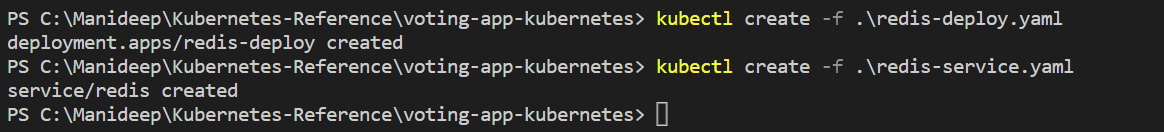
Description automatically generated

Create the Deployment for redis

Kubectl create -f redis-deploy.yaml

Create the Service for redis

Kubectl create -f redis-service.yaml



Create the Deployment for worker

Kubectl create -f worker-app-deploy.yaml

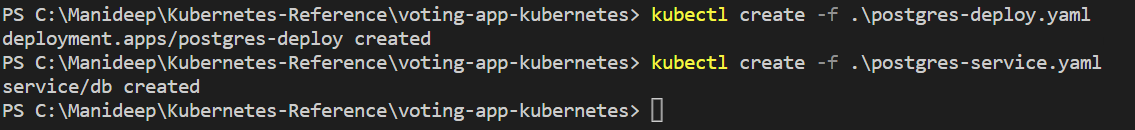


Create the Deployment for postgres

Kubectl create -f postgres-deploy.yaml

Create the Service for postgres

Kubectl create -f postgres-service.yaml

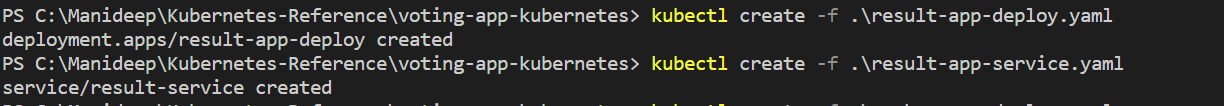


Create the Deployment for Result page

Kubectl create -f result-app-deploy.yaml

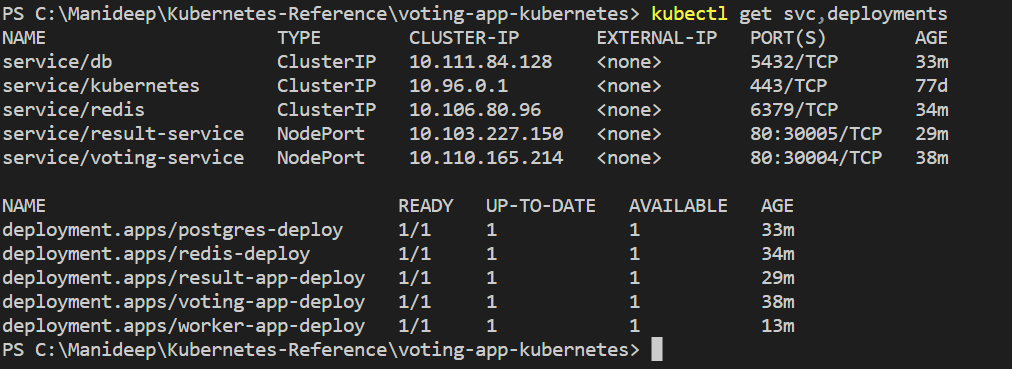
Create the Service for Result page

Kubectl create -f result-app-service.yaml



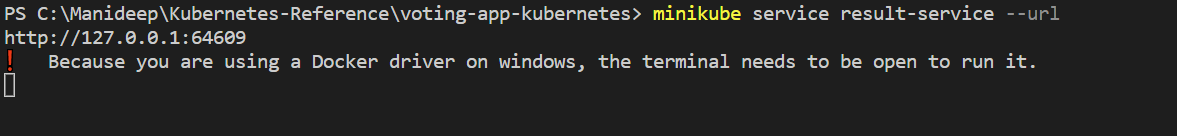
Fetch all the Created Deployments and Services

Kubectl get services, deployments



Get the IP address for the Result’s service

minikube service result-service --url



Open the address in the browser

Graphical user interface, application

Description automatically generated

**Deploying the Web application with Helm Charts**

