# ****Deploying a Complete Node.js Application in Kubernetes****

Pre-requisites:

1. Docker Desktop
2. Kubernetes

Download **Docker Desktop** to your computer. Then install it and make sure you get the output by running the docker version on your terminal.

**Create a Node.js Application**

You can create your Node.js app according to your preferences or use any node.js app already built. Sample node app code is given down below, so you can get an idea about the node application.

const express = require('express');  
const app = express();  
const port = 8080;app.listen(port, () => {  
console.log('listening for request on port 8080');  
});app.get('/', (req, res) => {  
console.log('request made');  
res.sendFile("./docs/index.html", { root: \_\_dirname })  
});app.get('/about', (req, res) => {  
res.sendFile('./docs/about.html', { root: \_\_dirname });  
});app.get('/about-us', (req, res) => {  
res.redirect('/about');  
});app.use((req, res) => {  
res.status(404).sendFile('./docs/404.html', { root: \_\_dirname });  
});

Here, you can see a simple node app created using express and I’ve set port **8080** for this application. Also, I’ve created a **docs folder** inside the project and placed all my HTML files inside it. I’ve set basic routing to access different pages of my node app and I’ve added a 404 route at the bottom of the code to capture other requests.

# Create a Dockerfile

FROM node:10  
WORKDIR /usr/src/app  
COPY package\*.json ./  
RUN npm install  
COPY . .  
EXPOSE 8080  
CMD ["node", "app.js"]

I’ve used the official **Node Docker image** from docker hub and exposed port 8080 as I did in my node app. Once you’re done with the Dockerfile, go ahead and create your docker image for your node application.

Don’t forget to create a docker container, then run and see the response in the browser to make sure you’ve done it correctly up to this point.

# Install Minikube

I’m using Minikube which is a one-node cluster and it’s really helpful for us to deploy our apps in Kubernetes.

# Create Kubernetes components

When you’re done up to this point, you can continue configuring Kubernetes components using YAML files. Let’s create our Kubernetes deployment first.

When you’re creating a Kubernetes Deployment, you need to add the path of your docker image to this configuration file. In the above code, you can see, I’ve indicated that place with #path to your docker image.

You can use any **container registry** to store your docker image and you should set its’ path to this configuration file. Also, if you’re a beginner in Kubernetes, you can use **Docker Hub** as your container registry which is free and push your Docker image to it.

If you’re using Docker Hub, first create an account on Docker Hub. Then, create a repository in Docker Hub and open your terminal and type docker login and provide your **username** and **password.** Once your login succeeded, you can tag your Docker image and push it to the Docker Hub using docker tag & docker push commands.

Now you can apply your Kubernetes deployment to the cluster using the below command.

$ kubectl apply -f node-app-deployment.yaml

Run kubectl get pods to see the pods and this process might take some time since it usually takes some time to pull the image from Docker Hub.

You can describe the pod using command and use

kubectl describe pod <pod\_name>

kubectl get pod

--watch command to see container creating live events

Once your pod has been created, you can start with creating Kubernetes Service.

You can create the Service using the below command. In Kubernetes services, you can allocate any node port you prefer in between 30000–32767. So I’ve selected 31110 as the node port and you will see this port once your node app started running on your browser. Also, I’ve set service type as LoadBalancer , because using Minikube, we can get an External-IP and open our app in the browser easily.

$ kubectl apply -f node-app-service.yaml

You can check your services inside the cluster by running the below command.

$ kubectl get service

Here, EXTERNAL-IP is still pending for the service. But using Minikube, you can get an External-IP. Run the below command to enable External-IP for your application.

$ minikube service <service\_name>

# Opening Node.js app in the browser

The above command will automatically open your Node.js application in the browser If not, you can go to the above URL on your browser, then you will see your app runs there. Also, notice that the node port has appended to the URL.