Automated Deployment of Node.js Applications with Docker Containers and Kubernetes Orchestration

Step 1: Create a Node.js Application

1. **Download the given Node.js Application with Routing.**
2. Open **CMD** in the same project directory.

Name	Date modified	Type	Size
test	08-11-2024 12:06	File folder	
.dockerignore	08-11-2024 12:06	DOCKERIGNORE F...	1 KB
dockerfile	08-11-2024 12:06	File	1 KB
index	08-11-2024 13:00	JS File	1 KB
package	08-11-2024 12:06	JSON Source File	1 KB
package-lock	08-11-2024 12:06	JSON Source File	52 KB

Step 2: Create a Dockerfile

1. **Create a Dockerfile** by running the following command:

echo. > Dockerfile

2. **Paste the following code** into the Dockerfile:

```
FROM node:latest

WORKDIR /usr/src/app

COPY package.json ./

RUN npm install

COPY . .

EXPOSE 4000

CMD [ "node", "index.js" ]
```

Step 3: Build Your Docker Image

1. **Open CMD** and run the following command to build your Docker image:

   docker build -t divyamurugan/divya .

   - Here, replace divyamurugan with your Docker Hub username and divya with your image name.

```
[+] Building 1993.5s (11/11) FINISHED                                                                    docker:desktop-linux
 => [internal] load build definition from dockerfile                                                                      0.1s
 => => transferring dockerfile: 167B                                                                                      0.0s
 => [internal] load metadata for docker.io/library/node:latest                                                            2.3s
 => [auth] library/node:pull token for registry-1.docker.io                                                               0.0s
 => [internal] load .dockerignore                                                                                         0.0s
 => => transferring context: 84B                                                                                          0.0s
 => [1/5] FROM docker.io/library/node:latest@sha256:840dad0077213cadd2d734d542ae11cd0f648200be29504eb1b6e2c995d2b75a       0.0s
 => [internal] load build context                                                                                         0.0s
 => => transferring context: 218B                                                                                         0.0s
 => CACHED [2/5] WORKDIR /usr/src/app                                                                                     0.0s
 => CACHED [3/5] COPY package.json ./                                                                                     0.0s
 => [4/5] RUN npm install                                                                                              1989.0s
 => [5/5] COPY . .                                                                                                        0.1s
 => exporting to image                                                                                                    1.6s
 => => exporting layers                                                                                                   1.5s
 => => writing image sha256:1616e9a449eb1490caaa3823221c052397a287473dabba4619c9bc2179a804ed                              0.0s
 => => naming to docker.io/library/divya                                                                                  0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/xp3iq5mqvxjrrpxmzoy6aq9tpe

What's next:e `node --trace-warnings ...` to show where the warning was created)
    View a summary of image vulnerabilities and recommendations → docker scout quickview

C:\Users\ibmtr\Desktop\Node App>
```

2. **Verify the image build** by checking Docker Desktop. The built image will appear in the Docker Desktop interface.

3. **Check your locally available images** by running the following command:

   docker images

   - This will list all available images, including the newly built one with the latest tag.

Step 4: Create Docker Container

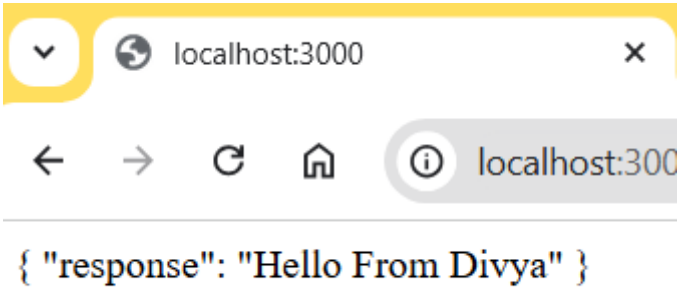
1. **Run the Docker container** using the following command:

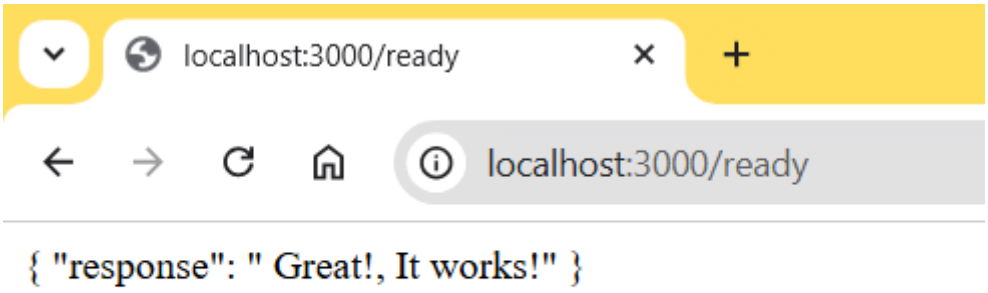
   docker run -d -p 3000:3000 divyamurugan/divya

- This will start your container and map the application's internal port (3000) to your local port (3000).

2. **Access the containerized Node.js application**:
   - Visit http://localhost:3000 to check if the application is running.
   - Test different routes like /will and /ready to confirm the application is responding.

```
C:\Users\ibmtr\Desktop\Node App>docker run -d -p 3000:3000 divya
a669e7f5a44182e79f8f2e1ae02198dfb5e68a97a6fa7236cca8fffc821629e9
```






```
C:\Users\ibmtr\Desktop\Node App>docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
                                      NAMES
a669e7f5a441   divya                                "docker-entrypoint.s…"   3 minutes ago  Up 3 minutes  0.0.0.0:3000->3000/tcp, 4
000/tcp
                                      great_taussig
a082d8277b42   gcr.io/k8s-minikube/kicbase:v0.0.45  "/usr/local/bin/entr…"   2 hours ago    Up 2 hours    127.0.0.1:51731->22/tcp,
127.0.0.1:51732->2376/tcp, 127.0.0.1:51734->5000/tcp, 127.0.0.1:51735->8443/tcp, 127.0.0.1:51733->32443/tcp   minikube
```

Step 5: Push Your Image to Docker Hub

1. **Login to Docker Hub**:
   - Run the following command:

docker login

   - Enter your Docker Hub credentials when prompted.

```
docker login
Authenticating with existing credentials...
Login Succeeded
```

1. **Push the image to Docker Hub**:
   - Run the following command to push your image:

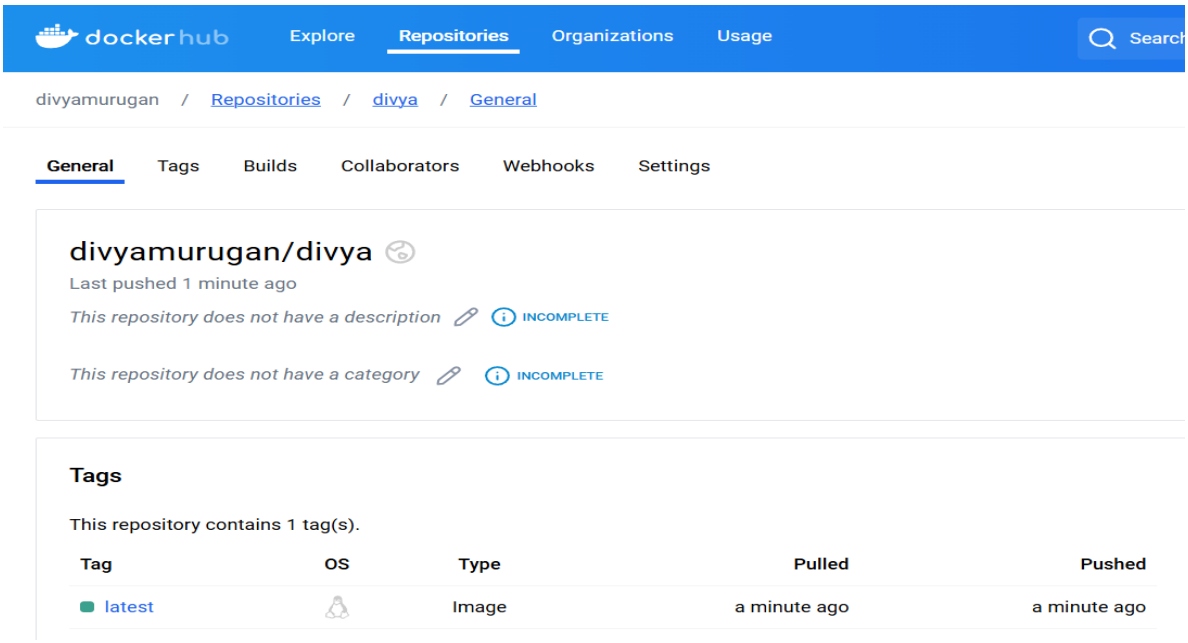
docker push divyamurugan/divya:latest

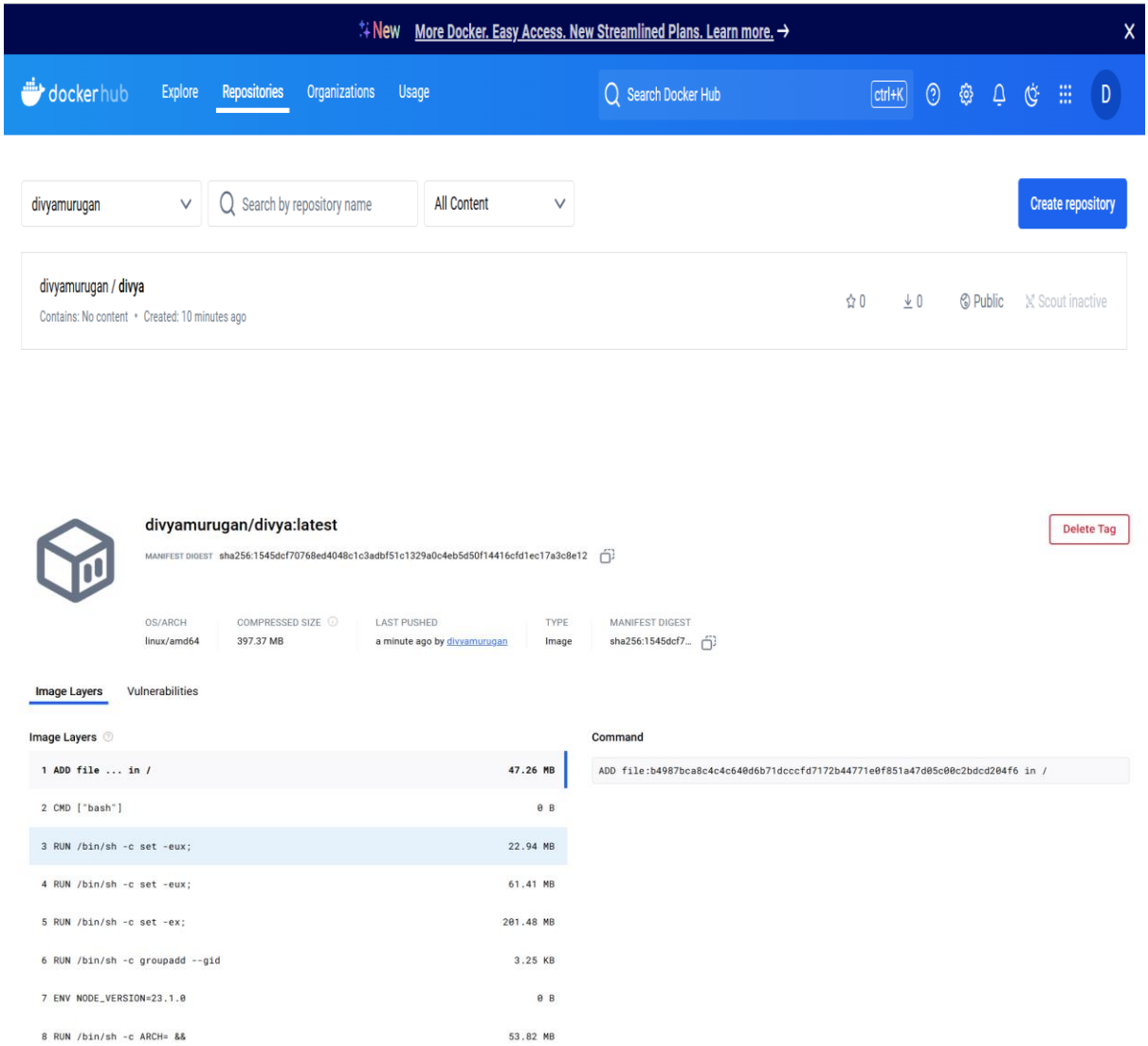
```
C:\Users\ibmtr\Desktop\Node App>docker tag divya:latest divyamurugan/divya:latest

C:\Users\ibmtr\Desktop\Node App>docker push divyamurugan/divya:latest
The push refers to repository [docker.io/divyamurugan/divya]
22f3593ebce1: Pushed
afa24fcfa264: Pushed
a8dd33642188: Pushed
218e84e35398: Pushed
af3ac94d1cf7: Pushed
2e6c05c90ce3: Pushed
454eacf69e50: Pushed
29f8baa3b3dc: Pushed
d23b5e6144a7: Pushing [==================================================> ]    586MB/587.5MB
e5ee1bd83fe3: Pushed
43da071b5e0c: Pushed
ef5f5ddeb0a6: Pushed
```

```
C:\Users\ibmtr\Desktop\Node App>docker push divyamurugan/divya:latest
The push refers to repository [docker.io/divyamurugan/divya]
22f3593ebce1: Pushed
afa24fcfa264: Pushed
a8dd33642188: Pushed
218e84e35398: Pushed
af3ac94d1cf7: Pushed
2e6c05c90ce3: Pushed
454eacf69e50: Pushed
29f8baa3b3dc: Pushed
d23b5e6144a7: Pushed
e5ee1bd83fe3: Pushed
43da071b5e0c: Pushed
ef5f5ddeb0a6: Pushed
latest: digest: sha256:1545dcf70768ed4048c1c3adbf51c1329a0c4eb5d50f14416cfd1ec17a3c8e12 size: 2838
```

1. **Verify Image on Docker Hub**:
   - Go to Docker Hub (https://hub.docker.com/) and log in with your credentials.
   - Navigate to your repository (e.g., divyamurugan/divya), and you should see your image once the push completes.

Click that image name and click latest





Step 6: Create Kubernetes Deployment and Service YAML Files

Kubernetes Deployment YAML:

1. **Create the deployment.yml file** with the following content:

apiVersion: apps/v1

kind: Deployment

metadata:

  name: nodeapp-deployment

  labels:

    app: nodeapp

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nodeapp
  template:
    metadata:
      labels:
        app: nodeapp
    spec:
      containers:
      - name: nodeserver
        image: divyamurugan/divya:latest  # Replace with your image name
        ports:
        - containerPort: 3000
```

EXPLANATION:

1. **apiVersion: apps/v1**
   - Tells Kubernetes you're using version 1 of the Deployment API.
2. **kind: Deployment**
   - You’re creating a **Deployment**, which manages how many copies (pods) of your app you want running.
3. **metadata:**
   - **name: nodeapp-deployment** – This is the name of your Deployment.

- **labels** – Labels help group and identify resources. Here, the label app: nodeapp is given to this Deployment.

4. **spec:**
   - **replicas: 1** – Tells Kubernetes to run **1** copy of your app (1 pod).
   - **selector** – Kubernetes uses this to find which pods belong to this Deployment using the label app: nodeapp.
5. **template:**
   - **metadata**: Defines the same label app: nodeapp for the pods.
   - **spec**: Defines how the containers inside the pods should run.
     - **containers**: This is where the app runs.
       - **name: nodeserver** – The name of the container.
       - **image: divyamurugan/divya -** This is the Docker image to use for your app (your Node.js app in this case).
       - **containerPort: 3000** – Tells Kubernetes that the container should listen on port **3000** (where your Node.js app runs).

Kubernetes Service YAML:

2. **Create the service.yml file** with the following content:

```
apiVersion: v1
kind: Service
metadata:
  name: nodeapp-service
spec:
  selector:
    app: nodeapp
  type: LoadBalancer
```

```
  ports:
  - protocol: TCP
    port: 5000
    targetPort: 3000
    nodePort: 31110
```

- Ensure that the app name in both the deployment and service YAML files matches (in this case, nodeapp).

EXPLANATION:

1. apiVersion: v1

- This specifies that you're using **version 1** of the Service API in Kubernetes.

2. kind: Service

- The kind is **Service**, meaning you're defining a network service for your application.

3. metadata:

- **name: nodeapp-service** – The name of the Service. This name is used to refer to the Service inside the Kubernetes cluster.

4. spec:

The spec section defines how the Service will behave.

- **selector:**
  - **app: nodeapp** – This tells the Service to route traffic to the pods with the label app: nodeapp. These are the pods defined in your **Deployment** earlier.
- **type: LoadBalancer**:
  - This exposes the Service externally (outside the Kubernetes cluster) using a cloud provider's load balancer.

- In a local setup like **Minikube**, this may not create an actual external load balancer, but Kubernetes will handle routing the traffic to the correct pod.

- **ports:**
  - **protocol: TCP** – The Service will use the **TCP** protocol (most web traffic uses TCP).
  - **port: 5000** – This is the port the **Service** listens on. It's the port that you can access from outside the cluster or within the cluster.
  - **targetPort: 3000** – This is the port on the pod where your application is running. Your Node.js app listens on **port 3000**, and the Service will forward traffic from **port 5000** to **port 3000** on the pod.
  - **nodePort: 31110** – This exposes the Service on a specific port on the nodes (your Kubernetes cluster machines). You can access the Service on **port 31110** from outside the Kubernetes cluster.

Step 7: Deploy Your Application in Kubernetes

1. **Check Minikube status**:
   - Run the following command to check the status of your Minikube setup:

minikube status

```
C:\Users\ibmtr\Desktop\Node App>minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

1. **Verify if any resources (pods, deployments, services) are running**:

kubectl get pods

kubectl get deployments

kubectl get svc

- Initially, you should see "no resources" because nothing is deployed yet.

```
C:\Users\ibmtr\Desktop\Node App>kubectl get pods
No resources found in default namespace.
```

```
C:\Users\ibmtr\Desktop\Node App>kubectl get deployments
No resources found in default namespace.
```

```
C:\Users\ibmtr\Desktop\Node App>kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   141m
```

1. **Apply the deployment and service files** to create the resources:

   kubectl apply -f deployment.yml

   kubectl apply -f service.yml

2. **Check the deployment status**:

   - Run the following command to verify that the deployment is created:

     kubectl get deployment

   - The deployment should be ready, and the pods will be created automatically.

3. **Check the status of the pods**:

   kubectl get pods

   - You should see the pods running as part of the deployment.

4. **Check the service status**:

   - Run the following command to verify the service:

     kubectl get svc

```
C:\Users\ibmtr\Desktop\Node App>kubectl apply -f deployment.yml
deployment.apps/nodeapp-deployment created

C:\Users\ibmtr\Desktop\Node App>kubectl get deployment
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
nodeapp-deployment   0/1     1            0           21s

C:\Users\ibmtr\Desktop\Node App>kubectl get deployment
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
nodeapp-deployment   0/1     1            0           34s

C:\Users\ibmtr\Desktop\Node App>kubectl get pods
NAME                                  READY   STATUS             RESTARTS   AGE
nodeapp-deployment-7657fb95d9-tb5fr   0/1     ImagePullBackOff   0          51s

C:\Users\ibmtr\Desktop\Node App>kubectl apply -f service.yml
service/nodeapp-service created

C:\Users\ibmtr\Desktop\Node App>kubectl get svc
NAME              TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes        ClusterIP      10.96.0.1        <none>        443/TCP          144m
nodeapp-service   LoadBalancer   10.106.182.242   <pending>     5000:31110/TCP   15s
```

Step 8: Access the Application

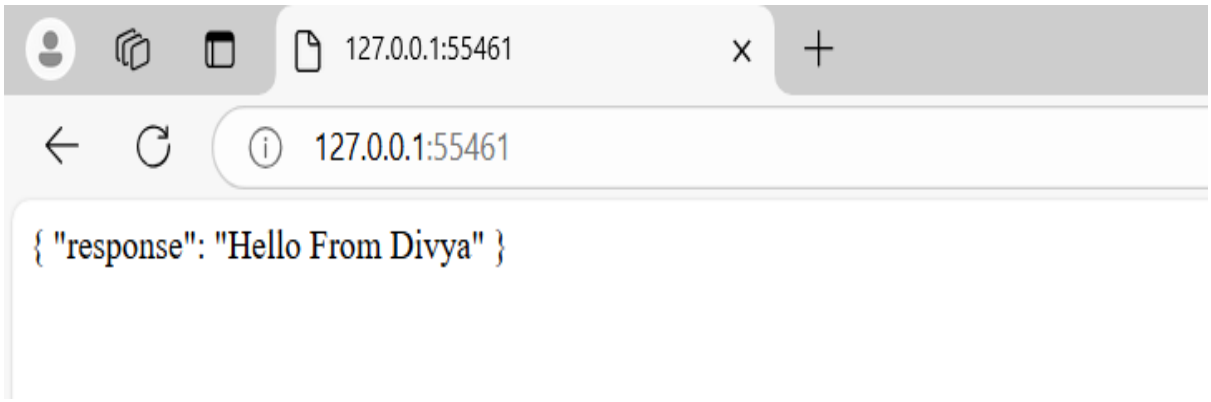
1. **Access the application from outside Minikube**:
   - Run the following command to open the service URL in your browser:

     minikube service nodeapp-service

2. **Get the URL for your service**:
   - Minikube will provide a URL where you can access your Node.js application in the browser.

3. **Verify the application**:
   - The homepage should load successfully, confirming the deployment is complete.

```
C:\Users\ibmtr\Desktop\Node App>minikube service nodeapp-service
|-----------|-----------------|-------------|---------------------------|
| NAMESPACE |      NAME       | TARGET PORT |            URL            |
|-----------|-----------------|-------------|---------------------------|
| default   | nodeapp-service |        5000 | http://192.168.49.2:31110 |
|-----------|-----------------|-------------|---------------------------|

X Exiting due to SVC_UNREACHABLE: service not available: no running pod for service nodeapp-service found
*
┌───────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐
│                                                                                                                   │
│    * If the above advice does not help, please let us know:                                                       │
│      https://github.com/kubernetes/minikube/issues/new/choose                                                     │
│                                                                                                                   │
│    * Please run `minikube logs --file=logs.txt` and attach logs.txt to the GitHub issue.                          │
│    * Please also attach the following file to the GitHub issue:                                                   │
│    * - C:\Users\ibmtr\AppData\Local\Temp\minikube_service_4dd957247a1a2f6e3db434b607d052c72ce97a47_0.log           │
│                                                                                                                   │
└───────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘
```



---------- *All the Best* ----------