**Study Guide: Deploying a Custom Node.js App with Docker & Kubernetes**

**Introduction**

This guide walks you through creating a **Node.js** application using **Express**, containerizing it with **Docker**, pushing the image to **Docker Hub**, and deploying it on **Kubernetes**.

**Step 1: Create a Node.js Application**

We’ll build a simple Node.js server that returns "Hello from Server".

**Initialize the Project**

1. Create a project directory:
2. mkdir k8s-web-hello && cd k8s-web-hello
3. Initialize a Node.js project:
4. npm init -y
5. Install **Express**:
6. npm install express

**Create the Application Code**

1. Create index.js:
2. touch index.js
3. Add the following code:
4. const express = require('express');
5. const os = require('os');
6. const app = express();
7. const port = 3000;
8. app.get('/', (req, res) => {
9. const hostname = os.hostname();
10. res.send(`Hello from ${hostname}`);
11. console.log(`Request received from ${hostname}`);
12. });
13. app.listen(port, () => {
14. console.log(`Server running on port ${port}`);
15. });
16. Update package.json to include a start script:
17. "scripts": {
18. "start": "node index.js"
19. }

**Step 2: Containerize the Application with Docker**

**Create a Dockerfile**

1. Create Dockerfile:
2. touch Dockerfile
3. Add the following content:
4. FROM node:alpine
5. WORKDIR /app
6. COPY package\*.json ./
7. RUN npm install
8. COPY . .
9. EXPOSE 3000
10. CMD ["npm", "start"]

**Build and Push the Docker Image**

1. Build the Docker image:
2. docker build -t <your-dockerhub-username>/k8s-web-hello .
3. Verify the image:
4. docker images
5. Log in to Docker Hub:
6. docker login
7. Push the image:
8. docker push <your-dockerhub-username>/k8s-web-hello

**Step 3: Deploy the Application in Kubernetes**

**Create a Deployment**

1. Deploy the app in Kubernetes:
2. kubectl create deployment k8s-web-hello --image=<your-dockerhub-username>/k8s-web-hello
3. Verify the deployment:
4. kubectl get deployments

**Expose the Deployment**

1. Expose it as a **NodePort** service:
2. kubectl expose deployment k8s-web-hello --type=NodePort --port=3000
3. Verify the service:
4. kubectl get services

**Access the Application**

1. Get the Minikube node’s IP:
2. minikube ip
3. Access the app via the node’s IP and port:
4. curl <node-ip>:<node-port>
5. Alternatively, open the service in a browser:
6. minikube service k8s-web-hello

**Key Takeaways**

* Containerize Node.js applications using Docker.
* Push images to Docker Hub for easy deployment.
* Deploy and expose applications in Kubernetes.
* Access Kubernetes services using Minikube or a cloud provider.

**Commands Summary**

| **Command** | **Description** |
| --- | --- |
| npm init -y | Initialize a Node.js project. |
| npm install express | Install Express framework. |
| docker build -t <username>/<image-name> . | Build a Docker image. |
| docker push <username>/<image-name> | Push the Docker image to Docker Hub. |
| kubectl create deployment <name> --image=<img> | Create a Kubernetes deployment. |
| kubectl expose deployment <name> --type=NodePort --port=<p> | Expose the Deployment using a NodePort Service. |

**Next Steps**

* Scale the deployment to multiple replicas.
* Configure **Ingress** for better traffic management.
* Deploy the application on a cloud provider using a **LoadBalancer**.

With this guide, you're ready to build, containerize, and deploy your own applications in Kubernetes! 🚀

### Study Material: Creating a Custom Node.js Application and Docker Image

---

#### \*\*Introduction\*\*

In this section, we’ll create a custom \*\*Node.js\*\* application using the \*\*Express\*\* framework, containerize it using \*\*Docker\*\*, and push the Docker image to \*\*Docker Hub\*\*. Finally, we’ll deploy the application in Kubernetes using the custom image.

---

### \*\*Step 1: Create a Node.js Application\*\*

Let’s create a simple Node.js application that responds with a "Hello from Server" message.

---

#### \*\*Initialize the Node.js Project:\*\*

1. Create a new directory for your project:

```bash

mkdir k8s-web-hello

cd k8s-web-hello

```

2. Initialize a new Node.js project:

```bash

npm init -y

```

3. Install the \*\*Express\*\* framework:

```bash

npm install express

```

---

#### \*\*Create the Application Code:\*\*

1. Create a file named `index.js`:

```bash

touch index.js

```

2. Add the following code to `index.js`:

```javascript

const express = require('express');

const os = require('os');

const app = express();

const port = 3000;

app.get('/', (req, res) => {

const hostname = os.hostname();

res.send(`Hello from ${hostname}`);

console.log(`Request received from ${hostname}`);

});

app.listen(port, () => {

console.log(`Web server is listening at port ${port}`);

});

```

3. Update the `scripts` section in `package.json` to include a `start` script:

```json

"scripts": {

"start": "node index.js"

}

```

---

### \*\*Step 2: Containerize the Application Using Docker\*\*

Now, let’s create a \*\*Dockerfile\*\* to containerize the Node.js application.

---

#### \*\*Create the Dockerfile:\*\*

1. Create a file named `Dockerfile`:

```bash

touch Dockerfile

```

2. Add the following content to the `Dockerfile`:

```dockerfile

# Use Node.js Alpine as the base image

FROM node:alpine

# Set the working directory

WORKDIR /app

# Copy package.json and package-lock.json

COPY package\*.json ./

# Install dependencies

RUN npm install

# Copy the rest of the application files

COPY . .

# Expose port 3000

EXPOSE 3000

# Start the application

CMD ["npm", "start"]

```

---

#### \*\*Build the Docker Image:\*\*

1. Build the Docker image:

```bash

docker build -t <your-dockerhub-username>/k8s-web-hello .

```

Replace `<your-dockerhub-username>` with your Docker Hub username.

2. Verify the image was built:

```bash

docker images

```

---

#### \*\*Push the Docker Image to Docker Hub:\*\*

1. Log in to Docker Hub:

```bash

docker login

```

2. Push the image to Docker Hub:

```bash

docker push <your-dockerhub-username>/k8s-web-hello

```

---

### \*\*Step 3: Deploy the Application in Kubernetes\*\*

Now that the Docker image is available on Docker Hub, let’s deploy it in Kubernetes.

---

#### \*\*Create a Deployment:\*\*

1. Create a Deployment using the custom image:

```bash

kubectl create deployment k8s-web-hello --image=<your-dockerhub-username>/k8s-web-hello

```

2. Verify the Deployment:

```bash

kubectl get deployments

```

---

#### \*\*Expose the Deployment:\*\*

1. Expose the Deployment using a \*\*NodePort\*\* Service:

```bash

kubectl expose deployment k8s-web-hello --type=NodePort --port=3000

```

2. Verify the Service:

```bash

kubectl get services

```

---

#### \*\*Access the Application:\*\*

1. Get the IP address of the Minikube node:

```bash

minikube ip

```

2. Access the application using the node’s IP and the NodePort:

```bash

curl <node-ip>:<node-port>

```

3. Alternatively, use the `minikube service` command to open the Service in your browser:

```bash

minikube service k8s-web-hello

```

---

### \*\*Key Takeaways\*\*

- You can create a custom Node.js application and containerize it using Docker.

- Push the Docker image to Docker Hub for easy access in Kubernetes.

- Use `kubectl` to create a Deployment and expose it using a Service.

- Access the application using the node’s IP and the assigned port.

---

### \*\*Commands Summary\*\*

| \*\*Command\*\* | \*\*Description\*\* |

|--------------------------------------------------|------------------------------------------------------|

| `npm init -y` | Initialize a Node.js project. |

| `npm install express` | Install the Express framework. |

| `docker build -t <username>/<image-name> .` | Build a Docker image. |

| `docker push <username>/<image-name>` | Push the Docker image to Docker Hub. |

| `kubectl create deployment <name> --image=<img>` | Create a Kubernetes Deployment. |

| `kubectl expose deployment <name> --type=NodePort --port=<p>` | Expose the Deployment using a NodePort Service. |

---

### \*\*Next Steps\*\*

Now that you’ve deployed a custom Node.js application in Kubernetes, you can:

- Scale the Deployment to multiple replicas.

- Explore advanced Service configurations like \*\*Ingress\*\*.

- Deploy your application to a cloud provider and use a real \*\*LoadBalancer\*\*.

---

With this knowledge, you’re ready to build, containerize, and deploy your own applications in Kubernetes! 🚀