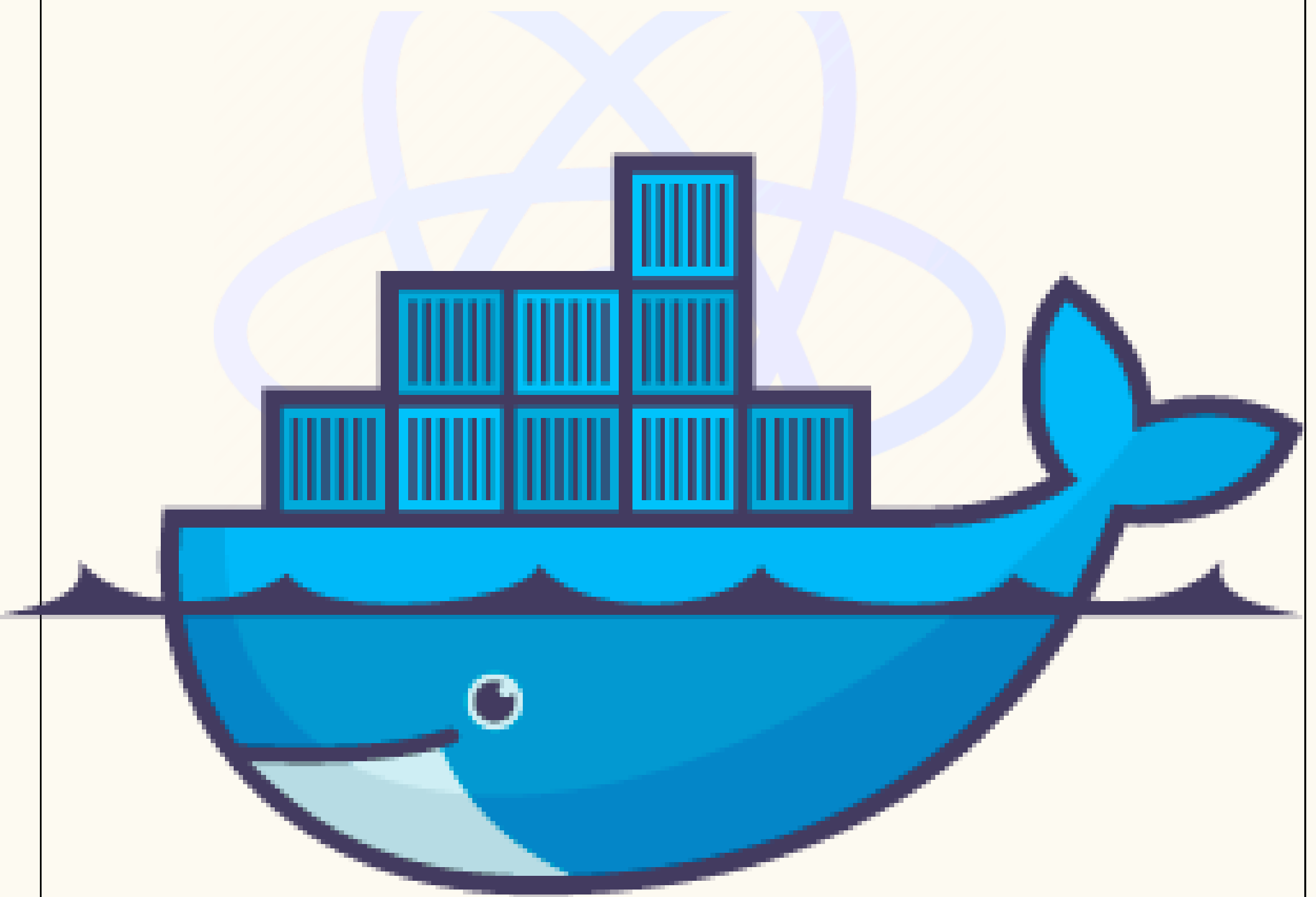


How to Dockerize a **Web** Application



Docker is a platform that enables developers to automate the deployment, scaling, and management of applications using containerization.

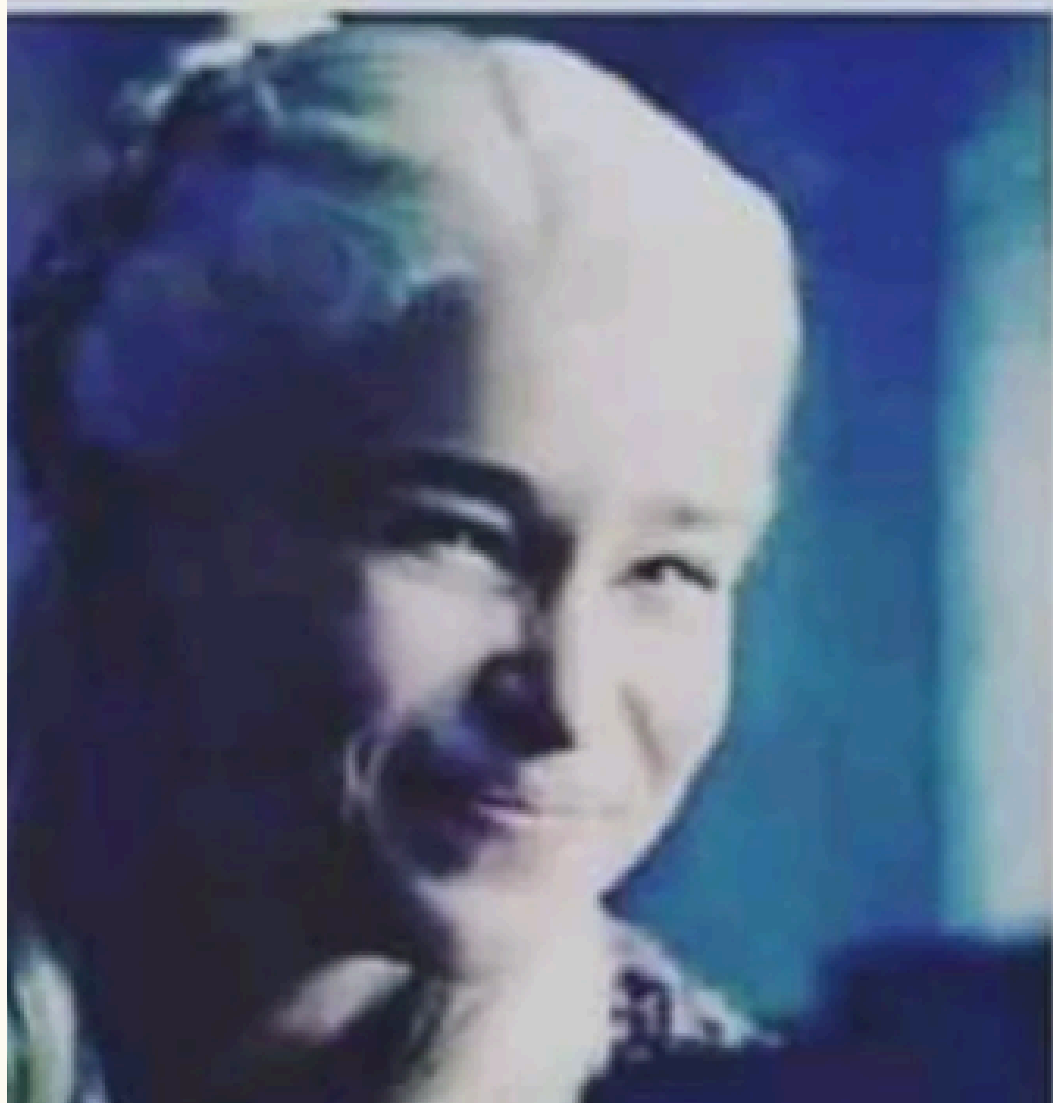


[/in/mulukenm](https://www.linkedin.com/in/mulukenm)

What issues does Docker fix?

- it works on my computer

- yes but we are not going to give your computer to the client



- **"It works on my computer":** Docker addresses this by providing a consistent, isolated environment for applications, ensuring that if something works within Docker, it should work similarly in other environments.

Advantages of using Docker

- By addressing the common problem of software running well in a developer's environment but failing in other environments. Docker helps mitigate this issue by providing a consistent and isolated environment for applications through containers.
- This ensures that if something works on your computer within Docker, it should work the same way in other environments where Docker is used, reducing the "it works on my computer" problem.
- **Simplified Dependency Management:** Bundles all necessary dependencies within the container.
- **Scalability:** Easily scale applications up or down with minimal overhead.
- **Isolation:** Each container operates independently, preventing conflicts between applications.



How to Dockerize a web application?

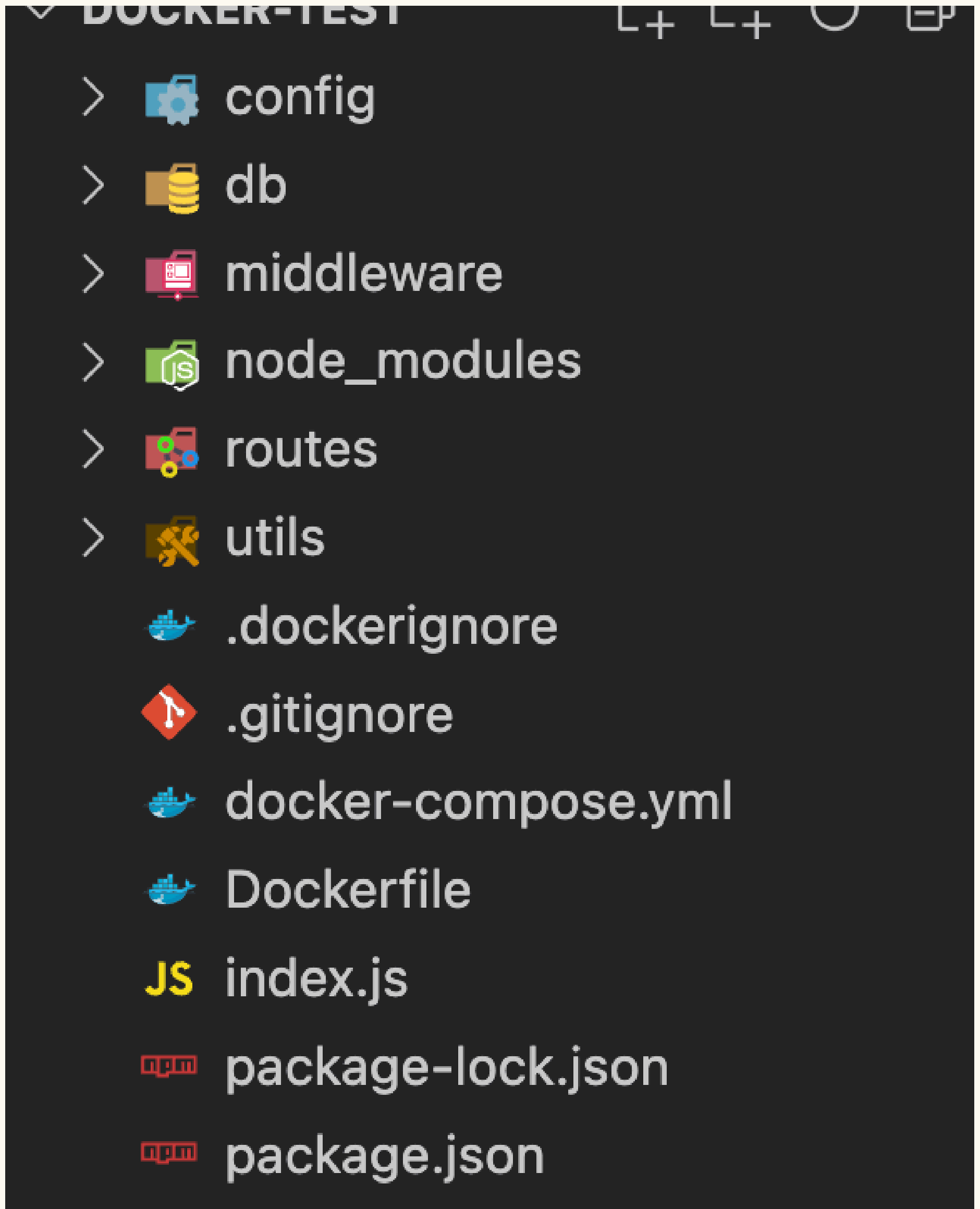
- Install Docker
 - Go to docker hub and install the version that matches with your OS
- Goto your application root folder and create the following files.
 - **Dockerfile** : main config file for your docker image
 - **.dockerignore**: a file similar to gitignore to exclude files from the docker image
 - **docker-compose.yml**: to add advanced config for the docker image

The process of Dockerizing is similar for different programming languages. In this lesson, we'll focus on Dockerizing a Node.js application.

1. Goto your project root and create the above mentioned files



Depending on your project setup, it will look something like this




Open the **Dockerfile file and add the following code into it**

Your Docker config may vary depending on your requirement

```
Dockerfile ×
Dockerfile > ...
1  # Use the official Node.js image as the base image
2  FROM node:lts-slim
3  # Set the working directory
4  WORKDIR /app
5  # Copy package.json and package-lock.json to the working directory
6  COPY package*.json ./
7  # Install dependencies
8  RUN npm install
9  # Copy the rest of the application code to the working directory
10 COPY . .
11 # Expose the port the app runs on
12 EXPOSE 3000
13 # Command to run the application
14 CMD ["npm", "start"]
15
```

now open **.dockerignore** file and add files to be excluded from the docker image

```
 .dockerignore
1  ./node_modules
```



[/in/mulukenm](https://www.linkedin.com/in/mulukenm)

Build the Docker Image.

Open a terminal in the root directory of your React application and run the following command to build the Docker image:

```
docker build -t my-nodejs-app .
```

Run the Docker Container

After building the Docker image, you can run it using the following command:

```
docker run -p 3000:80 my-nodejs-app
```

Then you can push the Docker image to Docker Hub or other private registry repositories like AWS and use it in production.



[/in/mulukenm](https://www.linkedin.com/in/mulukenm)

Optional: Use Docker Compose

If you have additional services or bigger application(e.g., a microservice API or a database), you can use Docker Compose to manage them.

Open the `docker-compose.yml` and add your config parameters. this one is optional

```
🚢 docker-compose.yml X
🚢 docker-compose.yml
1  version: '3'
2  services:
3    web:
4      build: .
5      ports:
6        - "3000:80"
7
```



**Run the application with Docker
Compose:**

docker-compose up

**This approach simplifies managing
multiple services and their
dependencies.**

THEN WHAT?



It works on every machine!



[/in/mulukennm](https://www.linkedin.com/in/mulukennm)