Docker Tutorial

1. Installing Docker

Windows and Mac:

- Download Docker Desktop from Docker Hub and follow the installation instructions.

Linux:

- Update your package index:

  ```
  sudo apt-get update
  ```

- Install Docker:

  ```
  sudo apt-get install docker-ce docker-ce-cli containerd.io
  ```

- Verify that Docker is installed correctly by running the hello-world image:

  ```
  sudo docker run hello-world
  ```

2. Creating a Simple Full-Stack Application

Structure

Your application structure might look like this:

```
/full-stack-app
  /client
    - package.json
    - // your React app files
  /server
    - package.json
```

```
    - // your Node.js app files
  - docker-compose.yml
```

3. Dockerizing the Node.js Application

1. **Create a Dockerfile** in your `/server` directory:

```
# Use the official Node.js 14 image as a parent image
FROM node:14

# Set the working directory
WORKDIR /usr/src/app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of your application's code
COPY . .

# Make port 3000 available outside this container
EXPOSE 3000

# Run the application
CMD ["node", "index.js"]
```

1. **Build your Docker image**:

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

1. **Run your Docker container**:

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

4. Dockerizing the React Application

1. **Create a Dockerfile** in your `/client` directory:

```
# Use the official Node.js 14 image to build your app
FROM node:14 as build

# 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 your application's code
COPY . .

# Build your app
RUN npm run build

# Use the official nginx image for a production build
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html

# Expose port 80
EXPOSE 80
```

1. **Build your Docker image**:

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

1. **Run your Docker container**:

```
docker run -p 80:80 my-react-app
```

5. Using Docker Compose for Full-Stack Deployment

Finally, to simplify running both containers together, use Docker Compose.

1. **Create a** `docker-compose.yml` at the root of your project:

```
version: '3'
services:
  server:
    build: ./server
    ports:
      - "3000:3000"
    volumes:
      - ./server:/usr/src/app
    environment:
      - NODE_ENV=production

  client:
    build: ./client
    ports:
      - "80:80"
    depends_on:
      - server
```

1. **Start your full-stack application** with Docker Compose:

```
docker-compose up --build
```

This command builds (or rebuilds) images as necessary and starts the containers defined in your `docker-compose.yml`. Your React application will now be accessible on port 80, and your

Node.js API will be accessible on port 3000.