github project link: https://github.com/katleo-rantle/containerized-react-app

working with docker multi stage builds
nothing fancy goal is to learn how to use docker to build and serve a react app

todo

- ✅ ~~spin up react app with `create-react-app` ✅ 2025-12-22~~
- ✅ ~~write multi stage docker file to build react app ✅ 2025-12-27~~
- ✅ ~~extend dockerFile to serve static files thru nginx server ✅ 2025-12-27~~

create build and serve a react app

a word on CRA:: in 2025 **Create React App (CRA)** has been officially deprecated and is no longer recommended.

```
npx create-react-app --template typescript containerized-react-app
```

npm start uses webpackdevServer

```
yarn create vite containerized-react-app --template react-ts
```

```
yarn run dev
```

`src/app.tsx` edit it if u want

✏️ 💡 **more info** ›

gemini: When you run the "dev" command, your tools (Vite, Next.js, etc.) prioritize **developer speed** over **user performance**.

If you want to see how your app actually performs or if you're ready to deploy, you must run the build command:

```
yarn build
```

This creates a `/dist` (Vite) or `/.next` (Next.js) folder containing the highly optimized version of your app.

```
yarn preview
```

This starts a local server that serves the *production* files, allowing you to check performance before you go live.

⚠️ Warning: Never deploy the output of a dev server to a real website. It will be 5–10x slower and may expose your source code or sensitive debugging information to the public.

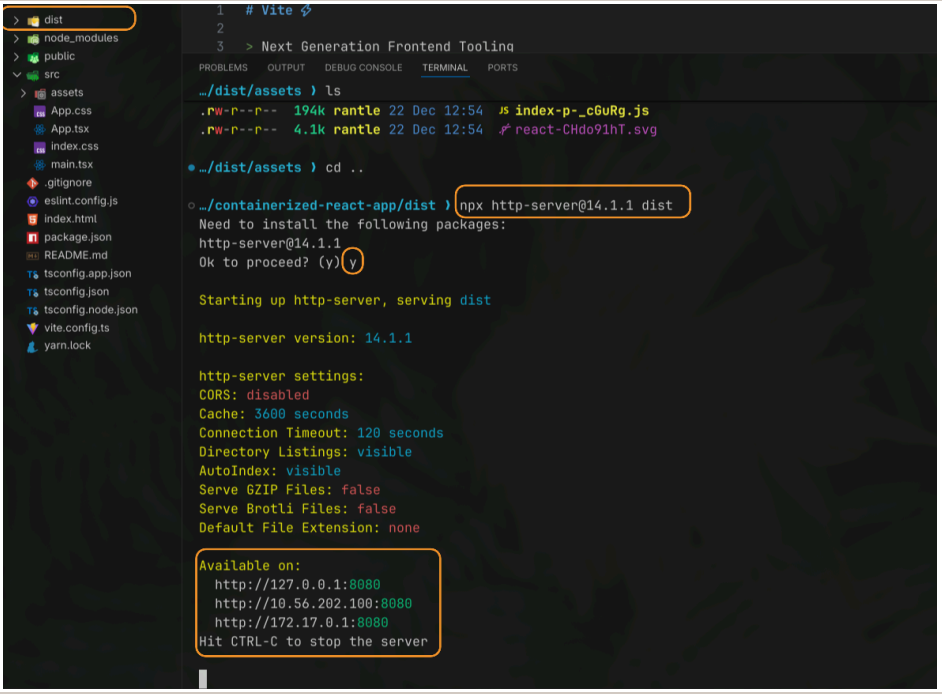
build app

```
yarn build
```

serve the output from build step

`dist` - folder containing build files
trgger package without installing

```
npx http-server@14.1.1 dist
```

note: no hot reload here need to run build command again	
didnt work coz i was already in the dist folder i needed to cd .. then npx http..	

multi stage docker file (Build and serve usecase)

Todo

- [x] ~~1: build production bundle (files)~~ ✅ ~~2025-12-22~~
- [ ] 2: serve the bundle with http server

1: build production bundle (files)

in root

```
touch dockerfile .dockerignore
```

```
node_modules
dist
```

alipine = lightweight downloads faster
4: run / install dep
5: copy everything else (safe to do coz we are ignoring some)

```
FROM node:22-alpine

WORKDIR /app

COPY package*.json .

RUN npm ci

COPY . .

RUN npm run build
```

🖊 💡 **yarn eqilavent** ›

```
FROM node:22-alpine

WORKDIR /app

# 1. Copy both the package.json AND the yarn.lock file
# This is crucial so Yarn can verify the exact versions

COPY package.json yarn.lock ./

# 2. Run the Yarn equivalent of 'npm ci' # For Yarn v1 (Classic): --frozen-lockfile # For Yarn v2+ (Modern): --immutable

RUN yarn install --frozen-lockfile

# 3. Copy the rest of your source code
COPY . .
```

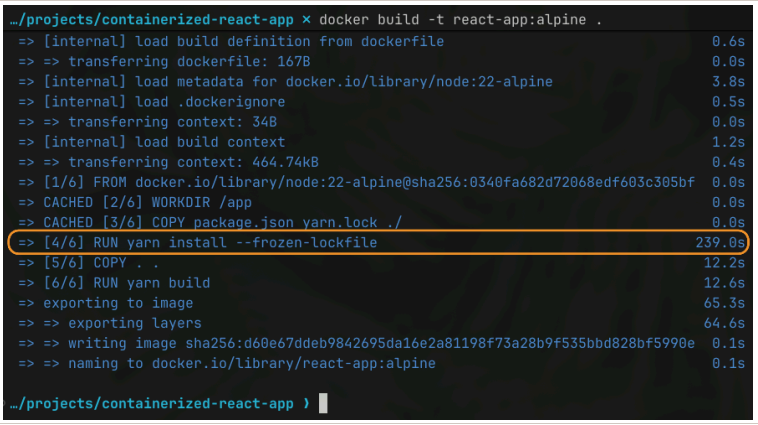
```
# 4. Build the application
RUN yarn build
```

create container from this image

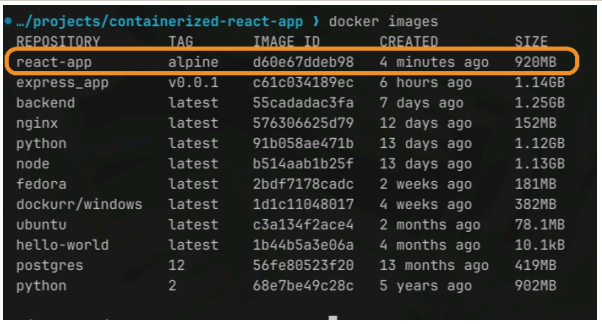
First we create the image
`.` = curr dir as context

```
docker build -t react-app:alpine .
```

out



docker images



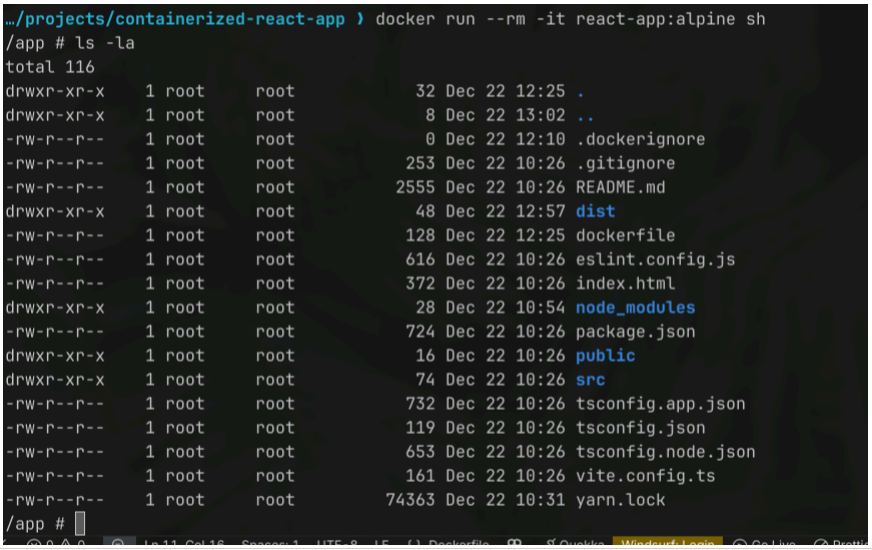
run container from image we just created
`sh` = shell command

```
docker run --rm -it react-app:alpine sh
```

we fall inside container in /app dir

```
ls -la

tree build # for npm

# for yarn / vite
tree dist

exit #out of container since we ran --rm container should be removed

# confirm no continers runnning
docker ps

# or
docker container ls
```

out > same structure as local machine



tree dist

2: serve the bundle with http server

name the 1st stage

```
# stage1: build production bundle
FROM node:22-alpine AS build

WORKDIR /app

COPY package.json yarn.lock ./

RUN yarn install --frozen-lockfile

COPY . .

RUN yarn build

# stage2: serve the bundle with http server nginx
FROM nginx:1.27.0

COPY --from=dist /app/dist /usr/share/nginx/html
```

create image > name it

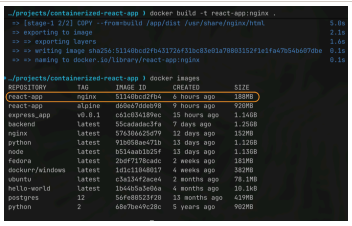
```
docker build -t react-app:nginx .
```

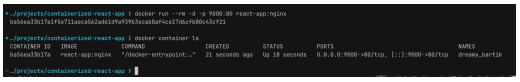
nginx will expose http server on port 80 inside container

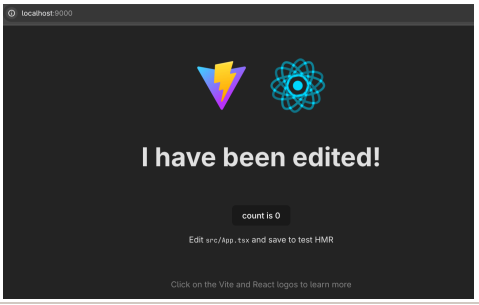
run container based on image
map local machine port 9000 to (direct traffic to) container port 80
use `react-app:nginx` image

```
docker run --rm -d -p 9000:80 react-app:nginx
```

outcomes







changes to source code > new image

OG code

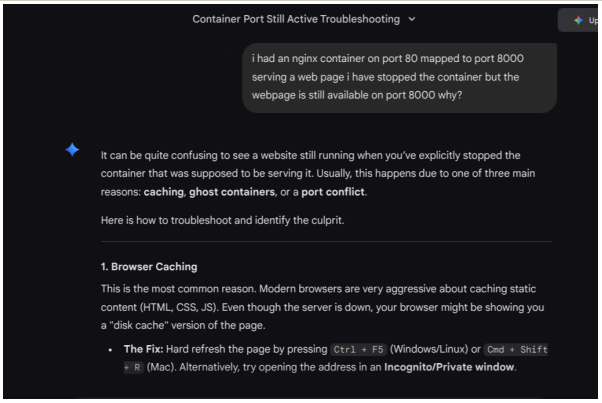
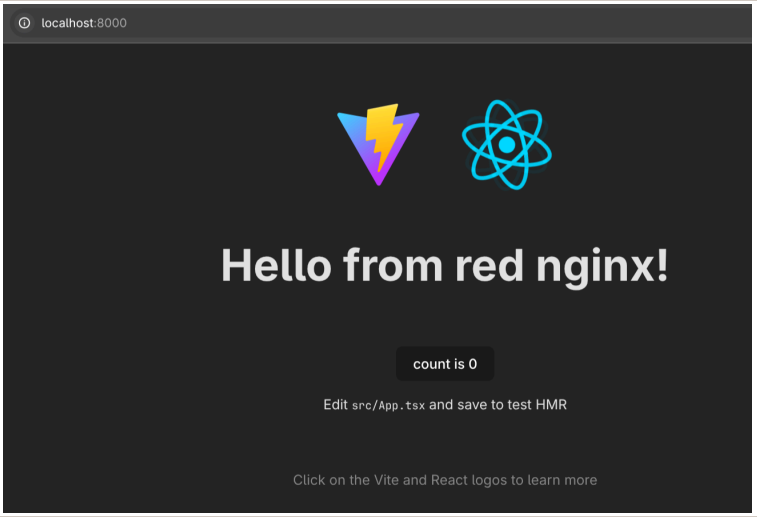
```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        <a href="https://vite.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React
logo" />
        </a>
      </div>
      <h1>I have been edited!</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.tsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </>
  )
}
```

Changes

```
      <h1>Hello  from blue nginx!</h1>
```

```
export default App
```

```
docker build -t react-app:blue .
```

container port still active	- `docker build -t react-app:red .` - `docker run --rm -d -p 8000:80 react-app:red`
 caching was the problem	

stop and remove containers

```
//stop containers
docker stop $(docker ps -q)

// or
docker stop $(docker contianer ls -q)


// remove containers
docker rm $(docker ps -aq)

// remove images
docker rmi -f $(docker images -q)
```

DONE!!!

what i learnt

To build and ship a react app from a single docker file
How different base images can be used for different stages of our build & ship process (a use case for Multi stage build)