

# Ambiente de desenvolvimento Nodejs com docker.

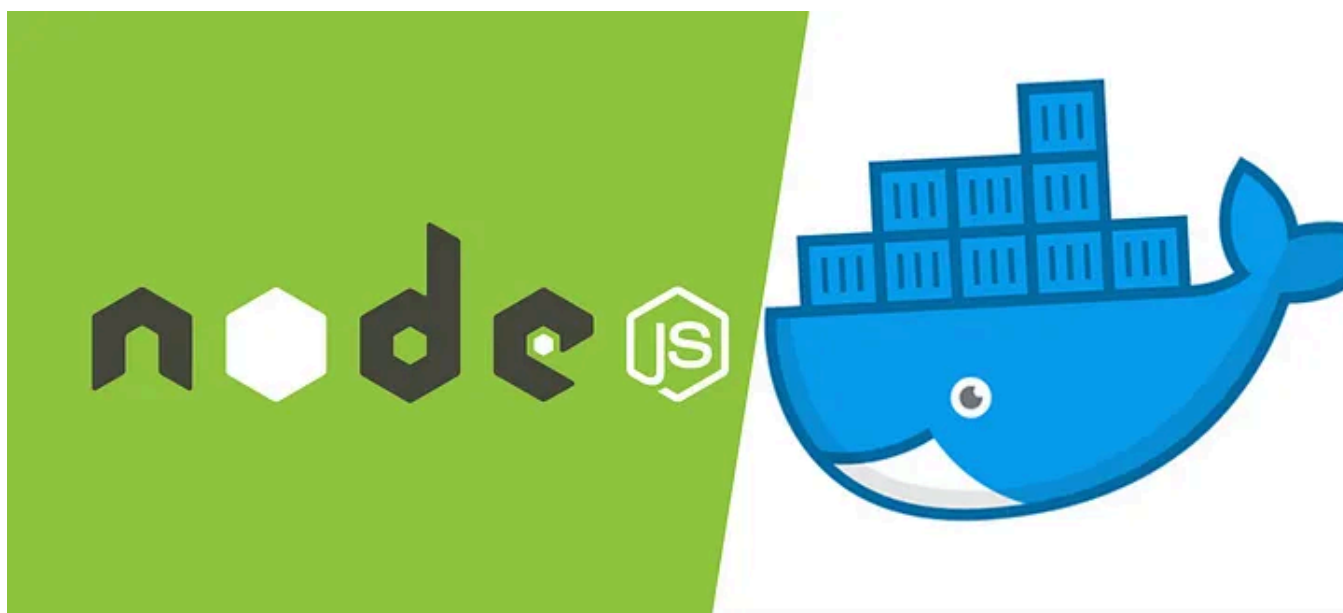


André Teles · Follow

8 min read · Oct 15, 2020



Share



Bem se você caiu aqui nesse artigo é porque você passando por problemas para configurar seu ambiente de desenvolvimento usando docker ou está querendo saber um pouco mais como isso poderá otimizar seu Workflow.

Em um primeiro momento se você é iniciante nessa coisa toda de docker, container e imagens sugiro acompanhar a documentação do docker para entender melhor como as coisas funcionam. É totalmente fundamental entender primeiramente esses conceitos, pois aqui vou focar em como configurar seu ambiente para que as coisas fiquem fáceis quando aquele membro novo da equipe entrar e gastar 2 ou 3 dias para configurar todo o ambiente e quando termina não funciona da maneira esperada. O melhor de tudo é que tem aquelas pessoas que falam “mas na minha máquina funciona”. Para essa galera, tenho um desabafo. Adivinham só, as pessoas

não vão usar a sua máquina, se na sua maquina funciona e na de outra pessoa não, adivinha novamente! Tem algo errado que não está certo. Mas enfim, o docker vai nos ajudar a minimizar esse tipo de problema e focar no que realmente importa, o negócio. Vamos lá!

Vamos definir primeiro as expectativa do que vamos cobrir por aqui:

- Vamos usar Nodejs.
- Vamos criar nossa própria imagem docker usando multi stage
- Veremos boas práticas ao criar sua imagens.
- Criaremos nosso próprio docker-compose.yml.
- Health check para coordenar seus containers.

Observação: Oque irei mostrar aqui é para ambiente de desenvolvimento 😊.

## Criando nosso Dockerfile.

Bem na própria documentação tem uma seção tratando apenas desses assuntos, mas vou tentar explicar alguns deles aqui.

### Labels:


Como pode ser visto aqui, as labels são um combinação de chave e valor na qual representa alguma informação significativa para pessoas que vão usar sua imagem ou em alguns casos devem seguir uma syntax pré definida como é usado no [traefik](#). Veja alguns exemplos:

```
FROM node:14.5-slim as base
LABEL org.opencontainers.image.author=
"andre.telestp@gmail.com"
LABEL org.opencontainers.image.title="Image example"
LABEL org.opencontainers.image.licenses=MIT
LABEL com.andretelestp.nodeversion=$NODE_VERSION
```

A iniciativa Opencontainers tem um repositório no qual estabelece alguns padrões na hora de criar suas labels.

### General Layers:

Bem, chamo a parte abaixo de Camadas gerais, pois ela vai fazer parte das outras imagens que iremos criar a partir da **as base**:



```
FROM node:14.5-slim as base
```

Se nunca viu isso você deve está se perguntando, WTF!!? Mas vamos continuar.

Nas camadas gerais temos as seguinte camadas:



```
ENV NODE_ENV=production
EXPOSE 3333
ENV PORT 3333
WORKDIR /app
COPY package*.json ./
COPY yarn.lock ./
RUN yarn config list
RUN yarn install --frozen-lockfile && yarn cache clean --force
ENV PATH /app/node_modules/.bin:$PATH
ENV TINI_VERSION v0.18.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini /tini
RUN chmod +x /tini
ENTRYPOINT ["/tini", "--"]
CMD ["yarn", "start"]
```

Primeiro criamos uma variável de ambiente `NODE_ENV`, posteriormente expomos a porta do serviço que irá executar dentro do container, nesse caso a 3333 e uma variável de ambientes caso quisermos mudar em runtime. Posteriormente definimos nosso diretório de trabalho como sendo `/app`, ou seja, dentro do container será criado um diretório `app`.

```
ENV NODE_ENV=production  
EXPOSE 3333  
ENV PORT 3333  
WORKDIR /app
```

Antes de entramos nas camadas de **COPY**, gostaria de fizesse uma reflexão:

*Se separado é tudo junto, porque tudo junto é separado?*



Me desculpem, não pude resistir.

Agora é sério. Faz sentido criarmos uma imagem na qual sempre que fizermos um build, instalarmos os pacotes em nossa máquina e enviarmos para o container e posteriormente distribuir para os colequinhas? Outra pergunta. Se você usa linux e o colequinho usa windows, você acredita que vai funcionar na máquina dele? Bem, segundo os teóricos dos antigos astronautas afirmam que não!

Pois, para evitar isso ao invés de enviarmos os pacotes (node\_modules) para dentro da máquina(container), mandamos apenas package.json e yarn.lock ou package-

lock.json e lá dentro efetuarmos a instalação dos pacotes específicos da plataforma que vai executar esta aplicação. É isso que é feito nas duas linhas de COPY.

```
COPY package*.json ./
COPY yarn.lock ./
```

Nas linhas seguintes fazemos listagem das configurações do yarn para que quando o build estiver sendo feito o usuário tenha um feedback das configurações, e posteriormente fazemos a instalação dos pacotes e limpamos o cache. Na linha seguinte mapeamos a variável \$PATH para que todo binário instalado em /app/node\_modules/.bin fique disponível globalmente no container. Para quem não entende, isso basicamente é o seguinte, se um binário não está no \$PATH do seu sistema você tem que entrar diretamente na pasta onde ele se encontra e executar ou indicar o local e executar, veja:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

andre in knowledge-base-backend on master
> node_modules/.bin/eslint
```

Ao contrário com o ENV PATH /app/node\_modules/.bin:\$PATH você teria que fazer apenas, assim:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

andre in knowledge-base-backend on master
```

[Open in app](#)

[Sign up](#)

[Sign in](#)

Medium



Search



Nessa primeira parte da imagem só restou a parte do **TINI**.

```
ENV TINI_VERSION v0.18.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini /tini
RUN chmod +x /tini
ENTRYPOINT ["/tini", "--"]
CMD ["yarn", "start"]
```

Bem, não vejo uma forma melhor de entender o que essa linha faz a não ser lendo essa ISSUE respondida pelo próprio criador do projeto. Para quem gosta de entender o porquê das coisas já adianta. Vale a pena lê, por mais que esteja em inglês, vale a pena o esforço.

## Multi-Stage builds

Finalizamos a primeira parte da nossa imagem. Agora vamos falar de uma parte muito legal. Sabe aqueles arquivos de dockerfiles espalhados pelo projeto, Dockerfile.build, Dockerfile etc. Pois, é segundo a documentação oficial isso é chamado Builder Pattern e, ao mesmo tempo ela diz o seguinte:

---

*Maintaining two Dockerfiles is not ideal.*

---

Então, se não é ideal então vamos usar multi-stages. Basicamente essa feature faz uma cópia do estágio anterior para outra, fazendo com que algumas dependências na qual é utilizada na imagem de produção, mas não na de desenvolvimento e isso consequentemente reduz o tamanho de suas imagens que, é uma boa prática também 😁. Show me the code:

### Layer dev

```
FROM base as dev
ENV NODE_ENV=development
RUN apt-get update -qq && apt-get install -qy \
    ca-certificates \
    bzip2 \
    curl \
    libfontconfig \
    --no-install-recommends
RUN yarn config list
RUN yarn install \
    && yarn cache clean --force
USER node
CMD ["yarn", "dev"]
```

Lembra da nossa imagem criada lá no início onde referenciamos as base, pois é, quando fizemos isso, foi criado um aliás para nos possibilitar referenciar em qualquer lugar do dockerfile mais tarde. Que nesse caso usamos agora para criar nossa imagem dev.

Conforme mostrado na imagem acima definimos `NODE_ENV=development`, instalamos apenas os pacotes no qual iremos usar em nosso ambiente de desenvolvimento, instalamos os pacotes do projeto. Agora um ponto importante, **USER node**.

Oque acontece é o seguinte, essa imagem por padrão já vem o esse usuário node, mas caso você opte por criar sua própria imagem eu diria que é um pré requisito criar um usuário sem privilégios. O motivo é que por padrão o docker executa containers com usuário root e quando esse namespace é mapeado para o container em execução significando que ele potencialmente tem acesso root ao host. Tenso né!

Outra dor de cabeça que nós usuários Linux temos é as permissões. Tenta omitir a linha `USER node` gerar uma imagem e altere algum arquivo, a não ser que você mande aquele `chmod 777` com certeza você não vai conseguir alterar nada.

Por fim, temos o comando que será executado quando o container estiver rodando `CMD ['yarn','start']`.

## Layer test

```
FROM dev as test
ENV NODE_ENV=test
USER node
COPY --chown=node:node . .
ARG MICROSCANNER_TOKEN
ADD https://get.aquasec.com/microscanner /
USER root
RUN chmod +x /microscanner
RUN /microscanner $MICROSCANNER_TOKEN --continue-on-failure
RUN yarn lint 66\
  yarn test
```

Mais uma vez usando o reaproveitamento de camadas temos a layer test que irá pegar as camadas de dev, definir a variável `NODE_ENV=test` mudar usuário para `node`, copiar todo o diretório no qual contém os arquivos do seu projeto. Adicionalmente outra boa prática é verificar as vulnerabilidades da sua imagem, para isso podemos usar o `microscanner`. `Microscanner` é um projeto muito bacana, você apenas precisa entrar no repositório seguir as orientações para conseguir um token no qual deve ser passado por argumento `MICROSCANNER_TOKEN` enquanto essa imagem estiver sendo construída, por fim, executamos o `yarn lint` e `yarn test`. Esse cenário é ideal para ambiente de CI, no qual você cria uma imagem para executar os teste e ela só será criada se tudo ocorrer como esperado.

## **docker-compose**

Praticamente na reta final só precisamos construir nosso arquivo `docker-compose.yml` para orquestrar nossos serviços. Na imagem abaixo temos um exemplo:



```
version: '2.4'
volumes:
  pgdatavoting:
services:
  app:
    build:
      context: .
      target: dev
    depends_on:
      database:
        condition: service_healthy
    environment:
      APP_URL: ${APP_URL}
      APP_SECRET: ${APP_SECRET}
      DB_HOST: ${DB_HOST}
      DB_USER: ${DB_USER}
      DB_PASS: ${DB_PASS}
      DB_NAME: ${DB_NAME}
      NODE_ENV: ${NODE_ENV}
    ports:
      - 3333:3333
      - 9229:9229
    volumes:
      - ./:/app
  database:
    image: postgres:12.3
    restart: on-failure
    healthcheck:
      test: pg_isready -h 127.0.0.1
      interval: 10s
      timeout: 5s
      retries: 5
    environment:
      POSTGRES_PASSWORD: ${DB_PASS}
    ports:
      - '5432:5432'
    volumes:
      - pgdatavoting:/var/lib/postgresql/data
```

Se você já manja dos paranâues esse arquivo não é muita novidade, geralmente a maioria dos cursos de docker ensinam uma estrutura semelhante a essa, mas gostaria de destacar apenas alguns pontos importantes que acho importante o entendimento.

### 1º Build

```
app:
  build:
    context: .
    target: dev
```

Aqui é onde tiramos proveito das nossas layers, na imagem acima estamos falando para que quando executarmos docker-compose up que seja criado nossa imagem

que está no diretório atual (context: . ) com **target: dev**. Top né!

## 2º Dependência entre os serviços.

Imagine que sua aplicação suba primeiro que seu banco, de cara já iria dar ruim, concorda?

*Ah, mas tempos a diretiva `depends_on` que o `docker-compose` nos fornece!*

Será?

Em um dado exemplo na documentação do docker você irá encontrar o seguinte:

*`depends_on` does not wait for `db` and `redis` to be “ready” before starting `web` - only until they have been started.*

Quer dizer, o `depends_on` irá aguardar o status do container esteja em execução e não o serviço que esteja executando dentro do container esteja pronto para uso. Isso é uma responsabilidade de sua aplicação fazer essa checagem. Para resolver isso temos a seguinte instrução:

```
depends_on:
  database:
    condition: service_healthy
```

Oque estamos dizendo ao `docker-compose` e que dependemos do serviço `database` esteja saudável para que nossa aplicação inicie.

```
healthcheck:
  test: pg_isready -h 127.0.0.1
  interval: 10s
  timeout: 5s
  retries: 5
```

Em nosso serviço criamos a entrada `healthcheck` fazemos um test com o comando `pg_isready`, pois estamos usando uma imagem do postgres. Seguindo no arquivo dizemos que em um intervalo de 10s, tente 5x e aguarde 5s em cada teste.

Isso é muito massa! Sua aplicação nunca irá subir sem que o banco esteja funcionando. Detalhe é que podemos fazer isso com redis, mysql ou qualquer serviço.

Bem acho que é isso!

Já sei, já sei... Você iria perguntar sobre como debugar sua aplicação NodeJS usando docker, certo?

Acho que tenho uma coisinha aqui....

Se já tiver a pasta .vscode já criada apenas coloque o código abaixo no arquivo **launch.json**, caso contrário crie e faça o mesmo processo.

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "name": "Debug Docker",
      "request": "attach",
      "protocol": "inspector",
      "address": "127.0.0.1",
      "port": 9229,
      "restart": true,
      "remoteRoot": "/app",
      "localRoot": "${workspaceFolder}",
      "skipFiles": [
        "<node_internals>/**/*.js",
        "${workspaceFolder}/node_modules/**/*.js",
      ],
    }
  ]
}
```

Depois disso. O pulo do gato:

```
"scripts": {
  "build": "tsc",
  "dev:server": "ts-node-dev --inspect=0.0.0.0 --transpile-only --ignore-watch node_modules src/server.ts",
  "typeorm": "ts-node-dev ./node_modules/typeorm/cli.js"
},
```

Na imagem acima estou usando ts-node-dev por causa do typescript, mas ponto aqui é o `--inspect=0.0.0.0`. Esse argumento fará com que tudo aconteça. Inicie o debug e seja feliz!

## Conclusão:

Bem espero que esse artigo te ajude como pontapé inicial para seu projetos NodeJS que sempre quis colocar o docker, mas não sabia por onde começar. Aqui você pode vê um repositório de exemplo caso queira seguir de exemplo.

### apteles/knowledge-base-backend

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com



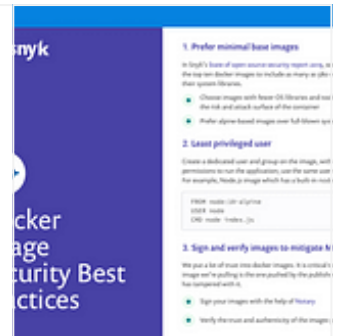
Qualquer dúvida ou sugestão é só ligar nós! Tchau BrigaduU.

## Referências e Links:

### 10 Docker Security Best Practices | Docker Best Practices

In this installment of our cheat sheets, we'd like to focus on Docker security and discuss docker security best...

snyk.io



### Compose file version 2 reference

These topics describe version 2 of the Compose file format. There are several versions of the Compose file format - 1...

docs.docker.com



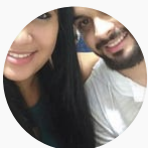
Docker

Docker Compose

Nodejs

Containers

JavaScript



Follow



## Written by André Teles

37 Followers

I'm a simple student, looking for knowlogment for become myself a great programmer.

### More from André Teles



André Teles in Training Center

## PHPUnit e Cobertura de Código — Uma Breve introdução a testes em PHP.

Atualmente estou querendo adotar nas minhas rotinas de estudos, o seguinte pensamento:

Sep 11, 2017 🖱 262 💬 2



### Introdução à Javascript Funcional

$\Lambda\lambda$

```
compose(  
  learning(),  
  every(),  
  day()  
)
```



André Teles

## Introdução à programação funcional com Javascript

Motivação para escrever esse artigo

Aug 25, 2020 4 1

[See all from André Teles](#)

## Recommended from Medium

### Amazon.com

#### Software Development Engineer

Seattle, WA

Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by \$25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

### Projects

#### NinjaPrep.io (React)

- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

#### HeatMap (JavaScript)

- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay



Alexander Nguyen in Level Up Coding

## The resume that got a software engineer a \$300,000 job at Google.

1-page. Well-formatted.




Jun 1 11.6K 154







 Mauro Di Pietro in Towards Data Science

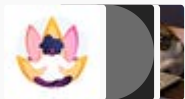
## GenAI with Python: RAG with LLM (Complete Tutorial)

Build your own ChatGPT with multimodal data and run it on your laptop without GPU

★ Jun 28 🖱 557 💬 10



### Lists



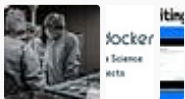
#### Stories to Help You Grow as a Software Developer

19 stories · 1188 saves



#### General Coding Knowledge

20 stories · 1366 saves



#### Coding & Development

11 stories · 692 saves




#### data science and AI

40 stories · 197 saves



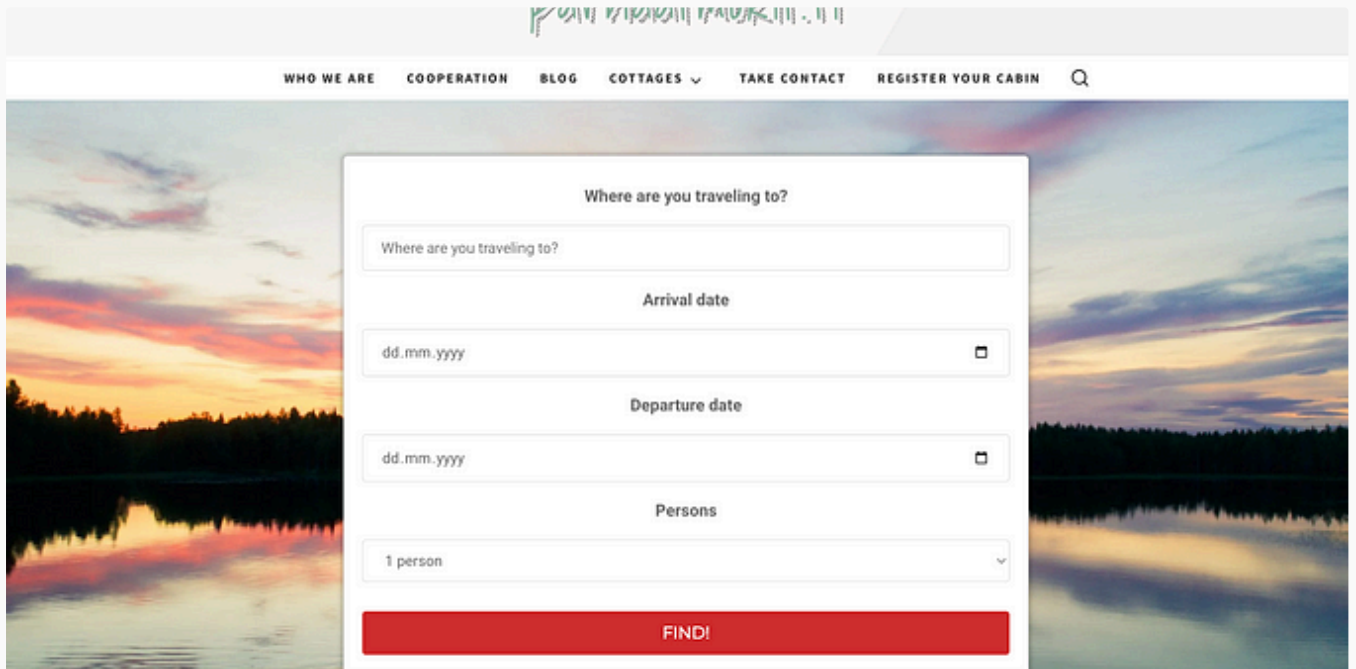


 Chinwendu E. Onyewuchi

## How to Dockerize Your TypeScript Application With Multi-Stage Build: A Step-By-Step Guide

Introduction

Feb 15  67

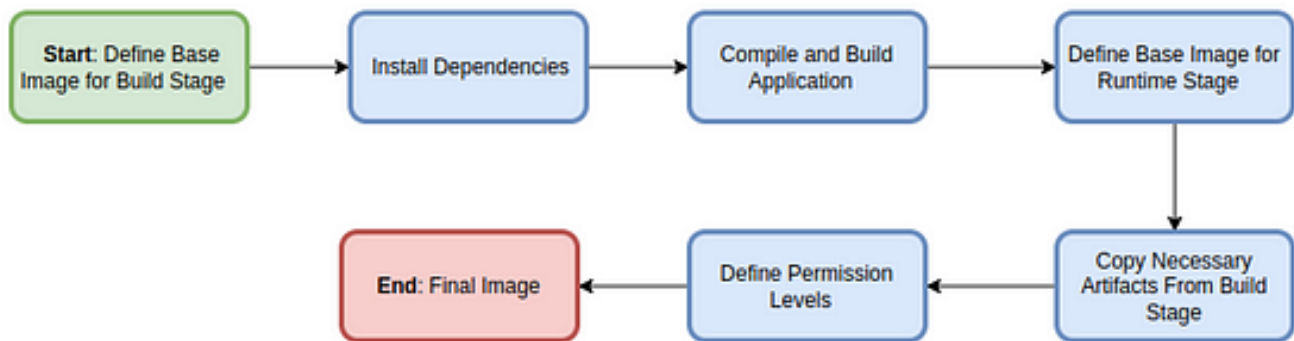


 Artturi Jalli

## I Built an App in 6 Hours that Makes \$1,500/Mo

Copy my strategy!

★ Jan 23 🖱 19.6K 💬 210



Ketan Gangal

## Docker 201: Multi-Stage Builds for Production

Introduction

Mar 5 🖱 53



Afan Khan in JavaScript in Plain English

## Microsoft is ditching React

Here's why Microsoft considers React a mistake for Edge.



Jun 6



2.4K



55



See more recommendations