



Reactの開発環境をdevcontainerで作る！

SIOS Tech lab | 龍ちゃん





アジェンダ

- ご挨拶
- 前提条件
- 使用するdocker-compose コマンドの確認
- 開発環境構築
- まとめ



龍ちゃん
@RyuReina_Tech

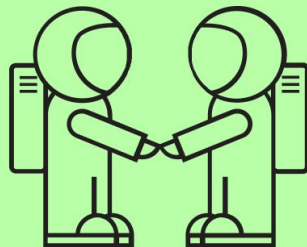


3年目のピヨピヨエンジニア

フロントエンド



デザイン業務



外部発信





今日の目標

Reactの環境をdevcontainerで作成する

前提条件



前提条件

- 開発環境がVS Codeの人である*
- docker composeが使用しているPCで実行できる
- フロントエンド
 - Viteを使った環境構築
 - npmかyarnで詰まったことがあればなおよい
- 萎えない心

*Dev Containers tutorial (<https://code.visualstudio.com/docs/devcontainers/tutorial>)

使用するコマンド確認



使用するコマンドの確認

`docker compose build`

サービスの構築

`docker compose run`

一度限りのコマンド実行

`docker compose up`

コンテナの作成と開始

開発環境構築



開発環境構築：全体の流れ

- 作成する環境の整理
- 環境作成
 - Reactプロジェクトの作成
 - 効率的なプロジェクトに改変
 - devcontainerでup
 - デバック機能の追加



作成する環境の整理：要件整理

Nodeバージョン	20.10.0
パッケージ管理ツール	npm
ビルドツール	Vite
プロジェクトタイプ	React+Typescript



作成する環境の整理：目指すディレクトリ構造

```
.
├── .devcontainer
│   ├── Dockerfile
│   └── devcontainer.json           // devcontainer設定ファイル
├── .dockerignore                 // ビルド時に無視するファイルパス
├── docker-compose.yml
└── frontend                     // フロントエンドプロジェクト
```



作成する環境の整理：環境構築の流れ

1. Reactプロジェクトの作成
2. 改善①：Volume Trickを使用してnode modulesをコンテナ内に収める
3. 改善②：Docker build時のキャッシュを活用
4. 一旦起動確認
5. devcontainerでコンテナを起動する
6. VS Codeでデバッグできる環境を整備する

作成する環境の整理：環境構築の流れ



ソースコード見にくいので、GitHubに上げています



全体の流れ

1. Reactプロジェクトの作成
2. 改善①：Volume Trickを使用してnode_modulesをコンテナ内に収める
3. 改善②：Docker build時のキャッシュを活用
4. 一旦起動確認
5. devcontainerでコンテナを起動する
6. VS Codeでデバッグできる環境を整備する



Reactプロジェクトの作成

- フロントプロジェクト作成用の一時的なNodeコンテナ作成
- Viteを使用してReactプロジェクトを作成する



Reactプロジェクトの作成

まずは以下のディレクトリ構造を作成しましょう

```
.
├── .devcontainer
│   ├── Dockerfile
│   └── devcontainer.json
├── .dockerignore
└── docker-compose.yml
```



Reactプロジェクトの作成

Dockerfile

```
ARG NODE_VER
FROM node:${NODE_VER}
RUN npm update -g npm

WORKDIR /home/node/app
USER node
```

.dockerignore

```
*/dist
*/node_modules
```



Reactプロジェクトの作成

```
version: "3.7"
services:
  react:
    build:
      context: .
      dockerfile: ../devcontainer/Dockerfile
      args:
        - NODE_VER=20.10.0
    tty: true
    volumes:
      - type: bind
        source: .
        target: /home/node/app
```

docker-compose.yml

1. Dockerfileを指定
2. NODE_VERを指定
3. ルートディレクトリをバインドマウント



Reactプロジェクトの作成

ファイルを作成したら以下のコマンドを実行

```
docker compose build
```

```
docker compose run --rm react npm create vite
```



Reactプロジェクトの作成

Viteのプロジェクトは以下の構成

ディレクトリ名	frontend
プロジェクトタイプ	React
JS or TS	Typescript



Reactプロジェクトの作成

```
.
├── .devcontainer
│   └── Dockerfile
├── .dockerignore
├── docker-compose.yml
└── frontend                                // viteが作ってくれる
```

上記のディレクトリ構造になっていれば成功ですう～



Reactプロジェクトの作成

frontend/package.jsonのファイルを変更

```
"scripts": {  
  "dev": "vite --host"  
},
```

参考：Vite ドキュメント (<https://ja.vitejs.dev/config/server-options>)



全体の流れ

1. Reactプロジェクトの作成
2. 改善①：Volume Trickを使用してnode_modulesをコンテナ内に収める
3. 改善②：Docker build時のキャッシュを活用
4. 一旦起動確認
5. devcontainerでコンテナを起動する
6. VS Codeでデバッグできる環境を整備する

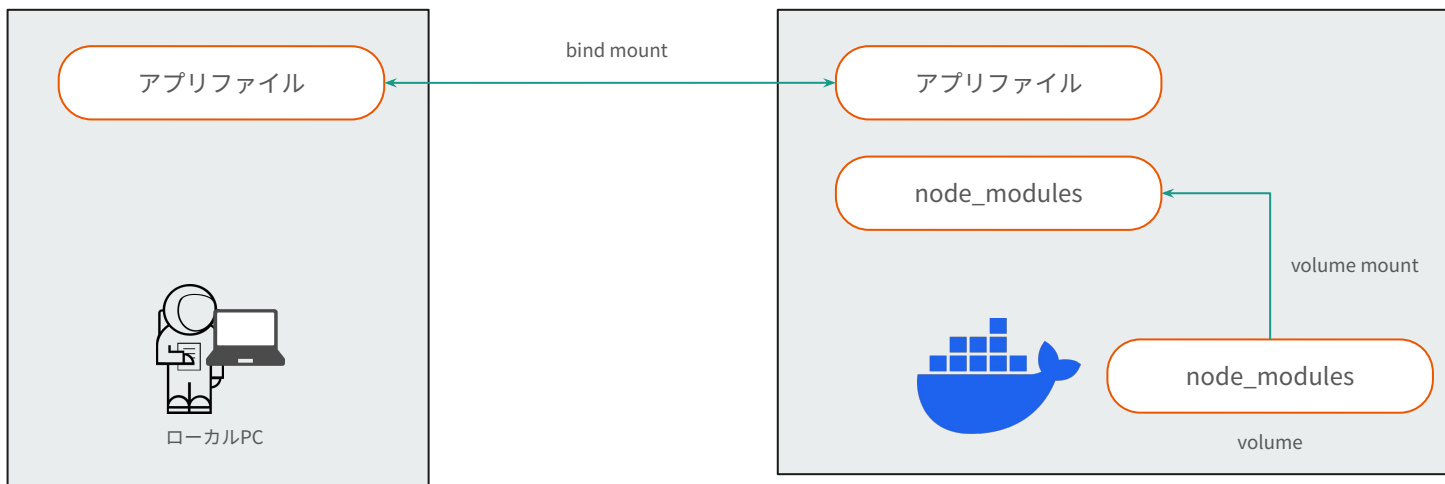


改善①：Volume Trickを使用してコンテナ内に収める

- Volume Trickを理解する（Volumeマウント）
- `node_modules`をvolumeマウントでコンテナ内に収める

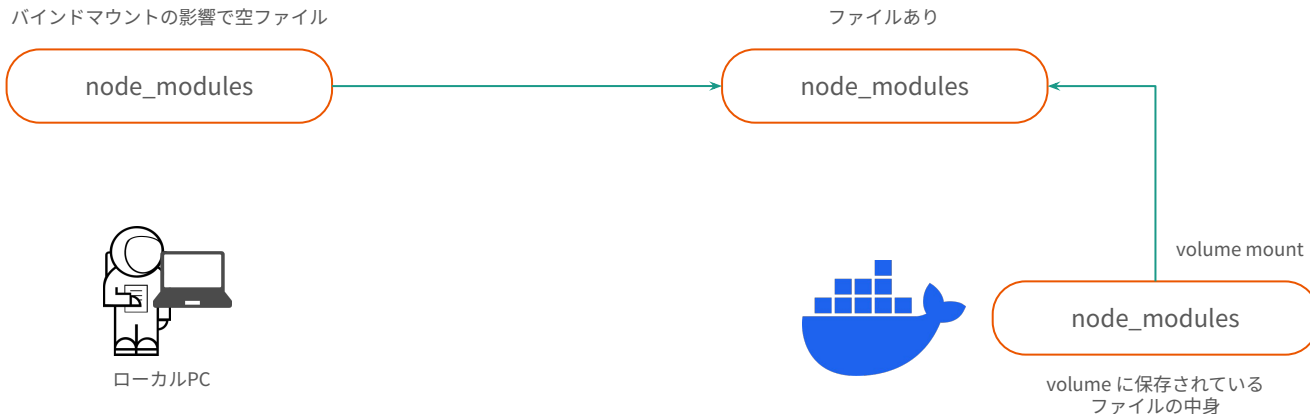
改善①：Volume Trickを使用してコンテナ内に収める

Volume Trickとは？→Volume mountでコンテナ内に情報を収める



改善①：Volume Trickを使用してコンテナ内に収める

Volume Trickとは？：今回の例



改善①：Volume Trickを使用してコンテナ内に収める

Dockerfileを変更

```
ARG NODE_VER
FROM node:${NODE_VER}
RUN npm update -g npm

WORKDIR /home/node/app

RUN chown node:node .
RUN mkdir node_modules && chown node:node node_modules

USER node
```

1. コンテナ内でnode_modulesを作成
2. 作業ディレクトリの権限をnodeユーザーにする



改善①：Volume Trickを使用してコンテナ内に収める

docker-compose.ymlを変更

1. バインドマウントで./frontendをマウントする
2. Volume Trick
 - a. node_modulesボリュームを作成する
 - b. ボリュームマウントで、コンテナ内のnode_modulesとボリュームをマウントする

```
version: "3.7"
services:
  react:
    build:
      context: .
      dockerfile: ../devcontainer/Dockerfile
    args:
      - NODE_VER=20.10.0
    tty: true
    volumes:
      - type: bind
        source: ./frontend
        target: /home/node/app
      - type: volume
        source: node_modules
        target: /home/node/app/node_modules
    ports:
      - 5173:5173
    volumes:
      node_modules:
```

バインドマウント

volume trick



全体の流れ

1. Reactプロジェクトの作成
2. 改善①：Volume Trickを使用してnode_modulesをコンテナ内に収める
3. 改善②：Docker build時のキャッシュを活用
4. 一旦起動確認
5. devcontainerでコンテナを起動する
6. VS Codeでデバッグできる環境を整備する



改善②：Docker build時のキャッシュを活用

- レイヤーキャッシュの特性を理解する
- レイヤーキャッシュを活用して適切に分割する

改善②：Docker build時のキャッシュを活用

```
USER node
COPY --chown=node:node ./frontend ./
RUN npm install
CMD ["npm", "run", "dev"]
```

どっちも動くよ

どっちが良いでしょうか？

```
COPY --chown=node:node ./package.json ./
USER node
RUN npm install
COPY --chown=node:node ./frontend ./
CMD ["npm", "run", "dev"]
```

改善②：Docker build時のキャッシュを活用

```
FROM golang:1.20-alpine
```

```
WORKDIR /src
```

```
COPY . .
```

```
RUN go mod download
```

```
RUN go build -o /bin/server ./cmd/client
```

```
RUN go build -o /bin/server ./cmd/server
```

```
ENTRYPOINT [ "/bin/server" ]
```

上から順に実行されて行く

一行ごと（レイヤ）にキャッシュ

ビルド時にキャッシュがあれば使用される

つまり....

分割できるところは分割や！



改善②：Docker build時のキャッシュを活用

Dockerfileを変更

1. 依存パッケージインストールに必要なファイルをコピー
2. 依存パッケージのインストール
3. フロントエンドプロジェクトファイルをコピー
4. 開発環境モードでビルド

分割

```
ARG NODE_VER  
FROM node:${NODE_VER}  
RUN npm update -g npm
```

```
WORKDIR /home/node/app
```

```
RUN chown node:node .  
RUN mkdir node_modules && chown node:node node_modules
```

```
COPY --chown=node:node ./frontend/package.json ./frontend/package-lock.json ./  
USER node
```

```
RUN npm install
```

```
COPY --chown=node:node ./frontend ./
```

```
CMD ["npm", "run", "dev"]
```

```
ARG NODE_VER
FROM node:${NODE_VER}
RUN npm update -g npm

WORKDIR /home/node/app

RUN chown node:node .
RUN mkdir node_modules && chown node:node node_modules

COPY --chown=node:node ./frontend/package.json ./frontend/package-lock.json ./
USER node
RUN npm install

COPY --chown=node:node ./frontend ./

CMD ["npm", "run", "dev"]
```



全体の流れ

1. Reactプロジェクトの作成
2. 改善①：Volume Trickを使用してnode_modulesをコンテナ内に収める
3. 改善②：Docker build時のキャッシュを活用
4. 一旦起動確認
5. devcontainerでコンテナを起動する
6. VS Codeでデバッグできる環境を整備する



起動確認

- Dockerfileとdocker-compose.ymlの確認
- ここまで上手くいっているかのチェックです！


```
version: "3.7"
services:
  react:
    build:
      context: .
      dockerfile: ../devcontainer/Dockerfile
    args:
      - NODE_VER=20.10.0
    tty: true
    volumes:
      - type: bind
        source: ./frontend
        target: /home/node/app
      - type: volume
        source: node_modules
        target: /home/node/app/node_modules
    ports:
      - 5173:5173

volumes:
  node_modules:
```

docker-compose.yml

```
ARG NODE_VER
FROM node:${NODE_VER}
RUN npm update -g npm

WORKDIR /home/node/app

RUN chown node:node .
RUN mkdir node_modules && chown node:node node_modules

COPY --chown=node:node ./frontend/package.json./frontend/package-lock.json./
USER node
RUN npm install

COPY --chown=node:node ./frontend ./

CMD ["npm", "run", "dev"]
```

../devcontainer/Dockerfile



起動確認

```
docker compose build
```

```
docker compose run --rm react
```

```
VITE v5.2.7 ready in 165 ms
```

```
→ Local:   http://localhost:5173/
```

```
→ Network: http://
```

```
→ press h + enter to show help
```

アクセスはできなくてOK

左の画面でLocalとNetwork両方であれば

できてるでしょう！

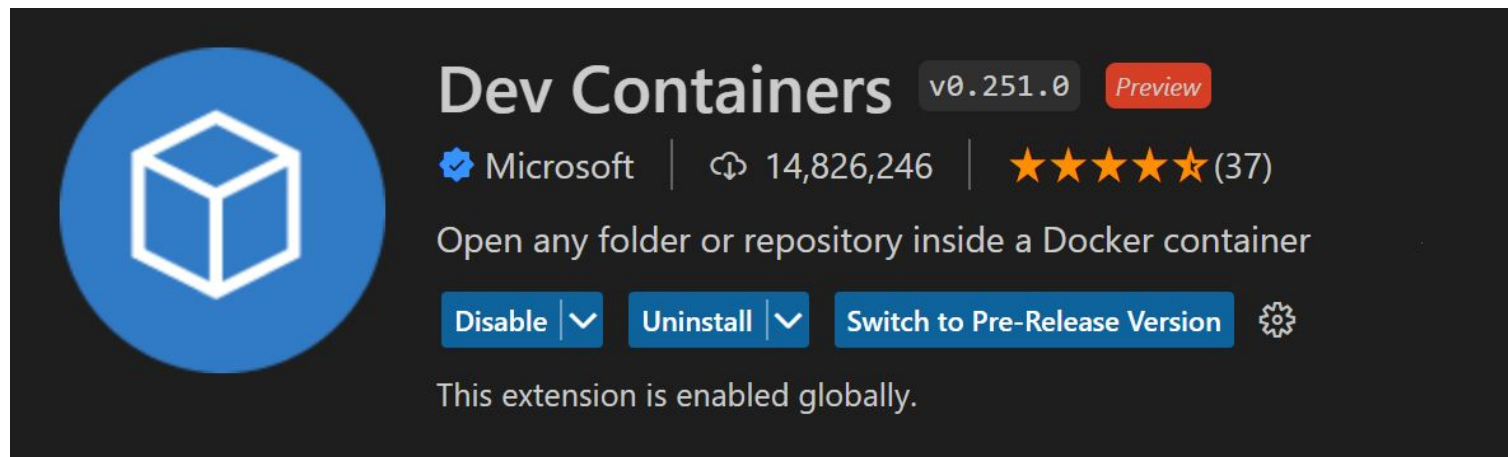


全体の流れ

1. Reactプロジェクトの作成
2. 改善①：Volume Trickを使用してnode_modulesをコンテナ内に収める
3. 改善②：Docker build時のキャッシュを活用
4. 一旦起動確認
5. devcontainerでコンテナを起動する
6. VS Codeでデバッグできる環境を整備する

devcontainerでコンテナを起動する

VS Codeの拡張機能が必要です。



The screenshot shows the Dev Containers extension page in VS Code. On the left is the extension's icon, a blue circle containing a white 3D cube. To the right of the icon, the text "Dev Containers" is displayed in a large, bold font. Below this, the publisher "Microsoft" is listed with a checkmark icon, followed by the download count "14,826,246" and a star rating of five stars with "(37)" reviews. A description reads "Open any folder or repository inside a Docker container". Below the description are three buttons: "Disable" with a dropdown arrow, "Uninstall" with a dropdown arrow, and "Switch to Pre-Release Version". To the right of these buttons is a gear icon for settings. Above the "Switch to Pre-Release Version" button, the version "v0.251.0" is shown in a grey box, and a red "Preview" badge is visible. At the bottom, a status message states "This extension is enabled globally."

参考：Dev Containers (<vscode:extension/ms-vscode-remote.remote-containers>)



devcontainerでコンテナを起動する

以下のファイルが必要です

```
.
├── .devcontainer
│   ├── Dockerfile
│   └── devcontainer.json
├── .dockerignore
├── docker-compose.yml
└── frontend
```

←これらのファイルがあれば起動
できます！



devcontainerでコンテナを起動する

最小構成

```
{  
  "name": "project-dev",  
  "dockerComposeFile": ["../docker-compose.yml"],  
  "service": "react",  
  "workspaceFolder": "/home/node/app",  
  "forwardPorts": [5173],  
  "remoteUser": "node"  
}
```

// VS Codeの上に表示される名前
// docker-composeファイルパス
// docker-compose内で起動したいサービス
// 最初に関くワークスペース
// サービスが動いているポート
// 実行ユーザー



devcontainerでコンテナを起動する

オレオレ設定のための知識

```
"customizations": {  
  "vscode": {  
    "extensions": [  
      "xxxxxxxxxxxxx",  
      "yyyyyyyyyyyyy"  
    ],  
    "settings": {  
      // コンテナ内に設定したい設定  
    }  
  }  
}
```

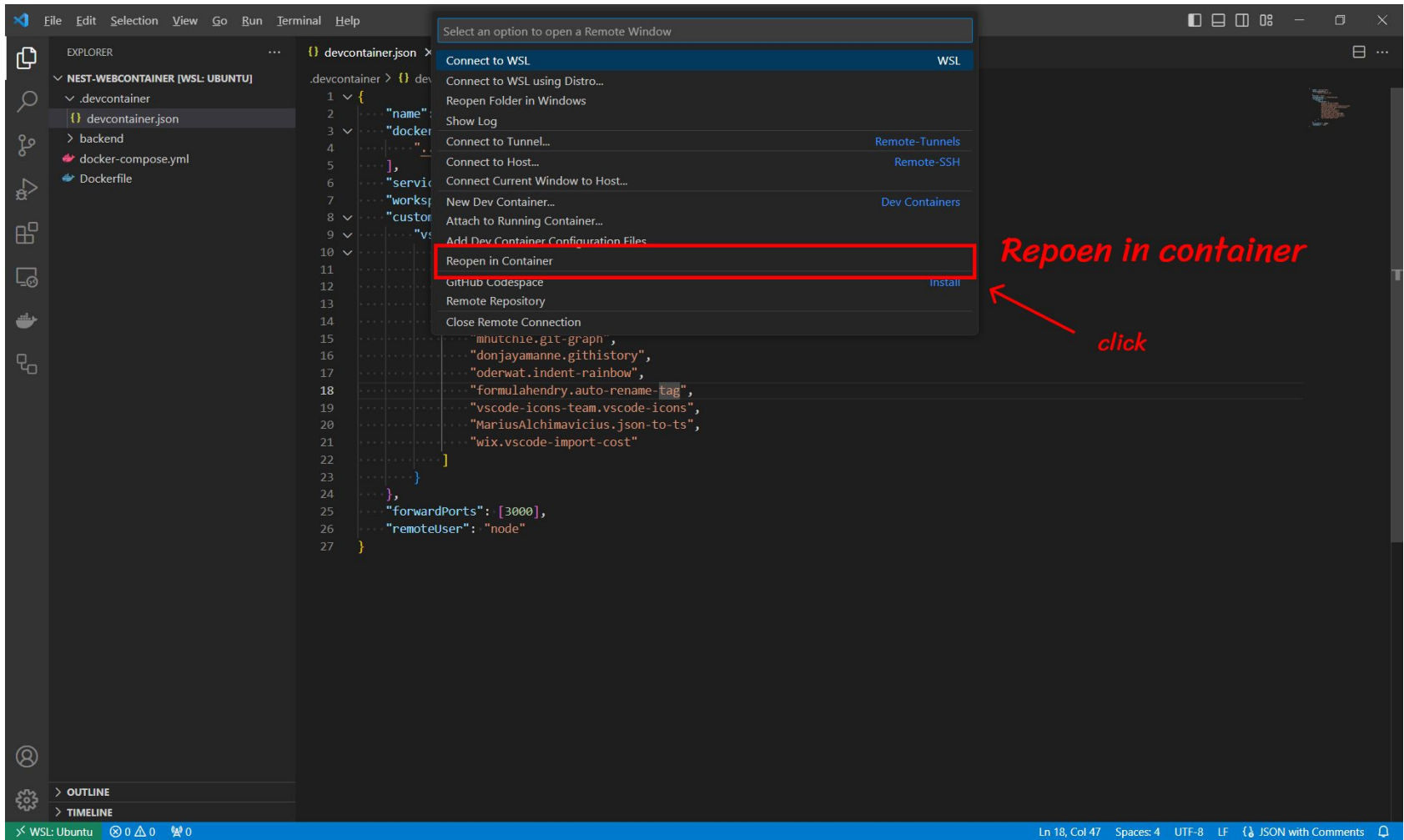
○ VSCodeの拡張機能
起動時にプリインストールする拡張機能

○ VSCodeの設定
起動時にコンテナ内のVS Codeに対して設定
することができる
プロジェクトで使用するフォーマッターとか
の設定を含めることができる

devcontainerでコンテナを起動する

devcontainerの起動方法





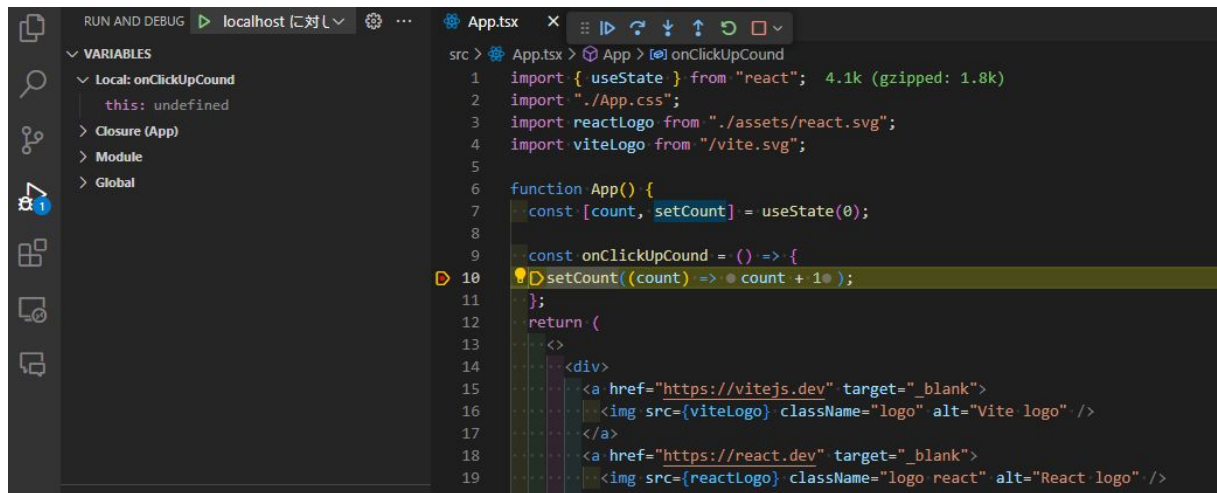


全体の流れ

1. Reactプロジェクトの作成
2. 改善①：Volume Trickを使用してnode_modulesをコンテナ内に収める
3. 改善②：Docker build時のキャッシュを活用
4. 一旦起動確認
5. devcontainerでコンテナを起動する
6. VS Codeでデバッグできる環境を整備する

デバック機能追加

デバック機能とは？



```
src > App.tsx > App > onClickUpCount
1 import { useState } from "react"; 4.1k (gzipped: 1.8k)
2 import "../App.css";
3 import reactLogo from "../assets/react.svg";
4 import viteLogo from "../vite.svg";
5
6 function App() {
7   const [count, setCount] = useState(0);
8
9   const onClickUpCount = () => {
10    setCount((count) => count + 1);
11  };
12  return (
13    <>
14    <div>
15      <a href="https://vitejs.dev" target="_blank">
16        <img src={viteLogo} className="logo" alt="Vite logo" />
17      </a>
18      <a href="https://react.dev" target="_blank">
19        <img src={reactLogo} className="logo react" alt="React logo" />
20      </a>
21    </div>
22  );
23 }
```

ちょっと止めて確認とか
(ブレイクポイント)

現在の値を確認とか

ステップ実行とか

いろいろできちゃう



デバック機能追加

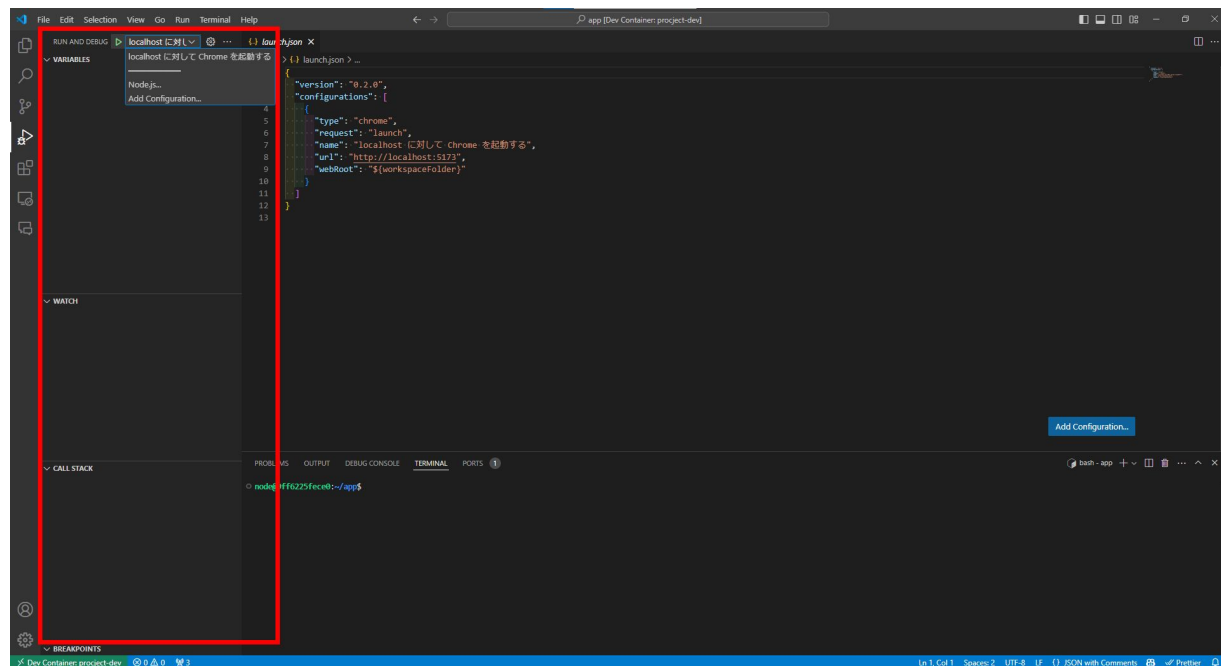
VS Codeで開いた際に、ルートディレクトリにある必要がある

```
./frontend
└─ .vscode
   └─ launch.json
```

ワークスペースに.vscodeファイルがあれば認識される

launch.jsonを記載することでデバック実行が可能になる

デバック機能追加



赤枠の表記なら

OK

—

まとめ

まとめ

こちらのリポジトリにまとまっています



これがdevcontianerのいいところ