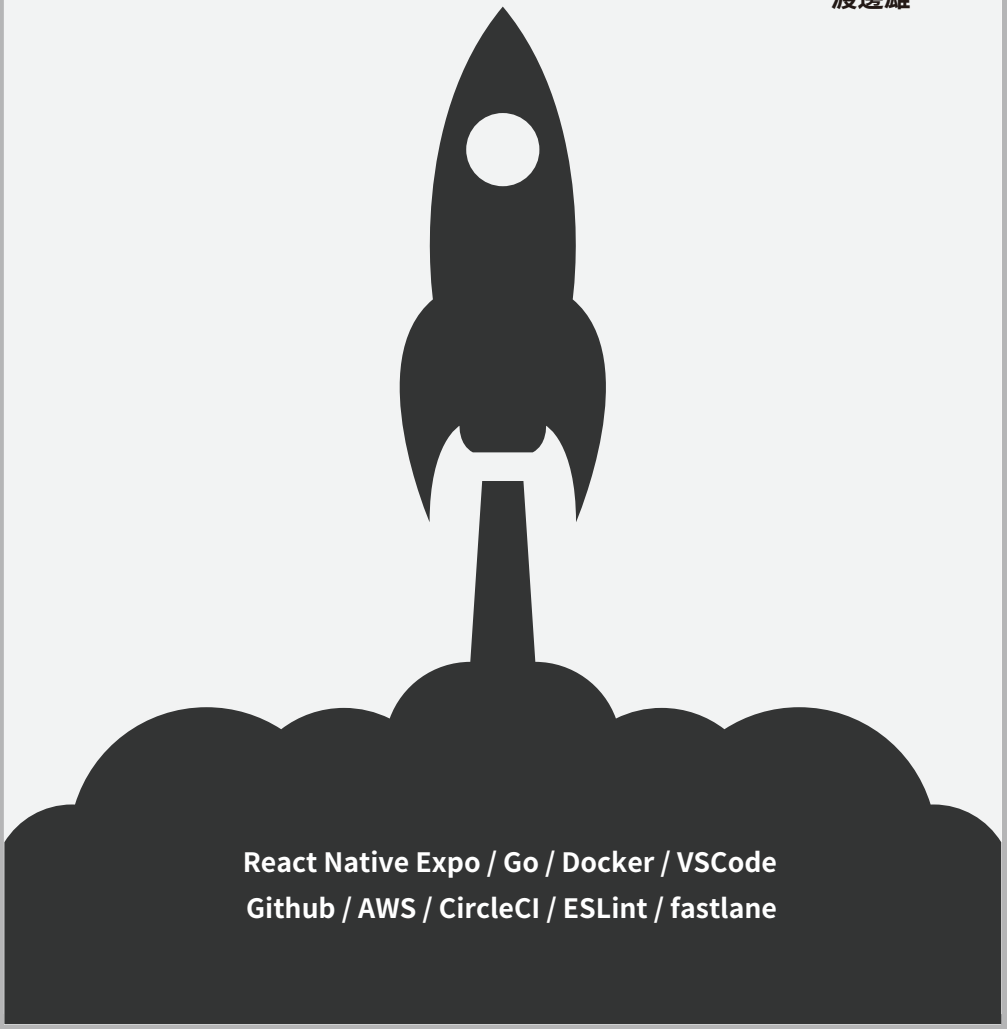
#1人チーム本

ノウハウなしでも1人でなんとかなる
React Native Expoを使ったアプリの
開発環境構築 / ビルド / デプロイ / 運用

渡邊雄



React Native Expo / Go / Docker / VSCode
Github / AWS / CircleCI / ESLint / fastlane

＃１人チーム本

渡邊雄　著

2019-04-14 版　　発行

はじめに

昨今、アプリ開発や Web 開発において様々な技術が出てきています。React Native での開発も最近では一般的となってきており、様々な知見が共有されるようになってきました。ただそれでも開発をするにあたってこれはどうしたらいいのだろうか？ ということも多くあります。また React Native では iOS・Android 両方を JavaScript で簡単に開発できるということもあり、1 人で開発をすることも珍しくないです。さらにスタートアップや規模の少ないベンチャーおよび新規事業開発の部署などではエンジニア 1 人でサーバサイドも兼任しないといけないということも想定されます。

本書では React Native での開発をする上で簡単に開発運用を行うには？ということについてを書きたいと思います。アプリ開発という面では React Native のみで完結することも多いですが、API もあるようなアプリを 1 人で開発するにはどうしたらよいのだろうか。それらも踏まえてアプリ開発運用を自分はどのようにやっているのかを紹介させていただけたらと思います。使うエディタやこういったプラグインを最低限入れておいた方がいい、どうやってプロジェクトを管理したり CI を回したらいいのかなどになります。

なおサーバサイドや React Native でのコード例を載せることを主とするわけではないのでご了承ください。

ソースコードについて

`https://github.com/watanabeyu/1team`

yml や package.json など設定ファイルについては本書で記載をしています。これらは折り返しなどがつき若干見辛いと思いますので、もしよろしければ GitHub からご確認ください。

対象読者について

本書では 1 人でサーバ含むアプリ開発をする上でどうやったらいいのか？という疑問について答えるような内容となっております。特に下記のような方にとってためになるような本になればよいなと思って書きました。

- 今までずっと 1 人でやっていたけどどうしたら効率よくできるのか？
- スタートアップや新規事業開発をすることになったけどエンジニア自分 1 人でサービスを作らないといけない
- React Native での開発を最速で行うにはどうしたらいいの？
- GitHub や CircleCI を組みわせるとどういうことができるの？
- 人が新しく入って来てもスムーズに開発が行えるような環境を作りたい

動作環境について

- OS: macOS High Sierra 10.13.6
- Node: 10.7.0
- npm: 6.2.0
- Watchman: 4.7.0

- Xcode: 10.1/10B61
- Android Studio: 2.3 AI-162.4069837
- expo-cli: 2.13.0

意見と質問

誤植や不正確な表現にお気づきの方は、著者 Twitter までご連絡ください。

https://twitter.com/hmktsu

またハッシュタグ (#1 人チーム本) をつけてツイートしていただけるとリプライもしやすいですし、他の人たちも情報についてを検索できるようになるので、みなさんで一緒に本を作っていくような形になるとよいなぁと思っています。

謝辞

Twitter や勉強会などリアルでも色々と親交がある皆様ありがとう

目次

第 1 章

本書で紹介する内容について

著者は会社の規模的に 1 人で普段から開発をしており、直近ではアプリを React Native Expo で、また Go で API の開発を行いました。その際に 1 人でも効率よく開発運用をでき、さらに他の人が参加してもスムーズになるような環境を構築したので本書で紹介したいと思い執筆することにしました。もちろん本書で紹介する内容がベストとは限らないのでご了承ください。

そこで本章ではどうしてそれらの技術を選定したのかを中心にお話します。

1.1 元となるアプリについて

著者は株式会社 g&h(`https://gandh.jp`) という音楽やダンスなどストリート系のサービスを提供する会社の CTO です。Weddy Weddy(`https://weddyweddy.net`) というダンス動画に特化した SNS アプリをリリースしています。

1.1.1 技術スタック

Weddy Weddy のサービス全体で使われている技術は下記のようになっております。なお 2 章から紹介する内容についてはこれら全てに触れているわけではないのでご了承ください。

- アプリ：React Native Expo で JavaScript のみで構築
- API：AWS の Elastic Beanstalk を用いることによって簡単にロードバランシングをし Go で記述
- データベース：AWS の Aurora という MySQL 互換 DB を使い、さらに ElastiCache の Redis を使用
- ファイルサーバ：S3 を Cloud Front を通して使用
- Web サイト：AWS の Elastic Beanstalk で Go をバックエンドにフロントを React SPA で作成
- プッシュ通知配信サーバ：AWS の ElasticBeanstalk で Go + Redis を使って構築
- ソースコード管理：簡易的な GitHub flow で GitHub を運用
- CI/CD：CircleCI を develop および master マージ時に回す

1.2 技術選定の理由について

1.2.1 アプリ：**React Native Expo**

どうして React Native なのかということについては、はじめにでも簡単に触れましたが iOS および Android 両方の開発を同時に進めることができるからです。ではその上でどうして Expo なのかということについては、2 章に Expo についてという項目で書いておりますのでそちらをご覧になってください。

1.2.2 API：Go

どうして Go で API を組むのかということについては、もちろん今流行りの言語という理由もありますが、静的型付けだったりコンパイル言語というところが選定の理由となりました。また Go を書いていくとわかるのですが、誰が書いても書き方が似てくるという特徴があります。1 人で開発をしていると API だけを開発するわけにもいかず、さらに運用段階に入ってしまうと API の大幅な改修というのはあまり行われなくなります。そうすると意外と忘れがちになってしまいますが、誰が書いても同じような書き方になるという点で少しソースコードを読むことで思い出しやすくなります。さらに他の人を API 開発に割り当てたとしても Go を書ければ割とスムーズに理解ができるのではないでしょうか。

以上の理由から Go を API 開発の言語として選定をしました。

1.2.3 Web サイト：Go + React SPA

こちらに関しては正直なところ失敗をしたと思っており、Next を TypeScript で Server Side Rendering するべきでした。

ただ一つだけどうして Go をバックエンドにしたのか、これは API の開発で他の人を割り当てることができれば Web サイトのバックエンドも割り当てられるのではないかという理由がありました。実際のところは人を割り当てることができなかったので Next にしておいた方がよかったのではないかと思います。

1.2.4 インフラ：AWS

Go との相性を考えると Google App Engine を使うという選択肢もありましたが、Weddy Weddy では曲と動画の合成を行う必要がありました。もし iOS および Android それぞれをネイティブで開発していれば端末内で曲と動画の合成を行い、その合成された動画をアップロードするということができますが、Expo で開発を行っていたのでネイティブ部分をいじることはできませんでした。そこで開発初期では Elastic Transcoder と Media Convert を組み合わせて合成をしたり、次のフェーズでは Lambda に ffmpeg をアップロードして S3 をトリガーに ffmpeg で合成するということを行っていました。最終的には速度を求めるために ffmpeg を搭載したサーバを用意する形に落ち着きましたが、インスタンス自体の柔軟性を考えると AWS で間違っていなかったのではないかと思います。

1.3 まとめ

このような技術で Weddy Weddy というアプリは作られました。そこで 2 章からは 1 人で開発する際にチーム開発をするのと同じような環境をどうやって作ったらよいのかということを紹介していきます。

第 2 章

アプリ UI 開発編

まずはアプリの UI を組み立てていくところから始めていきましょう。なぜ API からではなくアプリ開発から行うのか？というところですが、自分だけがエンジニアの場合だと他の人はどういったものを作っているのか？ということについてはデザインでしか確認することができません。デザインデータと Zeplin(https://zeplin.io) や InVision(https://www.invisionapp.com) を組み合わせることで簡単なモックを作成することができますが、全てのイベントをモック上で設定するのはかなり手間がかかってしまい現実的ではありません。また動き次第ではデザインをやっぱりこうしたいというのも開発していると直面してしまうことが多々あります。

そこで API などの連携は抜きにして、まずは UI を構築し、ダミーデータを表示するような形にして一連の流れを入れておくようにしていきましょう。

2.1 アプリ開発をするための環境構築

まずは React Native でアプリを開発するために必要なものをインストールしていきましょう。

2.1.1 Xcode のインストール

`https://itunes.apple.com/jp/app/xcode/id497799835?mt=12`

まずは App Store から Xcode をインストールしてください。

2.1.2 コマンドラインデベロッパーツールのインストール

次に Xcode をコマンドラインから操作するためのツールをインストールしてください。このツールは直接使うことはありませんが、これからインストールするためのものに必要となることが多いのでインストールしておきましょう。

```
$ xcode-select --install
```

2.1.3 Homebrew のインストール

Homebrew はパッケージマネージャです。例えば Node や MySQL、ffmpeg など Apple が用意していない様々な必要なものをインストールしてくれます。Linux では yum や apt などがパッケージマネージャに該当します。

では Homebrew の公式サイト (`https://brew.sh/index_ja`) を参考に進めていきます。こちらにアクセスをすると下記のようなインストールをするためのコマンドが表示されていますので、ターミナルを開いてコマンドを貼り付け実行してください。

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com
/Homebrew/install/master/install)"
```

Enter の入力やパスワードの入力が求められるので適宜対応してインストールしてください。

2.1.4 Node および npm のインストール

先ほどインストールした Homebrew 経由で Node をインストールしていきましょう。早速`$ brew install node`を実行してもよいのですが、Homebrew 経由で Node をインストールするとバージョンが一つのみに固定されてしまいます。Node はバージョンアップが頻繁に行われ、新しい機能を入れているけどそれが安定しないなどということもよくあります。なので Node のバージョンを管理するためのツールである nodebrew をまずはインストールしましょう。

```
$ brew install nodebrew
```

インストールをしたら次は nodebrew をコマンドとして使うためのパスを通しましょう。

```
$ echo "export PATH=$HOME/.nodebrew/current/bin:$PATH" >> ~/.bash_
profile
$ source ~/.bash_profile
$ which nodebrew
/usr/local/bin/nodebrew
```

which コマンドにて nodebrew までのパスが通っていたらインストール完了です。

では nodebrew のセットアップを行い、Node をインストールしてコマンドが使えるようにしましょう。

```
$ nodebrew setup
$ nodebrew ls-remote
$ nodebrew install-binary stable
$ nodebrew use stable
$ node -v
v10.7.0
$ npm -version
6.2.0
```

`$ node -v` とした際にバージョンが表示されれば Node のインストールは完了です。なお npm は Node と一緒にダウンロードされており、`$ npm -version` を実行してバージョンが表示されれば大丈夫です。

2.1.5　VSCode のインストール

`https://code.visualstudio.com/`

エディタとして使用する VSCode を上記 URL からインストールしましょう。Download for Mac というリンクがあるのでこちらから zip ファイルをダウンロードし、解凍してアプリケーションフォルダに入れればインストール完了です。

2.2 Expo について

https://expo.io

どうしてアプリを開発する際に React Native なのか？ についてははじめにおよび 1 章にて説明をしましたが、さらにその上でどうして Expo なのか？ということを疑問に思うのではないでしょうか。Expo は React Native でのアプリ開発を簡単にするためのものとなっており、Xcode や Android Studio を触る必要がなく、カメラやオーディオ・動画などメディアライブラリへのアクセスや各種センサー系など様々なコンポーネントを Expo 側で用意しているので、JavaScript のみでアプリ開発を行うことが簡単にできます。ある程度のアプリであるならば Expo を使うことで JavaScript のみで開発することができるのでないでしょうか。

さらに OTA(Over The Air) アップデートという機能も標準で備えており、アプリストアに申請をしなくてもアプリのアップデートを行うことができます。

ではその便利な Expo についてインストール方法や、どういうアプリを作るときに Expo が向いているのか？ ということについて説明していきます。

2.2.1 expo-cli およびクライアントアプリのインストール

まずは Expo を使うために expo-cli をインストールしましょう。

```
$ npm install -g expo-cli
```

第 3 章

API 開発編

アプリが接続する API の開発を行っていきましょう。

3.1 API 開発をするための環境構築

Go で API を開発するために必要なものをインストールしていきましょう。

3.1.1 Go のインストール

まずは Go のインストールをしましょう。Homebrew からもインストールできますし、Go の公式サイト (`https://golang.org/dl/`) でも pkg のダウンロードができますが、本書では Homebrew を入れているので Homebrew 経由で行いましょう。

```
$ brew install go
```

これでインストールは完了です。デフォルトの GOPATH は`$HOME/go` と

なっており、もし他のフォルダを GOPATH にしたいのであれば指定しましょう。なお後述しますが direnv による GOPATH をプロジェクトごとに切り替えるという方法もありますので気にせずデフォルトのままでよいのではないでしょうか。

次に VSCode の Extension である Go(https://marketplace.visualstudio.com/items?itemName=ms-vscode.Go) をインストールしてください。これで VSCode にて Go の開発をするための準備が整いました。

3.1.2 Docker のインストール

次に Docker をインストールしましょう。Docker はコンテナ型の仮想化環境を提供してくれるソフトウェアとなっており、サーバの各種設定やミドルウェアやツールなどをインストールした仮想環境を簡単に構築してくれます。もちろん自分で Docker コンテナを作成して共有してもよいですし、すでに共有されている Docker コンテナをベースに開発を進めても問題ありません。本書ではローカルの開発環境を Docker で構築していきます。Docker で行うことによりもし他に人が入ったとしてもローカル開発用のサーバを別途用意する必要もなく、また Go のバージョンを変更したいというときもベースコンテナのバージョンを変更するだけで可能となります。

では Docker を Homebrew 経由でインストールしましょう。

```
$ brew install docker
$ brew cask install docker
```

これで Docker のインストールが完了しました。

ターミナル上で Docker を操作してもよいのですが、簡単に Docker コンテナを立ち上げたり再起動するための GUI として kitematic をインストールしましょう。

```
$ brew cask install kitematic
$ brew install boot2docker
```

これで kitematic のインストールは完了です。

3.1.3 direnv のインストール

次に dirnev をインストールしましょう。direnv はプロジェクトごとに環境変数を変更してくれ、本書では GOPATH を変更するために使いたいと思いますのでインストールしましょう。

```
$ git clone https://github.com/direnv/direnv
$ cd direnv
$ sudo make install
```

次にシェルに hook を追加しましょう。普段自分が使っているシェルをまずは確認してください。

```
$ echo $SHELL
/bin/bash
```

普段使われているシェルは bash なので.bashrc に下記を追加します。もし zsh を使っているなど他のシェルを使っているようでしたらそれぞれのシェル用の設定を行なってください。

リスト 3.1: ~/.bashrc

```
export EDITOR="vi"
eval "$(direnv hook bash)"
```

これで direnv のインストールが完了しました。VSCode には Extensions として direnv(https://marketplace.visualstudio.com/itemdetails?itemName=Rubymaniac.vscode-direnv) があるので、こちらもインストールしてください。

3.2 プロジェクトを管理するための環境構築

コードを書く前にプロジェクトを管理するために設定しておいた方がいい項目について説明します。

3.2.1 git リポジトリの作成

API 開発用のリポジトリを GitHub 上に作成しましょう。アプリのリポジトリ同様にどのような名前でも問題はないですが、api や go などのキーワードが入っているとわかりやすいのではないでしょうか。さらに 2 章でプルリクエストテンプレートを作成したように、API 用のリポジトリでも同じようにプルリクエストテンプレートを作成しておきましょう。

また API 開発用のリポジトリとは別にプロジェクトのドキュメントを管理するためのリポジトリを作成しましょう。運用編にて詳しく解説をしますが、ドキュメント用リポジトリを用意しておくことで issue の管理を一本化

することが簡単になります。例えばアプリ内の文言が違うなどの単純なものに関してはアプリのみしか関与しないですが、API のレスポンスが想定外なためにバグが表示されている場合だとアプリと API 両方に関わってくる issue となります。それぞれに issue を立てるのは正直なところ手間がかかってしまいますが、一本化することで開発に関わる人間全員が issue を見ることができ、さらにラベリングすることで自分に該当するものなのか？などわかりやすくなります。さらに API のドキュメントなども含めておくことで、困ったらドキュメント管理用のリポジトリを見れば解決するという状況を作ることができます。

それらを踏まえた上でドキュメント管理用のリポジトリを作成してください。

3.2.2 Go のプロジェクト作成

Go のプロジェクトを作成しましょう。`bin` と `pkg` と `src` をプロジェクトのルートディレクトリ内に作成をしてください。src ディレクトリ内には go get してきたパッケージが入るようになっています。この src ディレクトリ内に自分のプロジェクト用ディレクトリを作成して main.go ファイルを作成すればプロジェクト作成の完了となります。

リスト 3.2: アップロードするファイル

```
.
├── Dockerfile
├── Dockerrun.aws.json
├── README.md
├── bin
├── docker-compose.yml
├── pkg
│   └── ...
└── src
    ├── github.com
```

第 4 章

アプリリリース編

3 章にて API を作成したので、2 章で作成したアプリ UI に API を組み込み、アプリを審査に提出してリリースしましょう。

4.1 API の組み込み

API の組み込みに関しては特段インストールするべきものなどはありません。基本的には一つ一つ組み込み、実機でテストを行っていけば大丈夫です。

4.2 アプリのストア申請

アプリをビルドして App Store および Google Play Store に提出しましょう。なお各ストアのデベロッパーアカウントはすでに作成済みという前提となっています。

4.2.1 スクリーンショットの作成

アプリを審査に提出する際に、アプリのスクリーンショットが必要となります。動作している画面をスクリーンショットしたものでも問題ないで

第 5 章

アプリ運用編

アプリをリリースして終わりではありません。日々アップデートをしたり、どうやったらダウンロードされるのか？ ということを考えていく必要があります。そこで本章ではより良いアプリを提供するために、効率よく改善を行うための方法について紹介します。

5.1 プロジェクト管理用リポジトリの運用

プロジェクト管理用リポジトリについては 3 章で軽く触れておきましたが、主にこの管理用リポジトリで求められるものは下記の 3 点です。

- API などのドキュメント管理
- issue の管理
- Milestones の管理

5.1.1 API などのドキュメント管理

「API などのドキュメント管理」という点ですが、3 章の API のドキュメントについてで述べているように、API のエンドポイントおよび API クラ

あとがき

最後まで読んでいただきありがとうございます。誤植や不正確な表現にお気づきの方は、著者 Twitter までご連絡ください。

https://twitter.com/hmktsu

またハッシュタグ (#1 人チーム本) をつけてツイートしていただけるとうれしいです。

＃１人チーム本

2019 年 4 月 14 日　v1.0.0

著　者　渡邊雄