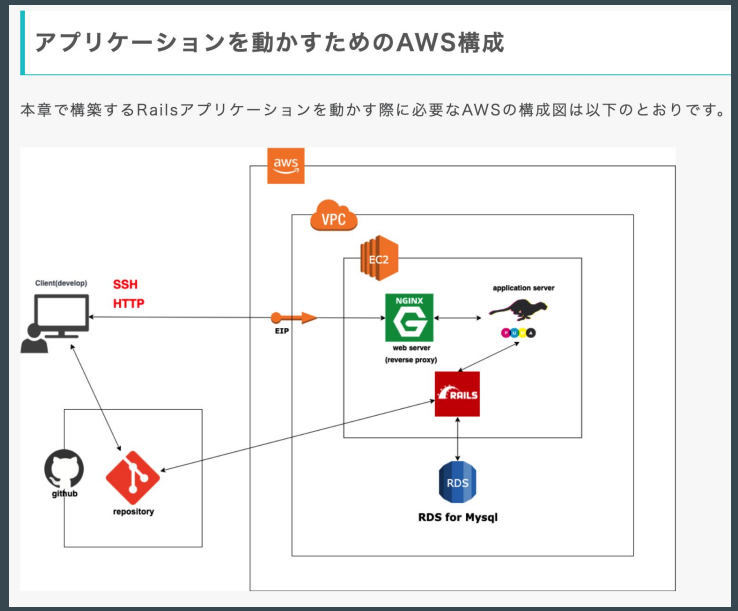
デプロイ講習会

目次

- Railsアプリケーションの動作の流れ(全体像の把握)
- Webサーバー(Nginx)とアプリケーションサーバー(puma)
  - Webサーバー(Nginx)のインストール(起動, 停止, 再起動含む), 設定
  - アプリケーションサーバー(puma)の設定
- ソフトウェアのインストール
  - ImageMagickのインストール
  - rbenv(ruby), Railsのインストール
- Railsアプリケーションの設定
  - bundle install, assets:precompile, db:migrate...

アプリケーション動作の流れ

カリキュラムの開発スキルアップ > AWSでWEBアプリケーションを公開しよう
「アプリケーションを動かすためのAWS構成」の図と比較しながら見ていこう！

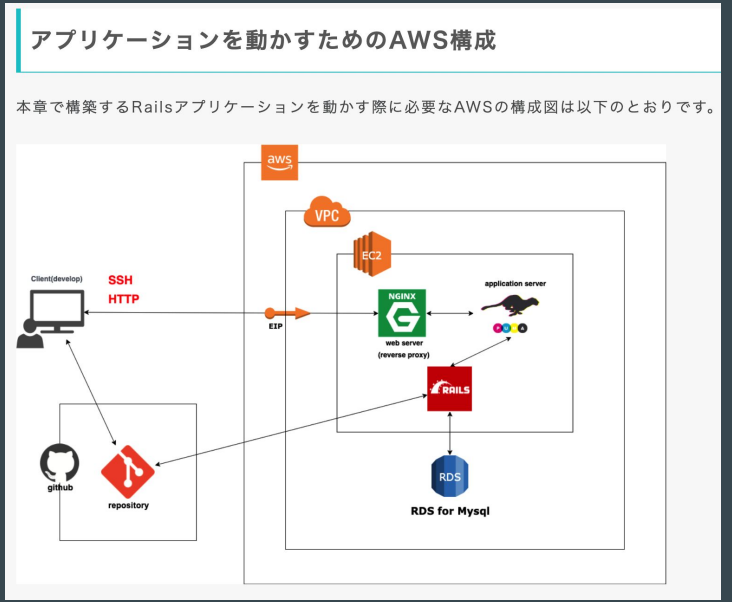


← この図を見てください

アプリケーション動作の流れ

注目すべきところ

EC2 (仮想サーバー)
Nginx（Webサーバー）
Rails
Puma（アプリケーションサーバー）



HTTPリクエストはどこへ？

http://example.com/hoge
のページが見たい！

ユーザー

HTTPリクエスト

http://example.com/hoge

WebサーバーがHTTPリクエストを捌く

HTTPリクエストはwebサーバーであるNginxへ向けて送られる

「http://example.com/hogeのページが見たい！」

ユーザー

HTTPリクエスト

http://example.com/hoge

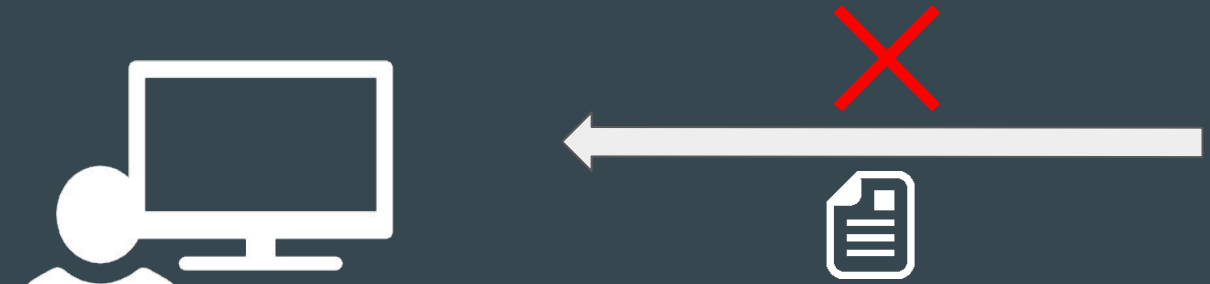
http://example.com/hogeが見たいんだね！対応するファイルを返してあげるね！

Webサーバー

HTTPリクエストはwebサーバーであるNginxへ向けて送られる

「http://example.com/hogeのページが見たい！」

困った。。僕は動的なコンテンツを提供することはできないんだ。
htmlファイルや画像ファイルといった静的コンテンツなら対応できるんだけど。。。
よし！アプリケーションサーバーのPumaくんに任せよっと！



ユーザー

Nginxは静的コンテンツであるhtml,css,jsファイルなどしかユーザーへお届け出来ません。
動的コンテンツはrails側で処理させてレスポンスさせます。

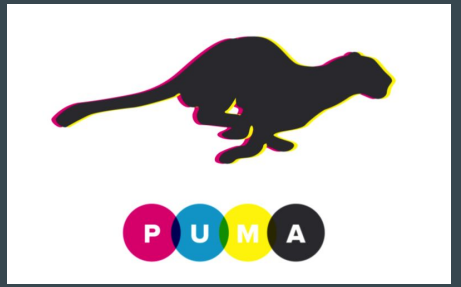
その流れは一体どうなっているのか？

リバースプロキシ(reverse proxy)

おーいアプリケーションサーバーのpumaくん!
ユーザーから
http://example.com/hoge
が見たい！ってきたから処理してくれーい！

Nginxはアプリケーションサーバーであるpumaにリクエストの内容を伝えます。
このNginxが働く仕組みを「リバースプロキシ」と言ったりします。

...むむ。分かりましたよ！ railsさんに処理させるように言うから！



Railsが処理して再度pumaへ返す

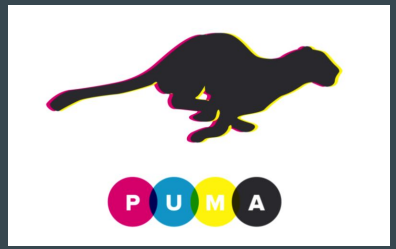
railsさん！railsさん！

http://exmple.com/hogeを処理して返してくれ！

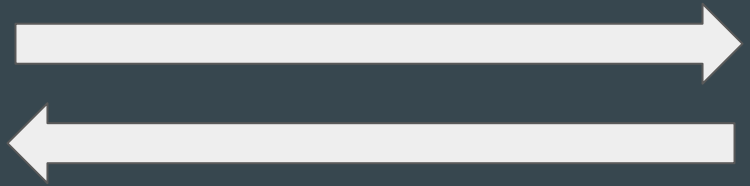
Nginxくんに返すからさ！よろしく頼むわー！

railsが行う処理はおなじみの MVCの流れです。

はいよー。
http://example.com/hogeね！
routes.rbに記述されているとおりに処理させるねー。



http://exmple.com/hoge処理して！



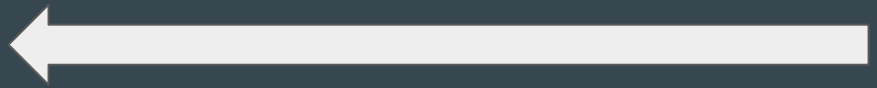
処理して返す！

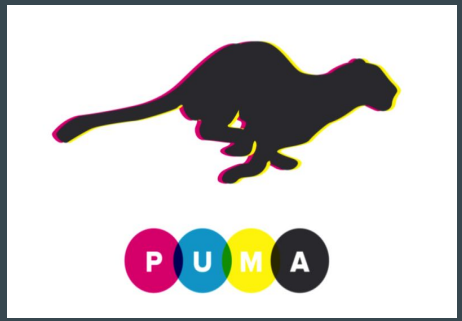
ユーザーへのレスポンスまで後少し！

早かったね！さすがpumaくん。
じゃあユーザーに届けてくるねー！
またよろしくー！

railsに処理させたファイルを持ってきたpumaくん。
Nginxにお渡しします。
この時Nginxがpumaから受け取るファイルは静的なものです。
railsのタグなどは純粋なhtmlに変換されています。
これならNginxくんも扱えるわけです。

Nginxくん！Nginxくん！
おまたせ！！railsさんに処理させたファイルをもってきたよ！
ユーザーさんにお届けしてあげて！

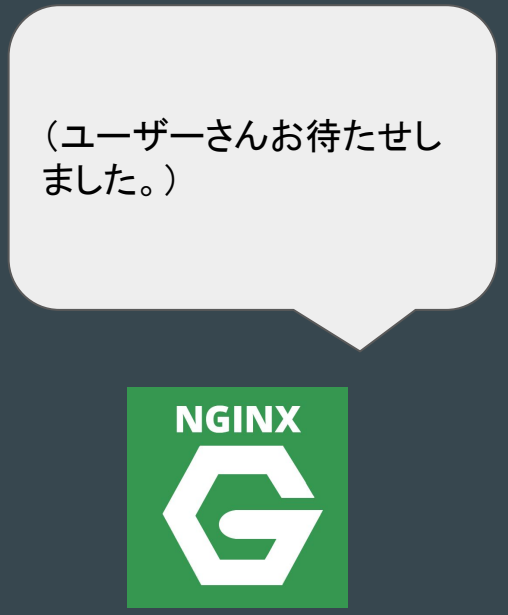




1回のHTTPリクエストの終わり

ユーザー

http://example.com/hogeに対応するファイル



アプリケーション動作のまとめ

1. ユーザーからのHTTPリクエスト
2. Nginxがリバースプロキシとして働き、Pumaへ繋ぐ
3. PumaからRailsに処理をさせてPumaへ返す
4. PumaがNginxへ返す
5. Nginxがユーザーへレスポンスを返す

皆さんのアプリケーションをAWSで公開するためには...

仮想サーバー(EC2)
Webサーバー(Nginx)
アプリケーションサーバー(Puma)
データベース(RDS)

これらの組み合わせが必要だということが分かります。

Webサーバー(Nginx)と
アプリケーションサーバー(puma)

Webサーバー(Nginx)のインストール(起動, 停止, 再起動含む), 設定

アプリケーションサーバー(puma)の設定

Nginxのインストール

HTTPリクエストは**Webサーバーである Nginxへ向けて送られる**ことを前の動画で説明しました。
webサーバーであるNginxが存在しないとHTTPリクエストに対してどうすることもできないので Nginxをインストールしています。

また、webサーバーへ向けられるHTTPリクエストであるHTTPには「ポート番号」というものが関わってきます。
ポート番号についての説明と、それに関わる問題については次のスライドから説明していきます。

ポート番号とは

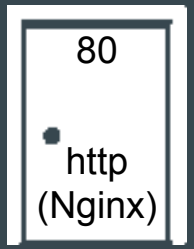
EC2(仮想サーバー)のIPアドレスを建物の住所に例えるなら、ポート番号は「部屋の番号は何号室か？」という例えになります。

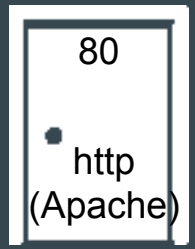
HTTPさんの通信には80番を指定、SSHさんの通信には22番を指定…と、いった具合です。

ポート番号の競合

ApacheとNginxの共存における問題

AWSを学ぼう > 2章【AWS基礎講座】の最後に**Apache**というWebサーバーをインストールしています。
ApacheとNginxではデフォルトでポート番号 80を使用するように設定されています。
2つが共存しているとHTTPの通信に対してどちらを使用するかわからないため、
結果的にエラーになってしまいます。時間があれば試してみてください。
この場合どちらかのポート番号を変更するか、どちらか一方を削除するのが安易です。
今回のカリキュラムでは Nginxを使用するので、既にインストールされている Apacheをアンインストールしています。





同じポート番号は
NG

Apacheのアンインストール

```
$ sudo yum -y remove httpd
```

試してみよう

https://qiita.com/ はhttps://qiita.com:443/

としているのと変らないです。

httpsが443というポート番号を使用しているということです。

逆にhttps://qiita.com:80/としてアクセスしてみましょう。

「え？httpsの部屋番号は80じゃないじゃん！！」

と、なってしまい接続できません。

これはhttpsのポート番号が 80番ではなく443番のためです。

Nginxの設定

Nginxをインストールすると/etc/nginx/にnginx.confというファイルが生成されます。
このnginx.confが設定ファイルのベースになります。

カリキュラムで指定している nginx.confの設定をいくつか説明します。
※設定の全てを説明はしません。詳細な設定については都度調べて実装します。完璧に覚えておく必要はありません。

/etc/nginx/nginx.conf

```
# For more information on configuration, see:
#   * Official English Documentation: http://nginx.org/en/docs/
#   * Official Russian Documentation: http://nginx.org/ru/docs/
#user nginx;
user ec2-user;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /var/run/nginx.pid;
# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;
```

Nginxの設定

/etc/nginx/nginx.confの設定

user ec2-user;

nginxの実行ユーザーの指定をしています。
カリキュラムではnginxをec2-userとして実行しています。

include /etc/nginx/conf.d/*.conf;

/etc/nginx/conf.d/にある設定ファイルを読み込みます。
***.conf**とすることで拡張子が .confとなるものがすべて読み込まれます。
インストール後の状態では、ファイルが存在しないため何も読み込まれません。
nginx.conf以外の設定ファイルを読み込ませたいときに使用します。
カリキュラムの例では nginx.confをベースファイルとして、アプリ名 .confを詳細設定のように
利用しています。

Nginxの設定

/etc/nginx/conf.d/アプリ名.confの設定

前のスライドで説明しましたが /etc/nginx/conf.d/以下の .confとする拡張子のファイルをすべて読み込んでいるため、/etc/nginx/conf.d/アプリ名 .confが読み込まれ、設定内容を反映します。

/etc/nginx/conf.d/<アプリ名>.conf

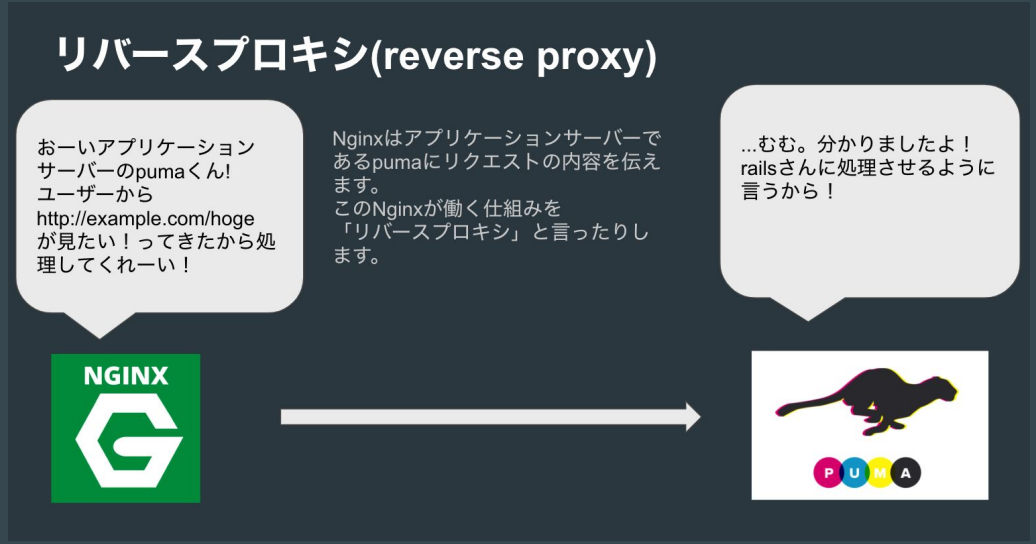
```
upstream puma {
    server unix:///home/ec2-user/<アプリ名>/tmp/sockets/puma.sock;
}
server {
    listen       80;
    server_name  <EC2のIPアドレスもしくはドメイン取得している方はドメイン名を設定>;
    root /home/ec2-user/<アプリ名>/public;

    access_log  /var/log/nginx/access.log  main;
    error_log /var/log/nginx/error.log;

    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
    keepalive_timeout   65;
    types_hash_max_size 2048;
    client_max_body_size 100M;
    include             /etc/nginx/mime.types;
```

Nginxの設定

/etc/nginx/conf.d/アプリ名.confの設定

```
upstream puma {
  server unix:///home/ec2-user/<アプリ名>/tmp/sockets/puma.sock;
}
```

こちらはアプリケーションサーバー（puma）との通信に必要な設定です。



Nginxの設定

/etc/nginx/conf.d/アプリ名.confの設定

listen　　80;

ポート80番を許可するという設定

HTTPのポートは80番を使用するためNginx側で80番を許可しないと接続ができません。

error_log /var/log/nginx/error.log;

nginxのエラーログの出力箇所の指定をしています。

カリキュラムの設定ではwebサーバーで起きたエラーは**/var/log/nginx/error.log**に

出力されます。エラーの原因を探るのに重要な設定です。

include　　/etc/nginx/mime.types;

MIMEタイプというもののマッピングをしています。

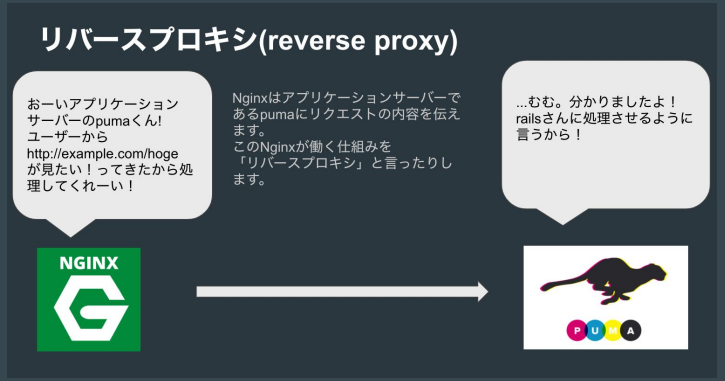
一言で言えばファイルの関連付けをしているところです。

この設定がないとcssなどが反映されません。

Nginxの設定

/etc/nginx/conf.d/アプリ名.confの設定

```
location / {
        proxy_pass http://puma;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_redirect off;
        proxy_connect_timeout 30;
}
```



ここでは主にリバースプロキシの設定をしています。

pumaへと連携している設定と思っていただければと思います。

前のスライドで説明した upstream pumaの部分はpumaへの窓口への通信というイメージで、このスライドでの内容は、連携の具体的な部分とイメージしておけばいいと思います。

リバースプロキシについては別動画の「アプリケーション動作の流れ」で概要について説明しています。

Nginxのコマンド

起動
$ sudo service nginx start

終了
$ sudo service nginx stop

再起動
$ sudo service nginx restart
※一度nginxを停止させるため、リクエストを受け付けできない時間が一瞬発生します。

設定の再読み込み
$ sudo nginx -s reload
※リクエストを受け付けできない時間が発生しません。

状態の確認
$ sudo service nginx status

設定ファイルのテスト、エラーがあれば内容を表示
$ sudo nginx -t

設定ファイルのテスト、設定ファイルの表示
$ sudo nginx -T
※includeされたファイルは展開され、1本の設定ファイルとして表示される

!! 注意 !!

EC2のOSがAmazonLinux2やCentOS7の場合は”service”の部分を”systemctl”に変更してください。

pumaの設定

config/puma.rb

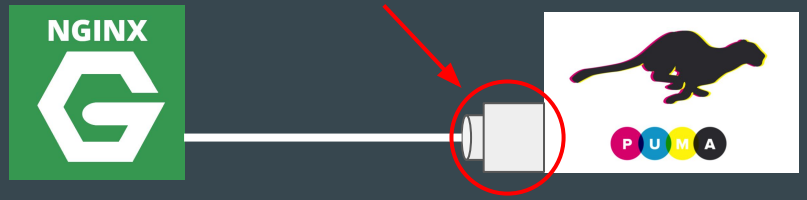
config/puma.rb

```
bind "unix://#{Rails.root}/tmp/sockets/puma.sock"

rails_root = Dir.pwd

# 本番環境のみデーモン起動

if Rails.env.production?

    pidfile File.join(rails_root, 'tmp', 'pids', 'puma.pid')

    state_path File.join(rails_root, 'tmp', 'pids', 'puma.state')

    stdout_redirect(

      File.join(rails_root, 'log', 'puma.log'),

      File.join(rails_root, 'log', 'puma-error.log'),

      true

    )

    # デーモン

    daemonize

end
```

pumaの設定

```
bind "unix://#{Rails.root}/tmp/sockets/puma.sock"
```

この部分!!



これはNginxがpumaへと接続するための”窓口”です。

“ソケット”という呼び方をしたりします。

```
if Rails.env.production?
  pidfile File.join(rails_root, 'tmp', 'pids', 'puma.pid')              # app/tmp/puma.pidにpumaのプロセス番号が記述される
  state_path File.join(rails_root, 'tmp', 'pids', 'puma.state')     # app/tmp/puma.stateにpumaの状態が記述される
  stdout_redirect(
        File.join(rails_root, 'log', 'puma.log'),                       # app/log/puma.logにpumaのログが記述される File.join(rails_root, 'log',
        'puma-error.log'),                  # app/log/puma-error.logにpumaのエラーログが記述される
    true
  )
  # デーモン
  daemonize                                                                  # アプリをデーモンとして起動する設定
end
```

ソフトウェアのインストール

ImageMagick

rbenv(ruby), Rails

ImageMagick

ImageMagick（イメージマジック）は画像を操作するためのソフトウェアです。

例えばImageMagickは、画像のリサイズやjpgからpngへ変換するような機能を実現します。

バージョンによってセキュリティに致命的な脆弱性を持っていますが、カリキュラムの「 AWSでアプリケーションを公開しよう」のインストールでは、脆弱性に対する対応プログラムを含むものをインストールできるようにしてあります。

公式URL : https://imagemagick.org/

ImageMagickインストールのコマンド解説

```
# ImageMagickに必要な各種パッケージをインストール
$ sudo yum -y install libpng-devel libjpeg-devel libtiff-devel gcc
# ImageMagickの圧縮ファイルをインストール
$ wget http://www.imagemagick.org/download/ImageMagick.tar.gz
#  ImageMagickの圧縮ファイルを解凍
$ tar -vxf ImageMagick.tar.gz
#  解答されたImageMagickディレクトリへ移動
$ cd ImageMagick-x.x.x-xx
# 設定
$ ./configure
# ビルド ※ソフトウェアが動くようにコンパイル等すること
$ make
# ビルドされたソフトウェアをインストール
$ sudo make install
```

ImageMagickのインストール

※ImageMagick-x.x.x-xxのxx部分はlsコマンドで確認ください。

```
$ sudo yum -y install libpng-devel libjpeg-devel libtiff-devel gcc
$ wget http://www.imagemagick.org/download/ImageMagick.tar.gz
$ tar -vxf ImageMagick.tar.gz
$ cd ImageMagick-x.x.x-xx
$ ./configure
$ make
$ sudo make install
```

rbenv(Ruby), Railsのインストール

rbenv(Ruby)のインストール

rbenvは、複数のRubyのバージョンを管理し、
プロジェクトごとにRubyのバージョンを指定して使うことを可能としてくれるツールです。
「AWSでアプリケーションを公開しよう」では rbenvを使ってRubyをインストールして管理します。

例えばruby2.5.5をインストールしたあとで ruby2.5.7を使用したいと言った場合にも
切り替えもインストールも楽になります。

rbenv(Ruby)のインストールコマンドの解説

① ~ ③

```
$ cd                                                                  # ホームディレクトリへ移動
$ mkdir ~/.rbenv                                                      #  .rbenvというディレクトリを作成
$ git clone https://github.com/rbenv/rbenv.git ~/.rbenv               # 作成した.rbenvの中へrbenvをgit clone
$ mkdir ~/.rbenv/plugins ~/.rbenv/plugins/ruby-build                  # .rbenv/plugins/ruby-buildというディレクトリ構造を作成
$ git clone https://github.com/rbenv/ruby-build.git ~/.rbenv/plugins/ruby-build # .rbenv/plugins/ruby-buildにruby-buildをgit clone
$ cd ~/.rbenv/plugins/ruby-build                                      # ~/.rbenv/plugins/ruby-buildに移動
$ sudo ./install.sh                                                   # install.shというシェルスクリプトでインストール
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bash_profile      # bash_profileに環境パスを設定
$ echo 'eval "$(rbenv init -)"' >> ~/.bash_profile                    # bash_profileにrbenvの初期化のコマンドを設定
$ source ~/.bash_profile                                              # bash_profileを反映
```

```
$ /***/***/.rbenv/bin/rbenv --version
↓
$ rbenv --version
```

プログラム名だけで実行できるようにするため!!

Railsアプリケーションの設定

bundle install, assets:precompile, db:migrate…

サーバー起動前の準備

① $ cd <アプリ名>

アプリケーションに移動

② $ bundle install --path vendor/bundle --without test development

オプションをつけてbundle installしている

③ $ bundle **exec** rails assets:precompile RAILS_ENV=production

アセットプリコンパイルをしている

④ $ bundle **exec** rake db:migrate RAILS_ENV=production

本番環境を指定してdb:migrate

②～④を掘り下げて説明していきます

② $ bundle install --path vendor/bundle --without test development

--path vendor/bundle

オプションに--path vendor/bundleを付けることでgemのインストール先を指定しています。
このようにすることでプロジェクトごとにgemを管理できます。
逆にオプションを付けずにbundle installとした場合にはgemがグローバル（system）にインストールされているということになります。
グローバルにインストールされるというのはインストールしたgemが開発マシン内のどこでも使える状態になるということです。
デプロイではRailsアプリケーションの中にgemをinstallしています。

--without test development

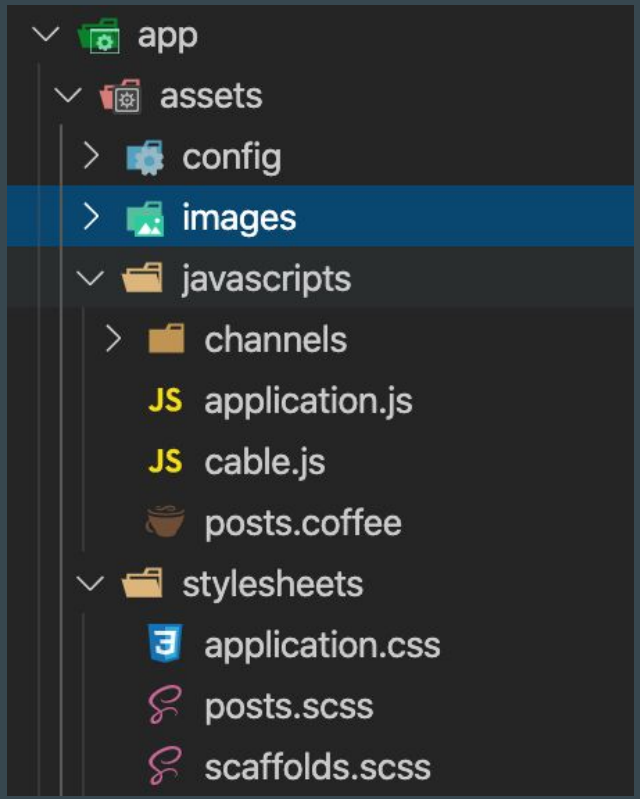
本番環境でbundle installする時にdevelopmentとtestグループのgemはインストールしないためのオプション。

③ $ bundle exec rails assets:precompile RAILS_ENV=production

アセットプリコンパイル

Railsは作業がしやすいようにJSやCSSのファイルが分割されていますが、”**最終的にJS、CSSそれぞれ一つのファイルに連結・圧縮する**”という部分をアセットプリコンパイルが担っています。

.coffeeや.scss形式のファイルもJSファイル、CSSファイルへとコンパイルされ一つに統合されます。



④ $ bundle exec rails db:migrate RAILS_ENV=production

RAILS_ENV=production

RAILS_ENV=productionをオプションに指定することで、本番環境を指定して db:migrateを行います。
このようにすることでdatabase.ymlのproduction箇所の情報でデータベースの構築を行います。

bundle exec rails...

bundle exec rails...とすると実行しようとしているRailsアプリケーションの設定を読み込んで実行するという認識でいればいいとおもいます。