特典B
本番環境の構築

書籍本編では、ローカル環境（開発者のパソコンでの環境）で開発・実行を行ってきました。実際の開発現場ではテストなどを行うステージング環境や、ユーザが使用する本番環境を構築する必要があります。実際に本番環境を構築しながら、環境構築にあたって必要な検討事項を学んでいきましょう。

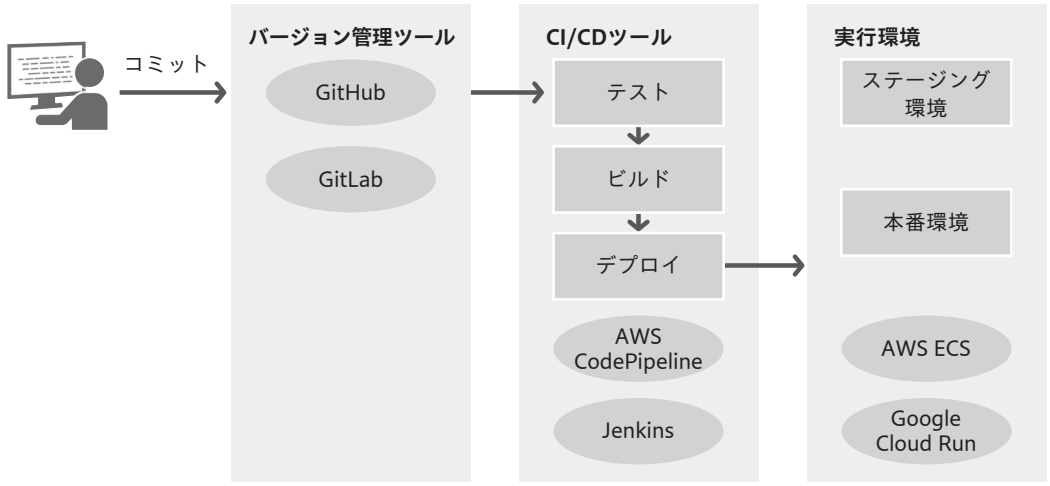
B-1 全体像

B-1-1 本番環境反映のイメージ

ローカル環境で開発したアプリケーションを、本番環境に反映する際の流れを説明します。大きく、次の3つの段階に分けられます。

① バージョン管理ツール
② CI/CDツール
③ 実行環境

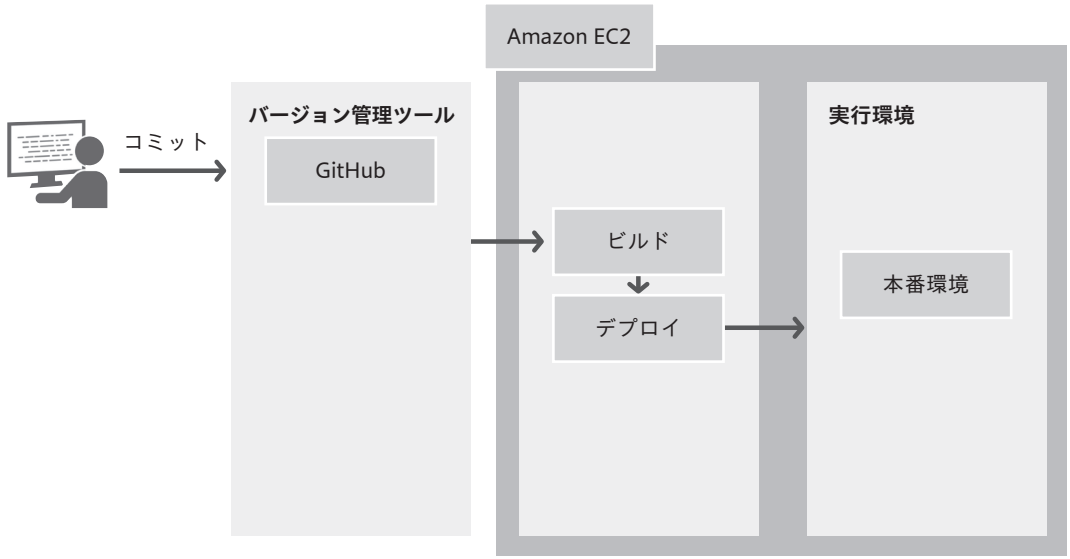
バージョン管理ツールで管理しているソースコードを、CI/CDでテスト、ビルド、デプロイし、実行環境に反映するという流れです。CI/CDとは「Continuous Integration（継続的インテグレーション）/ Continuous Delivery（継続的デリバリー）」の略称です。バージョン管理ツールで管理しているソースコードの変更を常にテストし、自動で実行環境に適用できるような状態にしておく開発手法のことを指します。各段階の詳細は後ほど解説します。



図B-1-1 環境構築検討資料

B-1-2 本書での構築イメージ

本特典では、説明と平行して本番環境を構築していきますが、説明のために最低限必要な構成で進めていきます。最低限必要な構成の構築イメージは次の通りです。



図B-1-2 環境構築イメージ

1つの汎用サーバー（Amazon EC2）の中でビルド、デプロイ、実行環境を作成します。Amazon EC2は、Amazon Elastic Compute Cloudの略で、Amazonが提供する「仮想サーバーサービス」です。EC2を利用することで、サーバー環境をクラウド上にすばやく作ることができます。特定のCI/CDツールに偏った説明とならないよう、基本的な手動でのビルド、デプロイを実施します。実際のプロジェクトではこのような構成にすることは少なく、CI/CDの段階ごとに別々のサービスを選択することが多いです。その選択のポイントも含めて解説していきます。

Purchase benefits

B-2 実行環境の構築

B-2-1 実行環境の検討

まず、サービスが動く実行環境を構築する必要があります。書籍本編では、Dockerコンテナでアプリケーションを実行しながら開発を進めてきました。DockerコンテナではOSやアプリケーショ

ン設定をファイルに定義しているので、どの開発者のパソコンでも同じ環境で開発できることがメリットでした。ステージング環境/本番環境でも同様にDockerコンテナで構築することで、環境の共通化が可能となります。

ステージング環境/本番環境にDockerコンテナ環境を構築する方法は、大きく分けて2通りあります。

1つ目は、汎用サーバーにDockerをインストールして構築する方法です。ローカル環境構築時にパソコンにDockerをインストールしたのと似たような手順で、汎用サーバーに構築します。物理的なサーバーがある場合はそちらに構築しても構いませんし、前述したEC2のような仮想サーバーサービスを使用しても構いません。後ほど実施するハンズオンもこちらの方法で行います。

2つ目は、Dockerコンテナをサポートしているコンテナサービスを選ぶ方法です。コンテナサービスとは、Dockerで作成したイメージをそのまま実行できるサービスです。具体的には次のようなサービスがあります。

- Amazon ECS（https://aws.amazon.com/jp/ecs/）
- Google Cloud Run（https://cloud.google.com/run）
- Microsoft Azure Container Instances
  （https://azure.microsoft.com/ja-jp/products/container-instances）

2通りのどちらを選択するかのポイントについては次の表の通りです（コンテナサービスによっても異なります）。コスト面や性能面でコンテナサービスを使用したほうが有利なことが多いので、基本はコンテナサービスを使用で検討し、難しい事情があれば汎用サーバーを使用するというアプローチがよいかもしれません。

表B-2-1 それぞれの利点

汎用サーバーにDockerをインストール	コンテナサービスを使用
• 物理サーバーへの構築ができる • シンプルなサーバー構成にできることが多い	• コンテナ以外のサーバー管理が不要 • スケールアップ（サーバーの性能を上げること）が柔軟にできることが多い • スケールアウト（サーバーの台数を増やすこと）が柔軟にできることが多い

コンテナサービスの選定方法に関する説明は省きますが、現在、使用しているクラウドサービスがあるのであれば、そのクラウドサービスのコンテナサービスを使用するのがスムーズでしょう。既にAWSを使用しているのであれば、Amazon ECSを選択するということです。

B-2-2 クラウドサービスに登録する

実際に実行環境を構築していきます。今回は汎用サーバー（EC2）にDockerをインストールして進めるので、AWSのクラウドサービスを利用します。まずは、AWSのアカウントを作成します。なお、AWSの管理画面は画面構成がしばしば変わります。画面の見た目は紙面と異なる可能性がありますので、ご了承ください。

AWSアカウント作成にあたり、下記の情報が必要なので、事前に準備しておきます。

- フルネーム（会社名）
- 使用可能なメールアドレス
- 電話着信を受けられる携帯電話か固定電話
- 有効なクレジットカードまたはデビットカード
- 住所

AWSアカウントを作成する手順は、公式にまとめられています（https://aws.amazon.com/jp/register-flow/）。こちらに従って、AWSアカウントを作成していきます。

リンク先のページでの「今すぐ無料サインアップ」ボタンを押下し、必要情報を入力していくことで作成できます。

なお、すでにAWSアカウントにログインしている状態で、別のAWSアカウントを作成しようとすると、ログイン中のAWSアカウント管理画面に遷移してしまい、AWSアカウントが作成できないことがあります。その場合、AWSアカウントからログアウトしてから、AWSアカウントを作成します。

B-2-3 AWS費用

AWSでの環境構築前に知っておくべき注意事項ですが、AWSの使用方法によって費用がかかります。今回の構成時に発生する費用は次の通りです。なお、2023/11時点の費用で計算していますが、今後、AWSの料金体系が変わることがあります。構築の際は https://aws.amazon.com/ の最新情報を確認してください。

EC2のオンデマンド費用

EC2を起動している時間帯に料金がかかります。https://aws.amazon.com/jp/ec2/pricing/on-demand/で計算が可能です。費用はEC2のインスタンス（スペック）によって異なります。今回利用するt3.smallの場合は、0.0272 USD（米ドル）となっています。仮に、24時間EC2を起動したままにすると0.0272 USD × 24 = 0.6528で、約0.7ドルがかかる計算となります。

図B-2-1　t3.large 費用

起動したままだと料金がかかり続けるので利用が終わったら忘れずにEC2を停止する点に注意しましょう。EC2の停止は、EC2の画面で「インスタンスを停止」で実施可能です。再開するときは、EC2の画面で「インスタンスを開始」を実施します。

ストレージ費用

EC2に割り当てるストレージ（データを保管する箇所）の費用がかかります。パソコンでいうところのハードディスクやSSDに該当します。ストレージの名称はElastic Block Storageです。https://aws.amazon.com/jp/ebs/pricing/ で計算が可能です。

図B-2-2　ストレージ費用

費用はストレージのボリュームタイプ（スペック）によって異なります。今回利用する基本的なストレージである「汎用 SSD (gp3) - ストレージ」であれば0.096USD/GB 月となります。今回は16GBを使用するので、1か月ストレージを使用した場合、0.096USD × 16GB = 1.536で、約2ドルがかかる計算となります。

ただし、AWSには特定の条件を満たした場合に無料になる利用枠があります（https://aws.amazon.com/jp/free/）。AWSに最初にサインアップした日から12か月間については、Amazon Elastic Block Storage30GBが無料となります。

図B-2-3　ストレージ無料枠

EC2を停止しても、ストレージの料金はかかり続けますので注意が必要です。EC2、ストレージともに不要になった場合はEC2の終了を行います。EC2の画面で「インスタンスを終了」で実施可能です。その後、Elastic Block Storeの画面で、対象のボリュームが削除されていることを確認してください。

データ転送費用

EC2からインターネットに転送する際の費用がかかります。https://aws.amazon.com/jp/ec2/pricing/on-demand/で計算が可能です。100GBまでは無料で、100GBを超えた場合に料金がかかります。

データ転送

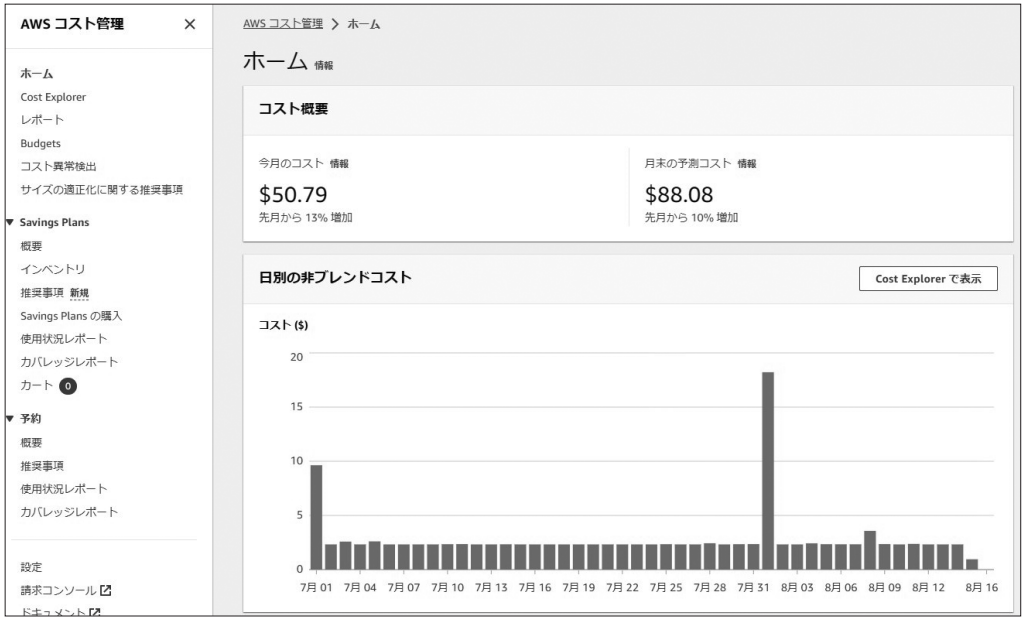
以下の料金は、Amazon EC2 に「受信 (イン)」/「送信 (アウト)」されるデータ転送量を基にしています。

リージョン: アジアパシフィック (東京)

	料金
インターネットから Amazon EC2 へのデータ転送 (イン)	
すべてのデータ受信	0.00USD/GB
Amazon EC2 からインターネットへのデータ転送 (アウト)	
AWS をご利用のお客様は、すべての AWS のサービスとリージョン (中国と GovCloud を除く) で合計 100 GB のインターネットへのデータ転送 (アウト) を 毎月無料でご利用いただけます。インターネットへのデータ転送 (アウト) のための 100 GB の無料利用枠は グローバルであり、AWS リージョンに個別にまたは別個に適用されるものではありません。	
最初の 10 TB/月	0.114USD/GB
次の 40 TB/月	0.089USD/GB
次の 100 TB/月	0.086USD/GB
150 TB/月以上	0.084USD/GB

図B-2-4　データ転送費用

AWSの料金体系は複雑なので、こまめに費用を確認するのが望ましいです。AWSコスト管理の画面で日々の費用を確認することが可能ですので、思わぬ費用が発生していないか、定期的に確認することが重要です。不要なサービスについては終了する、AWSのアカウント自体が不要になった場合はアカウント自体を削除するといった対応を行うようにしましょう。本書で作成したEC2も学習を中断するときは、EC2画面で「インスタンスを停止」して料金がかかり続けないようにします。学習を終了するときは、EC2画面で「インスタンスを終了」を行い削除しておきましょう。



図B-2-5　AWSコスト管理

また、AWS管理画面で「予算の作成」を行うことで、指定した金額を超過した場合にメールなどを通知することも可能です※B-1。必要に応じて活用していくとよいでしょう。

B-2-4 サーバーを構築する

それでは、サーバーを構築します。今回はAmazon EC2を使用します。まず、EC2を作成するリージョンを選択します。リージョンとは、クラウドサービスで利用するデータセンターの所在地のことを指します。今回は東京を指定します。AWS管理画面の上のプルダウンから「アジアパシフィック(東京) ap-northeast-1」を選択します。メニューのEC2から「インスタンスを起動」を選択します。メモリは2GB程度、ストレージは16GB程度が必要ですので、次のスペック以上を選択します。

```
名前＝（任意の名前）
AMI＝Amazon Linux 2023 AMI
インスタンス＝t3.small
キーペアなしで続行
セキュリティグループを作成する
ストレージ＝16GB
```

図B-2-6 EC2作成

※**B-1** https://docs.aws.amazon.com/ja_jp/cost-management/latest/userguide/budgets-create.html

各項目の解説をします。

名前

EC2につける名前です。他のインスタンスと区別をつけるために設定します。

AMI

Amazonマシンイメージの略です。EC2インスタンスを起動するためのテンプレートのようなものです。LinuxなどのOSが組み込まれています。「Amazon Linux 2023 AMI」はAmazonが提供するLinuxイメージです。長期サポートが提供され、使用している企業も多いことから、EC2を使用するうえで事実上の標準となっております。

インスタンスタイプ

インスタンスの性能を指定します。高性能なCPUや、多いメモリを指定すれば早く動作するようになりますが、料金も高くなるので、比較検討してちょうどよいタイプを選ぶことが大切です。今回のハンズオンでは、最低限動作するスペックとして「t3.small」を選択しています。

キーペアなしで続行

キーペアはSSH接続を行う場合に必要となるものです。SSH接続とは、EC2などのサーバーを遠隔操作できる仕組みです。後ほど紹介する手順ではSSHは使用せずにAWSセッションマネージャーで接続するためキーペアの作成は不要です。AWSセッションマネージャーは、AWSコンソール画面からEC2に接続する為の仕組みです。

セキュリティグループ

どのIPアドレスからEC2に接続を許可してよいかを設定するものです。IPアドレスはパソコンやサーバーの住所に該当するものです。IPアドレスにはIPv4という3.112.23.0のようなアドレスと、IPv6という2001:0db8:85a3:0000:0000:8a2e:0370:7334のようなアドレスがあります。以前はIPv4しかなかったのですが、IPv4では約42億のIPアドレスしか管理できないこと等から、より多数のIPアドレスが扱え、セキュリティや効率性も向上させたIPv6が策定されました。ただ、現時点ではIPv4が現役ということもあり、今回はIPv4での説明・設定とします。

IPアドレスは3.112.23.0のような3桁区切り×4の数字なのですが、セキュリティの設定を行う際にIPアドレスの範囲を定義するときがあります。IPアドレスの範囲を定義するにはサブネットマスクを使用します。サブネットマスクはIPアドレスの後にスラッシュ区切りで指定します。3.112.23.

0/29だと、IPアドレスが「3.112.23.0」でサブネットマスクが「29」になります。この場合のIPアドレスの範囲は3.112.23.0〜3.112.23.7になります。

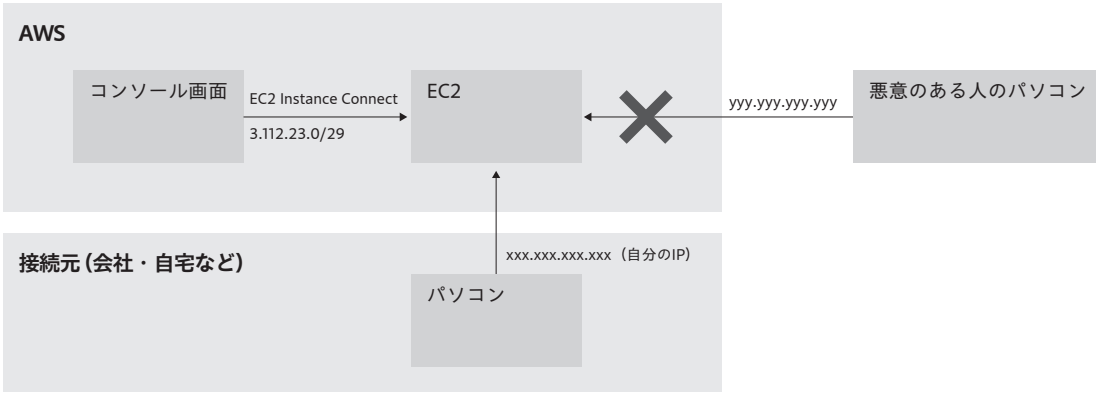
なぜそのIPアドレスの範囲になるのかについては、10進数（通常の数字）を2進数（コンピュータが扱う数字）に直して考える必要があります。

IPアドレス3.112.23.0の2進数表記は次の通りです。

- 3は2進数で00000011
- 112は2進数で01110000
- 23は2進数で00010111
- 0は2進数で00000000

したがって、IPアドレス3.112.23.0は2進数、00000011.01110000.00010111.00000000になります。サブネットマスクが/29の場合は、29桁目「00000011.01110000.00010111.00000」までは確定で、残りの3桁は「000」〜「111」のどれでもよいことを示します。つまり2進数での範囲は「00000011.01110000.00010111.00000000」〜「00000011.01110000.00010111.00000111」になります。これを10進数に直すと3.112.23.0〜3.112.23.7の範囲になるということです。

EC2に限らず、サーバーを構築するときは、どこからのアクセスを許可するか、拒否するのかを設定する必要があります。全てのアクセスを許可すると、悪意のある人から攻撃され、サーバーを乗っ取られたり、データを盗まれたりする恐れがあるからです。今回は、下の図にある通り、AWSのコンソール画面と、接続元のパソコンからは許可を設定する必要があります。逆に、悪意のある人のパソコンからのIPアドレスは拒否するようにしなければいけません（「EC2 Instance Connect」については後ほど説明します）。



図B-2-7　接続イメージ

AWSコンソールからのSSH接続を許可します。AWSコンソールのIPアドレスはhttps://ip-ranges.amazonaws.com/ip-ranges.jsonに定義があります。今回は東京リージョン（ap-northeast-1）（東京に実際のサーバーがある）にEC2を設定するので、ip-ranges.jsonに記載がある3.112.23.0/29を許可します。EC2作成画面（**図B-2-6**）の中にあるネットワークの設定箇所で「編集」ボタンを押下します（**図B-2-8**）。インバウンドセキュリティグループのルールでSSH 3.112.23.0/29を許可します（**図B-2-9**）。

図B-2-8　セキュリティグループ設定①

図B-2-9　セキュリティグループ設定②

次に、接続元（会社・自宅など）からの接続を許可します。Next.jsは3000のポートで起動するので、そちらのみを許可します。IPアドレスは住所のようなものであり、ポートはその住所に届けられる郵便物の部屋番号のようなものです。「自分のIP」と設定すればAWS側で接続元のIPアドレスを設定してくれます。それでは許可を設定します。インバウンドセキュリティグループのルールで「TCP 3000 自分のIP」を許可します（**図B-2-9**）。なお、ネット環境によっては接続元のIPアドレスが変わることがあります。その場合、「セキュリティグループ」の設定から、新しい自分のIPを設定しなおします。

設定が終わったら**図B-2-6**の「インスタンスを起動」を押下します。EC2の管理画面で、作成したインスタンスの状態が「実行中」になっていればOKです。

図B-2-10　EC2管理画面

次からはEC2に接続してコマンド操作を行い、環境を構築していきます。EC2の管理画面にて、先ほど作成したインスタンスを選択の上「接続」を押下します。EC2 Instance Connectタブが選択されていることを確認の上、再度「接続」を押下します。黒背景のターミナルが表示されればOKです。EC2 Instance ConnectとはAWS管理画面からEC2にアクセスできるサービスです。EC2に接続できたので、今後はこのコンソールからコマンド操作を実行していきます。

図B-2-11　EC2に接続

Yumパッケージマネージャーを使用してLinuxシステム上のすべてのパッケージ（アプリケーション）を最新のバージョンに更新します。

```
sudo yum update
```

yumとはLinuxでのパッケージを管理する管理ツールです。パッケージのインストールや更新を行います。「yum update」で、現在インストール済のパッケージを最新のバージョンに更新します。

EC2 Instance Connectで接続したec2-userというユーザでのままでは、権限がない為、yumのコマンドを実施できません。そこで、yumのコマンドの前に「sudo」を記載します。sudoはシステム管理者の権限が必要なタスク（今回だとyum）を、一般ユーザー（今回だとec2-user）として実行できるようになります。

ストレージ

サーバーの容量です。今回のハンズオンではDockerイメージを作成したりなど、多くの容量を使用するので16GBとします。

これで、実行環境としてのサーバーの準備ができました。後ほど、アプリケーションのデプロイ（配置）などを行って、アプリケーションが動作するように進めていきます。

Purchase benefits

B-3 バージョン管理システムの検討

B-3-1【ハンズオン】バージョン管理システムとの接続

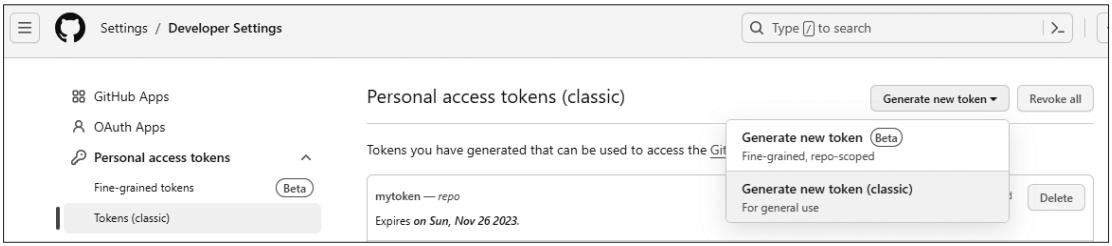
次に、GitHubのソースコードをEC2から取得できるよう接続します。サーバー内にGitを導入してfrontend、backendのイメージを取得できるようにします。

まずは、Gitをインストールします。

```
sudo yum install git
```

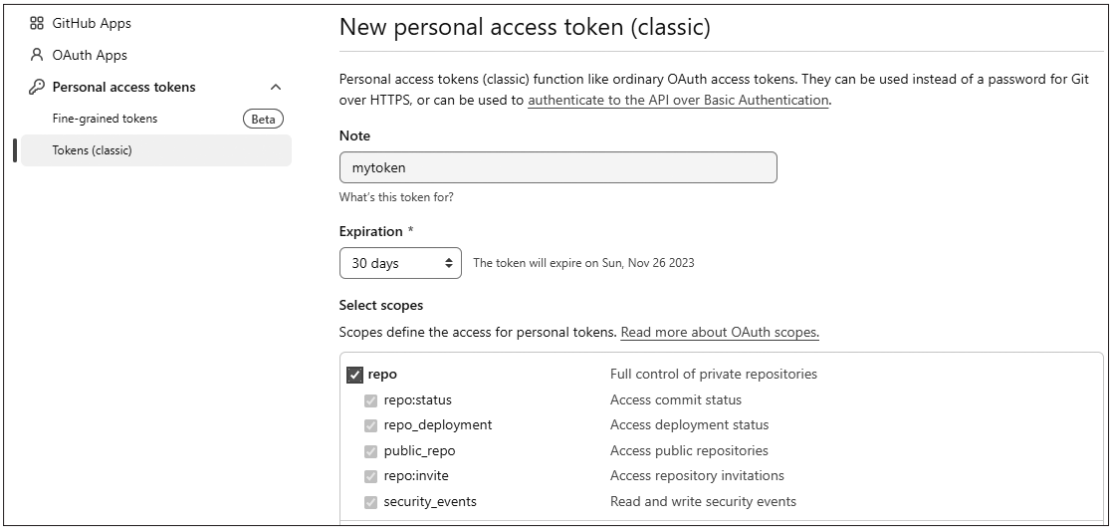
「yum install」でその後に続くパッケージであるGitをインストールするという意味になります。コンソールにおいて「Is this ok [y/N]:」のようにインストールしてよいかの確認があります。「y」を入力してEnterを押下してください。

Gitコマンドでリポジトリにアクセスできるよう、GitHubの管理画面からトークン（token）を作成します。トークンとはGitHubにアクセスするための鍵のようなものです。https://github.com/settings/tokens でGitHubのトークンページに移動して「Generate new token (classic)」でトークンを作成開始します。



図B-3-1　GitHubでトークン発行

「Note」にトークンを識別するための名前をつけます（何でもよいです）。次に「repo」にチェックをつけます。repoにチェックをつけることで、リポジトリのアクセスを許可する意味になります。「Generate Token」ボタンでトークンを発行すると、パスワードが表示されるので、忘れずに安全な場所に保存します。



図B-3-2　GitHubでトークン発行

書籍本編で作成したソースコードだと、多数のライブラリを使用している等の理由で、t3.smallのスペックだと動作しません。本書のサンプルコードにt3.smallで動作するコードを用意しているので、翔泳社サイトからダウンロードの上、使用してください。ダウンロードができたらGitHubに任意のリポジトリを作成してpushします。GitHubの使い方は11章を参照してください。

先ほど作成したトークンを使用してGit Cloneします。EC2のコンソールでGitHubのパスを指定しgit cloneします。

```
git clone https://github.com/{Githubのパス}.git
```

ユーザ名とパスワードを問い合わせされるので、GitHubユーザ名とtoken作成時のパスワードを指定します。

```
Username for 'https://github.com': {Githubユーザ名}
Password for 'https://{Github URL}@github.com': {token作成時パスワード}
```

ソースコードがEC2上に配置されれば成功です。

Purchase benefits

B-4 コンテナ環境の構築

B-4-1 Docker環境構築

Docker上でシステムを起動するので、Dockerをインストールしてサービスを起動します。「systemctl start」でサービスを起動するという意味になります。

```
sudo yum install docker
sudo systemctl start docker
```

Docker Composeをインストールします。Docker Composeを使用すると、単一のコマンドで複数のDockerコンテナを起動、停止、管理することができます。今回はfrontend、backend、DBをそれぞれのコンテナで起動して連携したいので、Docker Composeを使用します。Docker Composeのインストール方法はhttps://docs.docker.com/compose/install/standalone/にまとめられています。今回はLinux（EC2）にインストールするので、下記のコマンドを実施します。

```
sudo -i
curl -SL https://github.com/docker/compose/releases/download/v2.23.3/↵
docker-compose-linux-x86_64 -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
exit
```

sudo -iでrootユーザに変更します。curlでDocker Composeのプログラムをダウンロードして配置します。chmodで配置したプログラムに実行権限を付与します。「docker-compose -v」と実行して、Docker Composeのバージョンが表示されればOKです。

```
$ docker-compose -v
Docker Compose version v2.23.3
```


B-4-2【ハンズオン】Docker起動

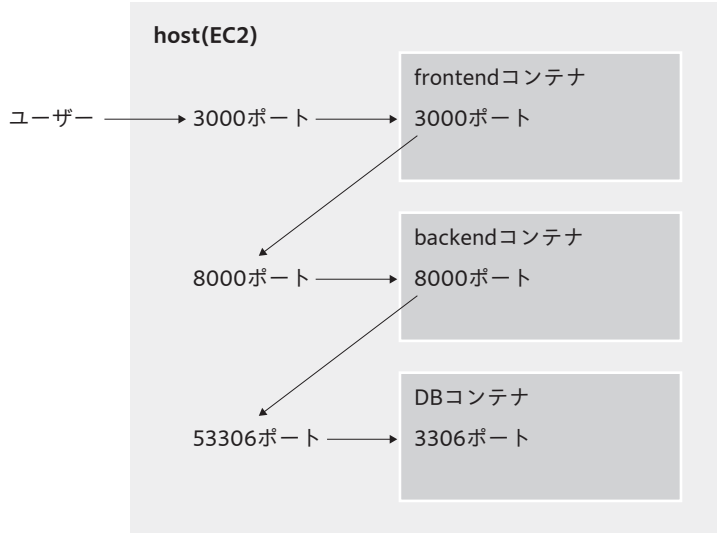
Docker Composeの定義を記載するために、docker-compose.ymlというファイルが必要になります。サンプルコードの中に含まれているので、中身を説明します。

```
version: "3.8"

services: ❶
  frontend:
    container_name: frontend #コンテナ名を指定
    build:
      context: frontend ❷
    tty: true
    volumes:
      - ./frontend:/frontend
    ports:
      - 3000:3000 ❸
    environment:
      - WATCHPACK_POLLING=true #ホットリロード（ソースコードを即時にアプリケーションに反映）
    extra_hosts:
      - host.docker.internal:host-gateway ❹
  backend:
    container_name: backend #コンテナ名を指定
    build:
      context: backend ❷
      args:
         environment_name=production #環境変数を指定。↵
本番用のbackend/config/settings/production.pyを参照する
    tty: true
    volumes:
      - ./backend:/backend
    ports:
      - 8000:8000 ❸
    depends_on:
      - db
    extra_hosts:
      - host.docker.internal:host-gateway ❹
```

```
  db:
    container_name: db #コンテナ名を指定
    image: mysql:8
    command:
      --collation-server=utf8mb4_0900_bin
      --transaction-isolation=READ-COMMITTED
    restart: always #host起動時に自動起動
    environment:
      MYSQL_ROOT_PASSWORD: password #初期パスワード
      TZ: Asia/Tokyo #タイムゾーンを東京に
    ports:
      - 53306:3306 ❸
```

サービスとしてfrontend、backend、dbという3サービスを定義します（❶）。サンプルコードの中でfrontendは/frontend/Dockerfile、backendは/backend/DockerfileにてDockerfileの起動を定義しています。contextにより、そのDockerfileの位置を指定します（❷）。次に示す図の通り、frontend、backend、dbはそれぞれ異なるコンテナで動作し、ネットワークも異なります。コンテナの動作元であるhost(EC2)ともネットワークが異なります。別のネットワークを呼び出せるように、hostとコンテナのポートを「ports:」で指定して紐づけます（❸）。また、frontendからbackend、backendからdbを呼び出せるよう、「extra_hosts: host.docker.internal:host-gateway」でコンテナからホストを呼び出せるようにします（❹）。



図B-4-1 ネットワークイメージ

Dockerイメージをビルドし、実行できる状態にします。

```
cd {Git Cloneしたdocker-compose.ymlのあるディレクトリ}
sudo docker-compose build
```

Dockerコンテナを起動します。

```
sudo docker-compose up -d
```

docker psコマンドで、Dockerの起動状態を確認することができます。docker-compose.ymlで定義したfrontend、backend、dbがそれぞれ起動していることを確認できます。

```
$ sudo docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED          ↵
STATUS         PORTS                                                    NAMES
20b91303b233   backend    "/bin/sh -c 'python …"   18 seconds ago   Up 5 seconds   ↵
 0.0.0.0:8000->8000/tcp, :::8000->8000/tcp                backend
8f8efb7291ee   frontend   "/bin/sh -c 'yarn st…"   19 seconds ago   Up 6 seconds   ↵
 0.0.0.0:3000->3000/tcp, :::3000->3000/tcp                frontend
97ac4670541e   mysql:8                  "docker-entrypoint.s…"   19 seconds ago   ↵
 Up 6 seconds   33060/tcp, 0.0.0.0:53306->3306/tcp, :::53306->3306/tcp   db
```

http://{IPアドレス}:3000/hello_backend_fetch_db で「hello !」と画面に表示されればOKです。{IPアドレス}の部分はAWSコンソールのEC2に表示されているパブリックIPv4アドレスを指定します。接続できない場合は、B-2-4項でセキュリティグループの設定を行ったときからIPアドレスが変わっている可能性があります。その場合、「セキュリティグループ」の設定から、新しい自分のIPを設定しなおします。

図B-4-2 グローバルIPアドレス

実はこの時点では、frontendは動作して「hello !」と画面に表示しているのですがfrontendからbackendを呼び出す際に、backendでエラーが発生しています。backend側にデータベースを作成していないからです。エラーの内容は、「sudo docker logs {コンテナ名} -t」で確認することができます。

```
sudo docker logs backend -t
（中略）
2023-12-03T01:32:23.119910722Z django.db.utils.OperationalError: (2013, "Lost ↵
connection to MySQL server at 'handshake: reading initial communication packet', ↵
system error: 11")
```

後ほど、backend側にデータベースを作成してエラーを解消していきます。

Purchase benefits

B-5 ビルド・デプロイ

バージョン管理システムと実行環境が決まったら、それぞれをつなぐ仕組みが必要です。具体的にはバージョン管理システムのソースコードをビルド（実行ファイルへ変換）してデプロイ（実行環境へ反映）します。

旧来のやり方として、ローカル環境でバージョン管理システムから取得したソースコードをビルドして手動でステージング／本番環境にデプロイする方法もあります。毎回その作業を行うことは手間がかかって大変です。

今回はCI/CDによる自動ビルド／自動デプロイについて学習していきましょう。

B-5-1 CI/CDとは

CI/CDとは「Continuous Integration（継続的インテグレーション）/ Continuous Delivery（継続的デリバリー）」の略称です。バージョン管理ツールで管理しているソースコードの変更を常にテストし、自動で実行環境に適用できるような状態にしておく開発手法のことを指します。

CI/CDの大まかな流れは次のようになっています。

① コードの変更
- 開発者がソースコードを変更する

② バージョン管理
- 変更内容がバージョン管理システムにコミットされる

③ 自動ビルド
    - CIサーバーが変更を検知して、自動的にビルドを開始する
④ 自動テスト
    - ビルドが成功した場合、テストが自動的に実行される
⑤ 自動デプロイ
    - テストが成功した場合、CDサーバーが自動的にデプロイを行う

必ずこの流れで行わなければならないということではありません。例えば、プロジェクトによっては、CI/CDでの自動テストは行わないという判断もあります。プロジェクトによって必要な実施事項と流れを選択することが重要です。

B-5-2 CI/CDツールの選定

ビルド／テスト／デプロイはセットでのサービス選定が必要です。代表的なサービスとしてはJenkins、CircleCI、AWS Codepipelineなどがあります。バージョン管理システムや実行環境との相性を含めて検討するとよいでしょう。例えば、実行環境にAWSを使用するのであれば、同じAWSのサービスであるAWS Codepipelineでの対応がスムーズと考えられます。

B-5-3 CI/CDソースコード取得

CI/CDでは特定ブランチに更新があったときに、自動でその内容が実行環境に反映されるようにします。そこで、ブランチと実行環境の紐づけを行う必要があります。例えば、次のように紐づけます。

- developブランチ→ステージング環境
- mainブランチ→本番環境

もちろん、必ずしも、この構成でないといけないというわけではないです。releaseブランチを活用する方法もあるので、プロジェクトの状況に応じて適宜読み替えてください。

また、Gitにpushされたタイミングで自動でCI/CDが実行されるのか、手動での承認が必要なのかを検討する必要があります。自動にしておくと、例えばmainブランチに誤ってpushしてしまった場合に、自動で本番環境にデプロイされてしまうという事故が想定されます。その対策として、ステージング環境は自動実行するよう設定して、本番環境は手動で実行にするなどの対応方法が考えられます。

B-5-4 CI/CD ビルド

ビルドに使用するマシンの選定や環境変数の設定を行います。CI/CDにおいてビルド時間を考慮することは次の理由から重要です。

開発プロセスの効率化

CI/CDは、開発者がコードの変更を早期にテストし、フィードバックを得ることを可能にします。ビルド時間が短い場合、開発者はコード変更の結果を素早く確認できます。これにより、問題が早期に発見され、修正が迅速に行われます。

コスト削減

CI/CDサービスは従量課金制（動いている時間だけお金がかかる）ことが多いです。ビルド時間を短縮することで、コスト削減に繋がります。

ビルド時間を短縮するために、ビルドするのに充分なCPUやメモリ、ディスク容量を選択する必要があります。スペックとビルド実行時間を比較しながら、プロジェクトにとって最適なビルドマシンの選定をするとよいでしょう。また、ビルドはビルドスクリプトによって実行します。ビルドスクリプトの中で、ライブラリや環境変数の指定などを指定します。自動実行するのでスクリプト化してGit上などで管理しておきます。

B-5-5 CI/CD デプロイ

デプロイ対象のリソースを設定します。ソースコードのブランチと実行環境の対象が紐づくように設定します。

特に本番環境時には、デプロイにあたってシステム停止をするのか、停止せずにリリースするのかを検討する必要があります。停止せずにリリースする方法としてブルーグリーンデプロイというものがあります。これは、旧アプリケーションと新アプリケーションを平行稼働させて、徐々に移行していくという方法です。システム要件に応じて検討、テストしてみてください。

最後に、デプロイ完了したら、メールなどで開発者に通知がいくようにするとよいでしょう。

B-6 実行

B-6-1 データ投入

コンテナ上に構築したMySQLにデータを投入していきます。MySQLでの操作を行うには、MySQLクライアントという操作用のツールが必要です。MySQLコンテナ内にMySQLクライアントの機能があるので、今回はそれを使用します。

MySQLのコンテナに接続し、MySQLコマンドにてappデータベースを作成します。「sudo docker exex -it コンテナ名 /bin/bash」でコンテナ内に移動してMySQLコマンドを実行することが可能になります。コンテナ名はdocker-compose.ymlで"db"と指定していました。

```
sudo docker exec -it db /bin/bash
bash-4.4# mysql -u root -p
Enter password: password

mysql> CREATE DATABASE app;
Query OK, 1 row affected (0.00 sec)

mysql> exit
Bye

bash-4.4# exit
exit
```

同様に、Backendコンテナに接続し、マイグレーションを実施します。

```
sudo docker exec -it backend /bin/bash
python manage.py migrate --settings config.settings.production
exit
```

再度、MySQLのコンテナに接続し、レコードを登録します。

```
sudo docker exec -it db /bin/bash
bash-4.4# mysql -u root -p
Enter password: password

mysql> INSERT INTO `app`.`hello` (`world`) VALUES ('123');
Query OK, 1 row affected (0.00 sec)

mysql> exit
Bye

bash-4.4# exit
exit
```


データベース作成とレコード作成を実施し、B-4-3項で発生していたエラーが解消したのでDockerを再起動します。

```
sudo docker-compose restart
```

B-6-2 画面表示

ブラウザで、http://{IPアドレス}:3000/hello_backend_fetch_db で「hello 123」と画面に表示されればOKです。

最後に、ソースコードの修正を反映してみましょう。frontend/app/hello_backend_fetch_db/page.tsx を「hello {data.message}!」から「new hello {data.message}!」に変更してCommit/Pushします。その後、EC2でgit pullしてビルド、起動します。

```
git pull
sudo docker-compose build
sudo docker-compose up -d
```

http://{IPアドレス}:3000/hello_backend_fetch_db を開いて「new hello 123!」と表示されればOKです。

ここまで、実際に本番環境を構築しながら、環境構築にあたって必要な検討事項を学びました。開発現場では、バージョン管理ツール、CI/CDツール、実行環境の選定から始める必要があります。選定のポイントとしては、コストや、自動化のしやすさ、スケールアップのしやすさ、既存インフラとの接続のしやすさがありますので、比較検討していくとよいでしょう。