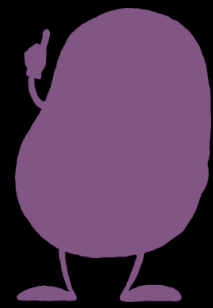
AWSとGitHubを使ってみよう勉強会

~ 第6回 AWS ECS Fargate環境の構築 ~

株式会社 豆蔵
ビジネスソリューション事業部



updated:2023/9/18

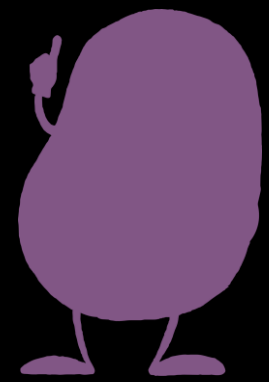
本日の内容

- 前回の課題の解説(60分)

※「前回の課題の回答」は解説の中で一緒に説明します

画面キャプチャやコマンド操作等はGitHubのssi-mz-studygroupユーザで行った例となります。キャプチャやコマンド等の該当部分は自分のユーザIDに読み替えてください
・ssi:simple server infra
・mz:mamezou

前回の課題の解説
・サンプルアプリの設定による動作変更
・課題の全体像とECS Fargateとは



Step1. REST APIで返す値を環境変数で指定できるようにサンプルアプリを修正する 再掲

- 変更内容
  - 環境変数CONFIG_VALの設定が
    - ある場合は、その設定値をREST APIで返す
    - ない場合は、デフォルト値として“Hello"

- 変更方法
  - MicroProfile Configを使って設定ファイルの設定値を環境変数で上書きできるようにする
  - MicroProfile Configについては以下を参照
    - https://developer.mamezou-tech.com/msa/mp/cntrn06-mp-config/

- ご自身でやってもらう方がいいと思いいいますがMicroProfileは目的外なので、ひな形リポジトリに回答例をアップしています
  - src/main/java/sample/HelloResource.java(変更)
  - src/main/resources/META-INF/microprofile-config.properties(追加)
  - src/test/java/sample/HelloResourceTest(変更)

対応後のサンプルアプリ

```java
import
org.eclipse.microprofile.config.inject.ConfigProperty;
…
@ApplicationScoped
@Path("hello")
public class HelloResource {
    @Inject
    @ConfigProperty(name = "config.val")
    private String configValue;
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return configValue;
    }
}
```

injection

MicroProfile Configの機能を利用

・/META-INF/microprofile-config.properties

```
config.val=Hello
```

override

・環境変数

```
CONFIG_VAL=Chanaged
```

- 同じキーが設定されている場合、優先度が高い設定源が優先される
- 環境変数は英字は小文字、記号は.(ドット)に置換した変数としても評価される

設定源	優先度
システムプロパティ	400
環境変数	300
/META-INF/microprofile-config.properties	100
config_ordinalが設定されている設定源	config_ordinalの値

※config_ordinalは任意の設定源に設定可能

サンプルアプリの設定の上書き実行

※GitHub Actionsでサンプルアプリをコンテナイメージにビルドしてコンテナレジストリにpushした後の操作

■ デフォルト設定でのコンテナ起動

```
# 最新のコンテナイメージの取得
sudo docker pull ghcr.io/[自分のGitHubユーザ名]/hello-app:latest
# コンテナの起動
sudo docker run -d --rm --name hello-app -p 7001:7001 ghcr.io/[自分のGitHubユーザ名]/hello-app:latest
# 起動ログの確認
sudo docker logs hello-app
# レスポンスの確認
curl localhost:7001/api/hello
> Helllo
```

■ 設定を変更してコンテナ起動

```
# コンテナの起動
sudo docker run -d --rm --name hello-app -p 7001:7001 -e CONFIG_VAL=Changed ghcr.io/[自分のGitHubユーザ名]/hello-app:latest
# 起動ログの確認
sudo docker logs hello-app
# レスポンスの確認
curl localhost:7001/api/hello
> Changed
```

latestタグのpullは意味あるの？

■ デフォルト設定でのコンテナ起動

```
# 最新のコンテナイメージの取得
sudo docker pull ghcr.io/[自分のGitHubユーザ名]/hello-app:latest
# コンテナの起動
sudo docker run -d --rm --name hello-app -p 7001:7001 ghcr.io/[自分のGitHubユーザ名]/hello-app:latest
# 起動ログの確認
sudo docker logs hello-app
# レスポンスの確認
curl localhost:7001/api/hello
> Helllo
```

docker runで一緒にpull してくれるのにわざわざpullする必要あるの？

■ 設定を変更してコンテナ起動

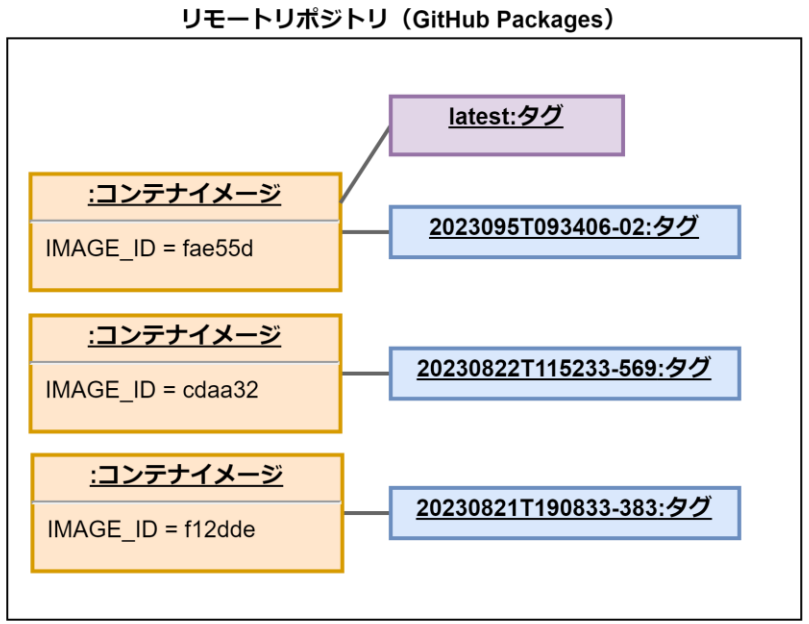
```
# コンテナの起動
sudo docker run -d --rm --name hello-app -p 7001:7001 -e CONFIG_VAL=Changed ghcr.io/[自分のGitHubユーザ名]/hello-app:latest
# 起動ログの確認
sudo docker logs hello-app
# レスポンスの確認
curl localhost:7001/api/hello
> Changed
```

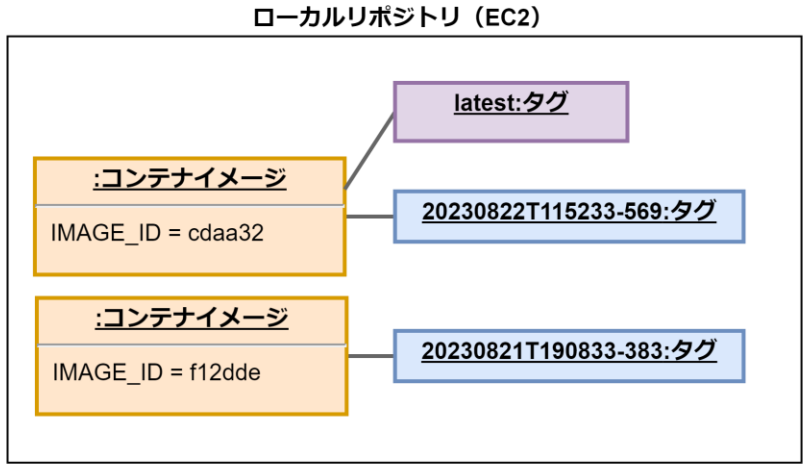
こっちではなんでdocker pullしてないの？

latestタグが更新されない理由

さっきの答えは

- docker pullはリモートリポジトリを常に見に行くがdocker runはローカルリポジトリに該当イメージがないときだけしかリモートリポジトリは見に行かない
- よってdocker runでローカルリポジトリにlatestのイメージがある場合、latestが更新されない





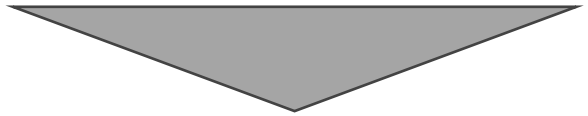
latestタグを使うとより深刻な問題が起きる可能性があります。それはなんでしょうか？

MAMEZOU-TECH

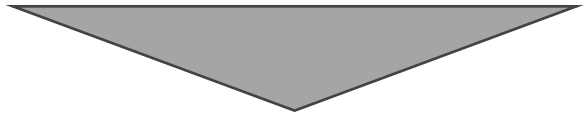
latestタグの問題（あまり使っちゃダメ）

latestタグの対象は移動する。なので

- 断面が不定になる
  - kubernetesやECSなどのコンテナオーケストレーションが持つ指定した断面に環境を戻すrollbackができなくなる
- 異なるバージョンが混じる
  - コンテナオーケストレーションでオートスケールやオートヒーリングされたときのlatestは既にデプロイされているコンテナイメージと異なっている可能性がある
  - これによりクラスタを構成するコンテナインスタンスごとに挙動が変わってしまう恐れがある



サンプルではコンテナイメージをピンポイントで指定できるようにタイムスタンプのタグをつけている



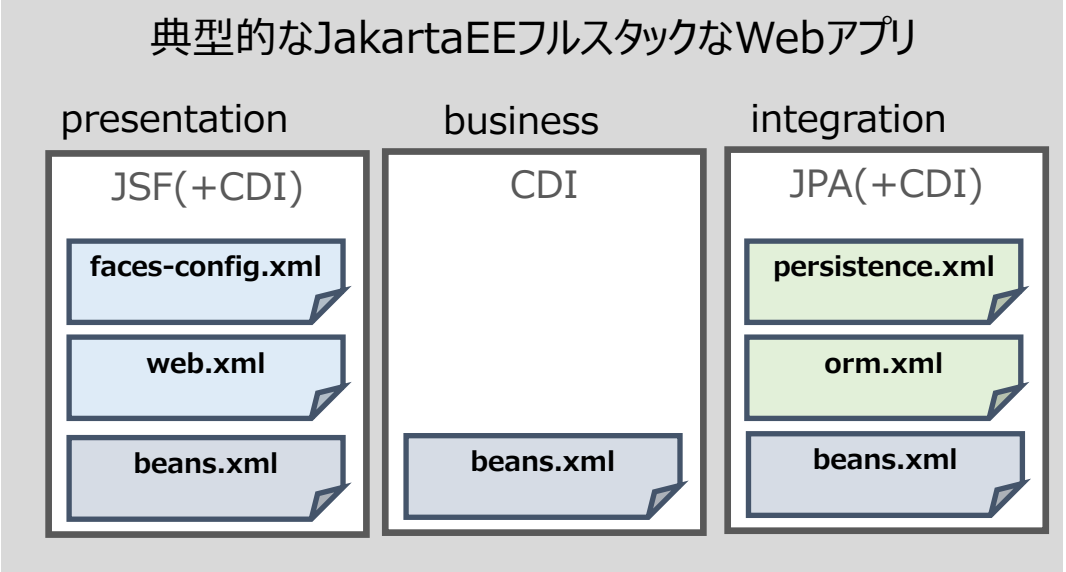
dockerイメージをリポジトリ管理する際のタグ戦略は非常に重要となる
（他のタグ戦略としては通番を振ったりコミットハッシュを使ったりするのが典型）

コンテナで環境変数の利用は定石- MicroProfile Config 登場背景

Configに対する2つのモチベーション

多すぎる設定ファイル

- JakartaEEには設定ファイルに対する共通的な仕様が定義されていない
- web.xmlやbeans.xml, persistence.xmlなど仕様ごとに必要な設定ファイルが存在する（徐々にアノテーションでも定義できるようになってはきてはいる）
- アプリケーション全体レベルでの設定の見通しが悪く、設定漏れや配置漏れも起きやすくハッキリいって不便（Springのapplication.ymlが裏山）



多すぎ、散らばりすぎ、勘弁して・・

CloudNativeへの追従

- マイクロサービスアーキテクチャのbest practiceとして知られる*The Twelve-Factor App*の“Ⅲ 設定”では「設定を環境変数に格納する」ことを推奨している
- これまでJakartaEEの仕様でこれを実現できるものはなかった（SpringBootでは普通にできるけど）

THE TWELVE-FACTOR APP

-Ⅲ 設定-

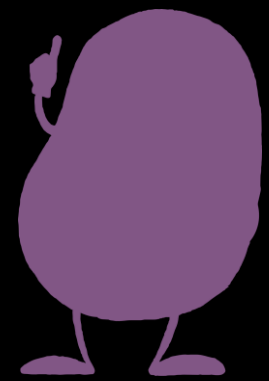
✓ 設定はコードや設定ファイルに含めず、環境変数で設定するようにしましょう。
✓ なぜならコードに含めてしまうと環境ごとにビルドが必要になり、設定ファイルの場合は環境ごとにファイルが必要となりスケールしにくくなるためです。
✓ 環境変数であればOSが異なっても取得方法は基本的に統一できます

引用:Twelve-Factor Appを噛み砕いてみた - Qiita @supreme0110

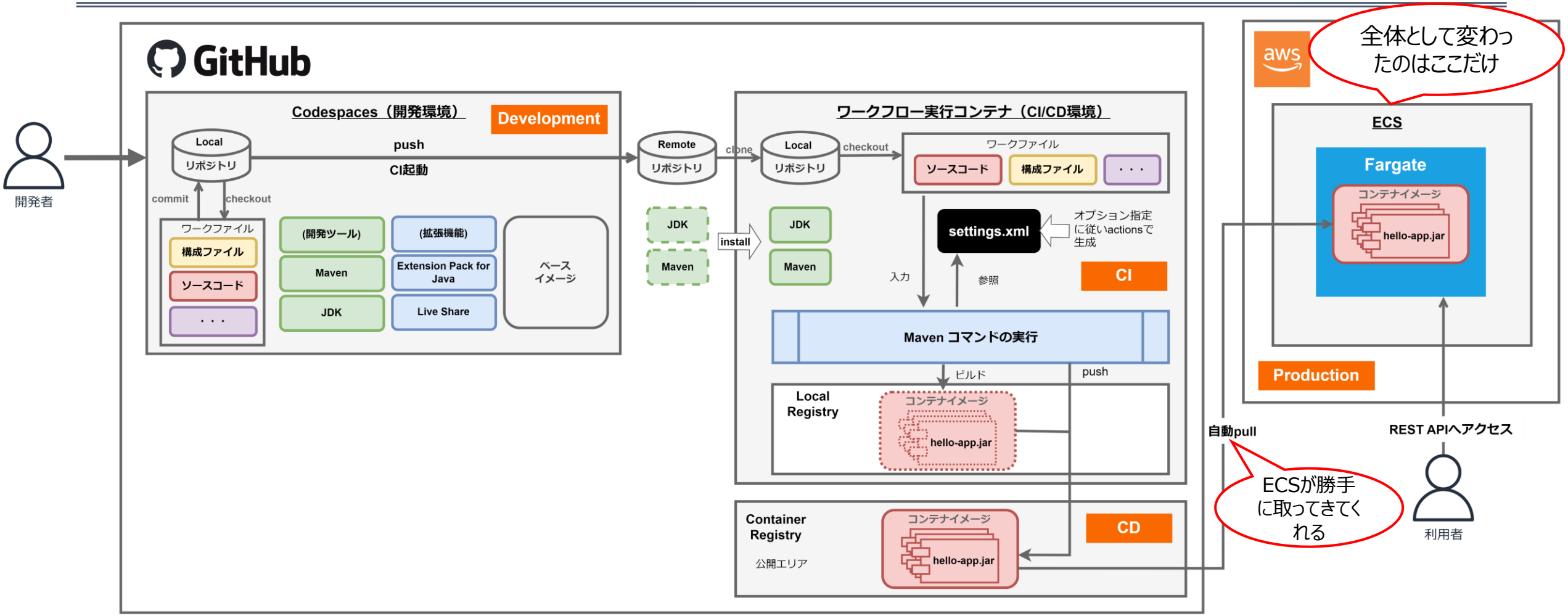
前回の課題の解説

・サンプルアプリの設定による動作変更

・課題の全体像とECS Fargateとは



これでまでの課題でやってきた全体像 – 振り返り



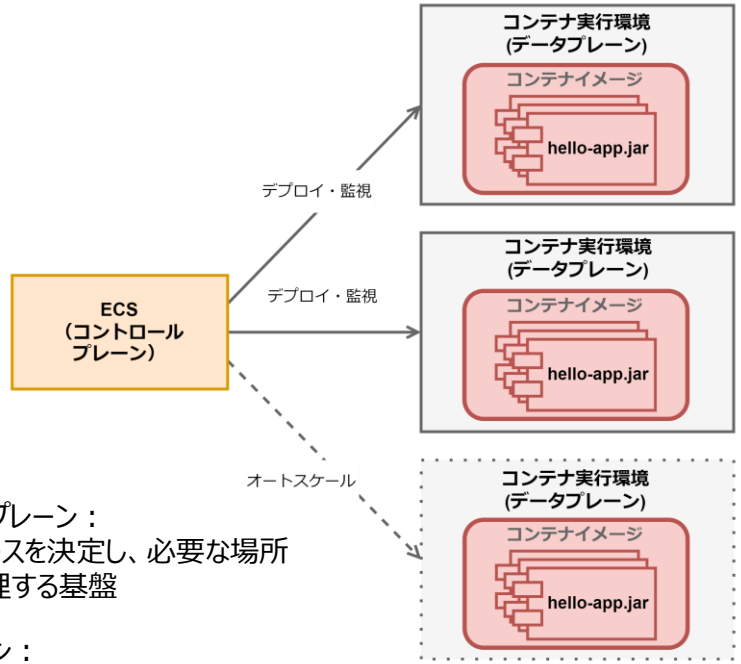
- ✓ コンテナの実行環境がEC2からECSに替わっただけ
- ✓ だがしかし、EC2で行っていたDocker環境の構築やコンテナの起動停止の運用周りは一切不要に

MAMEZOU-TECH

ECS Fargateとは

＜ECSとは＞

Amazon Elastic Container Service (Amazon ECS) は、コンテナ化されたアプリケーションのデプロイ、管理、スケーリングを容易にするフルマネージドコンテナオーケストレーションサービスです。※AWS公式の完全引用

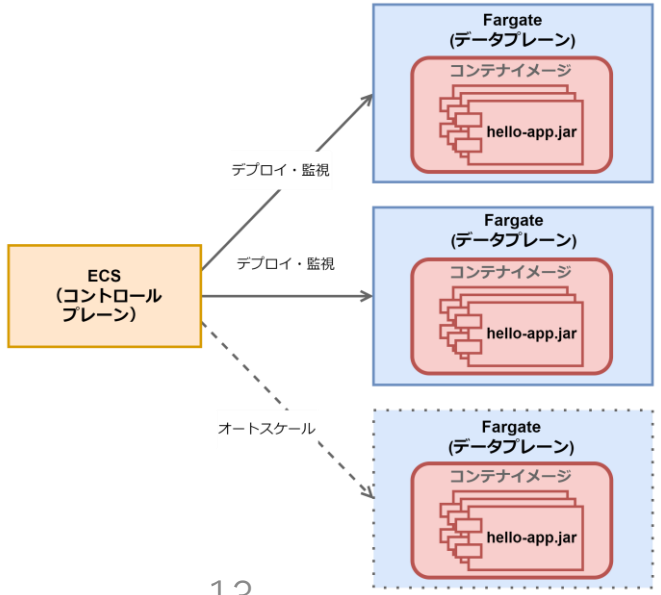


コントロールプレーン：
必要なリソースを決定し、必要な場所で実行・管理する基盤

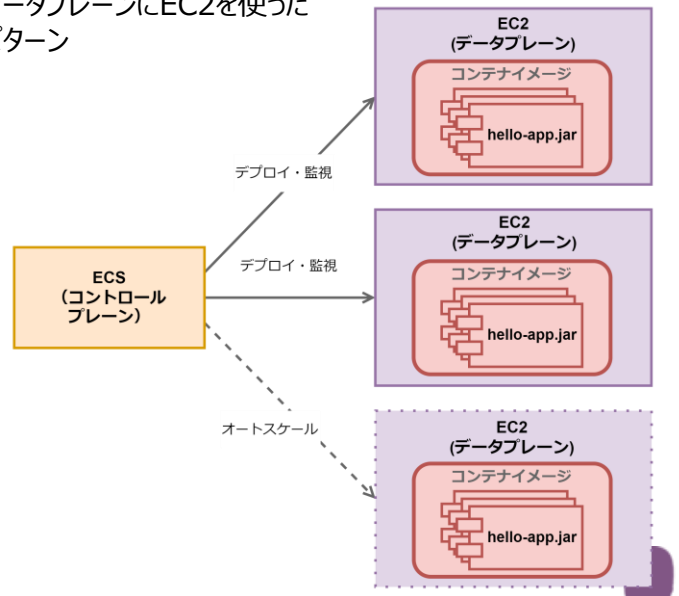
データプレーン：
リソースを指示されたとおりに実行する基盤

＜Fargateとは＞

AWS Fargate はAmazon ECSで使用できるテクノロジーであり、サーバーやAmazon EC2インスタンスのクラスターを管理することなくコンテナを実行できます。Fargate を使用すると、コンテナを実行するために仮想マシンのクラスターをプロビジョニング、設定、スケールする必要はありません。
これにより、サーバータイプの選択、クラスターをスケールするタイミングの決定、クラスターのパッキングの最適化を行う必要がなくなります。
※AWS公式の完全引用
⇒要はECSのデータプレーンに使えるサーバレス環境。EKSでも利用可



データプレーンにEC2を使ったパターン



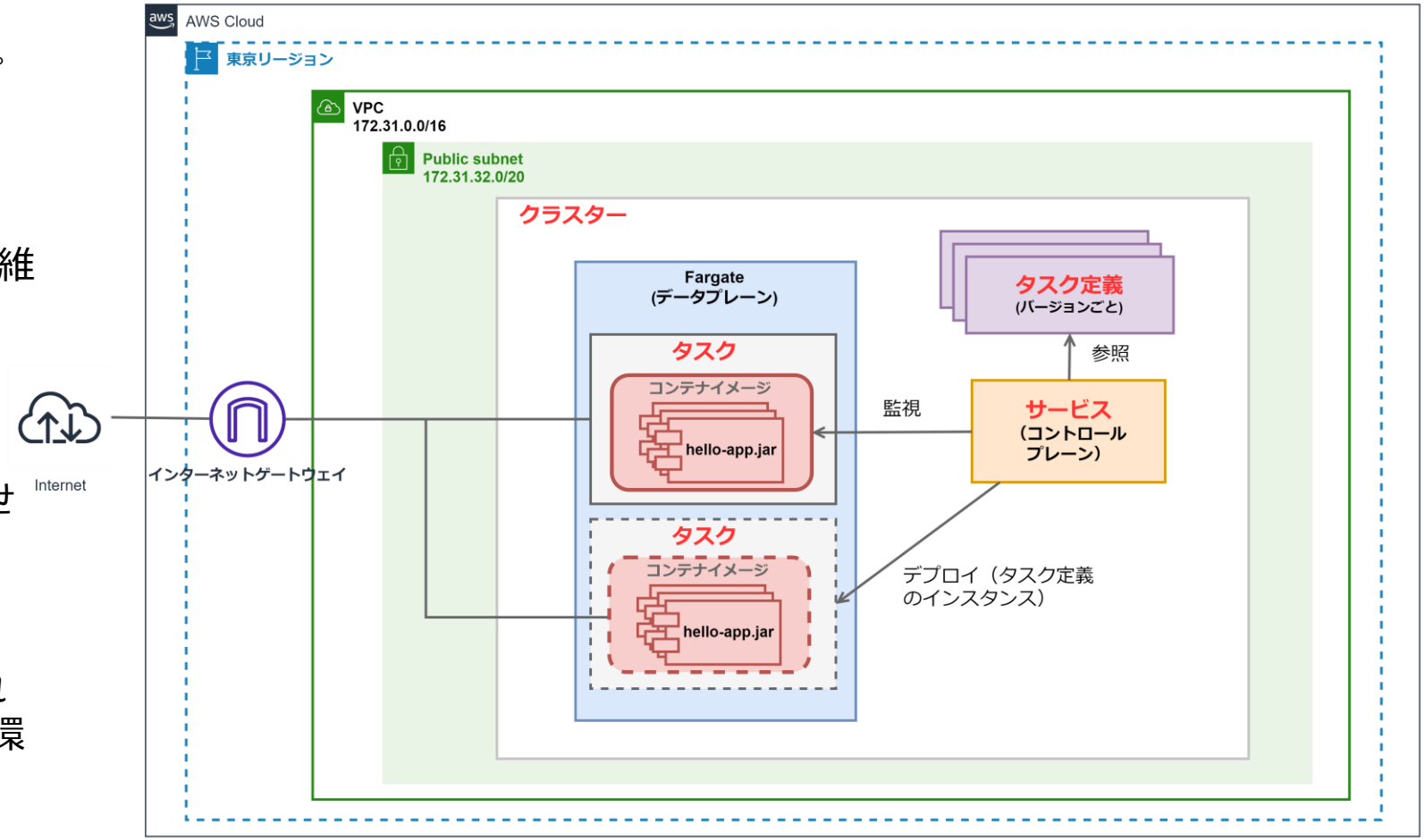
クラスター／サービス／タスク

■クラスター
- ECSリソースを管理するための論理グループ（なだけ）

■サービス
- ECS（コントロールプレーン）の実体
- タスクを監視しサービスで定義された状態を維持するようにタスクのローリングアップデートやオートスケール、オートヒーリングなど行う

■タスク定義
- なにを（コンテナイメージ）どのように（割り当てマシンリースや環境変数など）動作させるかを定義したもの
- 定義は世代ごとに管理される

■タスク
- タスク定義をもとにサービスにより実体化されたもの（サービスにより割り当てられた実行環境でコンテナが実行されている）



MAMEZOU-TECH

実際にやってみる

クラウド環境ではログの出力先は標準出力が基本

THE TWELVE-FACTOR APP

-XI.ログ-

✓ファイル出力ではなく標準出力に出力させ、ツールで1箇所に集約するようにしましょう。
✓なぜならスケールイン時に仮想サーバごと消える可能性があることと、環境(OS、IDE、ローカル/本番)が異なっても標準出力は常に行えるためです。

⇒そもそもサーバーレスでは基本的にローカルディスクは使えない

引用:Twelve-Factor Appを噛み砕いてみた - Qiita @supreme0110

おしまい。お疲れまでした