

使って学ぶ

# AWS ECS入門

FargateとLaravelでコンテナデプロイを体験

Project Aiiro 著



信頼性と拡張性の高い

Amazon Elastic Container Serviceを用いた、  
Webアプリケーションデプロイ

# 使って学ぶ AWS ECS

## 入門

— Fargate と Laravel でコンテナデプロイを体験 —

寺田晃大 著

2020 年 12 月 26 日 ver 1.0

### **■免責事項**

本書は情報の提供のみを目的としています。

したがって、本書を用いた開発、制作、運用は、必ずご自身の責任と判断によって行ってください。

これらの情報による開発、制作、運用の結果について、著者はいかなる責任も負いません。

### **■表記関係**

本書に記載されている会社名、製品名などは、一般に各社の登録商標または商標、商品名です。

会社名、製品名などについて、本文中では™、®、©などのマークは表示していません。

# はじめに

本書をお読みいただきありがとうございます。本書では、次の内容を説明します。

- ECS の利用に必要な AWS のネットワーク環境の構築
- サンプルのアプリケーションを Laravel で作成
- アプリケーションを ECS にデプロイ

本書を通して、ECS を扱う上での基本的な流れを伝えられれば幸いです。

## 対象読者

本書は次のような方を対象読者として想定しています。

- AWS をある程度利用した経験がある。
- ローカル開発環境で Docker を利用したことはあるが、ECS を利用したことがない。
- ECS を試したいが、ECS でアプリケーションをデプロイする方法がわからない。

## 本書で扱わないこと

本書では、目次の内容以外に、次の内容は扱いませんのでご注意ください。

- AWS アカウントの初期設定
- AWS のリソースを作成するための IAM の初期設定
- AWS CLI や Docker などの各種ツールの初期設定

## 開発環境

本書では AWS CLI、Docker や Docker Compose を使用します。各ツールの公式ホームページを参照してインストールしてください。

筆者の執筆時点の開発環境を次に記載しますので、ご参考にしてください。

- macOS: Catalina 10.15.7
- AWS CLI: 2.0.48
- Docker Desktop: 2.4.0.0
- Docker version: 19.03.13
- docker-compose version 1.27.4



# 目次

<b>はじめに</b>	i
対象読者 . . . . .	i
本書で扱わないこと . . . . .	i
開発環境 . . . . .	i
<b>第1章 Amazon ECS とは</b>	<b>1</b>
1.1 Amazon ECS クラスターとは . . . . .	1
1.2 Amazon ECS サービスとは . . . . .	1
1.3 Amazon ECS タスク定義とは . . . . .	2
1.4 Amazon ECR とは . . . . .	2
1.5 Fargate とは . . . . .	2
1.6 ECS のイメージ . . . . .	2
<b>第2章 ECS にデプロイするアプリケーションを用意する</b>	<b>5</b>
2.1 Web アプリケーションのネットワーク構成 . . . . .	5
2.2 Docker で Laravel アプリケーションを作成 . . . . .	6
2.2.1 プロジェクト用のディレクトリを作成 . . . . .	6
2.2.2 Composer のイメージ用の Dockerfile を作成 . . . . .	6
2.2.3 Laravel アプリケーションの生成 . . . . .	7
2.3 コマンドを作成する . . . . .	8
2.3.1 PHP のコンテナイメージを用意する . . . . .	8
2.3.2 コンテナで artisan コマンドを実行する . . . . .	10
2.4 Web ページを表示する . . . . .	13
2.4.1 Nginx のコンテナを用意する . . . . .	14
2.4.2 Docker Compose でアプリケーションを起動する . . . . .	16
2.4.3 アプリケーションの起動 . . . . .	18
<b>第3章 ECR にイメージを登録する</b>	<b>21</b>
3.1 ECR にリポジトリを作成する . . . . .	21
3.1.1 Nginx イメージのリポジトリを作成 . . . . .	21
3.1.2 PHP イメージのリポジトリを作成 . . . . .	24
3.2 リポジトリにコンテナイメージを登録する . . . . .	25
3.2.1 ECR にログインする . . . . .	25
3.2.2 プッシュ前にコンテナイメージをビルドする . . . . .	27
3.2.3 コンテナイメージにタグをつける . . . . .	27

---

3.2.4	ECR にコンテナイメージをプッシュ . . . . .	28
<b>第 4 章</b>	<b>AWS のネットワークを構築する</b>	<b>31</b>
4.1	VPC を作成する . . . . .	31
4.2	パブリックサブネットを作成する . . . . .	33
4.2.1	サブネットを作成する . . . . .	33
4.2.2	インターネットゲートウェイを作成 . . . . .	34
4.3	VPC にインターネットゲートウェイをアタッチする . . . . .	36
4.3.1	ルートテーブルを作成する . . . . .	38
4.3.2	ルートテーブルにルートを追加する . . . . .	40
4.3.3	サブネットとルートテーブルを紐付ける . . . . .	42
4.3.4	「パブリック IP アドレスの自動」を有効にする . . . . .	44
4.4	セキュリティグループを設定する . . . . .	46
4.5	ECS で使用する IAM ロールを作成する . . . . .	49
4.5.1	ECS タスク用の IAM ロールを作成する . . . . .	49
4.5.2	ECS サービスにリンクしたロールを作成する . . . . .	54
4.6	CloudWatch の設定 . . . . .	54
4.6.1	CloudWatch にロググループを作成する . . . . .	54
<b>第 5 章</b>	<b>バッチ処理を実行する</b>	<b>57</b>
5.1	クラスターを作成する . . . . .	57
5.1.1	クラスターの作成画面を開く . . . . .	57
5.1.2	クラスターの設定を選択・入力する . . . . .	58
5.2	タスク定義を用意する . . . . .	60
5.2.1	タスク定義ファイルの作成 . . . . .	60
5.2.2	AWS Systems Manager パラメータストアに秘密情報 報を登録する . . . . .	62
5.2.3	タスク定義の登録 . . . . .	65
5.3	タスクを実行する . . . . .	67
<b>第 6 章</b>	<b>Web アプリケーションをデプロイする</b>	<b>71</b>
6.1	タスク定義を用意する . . . . .	71
6.1.1	タスク定義ファイルの作成 . . . . .	71
6.1.2	AWS Systems Manager パラメータストアに秘密情報 報を登録する . . . . .	73
6.1.3	タスク定義の登録 . . . . .	76
6.2	アプリケーションのデプロイ (ALB なし) . . . . .	79
6.2.1	サービスの作成 . . . . .	79
6.2.2	動作確認 . . . . .	80
6.2.3	サービスの削除 . . . . .	83

---

6.3	アプリケーションのデプロイ (ALB あり) . . . . .	85
6.3.1	パブリックサブネットを追加する . . . . .	85
6.3.2	サブネットとルートテーブルを紐付ける . . . . .	87
6.3.3	「パブリック IP アドレスの自動」を有効にする . . . . .	88
6.3.4	ALB (ロードバランサー) を用意する . . . . .	89
6.3.5	ターゲットグループの ARN を確認 . . . . .	94
6.3.6	サービスの作成 . . . . .	94
6.3.7	動作確認 . . . . .	95
6.3.8	サービスの削除 . . . . .	99
6.3.9	AWS リソースの削除 . . . . .	100



## 第 1 章

# Amazon ECS とは

Amazon ECS(Elastic Container Service)<sup>\*1</sup>は、AWS が提供するフルマネージド型のコンテナオーケストレーションサービスです。AWS が提供する EC2 との比較すると、EC2 はインスタンスを起動してから、アプリケーションの実行に必要なミドルウェアなどをインストールする必要があります。一方、ECS では、事前に用意した Docker のコンテナイメージを元に、アプリケーションの実行環境を構築できます。

### 1.1 Amazon ECS クラスターとは

ECS ではクラスター、サービス、タスク、タスク定義といった要素が存在します。その中でクラスター<sup>\*2</sup>は、サービスまたはタスクを内包する論理グループです。

### 1.2 Amazon ECS サービスとは

サービス<sup>\*3</sup>の役割は、タスクを管理することです。サービスは作成時に、タスク定義から指定された数のタスクを起動します。もし何らかの理由でタスクが終了した場合、サービスは新たなタスクを起動し、タスクが必要な数となるように維持します。またサービスは、ロードバランサーを使用する際に、ロードバランサーとの紐付けの対象となります。

<sup>\*1</sup> <https://aws.amazon.com/jp/ecs/>

<sup>\*2</sup> [https://docs.aws.amazon.com/ja\\_jp/AmazonECS/latest/developerguide/clusters.html](https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/clusters.html)

<sup>\*3</sup> [https://docs.aws.amazon.com/ja\\_jp/AmazonECS/latest/developerguide/ecs\\_services.html](https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/ecs_services.html)

## 1.3 Amazon ECS タスク定義とは

---

タスク定義<sup>\*4</sup>は、タスクを作成する際の元となる定義です。タスク定義には、コンテナが使用する CPU やメモリ、ネットワークモードなどを設定します。

また、タスク定義には、複数のコンテナの設定を記述できます。たとえば、Nginx と PHP のコンテナを 1 つのタスク定義内に登録できます。このタスク定義からタスクが作られ、そのタスクの中でコンテナが動作します。

## 1.4 Amazon ECR とは

---

Amazon ECR(Elastic Container Registry)<sup>\*5</sup>は、AWS が提供する Docker のコンテナレジストリです。ECR に登録したコンテナイメージは、ECS のタスクで使用します。

## 1.5 Fargate とは

---

Fargate<sup>\*6</sup>とは、コンテナ向けのサーバレスコンピューティングエンジンのことです、コンテナの実行環境です。ECS には EC2 タイプと Fargate タイプがありますが、Fargate タイプは EC2 タイプと異なり、インスタンスタイプの選択や更新をする必要がありません。その代わり、Fargate を使用する際は、CPU やメモリなどのリソースを設定します。

## 1.6 ECS のイメージ

---

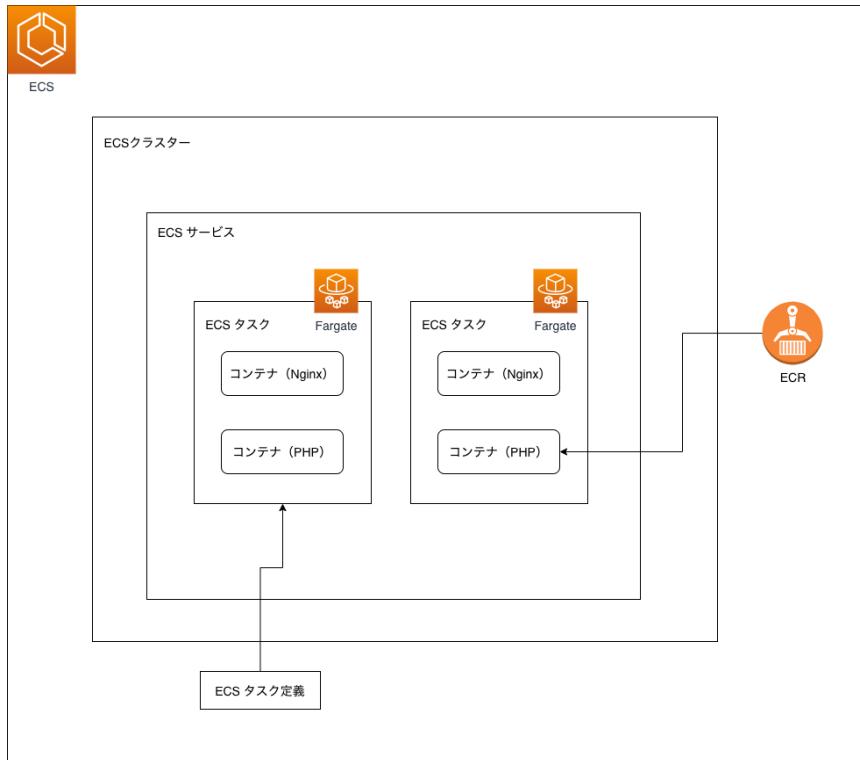
ECS のイメージは次の図のようになります（図 1.1）。

---

<sup>\*4</sup> [https://docs.aws.amazon.com/ja\\_jp/AmazonECS/latest/developerguide/task\\_definitions.html](https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/task_definitions.html)

<sup>\*5</sup> <https://aws.amazon.com/jp/ecr/>

<sup>\*6</sup> <https://aws.amazon.com/jp/fargate/>



▲図 1.1



## 第 2 章

# ECS にデプロイするアプリケーションを用意する

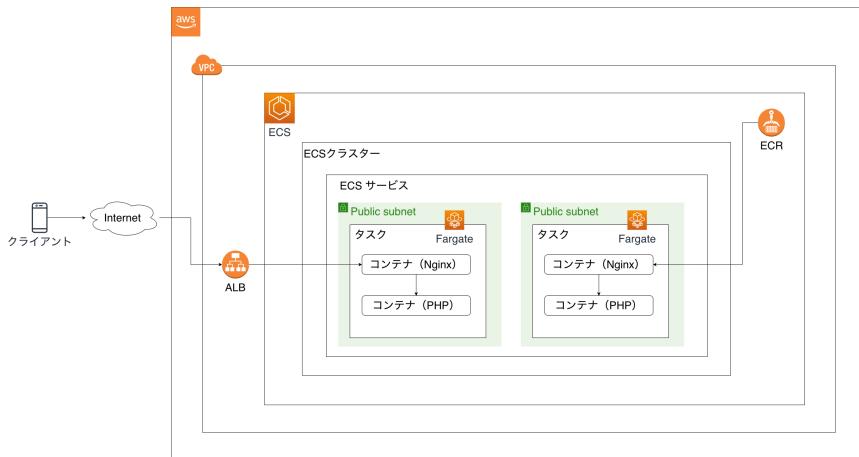
本書では、ECS で実行するバッチ処理と Web アプリケーションに、PHP のフレームワークである Laravel<sup>\*1</sup> を使用します。本章では、ECS で使用する Laravel アプリケーションの作成方法を説明します。

## 2.1 Web アプリケーションのネットワーク構成

本書で使用する Laravel アプリケーションは、Nginx と PHP のコンテナを組み合わせて動作します。アプリケーションを作成する前に、次の図（図 2.1）で ECS にアプリケーション（ALB あり）をデプロイしたときの、ネットワーク構成を確認しておきましょう。

---

<sup>\*1</sup> <https://laravel.com/>



▲図 2.1

## 2.2 Dockerで Laravel アプリケーションを作成

### 2.2.1 プロジェクト用のディレクトリを作成

まずはプロジェクト用のディレクトリを作成します。以降、このディレクトリをプロジェクトのルートディレクトリとして扱います。

```
$ mkdir ecs-hands-on
$ cd ecs-hands-on
```

### 2.2.2 Composer のイメージ用の Dockerfile を作成

Laravel プロジェクトのひな型は、PHP のパッケージ管理ツールである、「Composer」<sup>\*2</sup>を使って生成します。

Docker Hub<sup>\*3</sup>で公開されているイメージを用いることもできますが、今回は Composer を PHP:7.4 のイメージを元に、Composer をインストールしたイメージを作成してみましょう。

まずは、イメージの設定を記述する Dockerfile を、docker/composer 配下に作成します。

```
$ mkdir -p docker/composer
$ touch docker/composer/Dockerfile
```

作成した Dockerfile を、次のように編集します。

<sup>\*2</sup> <https://getcomposer.org/>

<sup>\*3</sup> [https://hub.docker.com/\\_/composer](https://hub.docker.com/_/composer)

## ▼ docker/composer/Dockerfile

```
FROM php:7.4-fpm

WORKDIR /application

RUN apt-get update && apt-get install -y \
    git \
    zip \
    unzip \
&& curl -sS https://getcomposer.org/installer | php \
&& mv composer.phar /usr/local/bin/composer
```

**Composer のイメージのビルド**

次に、この Dockerfile を使ってイメージをビルドします。

```
$ docker build -t ecs-hands-on/composer:latest -f ./docker/composer/Dockerfile .
```

docker images を実行して、ビルドしたイメージを確認します。

```
$ docker images | head -n 2
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
ecs-hands-on/composer    latest    f7c0d502bb45  2 minutes ago  407M
>B
```

**2.2.3 Laravel アプリケーションの生成**

この Composer のイメージを使って、Laravel プロジェクトを生成します。

次のコマンドでは、-v オプションでカレントディレクトリをコンテナ内の /application にマウントしています。そして、コンテナが起動したときに composer create-project を実行します。

```
$ docker run -v $(pwd):/application ecs-hands-on/composer:latest composer create-project --prefer-dist laravel/laravel src "8.*"
```

実行した結果、ローカルに src ディレクトリが作られ、その中に Laravel のプロジェクトが生成されていれば OK です。

## ▼ Laravel プロジェクト生成ファイル

```
src
├── .editorconfig
├── .env
├── .env.example
├── .gitattributes
├── .gitignore
├── .styleci.yml
└── README.md
```

```
├── app
├── artisan
├── bootstrap
├── composer.json
├── composer.lock
├── config
├── database
├── package.json
├── phpunit.xml
├── public
├── resources
├── routes
├── server.php
├── storage
├── tests
└── vendor
    └── webpack.mix.js
```

## 2.3 コマンドを作成する

---

Laravel では、artisan を使用して実行する、独自のコマンドを作成できます。アプリケーションのひな型ができたので、次は ECS でバッチ処理として実行するコマンドを作成します。

### 2.3.1 PHP のコンテナイメージを用意する

「2.1 Web アプリケーションのネットワーク構成」で確認したように、Laravel を Web アプリケーションとして動作させるためには、Nginx と PHP のコンテナが必要となります。

ただし、Laravel のコマンドを実行する際は、PHP のコンテナだけで動作が確認できます。そのため、まずは PHP のコンテナイメージから作成しましょう。

Nginx のコンテナについては、「2.4.1 Nginx のコンテナを用意する」で説明します。

#### .dockerrcignore を作成

Laravel では、動作に必要なパッケージを vendor や node\_modules ディレクトリにインストールします。本書では、これらのパッケージのインストールは、後述の「Dockerfile の作成」で作成するイメージのビルド時に実行します。

また、アプリケーションが参照する環境変数は、ローカル環境と ECS 環境で取得元や値が変わります。ローカル環境では.env ファイルから読み込みますが、ECS 環境では、AWS Systems Manager のパラメータストアから取得

します。

そのため、イメージのビルド時に必要のないディレクトリ（vendor, node\_modules）やファイル (.env) を.dockerignore を使って除外します。プロジェクトのルートディレクトリに.dockerignore を作成して次の設定を追加します。

#### ▼ .dockerignore

```
src/vendor
src/node_modules
src/.env
```

### Dockerfile の作成

Laravel を実行する PHP のコンテナイメージを用意します。docker/laravel 配下に Dockerfile を作成し、次のように編集します。

#### ▼ docker/laravel/Dockerfile

```
FROM php:7.4-fpm

COPY ./src /application

WORKDIR /application

COPY --from=ecs-hands-on/composer /usr/local/bin/composer /usr/local/bin/composer ①

RUN apt-get update && apt-get install -y \
    git \
    zip \
    unzip \
&& composer install ②

RUN php artisan cache:clear \
&& php artisan config:clear \
&& php artisan route:clear \
&& php artisan view:clear

RUN chown -R www-data:www-data storage
```

①では、「2.2 Docker で Laravel アプリケーションを作成」でビルドしたイメージから composer をコピーしています。なぜコピーしているかというと、②で composer install を実行するときに composer の実行ファイルが必要となります。php:7.4-fpm のイメージには、実行ファイルが含まれていません。そこで、作成済みのイメージから composer の実行ファイルを取り込んでいます。

### PHP のイメージをビルド

Laravel を実行する PHP のコンテナイメージをビルドします。

```
$ docker build -t ecs-hands-on/laravel:latest -f ./docker/laravel/Dockerfile .
```

### 2.3.2 コンテナで artisan コマンドを実行する

用意したイメージを使って、コンテナ内から artisan コマンドを実行してみます。まずは、docker run コマンドでコンテナを起動します。

```
$ docker run -it ecs-hands-on/laravel:latest bash
```

コンテナ内に入ると、Dockerfile で設定した WORKDIR の /application ディレクトリがカレントディレクトリになっています。

artisan を使って、Laravel のバージョンと実行環境を確認するコマンドを実行します。

```
root@a305e70f6321:/application# php artisan -v  
←コンテナ内で実行
```

#### ▼ 実行結果

```
Laravel Framework 8.6.0
```

```
root@a305e70f6321:/application# php artisan env  
←コンテナ内で実行
```

#### ▼ 実行結果

```
Current application environment: production
```

上記のコマンドを実行すると、環境が production と表示されます。

事前に作成した.dockerignore で .env を除外しているため、コンテナ内には .env が存在しないのに、なぜ production と表示されるのでしょうか。

その理由は、.env が存在せず、かつ環境変数に APP\_ENV が設定されていない場合、Laravel はデフォルトでその環境を production と判定するためです。

動作が確認できたらコンテナを停止します。コンテナの停止はコンテナ内で exit を実行するか、<Ctrl+D>を実行します。

### コマンドファイルの作成

新しく追加する独自コマンド用のファイルを生成します。コマンドファイルの生成には、php artisan make:command <クラス名>を実行します。

ここではコンテナにログインするのではなく、docker run にファイル生成のコマンドを引数で渡して、実行する方法を用います。

```
$ docker run -v $(pwd)/src:/application -it ecs-hands-on/laravel:late>
>st php artisan make:command HelloWorld
```

コマンドを実行すると、ローカル環境に  
src/app/Console/Commands/HelloWorld.php が作られます。

▼ src/app/Console/Commands/HelloWorld.php

```
<?php

namespace App\Console\Commands;

use Illuminate\Console\Command;

class HelloWorld extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'command:name';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Command description';

    /**
     * Create a new command instance.
     *
     * @return void
     */
    public function __construct()
    {
        parent::__construct();
    }

    /**
     * Execute the console command.
     *
     * @return int
     */
    public function handle()
    {
        return 0;
    }
}
```

生成した HelloWorld.php を次のように変更します。

**▼ src/app/Console/Commands/HelloWorld.php**

```
class HelloWorld extends Command
{
    (省略)

    -     protected $signature = 'command:name';
    +     protected $signature = 'print:helloworld'; // ①

    public function handle()
    {
        $this->line('Hello ' . config('app.word')); // ②
        return 0;
    }

    (省略)
```

php artisan で実行するコマンドの名前を print:helloworld に変更します(①)。

②の処理は config() を使用して、src/config/app.php の連想配列から、キー名が word の値を取得し、'Hello' と取得した文字を連結してコンソールに出力します。

**config の追加**

config() を使って値を参照するようにしたので、次は src/config/app.php に設定を追加します。

**▼ src/config/app.php**

```
<?php

return [
    'word' => env('APP_WORD'), // ①

    省略

    'name' => env('APP_NAME', 'Laravel'),
```

env() は環境変数から値を取得する関数です(①)。ただし、.env ファイルが存在する場合、環境変数ではなく .env ファイルから値を読み取ります。ローカル環境では、.env ファイルから値を読み取ります。

**.env にパラメータを追加**

続いて、env() で取得する値を .env ファイルに加えます。

**▼ src/.env**

```
+ APP_WORD=World
APP_NAME=Laravel
```

```
APP_ENV=local
```

### コマンドを登録する

作成したコマンドを実行するには、src/app/Console/Kernel.php に登録しておく必要があります。コマンドを登録するには、\$commands の配列に対象のコマンドクラスを追加します。

#### ▼ src/app/Console/Kernel.php

```
class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     *
     * @var array
     */
    protected $commands = [
        // ...
        \App\Console\Commands\HelloWorld::class
    ];
}
```

### 動作確認

コマンドの追加ができたので、動作確認を行います。Laravel コンテナを起動して、コマンドを実行します。このとき Hello World (World は.env から取得した文字列) と表示されていれば OK です。

```
$ docker run -v $(pwd)/src:/application -it ecs-hands-on/laravel:late
>st bash
```

コンテナが起動したら、作成したコマンドを実行します。

```
root@a305e70f6321:/application# php artisan print:helloworld
←コンテナ内で実行
```

```
root@a6c0efaa289d:/application# php artisan print:helloworld
Hello World
root@a6c0efaa289d:/application#
.evnの値が表示されている
```

## 2.4 Web ページを表示する

---

ここからは、Laravel で用意した Web ページを表示する方法について、説

明します。動作確認に使用する Web ページは次の 2 種類です。

- Laravel のデフォルトの Welcome ページ
- 新規に作成する、Hello World を表示するページ

### 2.4.1 Nginx のコンテナを用意する

ECS でデプロイした Web ページを表示するために必要な、Nginx のコンテナを用意します。

#### Dockerfile の作成

Nginx のコンテナイメージ用の Dockerfile を追加します。  
docker/nginx/Dockerfile を作成して、下記のように編集します。

##### ▼ docker/nginx/Dockerfile

```
FROM node:14.11.0 as node-build

COPY ./src /application

WORKDIR /application

RUN apt-get update \
&& npm install \ ①
&& npm run prod ②

FROM nginx:1.19.2

WORKDIR /application

RUN apt-get update

COPY ./docker/nginx/ecs/conf.d/default.conf /etc/nginx/conf.d/de>
>fault.conf ③
COPY --from=node-build /application/public /application/public/ >
>④

EXPOSE 80
```

JavaScript や CSS などのアセットをコンパイルために、NPM のパッケージを node\_modules にインストールします (①)。

npm run prod (②) を実行して、アセットをコンパイルします。これによって、application 配下に public/js/app.js と public/css/app.css が生成されます。

ECS で Nginx のコンテナを動かすためには、設定ファイル (③) が必要です。この設定ファイルについては、次の「ECS で使用する Nginx の設定ファイル」で説明します。

この Dockerfile ではマルチステージビルドを使用して、一行目の

node:14.11.0 のイメージでコンパイルしたアセットをコピーしています (④)。マルチステージビルドを使用することで、Nginx のイメージにはアセットのコンパイルに必要な node\_modules やミドルウェアをインストールする必要がなくなっています。

### ECS で使用する Nginx の設定ファイル

ECS で使用する Nginx の設定ファイルとして、Dockerfile に記述した docker/nginx/ecs/conf.d/default.conf を用意しましょう。

Laravel の公式ドキュメント<sup>\*4</sup>を参考に、次のように設定します。

#### ▼ docker/nginx/ecs/conf.d/default.conf

```
upstream php-fpm {
    server localhost:9000; ①
}

server {
    listen 80;
    server_name localhost;
    root /application/public;

    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Content-Type-Options "nosniff";

    index index.php;

    charset utf-8;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location = /favicon.ico { access_log off; log_not_found off; }
    location = /robots.txt { access_log off; log_not_found off; }

    error_page 404 /index.php;

    location ~ \.php$ {
        fastcgi_pass php-fpm;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ /\.well-known.* {
}
```

<sup>\*4</sup> <https://laravel.com/docs/8.x/deployment#nginx>

```
        deny all;  
    }  
}
```

今回のアプリケーションの構成では、Nginx がリクエストを受け取り、PHP のコンテナに流します。そのため、upstream でリクエストを流す先を定義しているのですが、ECS 上ではこのとき対象のホストを localhost とします(①)。後述の「Docker Compose で使用する Nginx の設定ファイル」に出てくる docker-compose を使用するときは、ホストにコンテナ名を指定するので注意してください。

### Nginx のイメージをビルド

```
$ docker build -t ecs-hands-on/nginx:latest -f ./docker/nginx/Dockerfile .
```

### ビルド結果を確認

ビルドしたコンテナイメージにアセットが生成されているかを確認します。ビルドしたイメージからコンテナを起動して、public ディレクトリ配下にコンパイルされた JS や CSS があるか確認します。

```
$ docker run -it ecs-hands-on/nginx:latest ls -la public/js public/css
```

#### ▼ 実行結果

```
public/css:  
total 8  
drwxr-xr-x 2 root root 4096 Sep 24 15:22 .  
drwxr-xr-x 4 root root 4096 Sep 26 13:59 ..  
-rw-r--r-- 1 root root 0 Sep 26 13:59 app.css  
  
public/js:  
total 100  
drwxr-xr-x 2 root root 4096 Sep 26 13:59 .  
drwxr-xr-x 4 root root 4096 Sep 26 13:59 ..  
-rw-r--r-- 1 root root 88721 Sep 26 13:59 app.js  
-rw-r--r-- 1 root root 336 Sep 26 13:59 app.js.LICENSE.txt
```

## 2.4.2 Docker Compose でアプリケーションを起動する

Docker Compose を使ってコンテナを起動し、ローカル環境で動作を確認します。

### Docker Compose で使用する Nginx の設定ファイル

docker-compose では、ECS と比べて Nginx の設定が一部異なります。

docker-compose 起動時に設定ファイルを書き換えることもできますが、本書では docker-compose 用の Nginx のファイルを用意します。

docker/nginx/local/conf.d/default.conf を作成して次のように設定します。

#### ▼ docker/nginx/local/conf.d/default.conf

```
upstream php-fpm {
    server laravel:9000; ①
}

server {
    listen 80;
    server_name localhost;
    root /application/public;

    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Content-Type-Options "nosniff";

    index index.php;

    charset utf-8;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location = /favicon.ico { access_log off; log_not_found off; }
    location = /robots.txt { access_log off; log_not_found off; }

    error_page 404 /index.php;

    location ~ \.php$ {
        fastcgi_pass php-fpm;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ /\.well-known.* {
        deny all;
    }
}
```

ECS 用の設定ファイルと違い、docker-compose 用の設定ファイルでは、ホストにコンテナ名を指定します（①）。

#### **docker-compose.yml の作成**

プロジェクトルート配下に docker-compose.yml を用意して、次のように編

集します。

#### ▼ docker-compose.yml

```
version: '3'

services:
  nginx:
    image: ecs-hands-on/nginx:latest ①
    volumes:
      - ./docker/nginx/local/conf.d/default.conf:/etc/nginx/conf>
        .d/default.conf ②
    ports:
      - 8080:80
    expose:
      - 8080
    depends_on:
      - laravel
  laravel:
    image: ecs-hands-on/laravel:latest
    volumes:
      - ./src:/application
    env_file:
      - ./src/.env
```

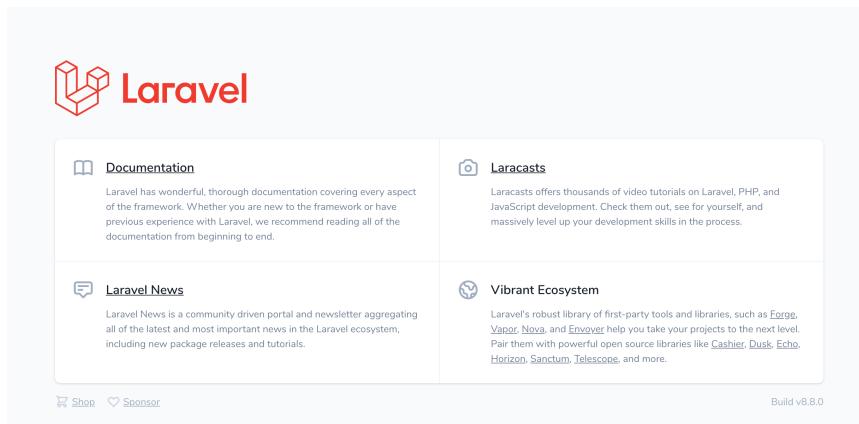
「Nginx のイメージをビルド」で作成した、Nginx のイメージを指定します(①)。

「Docker Compose で使用する Nginx の設定ファイル」で作成した設定ファイルを使用します(②)。

### 2.4.3 アプリケーションの起動

```
$ docker-compose up
```

http://localhost:8080/ にアクセスして、Laravel の welcome ページが表示されていれば ok です。docker-compose を停止する際は、<Ctrl+C>を押します。



## ルーティングの追加

Hello World を表示する、新規ページ用のルーティングを追加するために、src/routes/web.php を編集します。

### ▼ src/routes/web.php

```
Route::get('/', function () {
    return view('welcome');
});

+ Route::get('/hello', function () {
+     return 'Hello ' . config('app.word'); // ①
+ });

config() を使用して、src/config/app.php の連想配列から、キー名が word の値を取得します (①)。
```

## 動作確認

ルーティングの追加ができたので、動作確認を行います。  
docker-compose を起動して、<http://localhost:8080/hello> にアクセスします。  
アクセスしたページで Hello World と表示されていれば ok です。

Hello World

.envの値が表示されている

## 第 3 章

# ECR にイメージを登録する

ECS では、デプロイするコンテナのイメージを ECR から取得します。本章では、ECR<sup>\*1</sup> に Nginx と PHP のコンテナイメージを登録する方法を説明します。

### 3.1 ECR にリポジトリを作成する

本書では Nginx と PHP のコンテナイメージを別々のリポジトリで管理します。そのためのリポジトリを ECR に作成しましょう。それぞれのリポジトリ名は次のように設定します。

Nginx イメージのリポジトリ	ecs-hands-on/nginx
Laravel イメージのリポジトリ	ecs-hands-on/laravel

#### 3.1.1 Nginx イメージのリポジトリを作成

リポジトリ作成画面を開く

- AWS のコンソール画面で ECR > リポジトリの一覧画面を開き、「リポジトリを作成」ボタン（①）を押下します（図 3.1）。

\*1 <https://ap-northeast-1.console.aws.amazon.com/ecr/repositories?region=ap-northeast-1#>



▲図 3.1

### リポジトリの設定を入力する

- リポジトリ作成画面が表示されたら、リポジトリ名に `ecs-hands-on/nginx` を入力し (①)、「リポジトリを作成」ボタン (②) を押下します (図 3.2)。

ECR > リポジトリ > リポジトリを作成

## リポジトリを作成

**アクセスとタグ**

リポジトリ名

ecs-hands-on/nginx

リポジトリ名には名前空間を含めることができます (例: namespace/repo-name)。

タグのイミュータビリティ

タグのイミュータビリティを有効にすると、同じタグを使用した後続イメージのプッシュによる が上書きを防ぎます。タグのイミュータビリティを無効にするとイメージタグを上書きできるようになります。

無効

**イメージスキャンの設定**

プッシュ時にスキャン

プッシュ時にスキャンを有効にすると、各イメージがリポジトリにプッシュされた後に自動的にスキャンされます。無効にした場合、スキャン結果を取得するには、各イメージのスキャンを手動で開始する必要があります。

無効

**暗号化設定**

KMS 暗号化

デフォルトの暗号化設定を使用する代わりに、AWS Key Management Service (KMS) を使用して、このリポジトリに保存されているイメージを暗号化できます。

無効

① KMS 暗号化設定は、リポジトリの作成後に変更または無効にすることはできません。

キャンセル リポジトリを作成

▲図 3.2

- リポジトリの作成に成功すると一覧画面に遷移し、追加したリポジトリが表示されます（図 3.3）。

リポジトリ ecs-hands-on/nginx は正常に作成されました

リポジトリ (1)

リポジトリ名	URI	作成時刻	タグのイミュータビリティ	プッシュ時にスキャン
ecs-hands-on/nginx		20/10/17 17:40:33	無効	無効

▲図 3.3

### 3.1.2 PHP イメージのリポジトリを作成

「3.1.1 Nginx イメージのリポジトリを作成」と同様に、PHP のイメージを登録するリポジトリを作成します。

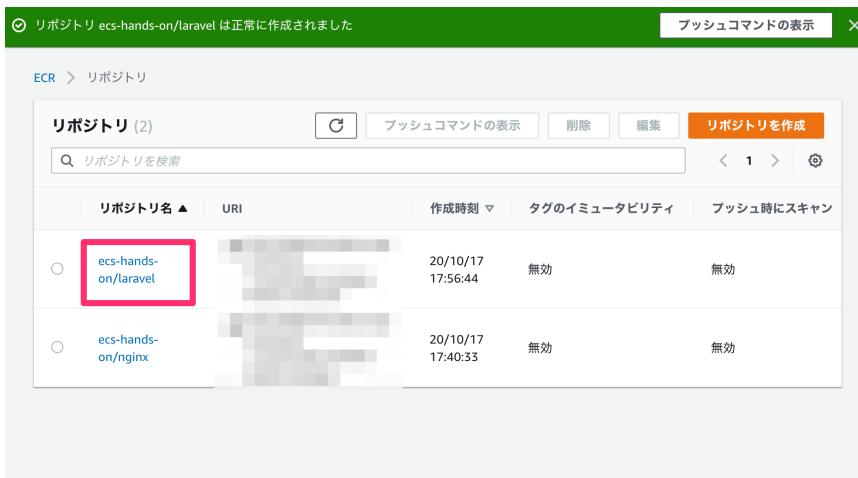
#### リポジトリを作成する

- リポジトリ名には `ecs-hands-on/laravel` (①) と入力し、「リポジトリを作成」ボタン (②) を押下します (図 3.4)。



▲図 3.4

- リポジトリの作成に成功すると一覧画面に遷移し、追加したリポジトリが表示されます (図 3.5)。



▲図 3.5

## 3.2 リポジトリにコンテナイメージを登録する

リポジトリが用意できたので、次はコンテナイメージをリポジトリに登録します。

### 3.2.1 ECR にログインする

ECR にイメージを登録するには、AWS CLI を用います。AWS CLI で ECR に対してイメージをプッシュするためには、ログインしておく必要があります。

#### プッシュコマンドを表示する

- Nginx か PHP リポジトリのイメージ一覧画面を開き、右上の「プッシュコマンドの表示」ボタン（①）を押下します（図 3.6）。

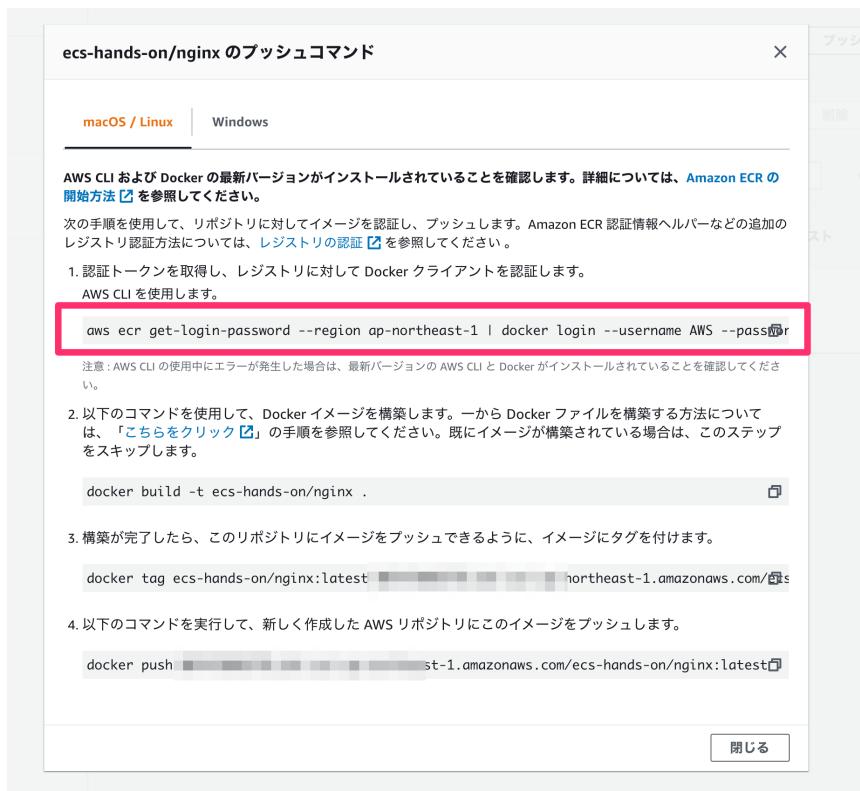
## 第3章 ECR にイメージを登録する



▲図 3.6

### ログインコマンドを使用する

- ダイアログの最初に記載されている、aws ecr get-login-password コマンドを使用します。(図 3.7)



▲図 3.7

**▼ECR ログインコマンド**

```
$ aws ecr get-login-password --region ap-northeast-1 | docker login --username AWS --password-stdin <AWSアカウントID>.dkr.ecr.ap-northeast-1.amazonaws.com
```

ログインに成功するとメッセージが表示されます。

**▼実行結果**

```
Login Succeeded
```

### 3.2.2 プッシュ前にコンテナイメージをビルドする

リポジトリの作成と ECR へのログインが完了したことで、コンテナイメージを登録する準備が整いました。イメージをプッシュする前に、もう一度コンテナイメージをビルドしておきます。

#### Nginx のイメージをビルド

Nginx のコンテナイメージのビルドは「Nginx のイメージをビルド」と同様です。

- プロジェクトルート配下で docker build を実行します。

**▼Nginx イメージのビルド**

```
$ docker build -t ecs-hands-on/nginx:latest -f ./docker/nginx/Dockerfile .
```

#### PHP のイメージをビルド

PHP のコンテナイメージも同様にビルドします。

**▼PHP イメージのビルド**

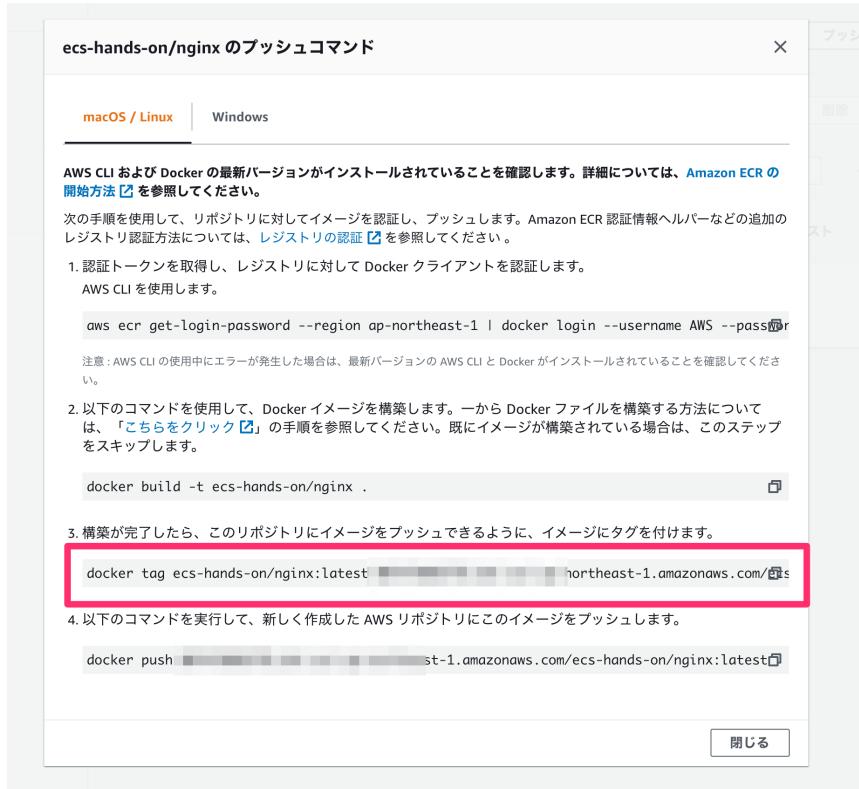
```
$ docker build -t ecs-hands-on/laravel:latest -f ./docker/laravel/Dockerfile .
```

### 3.2.3 コンテナイメージにタグをつける

ECR にプッシュするときは、コンテナイメージにタグをつける必要があります。

#### Nginx のコンテナイメージにタグをつける

- ダイアログのコマンドから docker tag のコマンドをコピーして、実行します（図 3.8）。



▲ 図 3.8

#### ▼ Nginx コンテナイメージにタグ付け

```
$ docker tag ecs-hands-on/nginx:latest <AWSアカウントID>.dkr.ecr.ap-northeast-1.amazonaws.com/ecs-hands-on/nginx:latest
```

#### PHP のコンテナイメージにタグをつける

- Nginx と同様にダイアログのコマンドをコピーして、実行します。

#### ▼ Nginx のコンテナイメージにタグ付け

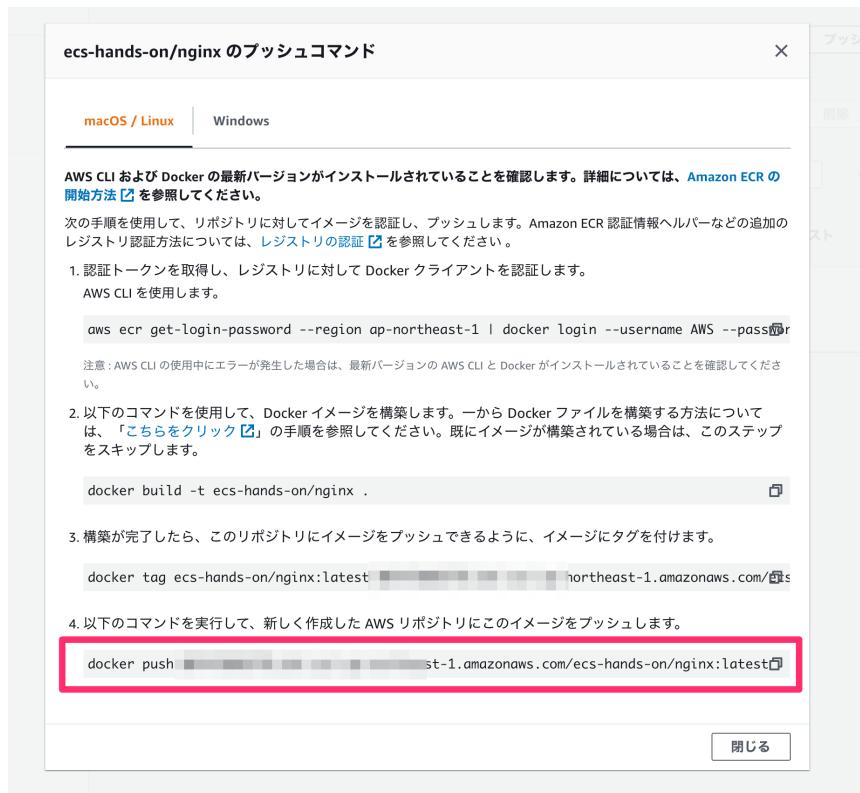
```
$ docker tag ecs-hands-on/laravel:latest <AWSアカウントID>.dkr.ecr.ap-northeast-1.amazonaws.com/ecs-hands-on/laravel:latest
```

### 3.2.4 ECR にコンテナイメージをプッシュ

タグを設定したコンテナイメージを ECR にプッシュします。

#### Nginx のコンテナイメージをプッシュ

- ダイアログの docker push コマンドをコピーして、ターミナルで実行します（図 3.9）。



▲ 図 3.9

## ▼ Nginx のコンテナイメージをプッシュ

```
$ docker push <AWSアカウントID>.dkr.ecr.ap-northeast-1.amazonaws.>
>com/ecs-hands-on/nginx:latest
```

- イメージをプッシュすると、一覧画面にプッシュしたイメージが表示されます（図 3.10）。



▲ 図 3.10

## PHP のコンテナイメージをプッシュ

- Nginx と同様にダイアログのコマンドをコピーして、実行します。

### ▼ PHP のコンテナイメージをプッシュ

```
$ docker push <AWSアカウントID>.dkr.ecr.ap-northeast-1.amazonaws.>
>com/ecs-hands-on/laravel:latest
```

- イメージをプッシュしたあと、一覧画面にイメージが表示されます。  
(図 3.11)



▲ 図 3.11

## 第 4 章

# AWS のネットワークを構築する

本章では、ECS を使用するために必要な VPC やサブネットなどのリソースを作成します。

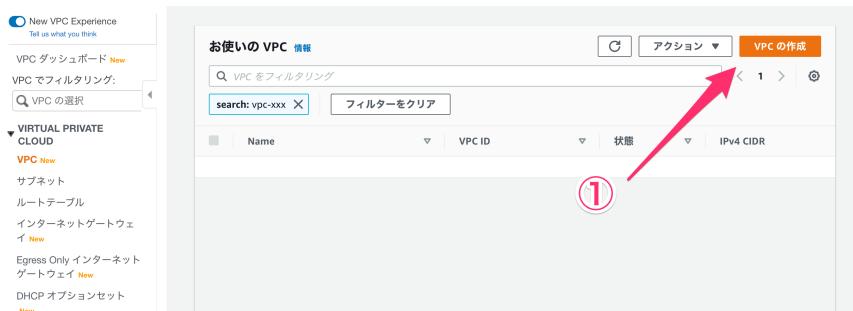
## 4.1 VPC を作成する

まず最初に VPC を作成します。VPC は次のように設定します。

名前	ecs-hands-on
IPv4 CIDR ブロック	10.0.0.0/16

### VPC 作成画面を開く

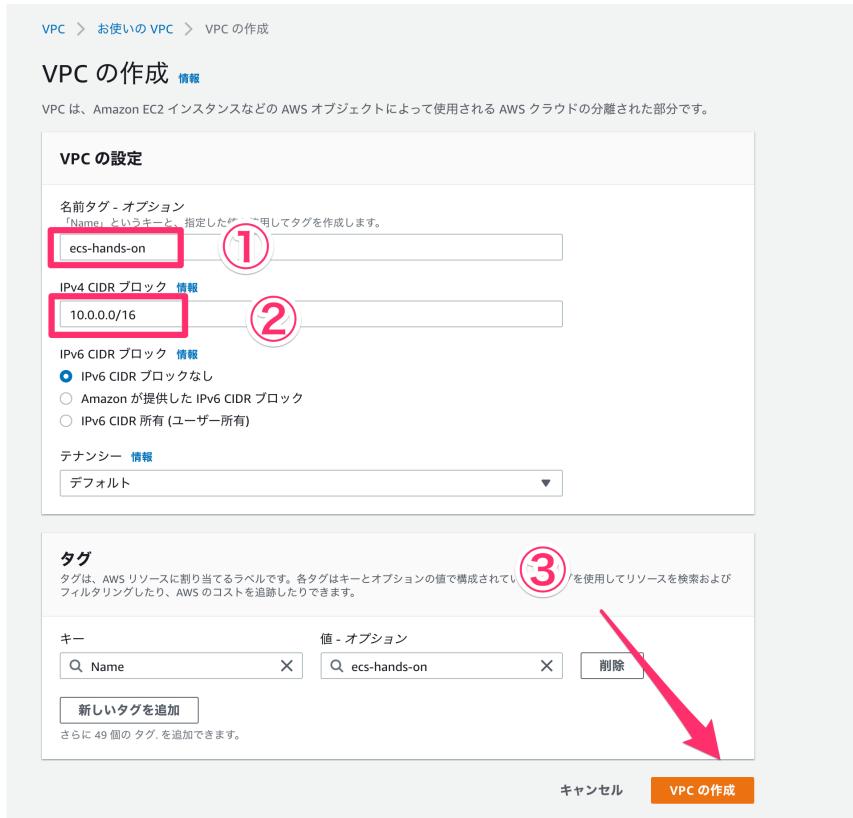
- VPC のダッシュボードにアクセスし、VPC の作成ボタン (①) を押下します。(図 4.1)



▲図 4.1

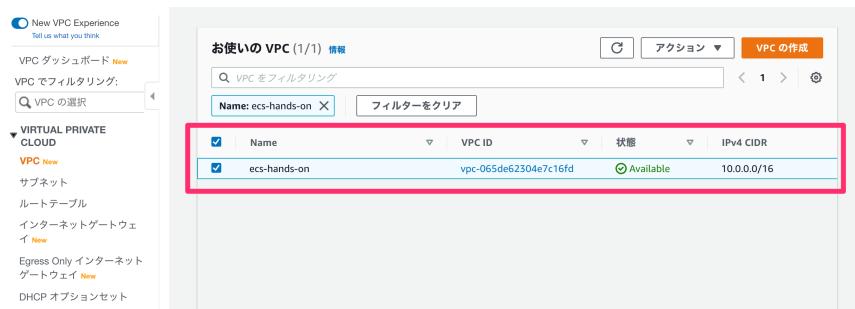
## VPC の設定を入力する

- 名前タグに ecs-hands-on を入力します (①)。
- IPv4 CIDR ブロックには、10.0.0.0/16 を入力します (②)。
- VPC の作成ボタン (③) を押下して、VPC を作成します (図 4.2)。



▲図 4.2

- VPC ダッシュボードの一覧で、作成した VPC が表示されます (図 4.3)。



▲図 4.3

## 4.2 パブリックサブネットを作成する

### 4.2.1 サブネットを作成する

次は VPC に紐づくサブネットを作成します。サブネットは次のように設定します。

名前タグ	ecs-hands-on
VPC	「4.1 VPC を作成する」の VPC
アベイラビリティーゾーン	ap-northeast-1a
IPv4 CIDR ブロック	10.0.0.0/24

#### サブネットの作成画面を開く

- VPC > サブネットのダッシュボード左上のサブネットの作成ボタン (①) を押下します (図 4.4)。



▲図 4.4

### サブネットの設定を入力する

- 名前タグ (①) に ecs-hands-on を入力します。
- VPC (②) には、「4.1 VPC を作成する」の VPC を選択します。
- アベイラビリティーゾーン (③) は、ap-northeast-1a を使用します。
- IPv4 CIDR ブロック (④) は、10.0.0.0/24 を入力します。
- 各項目が入力できたら右下の作成ボタンをクリックします (図 4.5)。



▲図 4.5

ダッシュボードのサブネット一覧に作成したサブネットが表示されます (図 4.6)。

Name	サブネット ID	状態	VPC	IPv4 CIDR
ecs-hands-on	subnet-0255f3402d19bc3fb	available	vpc-065de62304e7c16fd   ...	10.0.0.0/24

▲図 4.6

### 4.2.2 インターネットゲートウェイを作成

AWS ネットワークの内部からインターネットへの接続ができるようにします。そのためには、インターネットゲートウェイとルートテーブルを用意して、VPC にアタッチする必要があります。まずは、インターネットゲートウェイの設定をします。

### インターネットゲートウェイの作成画面を開く

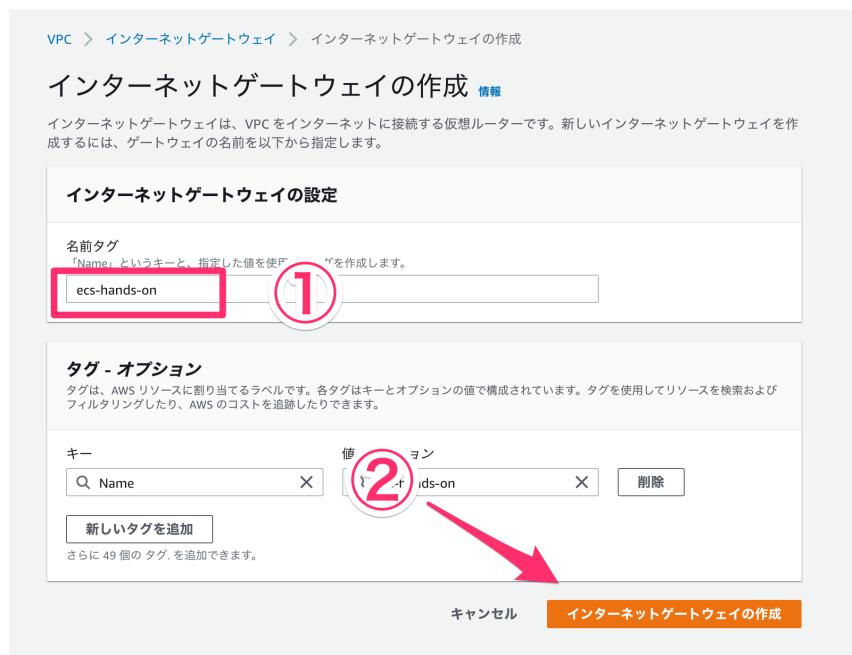
- VPC > インターネットゲートウェイのダッシュボード右上のインターネットゲートウェイの作成ボタン (①) をクリックします (図 4.7)。



▲図 4.7

### インターネットゲートウェイの設定を入力する

- 名前タグに ecs-hands-on を入力 (①) し、インターネットゲートウェイの作成ボタン (②) を押下します (図 4.8)。



▲図 4.8

- ダッシュボードのインターネットゲートウェイ一覧に作成したインターネットゲートウェイが表示されます (図 4.9)。



▲図 4.9

## 4.3 VPC にインターネットゲートウェイをアタッチする

「4.2.2 インターネットゲートウェイを作成」で作成したインターネットゲートウェイを VPC にアタッチします。

### インターネットゲートウェイのアタッチ画面を開く

- 作成済みのインターネットゲートウェイを右クリックします。
- 表示されたメニューから、VPC にアタッチ（①）を選択します（図 4.10）。



▲図 4.10

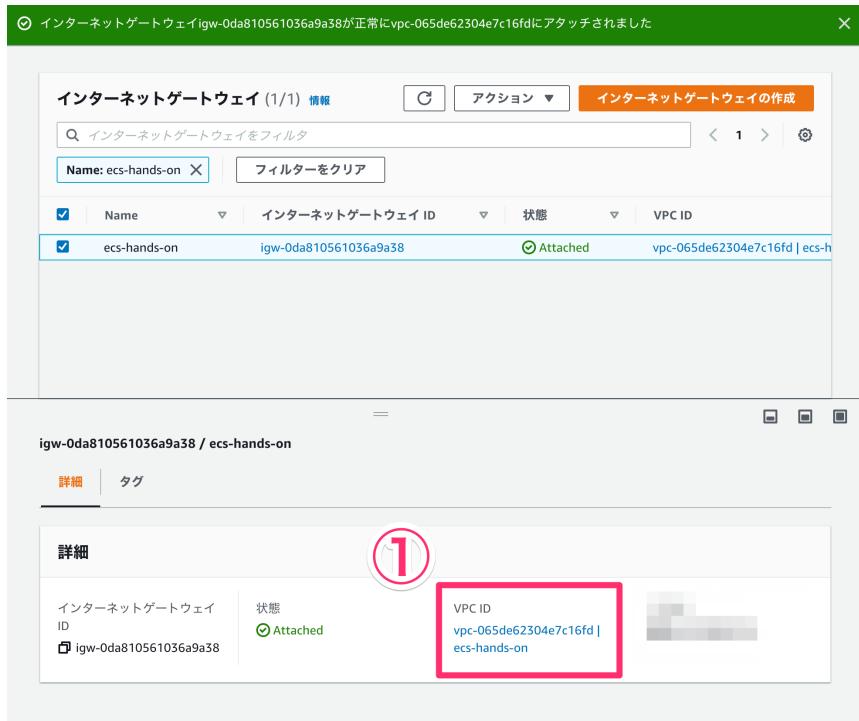
### インターネットゲートウェイをアタッチ

- 使用可能な VPC に「4.1 VPC を作成する」で作成した VPC (①) を設定します。
- 次にインターネットゲートウェイのアタッチボタン (②) をクリックします (図 4.11)。



▲ 図 4.11

- インターネットゲートウェイの詳細情報で、インターネットゲートウェイが VPC にアタッチされていることを確認 (①) できます (図 4.12)。



▲図 4.12

### 4.3.1 ルートテーブルを作成する

インターネットゲートウェイが用意できたので、次はルートテーブルを設定します。ルートテーブルの設定内容を次に示します。

名前タグ	ecs-hands-on
VPC	「4.1 VPC を作成する」の VPC

#### ルートテーブルの作成画面を開く

- VPC > ルートテーブルのダッシュボード左上のルートテーブルの作成ボタン (①) をクリックします (図 4.13)。

## 4.3 VPC にインターネットゲートウェイをアタッチする



▲ 図 4.13

### ルートテーブルの設定を入力する

- 名前タグ (①) に ecs-hands-on と入力します。
- VPC (②) には、「4.1 VPC を作成する」で作成した VPC を設定します。
- 各項目の設定ができたら、作成ボタンを押下します (図 4.14)。

The screenshot shows the 'Create Route Table' page. In the 'Tag Name' field (①), 'ecs-hands-on' is entered. In the 'VPC' dropdown (②), 'vpc-065de62304e7c16fd' is selected. Below the form, a note says 'このリソースには現在、タグがありません' (There are no tags for this resource currently). At the bottom right, there are 'Cancel' and 'Create' buttons, with 'Create' being highlighted by a red arrow.

▲ 図 4.14

- ダッシュボードのルートテーブル一覧に作成したルートテーブルが表示されます (図 4.15)。



▲ 図 4.15

### 4.3.2 ルートテーブルにルートを追加する

「4.2.1 サブネットを作成する」で作成したサブネットをパブリックサブネットにするためのルートを、ルートテーブルに追加します。ルートは次のように設定します。

<b>送信先</b>	0.0.0.0/0
<b>ターゲット</b>	「4.2.2 インターネットゲートウェイを作成」のインターネットゲートウェイ

#### ルートの編集画面を開く

- ルートテーブルの一覧から、「ルートテーブルの設定を入力する」で作成したルートテーブルを選択し (①)、画面下部のルートタブのルートの編集ボタン (②) を押下します (図 4.16)。

## 4.3 VPC にインターネットゲートウェイをアタッチする

The screenshot shows the AWS Lambda console interface. At the top, there is a search bar with the text 'search : ecs-hands-on' and a 'フィルターの追加' (Add Filter) button. Below the search bar is a table with columns: Name, ルートテーブル ID (Route Table ID), 明示的に関連付けられた (Associated), Edge associations (Edge Associations), and メイン (Main). Two rows are listed: 'rtb-047ff62916b9a1690' and 'ecs-hands-on'. The 'ecs-hands-on' row is highlighted with a red box and has a circled '1' above it. A red arrow points from the circled '1' down to the 'Route' tab in the navigation bar of the detailed view below. The detailed view shows the route table configuration with tabs for Overview, Route (circled by a red arrow), Subnet Association, Edge Associations, Route Propagation, and Tags. The 'Route' tab is selected. It displays one route entry: 送信先 (Destination) 10.0.0.0/16, ターゲット (Target) local, ステータス (Status) active, and 伝播済み (Propagated) いいえ (No). The status is highlighted with a red box and circled '2'.

▲ 図 4.16

### ルートの設定を入力する

- ルートの編集画面に遷移したあと、ルートの追加ボタン（①）をクリックします（図 4.17）。

The screenshot shows the 'Route Editor' page. The title bar says 'ルートテーブル > ルートの編集'. The main area is titled 'ルートの編集'. It contains a table with columns: 送信先 (Destination), ターゲット (Target), ステータス (Status), and 伝播済み (Propagated). One row is shown: 10.0.0.0/16, local, active, いいえ (No). Below the table is a button labeled 'ルートの追加' (Add Route) with a circled '1' above it. A red arrow points from the circled '1' to the 'Route' tab in the navigation bar at the bottom of the screen. At the bottom right are 'キャンセル' (Cancel) and 'ルートの保存' (Save Route) buttons.

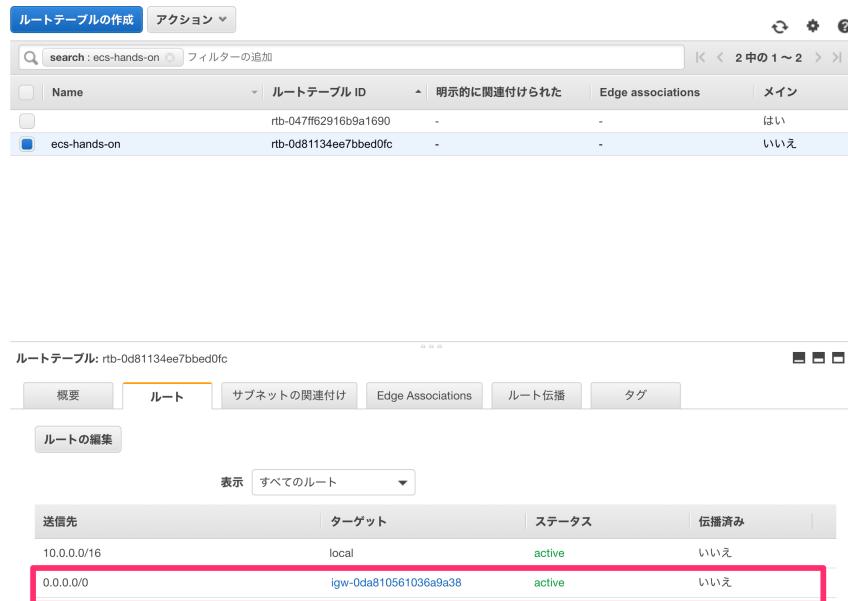
▲ 図 4.17

- 送信先（①）に 0.0.0.0/0 を設定します。
- ターゲット（②）には、「4.2.2 インターネットゲートウェイを作成」で作成したインターネットゲートウェイを指定します。
- 各項目が設定できたら、ルートの保存ボタン（③）をクリックします（図 4.18）。



▲ 図 4.18

- ルートテーブルの設定に、新しいルートが追加されていることを確認します（図 4.19）。

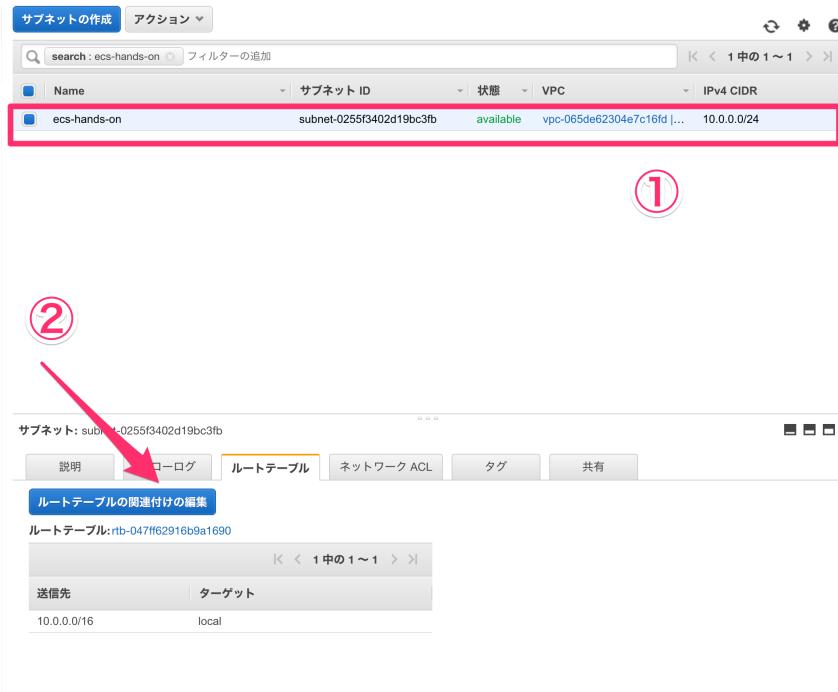


▲ 図 4.19

### 4.3.3 サブネットとルートテーブルを紐付ける

#### ルートテーブルの関連付けの編集画面を開く

- VPC > サブネットの一覧から、「4.2.1 サブネットを作成する」で作成したサブネットを選択します（①）。
- 画面下部のルートテーブルタブの、ルートテーブルの関連付けの編集ボタン（②）をクリックします（図 4.20）。



▲ 図 4.20

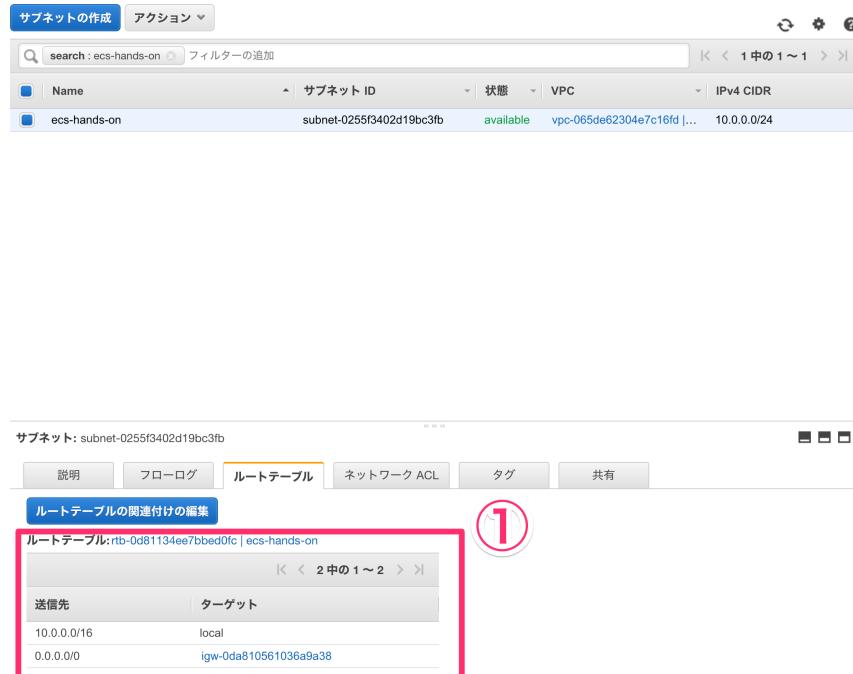
#### ルートテーブル ID を設定する

- ルートテーブルの関連付けの編集画面で、ルートテーブル ID に「4.3.1 ルートテーブルを作成する」で作成したルートテーブルの ID を設定 (①) します。
- 設定ができたら保存ボタン (②) を押下します (図 4.21)。



▲ 図 4.21

- サブネットのルートテーブルが変更されていること (①) を確認します (図 4.22)。



▲ 図 4.22

#### 4.3.4 「パブリック IP アドレスの自動」を有効にする

「4.2.1 サブネットを作成する」で作成したサブネットは、パブリック IP アドレスの自動割り当てが有効になっていません。このサブネットに配置されたコンテナへのインターネットからのアクセス、そしてコンテナが ECR からイメージを pull できるようにするために、パブリック IP アドレスの自動割り当てを有効化します。

##### 自動割り当て IP 設定の変更画面を開く

- 「4.2.1 サブネットを作成する」で作成したサブネットを右クリックします。表示されたメニューから、自動割り当て IP 設定の変更（①）を選択します（図 4.23）。

#### 4.3 VPC にインターネットゲートウェイをアタッチする



▲ 図 4.23

#### IPv4 の自動割り当ての有効化

- IPv4 の自動割り当てにチェックを入れ (①)、保存ボタン (②) をクリックします (図 4.24)。



▲ 図 4.24

- サブネットの説明タブでパブリック IPv4 アドレスの自動が「はい」に設定されていることを確認します (図 4.25)。

▲図 4.25: キャプション

## 4.4 セキュリティグループを設定する

ECS でデプロイする Web アプリケーションに対して、HTTP でアクセスできるようにするためのセキュリティグループを作成します。セキュリティグループの設定内容を次に示します。

セキュリティグループ名	ecs-hands-on
説明	任意の内容
VPC	「4.1 VPC を作成する」で作成した VPC
インバウンドルールのタイプ	HTTP
ソース	「任意の場所」(0.0.0.0/0, :/0)

### セキュリティグループの作成画面を開く

- VPC > セキュリティグループのダッシュボード右上のセキュリティグループを作成ボタンを選択します（図 4.26）。



▲図 4.26

### セキュリティグループの内容を設定

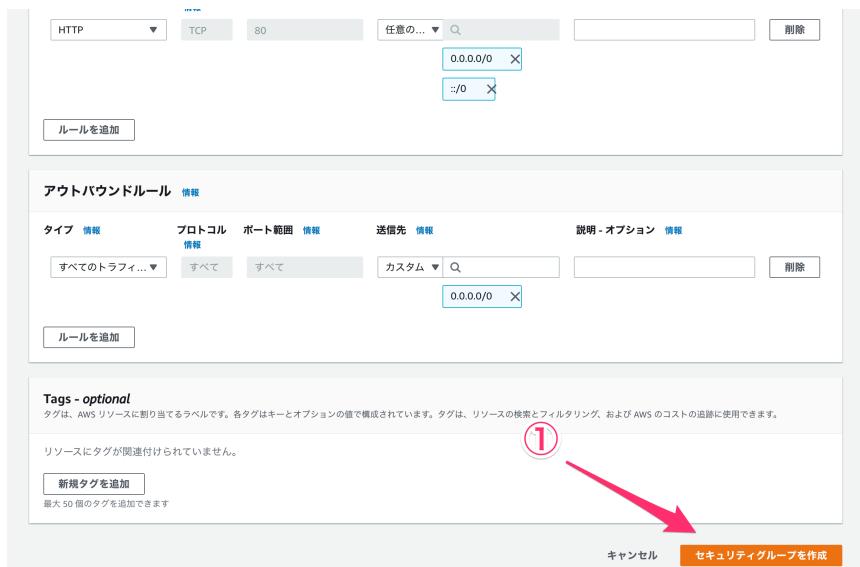
セキュリティグループの作成画面で、セキュリティグループの情報を入力します。

- セキュリティグループ名 (①) に、ecs-hands-on を設定します。説明 (②) は任意の内容を入力します。
- VPC (③) には、「4.1 VPC を作成する」で作成した VPC を選択します。
- インバウンドルールには、タイプ (④) に HTTP を設定します。
- すべての IP からアクセスできるようにするために、ソース (⑤) には「任意の場所」を設定します (図 4.27)。



▲ 図 4.27

- 各項目が設定できたら、セキュリティグループを作成ボタン（①）をクリックします（図 4.28）。



▲ 図 4.28

ダッシュボードのセキュリティグループ一覧に、作成したセキュリティグループが表示されることを確認します（図 4.29）。



▲図 4.29

## 4.5 ECS で使用する IAM ロールを作成する

ECS で使用する IAM ロールを作成します。今回用意するロールは 2 種類です。一つは、タスクのコンテナで使用する、タスク用の IAM ロールです。<sup>\*1</sup>もう一つは、クラスターの作成などに必要な、ECS 用のサービスにリンクしたロール<sup>\*2</sup>です。

### 4.5.1 ECS タスク用の IAM ロールを作成する

ECS タスク用の IAM ロールを作成します。設定内容は次のとおりです。

ロール	ECSTaskRole
信頼されたエンティティ	ecs-tasks.amazonaws.com
ポリシーその 1	AmazonECSTaskExecutionRolePolicy
ポリシーその 2	AmazonSSMReadOnlyAccess

#### IAM ロール作成画面を開く

- IAM ダッシュボード > アクセス管理 > ロール画面の左上のロールの作成ボタン（①）をクリックします（図 4.30）。

<sup>\*1</sup> [https://docs.aws.amazon.com/ja\\_jp/AmazonECS/latest/developerguide/task-iam-roles.html](https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/task-iam-roles.html)

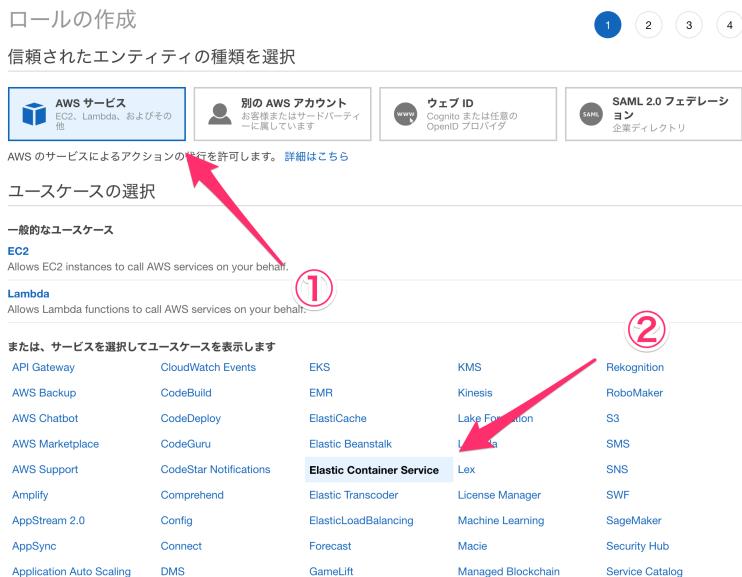
<sup>\*2</sup> [https://docs.aws.amazon.com/ja\\_jp/AmazonECS/latest/developerguide/using-service-linked-roles.html](https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/using-service-linked-roles.html)



▲ 図 4.30

### IAM ロールを設定する

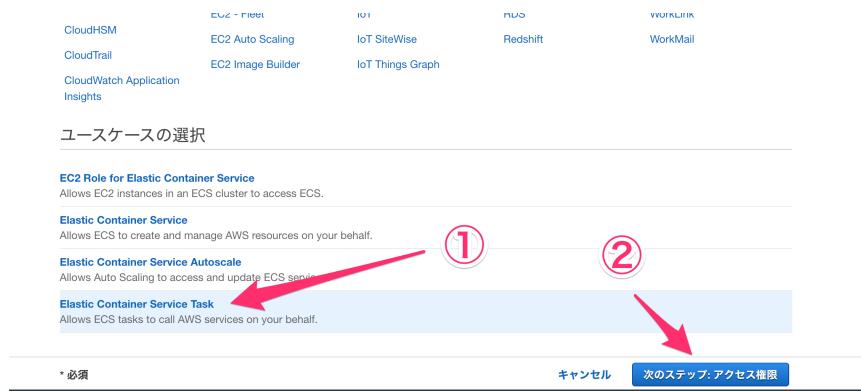
- 信頼されたエンティティの種類で「AWS サービス」を選択し (①)、サービスの一覧から「Elastic Container Service」(②) をクリックします (図 4.31)。



▲ 図 4.31

- 続いてユースケースの選択で「Elastic Container Service Task」(①) を選び、「次のステップ：アクセス権限」(②) をクリックします (図 4.32)。

## 4.5 ECS で使用する IAM ロールを作成する



▲ 図 4.32

- ロールには 2 つのポリシーをアタッチします。1 つめは「AmazonECSTaskExecutionRolePolicy」をチェック (①) します (図 4.33)。



▲ 図 4.33

- さらに「AmazonSSMReadOnlyAccess」のポリシーをチェック (①) し、次のステップ：タグボタン (②) をクリックします (図 4.34)。



▲ 図 4.34

- 次のステップボタンを押下します（図 4.35）。



▲ 図 4.35

- ロール名（①）に ECSTaskRole と入力します。
- 信頼されたエンティティ（②）に ecs-tasks.amazonaws.com が設定さ

れていることを確認します。<sup>\*3</sup>

- 2つのポリシー（③）が設定されていることを確認します。
- 設定ができたら、ロールの作成ボタンをクリックします（図 4.36）。



▲ 図 4.36

- IAM ロールの一覧に作成したロールが表示されていることを確認します（図 4.36）。

Role ECSTaskRole was created successfully.		
<a href="#">Role creation</a> <a href="#">Delete role</a> <span style="float: right;">✖</span>		
<input type="text" value="ECSTaskRole"/> <span style="float: right;">Search</span>	1 result found	
<b>Role Name</b>	<b>Trusted Entity</b>	<b>Last Activity</b>
<input checked="" type="checkbox"/> ECSTaskRole	AWS Service: ecs-tasks	None

▲ 図 4.37

<sup>\*3</sup> [https://docs.aws.amazon.com/ja\\_jp/AmazonECS/latest/developerguide/task-iam-roles.html](https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/task-iam-roles.html)

## 4.5.2 ECS サービスにリンクしたロールを作成する

ECS サービスにリンクしたロールは、AWS CLI を使用して作成します。

```
$ aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

### ▼ 実行結果

```
{  
    "Role": {  
        "Path": "/aws-service-role/ecs.amazonaws.com/",  
        省略  
        "AssumeRolePolicyDocument": {  
            "Version": "2012-10-17",  
            "Statement": [  
                {  
                    "Action": [  
                        "sts:AssumeRole"  
                    ],  
                    "Effect": "Allow",  
                    "Principal": {  
                        "Service": [  
                            "ecs.amazonaws.com"  
                        ]  
                    }  
                }  
            ]  
        }  
    }  
}
```

## 4.6 CloudWatch の設定

ECS で稼働するコンテナが出力するログは、CloudWatch に保存します。その際にログの出力先として必要となる、ロググループを作成します。

### 4.6.1 CloudWatch にロググループを作成する

CloudWatch のロググループは次の内容で作成します。

ロググループ名	ecs-hands-on
---------	--------------

#### ロググループの作成画面を開く

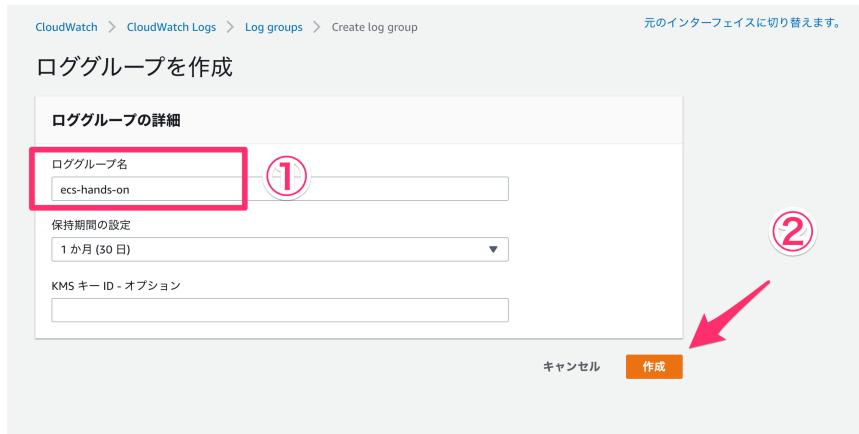
- CloudWatch > ログ > ロググループのダッシュボード画面でロググループを作成ボタン（①）をクリックします（図 4.38）。



▲図 4.38

### ロググループの設定を入力する

- ロググループ名に ecs-hands-on を設定し (①)、作成ボタン (②) を押下します (図 4.39)。



▲図 4.39

- ダッシュボードでロググループの一覧に作成したロググループが表示されていることを確認します (図 4.40)。



▲図 4.40



## 第 5 章

# バッチ処理を実行する

本章から AWS ECS を使用してアプリケーションを実行していきます。本章では、「2.3 コマンドを作成する」で作成した Laravel のコマンドを、ECS で実行する方法について確認します。

## 5.1 クラスターを作成する

今回使用するコマンドは、ECS のタスクとして実行します。タスクはクラスターの中で実行するため、まずはクラスターを作成します。クラスターの設定を次に示します。

クラスターのテンプレート	「ネットワーキングのみ」
クラスター名	ecs-hands-on

### 5.1.1 クラスターの作成画面を開く

- Amazon ECS > クラスターのダッシュボードを開き、クラスターの作成ボタン（①）を押下します（図 5.1）。



▲図 5.1

### 5.1.2 クラスターの設定を選択・入力する

- クラスターのテンプレートから「ネットワーキングのみ」を選択し(①)、次のステップボタン(②)をクリックします(図 5.2)。

#### クラスターの作成



▲図 5.2

- クラスターネーム(①)にecs-hands-onと入力し、作成ボタン(②)をクリックします(図 5.3)。

## クラスターの作成



▲図 5.3

- 作成に成功すると、完了メッセージが表示されます（図 5.4）。

## 起動ステータス

コンテナインスタンスが起動中です。実行状態でアクセス可能になるまでに数分かかることがあります。新しいコンテナインスタンスの使用時間はすぐに開始され、インスタンスを停止または終了するまで費用が発生し続けます。



▲図 5.4

「4.5.2 ECS サービスにリンクしたロールを作成する」でロールを作成していない場合、クラスターの作成に失敗し、「Unable to assume the service linked role. Please verify that the ECS service linked role exists」が表示されることがあります（図 5.5）。

クラスターを作成しようとしてこのエラーが発生した場合、自動的にリンクしたロールが作られていることがあるため、ECS クラスターのダッシュボードに戻り、最初から作成をやり直すと成功することができます。

## 第5章 バッチ処理を実行する

### 起動ステータス

コンテナインスタンスが起動中です。実行状態でアクセス可能になるまでに数分かかることがあります。新しいコンテナインスタンスの使用時間はすぐに開始され、インスタンスを停止または終了するまで費用が発生し続けます。



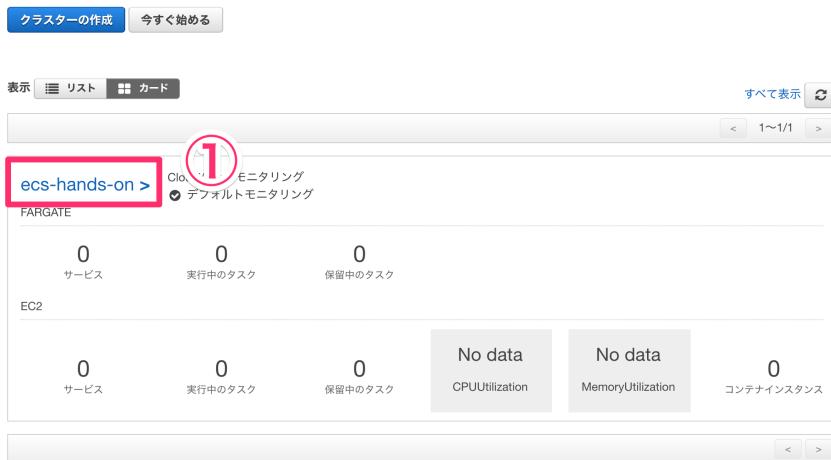
▲図 5.5

- クラスターのダッシュボードに作成したクラスターが表示されること  
(①) を確認します(図 5.6)。

### クラスター

Amazon ECS クラスターは、タスクリクエストを実行できる 1 つ以上のコンテナインスタンスのリージョングループです。各アカウントは、最初に Amazon ECS サービスを使用するときにデフォルトのクラスターを受け取ります。クラスターは、複数の Amazon EC2 インスタンスタイプを含むことができます。

詳細については、[ECS のドキュメント](#)を参照してください。



▲図 5.6

## 5.2 タスク定義を用意する

タスクを使用するためには、タスク定義を用意する必要があります。タスク定義には、CPU やメモリ、コンテナの定義などの設定を記述します。

### 5.2.1 タスク定義ファイルの作成

プロジェクトルート配下に `ecs-task-definition-for-command.json` を作成

し、次のように設定してください。<AWS アカウント ID>はご自身の ID に置き換えてください。

#### ▼ ファイル構成

```
.
├── docker
├── docker-compose.yml
└── ecs-task-definition-for-command.json
    └── src
```

タスク定義は次のように、`--generate-cli-skeleton` オプションを指定してコマンドを実行することで、ひな型を生成できます。

```
$ aws ecs register-task-definition --generate-cli-skeleton
```

#### ▼ ecs-task-definition-for-command.json

```
{
  "family": "ecs-hands-on-for-command", // ①
  "taskRoleArn": "arn:aws:iam::<AWSアカウントID>:role/ECSTaskRole",
  // ②
  "executionRoleArn": "arn:aws:iam::<AWSアカウントID>:role/ECSTaskRole", // ③
  "networkMode": "awsvpc",
  "containerDefinitions": [
    {
      "name": "laravel",
      "image": "<AWSアカウントID>.dkr.ecr.ap-northeast-1.amazonaws.com/ecs-hands-on/laravel:latest", // ④
      "essential": true,
      "environment": [
        {
          "name": "APP_ENV", // ⑤
          "value": "production"
        }
      ],
      "secrets": [
        {
          "name": "APP_WORD",
          "valueFrom": "arn:aws:ssm:ap-northeast-1:<AWSアカウントID>:parameter/APP_WORD" // ⑥
        }
      ],
      "readonlyRootFilesystem": false,
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "ecs-hands-on", // ⑦
          "awslogs-region": "ap-northeast-1",
          "awslogs-datetime-format": "%Y-%m-%d %H:%M:%S",
          "awslogs-stream-prefix": "laravel"
        }
      }
    }
  ]
}
```

```
        }
    },
],
"requiresCompatibilities": [
    "FARGATE" // ⑧
],
"cpu": "256",
"memory": "512"
}
```

family (①) はタスク定義を登録したときに、タスク定義名として使用されます。

「4.5 ECS で使用する IAM ロールを作成する」で作成した ECS 用の IAM ロールを使用します (②、③)。

「3.2.4 ECR にコンテナイメージをプッシュ」で ECR にプッシュした Laravel のコンテナイメージを指定します (④)。

⑤では、環境変数を設定しています。environment を使用するには、name と value をキーとしたオブジェクトを追加します。

ECS では、DB の接続情報や API キーなどの秘密情報を扱うとき、AWS Systems Manager パラメータストアを使用できます<sup>\*1</sup>。

⑥では AWS Systems Manager パラメータストアに登録した値を、secrets で参照しています。パラメータの登録方法は、「5.2.2 AWS Systems Manager パラメータストアに秘密情報を登録する」で後述します。

このタスク定義では、ログドライバとして awslogs を指定しています。ログの出力先として、「4.6 CloudWatch の設定」で作成したロググループを設定しています (⑦)。

ECS のコンテナタイプは FARGATE を使用します (⑧)。

## 5.2.2 AWS Systems Manager パラメータストアに秘密情報を登録する

「5.2.1 タスク定義ファイルの作成」でタスク定義に設定した秘密情報を Systems Manager に登録します。登録するパラメータは次のとおりです。

名前	APP_WORD
タイプ	安全な文字列
値	World

### パラメータストアにアクセスする

---

<sup>\*1</sup> [https://docs.aws.amazon.com/ja\\_jp/AmazonECS/latest/developerguide/specifying-sensitive-data.html](https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/specifying-sensitive-data.html)

- AWS Systems Manager > アプリケーション管理 > パラメーターストア<sup>\*2</sup>にアクセスし、パラメータの作成ボタン（①）をクリックします（図 5.7）。



▲図 5.7

### パラメータを入力する

- 名前（①）を APP\_WORD と設定します。
- タイプに安全な文字列（②）を選択します（図 5.8）。

<sup>\*2</sup> <https://ap-northeast-1.console.aws.amazon.com/systems-manager/parameters?region=ap-northeast-1>

## 第5章 バッチ処理を実行する

AWS Systems Manager > パラメータストア > パラメータを作成

### パラメータを作成

パラメータの詳細

名前  ①

説明 — *Optional*

利用枠  
パラメータストアは標準パラメータおよび詳細パラメータを提供します。

標準  
パラメータの上限は 10,000 です。パラメータ値サイズの上限値は 8 KB です。パラメータポリシーは使用できません。追加料金はありません。

詳細  
10,000 以上のパラメータを作成できます。パラメータ値サイズの上限値は 8 KB です。パラメータポリシーを使用するよう設定できます。詳細パラメータごとに請求が発生します。

タイプ  
 文字列  
任意の文字列値。  
 文字列のリスト  
カンマ区切りの文字列。

安全な文字列  
アカウントまたは別のアカウントの KMS キーを使用して、機密データを暗号化します。 ②

KMS キーポリシー  
 現在のアカウント  
このアカウントのデフォルトの KMS キーを使用するか、このアカウント用のカスタマー管理キーを指定します。 詳細はこちら

別のアカウント  
別のアカウントの KMS キーを使用する 詳細はこちら

▲図 5.8

- 値 (①) に任意の文字を入力します。ここでは World を設定しています。
- 各項目が入力できたら、パラメータを作成ボタン (②) をクリックします (図 5.9)。

□ 先手取りのリスト  
カンマ区切りの文字列。

安全な文字列  
アカウントまたは別のアカウントの KMS キーを使用して、機密データを暗号化します。

KMS キーポリシー  
 現在のアカウント  
このアカウントのデフォルトの KMS キーを使用するか、このアカウント用のカスタマー管理キーを指定します。 詳細はこちら

別のアカウント  
別のアカウントの KMS キーを使用する 詳細はこちら

KMS キー ID

① You have selected the default AWS managed key. All users in the current AWS account and Region will have access to this parameter. To restrict access to the parameter, use a customer managed key (CMK) instead.  
[Learn more](#) ②

値  
 ①

最大長は 4096 文字です。

タグ — *Optional*  
タグを使用して、パラメータへのアクセスを監理および制限できます。

タグを追加 ②

キャンセル パラメータを作成

▲図 5.9

- パラメータの一覧に追加したパラメータが表示されていることを確認します（図 5.10）。

名前	利用枠	種類	最終変更日
APP_WORD	選択	SecureString	Sun, 18 Oct 2020 08:46:28 GMT

▲図 5.10

### 5.2.3 タスク定義の登録

タスク定義ファイルと Systems Manager パラメータストアを作成し、タスク定義を登録する用意ができました。

次の aws ecs コマンドを実行して、タスク定義を登録しましょう。

```
$ aws ecs register-task-definition --cli-input-json file://ecs-task-definition-for-command.json
```

#### ▼実行結果

```
{
  "taskDefinition": {
    "taskDefinitionArn": "arn:aws:ecs:ap-northeast-1:<AWSアカウントID>:task-definition/ecs-hands-on-for-command:1",
    "containerDefinitions": [
      {
        "name": "laravel",
        "image": "<AWSアカウントID>.dkr.ecr.ap-northeast->1.amazonaws.com/ecs-hands-on/laravel:latest",
        "cpu": 0,
        "portMappings": [],
        "essential": true,
        "environment": [
          {
            "name": "APP_ENV",
            "value": "production"
          }
        ],
        "mountPoints": [],
        "volumesFrom": [],
        "secrets": [
          {
            "name": "APP_WORD",
            "value": "<AWSアカウントID>.awssecretmanager.ap-northeast->1.amazonaws.com/ecs-hands-on/word:latest"
          }
        ]
      }
    ],
    "family": "ecs-hands-on-for-command"
  }
}
```

```
        "valueFrom": "arn:aws:ssm:ap-northeast-1:>
>:<AWSアカウントID>:parameter/APP_WORD"
    }
],
"readonlyRootFilesystem": false,
"logConfiguration": {
    "logDriver": "awslogs",
    "options": {
        "awslogs-datetime-format": "%Y-%m-%d %H:>
>%M:%S",
        "awslogs-group": "ecs-hands-on",
        "awslogs-region": "ap-northeast-1",
        "awslogs-stream-prefix": "laravel"
    }
}
],
"family": "ecs-hands-on-for-command",
"taskRoleArn": "arn:aws:iam::<AWSアカウントID>:role/ECSTa>
>skRole",
"executionRoleArn": "arn:aws:iam::<AWSアカウントID>:role/>
>ECSTaskRole",
"networkMode": "awsvpc",
"revision": 1,
"volumes": [],
"status": "ACTIVE",
"requiresAttributes": [
    省略
],
"placementConstraints": [],
"compatibilities": [
    "EC2",
    "FARGATE"
],
"requiresCompatibilities": [
    "FARGATE"
],
"cpu": "256",
"memory": "512"
}
}
```

- Amazon ECS > タスク定義の一覧に作成したタスク定義が表示されていることを確認します（図 5.11）。



▲図 5.11

## 5.3 タスクを実行する

タスクの実行には、aws ecs run-task コマンドを、クラスターなどのリソースを指定して実行します。サブネット ID とセキュリティグループ ID はそれぞれ、「4.2.1 サブネットを作成する」と「4.4 セキュリティグループを設定する」で作成した ID を指定します。

```
$ aws ecs run-task \
--cluster ecs-hands-on \
--task-definition ecs-hands-on-for-command \
--overrides '{"containerOverrides": [{"name": "laravel", "command": ["sh", "-c", "php artisan print:helloWorld"]}]}' \
--launch-type FARGATE \
--network-configuration "awsvpcConfiguration={subnets=[<作成済みのサブネットID>], securityGroups=[<作成済みのセキュリティグループID>], assignPublicIp=ENABLED}"
```

### ▼ 実行結果

```
{
  "tasks": [
    {
      "attachments": [
        省略
      ],
      "availabilityZone": "ap-northeast-1a",
      "clusterArn": "arn:aws:ecs:ap-northeast-1:<AWSアカウントのID>:cluster/ecs-hands-on",
      "containers": [
        {
          省略
        }
      ],
      "cpu": "256",
```

```
        "createdAt": "2020-10-18T18:04:50.440000+09:00",
        "desiredStatus": "RUNNING",
        "group": "family:ecs-hands-on-for-command",
        "lastStatus": "PROVISIONING",
        "launchType": "FARGATE",
        "memory": "512",
        "overrides": [
            "containerOverrides": [
                {
                    "name": "laravel",
                    "command": [
                        "sh",
                        "-c",
                        "php artisan print:helloworld"
                    ]
                }
            ],
            "inferenceAcceleratorOverrides": []
        },
        "platformVersion": "1.3.0",
        "tags": [],
        "taskArn": "arn:aws:ecs:ap-northeast-1:<AWSアカウントのID>:task ecs-hands-on/9c41390de957424ab4fbadce6b95cc80",
        "taskDefinitionArn": "arn:aws:ecs:ap-northeast-1:<AWSアカウントのID>:task-definition/ecs-hands-on-for-command:1",
        "version": 1
    ],
    "failures": []
}
```

- タスクが完了した後に、AWS ECS > クラスターの詳細画面でタスクタブ (①) からステータスが Stopped (②) を選択すると、終了したタスクが表示されます。

タスクが実行中の場合は、Running の方に表示されます（図 5.12）。

### 5.3 タスクを実行する

The screenshot shows the AWS ECS Cluster details page for 'ecs-hands-on'. The left sidebar lists services like Amazon EKS, Clusters, and AWS Marketplace. The main area displays cluster statistics: 0 registered tasks, 0 active tasks, 0 pending tasks, 0 failed tasks, and 0 running tasks. Below this is a table of tasks. A red arrow points to the 'Running' status of the first task (ID: 9c41390...), which is highlighted with a red box. Another red arrow points to the 'Stopped' button in the task's row. A third red circle highlights the task ID '9c41390...'. The top right has buttons for 'Clusterの更新' and 'Clusterの削除'.

▲ 図 5.12

- 「4.6 CloudWatch の設定」で作成したロググループにアクセスし、追加されているログストリームを選択します (①) (図 5.13)。

The screenshot shows the AWS CloudWatch Logs Log groups page for 'ecs-hands-on'. It displays log group details: retention period (1 month), creation time (17 hours ago), bytes saved (0), and ARN. Below this is a table of log streams. A red box highlights the first log stream ('laravel/laravel/9c41390de957424ab4fbadce6b95cc80') with a red circle labeled ①. The table includes columns for Log stream, Last event time, and ARN. The bottom navigation bar shows tabs for 'ログストリーム' (selected), 'メトリクスフィルター', 'サブスクリプションフィルター', and '寄稿者のインサイト'.

▲ 図 5.13

- ログストリーム (①) の内容を確認すると、「2.3 コマンドを作成する」のコマンドの結果が出力されている (②) ことが確認できます (図 5.14)。

## 第5章 バッチ処理を実行する



▲図 5.14

## 第 6 章

# Web アプリケーションをデプロイする

本章では、「2.4.2 Docker Compose でアプリケーションを起動する」で使用した、Nginx と PHP のコンテナイメージを組み合わせて動作する Web アプリケーションを、ECS で実行する方法について確認します。

ECS では、ECS サービスを使って Web アプリケーションをデプロイできます。サービス作成時に必要なタスクの数を設定すると、サービスはタスクの数が指定された数になるように、タスクを維持します。本章では、このサービスを用いて Web アプリケーションをデプロイします。

## 6.1 タスク定義を用意する

「5.1 クラスターを作成する」にてクラスターは作成済みのため、タスク定義を用意するところから始めます。

### 6.1.1 タスク定義ファイルの作成

プロジェクトルート配下に `ecs-task-definition-for-web.json` を作成し、次のように設定してください。

<AWS アカウント ID>はご自身の ID に置き換えてください。

#### ▼ ファイル構成

```
.  
|   └── docker  
|   └── docker-compose.yml  
|   └── ecs-task-definition-for-command.json  
|   └── ecs-task-definition-for-web.json  
└── src
```

## ▼ ecs-task-definition-for-web.json

```
{  
    "family": "ecs-hands-on-for-web",  
    "taskRoleArn": "arn:aws:iam::<AWSアカウントID>:role/ECSTaskRole",  
    "executionRoleArn": "arn:aws:iam::<AWSアカウントID>:role/ECSTaskRole",  
    "networkMode": "awsvpc",  
    "containerDefinitions": [  
        {  
            "name": "nginx",  
            "image": "<AWSアカウントID>.dkr.ecr.ap-northeast-1.amazonaws.com/ecs-hands-on/nginx:latest",  
            "portMappings": [  
                {  
                    "containerPort": 80,  
                    "hostPort": 80,  
                    "protocol": "tcp"  
                }  
            ],  
            "essential": true,  
            "dependsOn": [ // ①  
                {  
                    "containerName": "laravel",  
                    "condition": "START"  
                }  
            ],  
            "readonlyRootFilesystem": false,  
            "logConfiguration": {  
                "logDriver": "awslogs",  
                "options": {  
                    "awslogs-group": "ecs-hands-on",  
                    "awslogs-region": "ap-northeast-1",  
                    "awslogs-stream-prefix": "nginx"  
                }  
            }  
        },  
        {  
            "name": "laravel",  
            "image": "<AWSアカウントID>.dkr.ecr.ap-northeast-1.amazonaws.com/ecs-hands-on/laravel:latest",  
            "essential": false,  
            "environment": [  
                {  
                    "name": "APP_ENV",  
                    "value": "production"  
                },  
                {  
                    "name": "APP_DEBUG",  
                    "value": "true"  
                },  
                {  
                    "name": "LOG_CHANNEL",  
                    "value": "awslogs"  
                }  
            ]  
        }  
    ]  
}
```

```
        "value": "stderr"
    }
],
"secrets": [
{
    "name": "APP_KEY",
    "valueFrom": "arn:aws:ssm:ap-northeast-1:<AWSアカウントID>:parameter/APP_KEY"
},
{
    "name": "APP_WORD",
    "valueFrom": "arn:aws:ssm:ap-northeast-1:<AWSアカウントID>:parameter/APP_WORD"
}

],
"privileged": false,
"readonlyRootFilesystem": false,
"logConfiguration": {
    "logDriver": "awslogs",
    "options": {
        "awslogs-group": "ecs-hands-on",
        "awslogs-region": "ap-northeast-1",
        "awslogs-datetime-format": "%Y-%m-%d %H:%M:%S",
        "awslogs-stream-prefix": "laravel"
    }
}
],
"requiresCompatibilities": [
    "FARGATE"
],
"cpu": "256",
"memory": "512"
}
```

dependsOn を使用することで、nginx のコンテナは、laravel のコンテナの起動を待って起動します（①）。

### 6.1.2 AWS Systems Manager パラメータストアに秘密情報を登録する

Laravel の動作に必要な APP\_KEY をパラメータストアに追加します。本来は ECS 用の APP\_KEY を生成するところですが、今回は手順の簡略のためローカル環境で使用した src/.env の APP\_KEY をコピーして使用します。

登録するパラメータを次に示します。

名前	APP_KEY
タイプ	安全な文字列
値	src/.env の APP_KEY

```
$ cat src/.env | grep '^APP_KEY'
```

▼src/.env の APP\_KEY をコピー

```
APP_KEY=base64:6cB51ynJgQ0WHkgSwieNzyln6TSdaxznfYn6bLWQvb8=
```

- パラメータストアにアクセスし、パラメータの作成ボタン（①）をクリックします（図 6.1）。



▲図 6.1

- 名前（①）に APP\_KEY を入力し、タイプ（②）に安全な文字列を選択します（図 6.2）。

AWS Systems Manager > パラメータストア > パラメータを作成

### パラメータを作成

**パラメータの詳細**

名前  ①

説明 — *Optional*

利用規約  
パラメータストアは標準パラメータおよび詳細パラメータを提供します。

標準  
パラメータの上限は 10,000 です。パラメータ値サイズの上限は 4 KB です。パラメータポリシーは使用できません。追加料金はありません。

詳細  
10,000 以上のパラメータを作成できます。パラメータ値サイズの上限は 8 KB です。パラメータポリシーを使用するよう設定できます。詳細パラメータごとに請求が発生します。

タイプ  文字列  
任意の文字列。

文字列のリスト  
カンマ区切りの文字列。

安全な文字列  
アカウントまたは別のアカウントの KMS キーを使用して、機密データを暗号化します。 ②

KMS キースース  
 現在のアカウント  
このアカウントのデフォルトの KMS キーを使用するか、このアカウント用のカスタマー管理キーを指定します。 詳細はこちら

別のアカウント  
別のアカウントの KMS キーを使用する 詳細はこちら

... 2

▲図 6.2

- 値 (①) にコピーした APP\_KEY の値 (base64 から始まる文字列) を設定したら、パラメータを作成ボタン (②) をクリックします (図 6.3)。

現在のアカウント  
このアカウントのデフォルトの KMS キーを使用するか、このアカウント用のカスタマー管理キーを指定します。 詳細はこちら

別のアカウント  
別のアカウントの KMS キーを使用する 詳細はこちら

KMS キー ID  
alias/aws/ssm

You have selected the default AWS managed key. All users in the current AWS account and Region will have access to this parameter. To restrict access to the parameter, use a customer managed key (CMK) instead.  
Learn more ③

値  
base64:6cB51ynJgQOWHkgSwieNzyln6TSdaxznfYn6bLWQvb8= ①

最大長は 4096 文字です。

タグ — *Optional*  
タグを使用して、パラメータへのアクセスを整理および制限できます。

タグを追加 ④

キャンセル パラメータを作成

▲図 6.3

- 一覧画面で追加したパラメータが登録されていることを確認します (図 6.4)。



▲図 6.4

### 6.1.3 タスク定義の登録

パラメータストアへの APP\_KEY の追加ができたので、次はタスク定義を登録しましょう。

```
$ aws ecs register-task-definition --cli-input-json file://ecs-task-definition-for-web.json
```

#### ▼ 実行結果

```
{
  "taskDefinition": {
    "taskDefinitionArn": "arn:aws:ecs:ap-northeast-1:<AWSアカウントID>:task-definition/ecs-hands-on-for-web:1",
    "containerDefinitions": [
      {
        "name": "nginx",
        "image": "<AWSアカウントID>.dkr.ecr.ap-northeast-1.amazonaws.com/ecs-hands-on/nginx:latest",
        "cpu": 0,
        "portMappings": [
          {
            "containerPort": 80,
            "hostPort": 80,
            "protocol": "tcp"
          }
        ],
        "essential": true,
        "environment": [],
        "mountPoints": [],
        "volumesFrom": [],
        "dependsOn": [
          {
            "containerName": "laravel",
            "condition": "START"
          }
        ],
        "readonlyRootFilesystem": false,
        "logConfiguration": {
          "logDriver": "awslogs",
          "options": {
            "awslogs-group": "/aws/containerLogs/ecs-hands-on-for-web/lambda@on",
            "awslogs-region": "ap-northeast-1",
            "awslogs-stream-prefix": "lambda"
          }
        }
      }
    ]
  }
}
```

```
        "awslogs-group": "ecs-hands-on",
        "awslogs-region": "ap-northeast-1",
        "awslogs-stream-prefix": "nginx"
    }
}
},
{
    "name": "laravel",
    "image": "<AWSアカウントID>.dkr.ecr.ap-northeast->
>1.amazonaws.com/ecs-hands-on/laravel:latest",
    "cpu": 0,
    "portMappings": [],
    "essential": false,
    "environment": [
        {
            "name": "LOG_CHANNEL",
            "value": "stderr"
        },
        {
            "name": "APP_DEBUG",
            "value": "true"
        },
        {
            "name": "APP_ENV",
            "value": "production"
        }
    ],
    "mountPoints": [],
    "volumesFrom": [],
    "secrets": [
        {
            "name": "APP_KEY",
            "valueFrom": "arn:aws:ssm:ap-northeast-1>
>:<AWSアカウントID>:parameter/APP_KEY"
        },
        {
            "name": "APP_WORD",
            "valueFrom": "arn:aws:ssm:ap-northeast-1>
>:<AWSアカウントID>:parameter/APP_WORD"
        }
    ],
    "privileged": false,
    "readonlyRootFilesystem": false,
    "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
            "awslogs-datetime-format": "%Y-%m-%d %H:>
>%M:%S",
            "awslogs-group": "ecs-hands-on",
            "awslogs-region": "ap-northeast-1",
            "awslogs-stream-prefix": "laravel"
        }
    }
}
```

```

        }
    ],
    "family": "ecs-hands-on-for-web",
    "taskRoleArn": "arn:aws:iam::<AWSアカウントID>:role/ECSTaskRole",
    "executionRoleArn": "arn:aws:iam::<AWSアカウントID>:role/>ECSTaskRole",
    "networkMode": "awsvpc",
    "revision": 1,
    "volumes": [],
    "status": "ACTIVE",
    "requiresAttributes": [
        省略
    ],
    "placementConstraints": [],
    "compatibilities": [
        "EC2",
        "FARGATE"
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "256",
    "memory": "512"
}
}

```

- Amazon ECS > タスク定義の一覧に作成したタスク定義が表示されていることを確認します（図 6.5）。



▲図 6.5

## 6.2 アプリケーションのデプロイ (ALB なし)

ECS サービスを使ってアプリケーションをデプロイする際、ALB とサービスを紐付けることができます。まずは、ALB を使用しない例を説明します。ALB とサービスを紐付ける方法については、「6.3 アプリケーションのデプロイ (ALB あり)」で紹介します。

### 6.2.1 サービスの作成

```
$ aws ecs create-service \
--cluster ecs-hands-on \
--service-name ecs-hands-on-laravel \
--task-definition ecs-hands-on-for-web \
--launch-type FARGATE \
--desired-count 1 \
--network-configuration "awsvpcConfiguration={subnets=[<作成済みのサブネットID>], securityGroups=[<作成済みのセキュリティグループID>], assignPublicIp=ENABLED}"
```

--desired-count は必要なタスクの数を指定します。サブネット ID とセキュリティグループ ID はそれぞれ、「4.2.1 サブネットを作成する」と「4.4 セキュリティグループを設定する」で作成した ID を指定します。

#### ▼ 実行結果

```
{
  "service": {
    "serviceArn": "arn:aws:ecs:ap-northeast-1:<AWSアカウントのID>:service/ecs-hands-on/ecs-hands-on-laravel",
    "serviceName": "ecs-hands-on-laravel",
    "clusterArn": "arn:aws:ecs:ap-northeast-1:<AWSアカウントのID>:cluster/ecs-hands-on",
    "loadBalancers": [],
    "serviceRegistries": [],
    "status": "ACTIVE",
    "desiredCount": 1,
    "runningCount": 0,
    "pendingCount": 0,
    "launchType": "FARGATE",
    "platformVersion": "LATEST",
    "taskDefinition": "arn:aws:ecs:ap-northeast-1:<AWSアカウントのID>:task-definition/ecs-hands-on-for-web:1",
    "deploymentConfiguration": {
      "maximumPercent": 200,
      "minimumHealthyPercent": 100
    },
    "deployments": [
      省略
    ],
  }
}
```

```

        "roleArn": "arn:aws:iam::<AWSアカウントのID>:role/aws-serv
>ice-role/ecs.amazonaws.com/AWSServiceRoleForECS",
      "events": [],
      省略
    }
}

```

## 6.2.2 動作確認

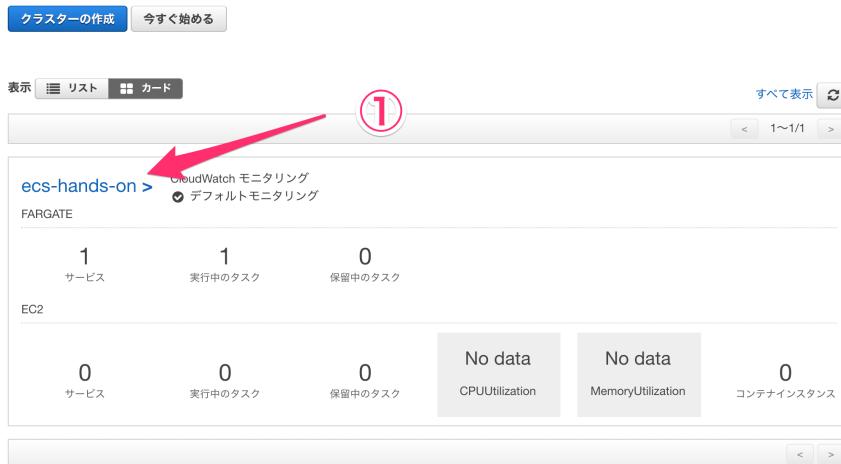
### クラスターの詳細画面を開く

- ECS のダッシュボードにアクセスし、クラスター名のリンク（①）をクリックします（図 6.6）。

### クラスター

Amazon ECS クラスターは、タスククリエストを実行できる 1 つ以上のコンテナインスタンスのリージョングループです。各アカウントは、最初に Amazon ECS サービスを使用するときにデフォルトのクラスターを受け取ります。クラスターは、複数の Amazon EC2 インスタンスタイプを含むことができます。

詳細については、[ECS のドキュメント](#)を参照してください。



▲図 6.6

### タスクの一覧画面を開く

- サービスタブ（①）を表示します。サービスの一覧に作成したサービスが表示されていることを確認します（②）。
- 確認ができたらタスクタブ（③）を選択します（図 6.7）。

## 6.2 アプリケーションのデプロイ (ALB なし)

クラスター > ecs-hands-on

クラスター : ecs-hands-on

クラスター上のリソースの詳細を取得します。

Cluster ARN: [REDACTED]

ステータス: ACTIVE

登録済みコンテナインスタンス: 0

保留中のタスクの数: 0 個の Fargate、0 個の EC2

実行中のタスクの数: 1 個の Fargate、0 個の EC2

アクティブサービス数: 1 個の Fargate、0 個の EC2

ドレインングサービス数: 0 個の Fargate、0 個の EC2

最終更新日: 2020年10月18日 10:54:41 午後 (0 分前)

サービス	タスク	ECS インスタンス	メトリクス	タスクのスケジューリング	Tags	キャッシュティープロバイダー
<input type="checkbox"/> サービス名	ステータス	サービス	タスク定義	必要な...	実行中...	起動タ...
<input type="checkbox"/> ecs-hands-on-laravel	ACTIVE	REPLICA	ecs-hands-on-for-web:1	1	1	FARGATE LATEST...

▲図 6.7

### タスクの詳細画面を開く

- タスクの一覧 (①) でタスクが実行されていることを確認します。
- タスク ID のリンク (②) をクリックし、タスクの詳細画面に移動します (図 6.8)。

クラスター > ecs-hands-on

クラスター : ecs-hands-on

クラスター上のリソースの詳細を取得します。

Cluster ARN: [REDACTED]

ステータス: ACTIVE

登録済みコンテナインスタンス: 0

保留中のタスクの数: 0 個の Fargate、0 個の EC2

実行中のタスクの数: 1 個の Fargate、0 個の EC2

アクティブサービス数: 1 個の Fargate、0 個の EC2

ドレインングサービス数: 0 個の Fargate、0 個の EC2

新しいタスクの実行

必要なタスクのステータス: (Running) Stopped

タスク	タスク定義	コンテ...	前回の...	必要な...	開始時...	開始元...	グループ...	起動タ...	プラッ...
<input type="checkbox"/> タスク...	ecs-hands-on-for-web:1	--	RUNNING	RUNNING	2020-1...	ecs-svc...	service:...	FARGATE	1.3.0

▲図 6.8

- Nginx と Laravel (PHP) のコンテナがタスク内で実行されていることを確認します (①)。
- ネットワークにて、割り当てられているパブリック IP を確認できます (②) (図 6.9)。

The screenshot shows the AWS CloudWatch Tasks console for an ECS task named 'ecs-hands-on'. The task has the following details:

- クラスター: ecs-hands-on
- タスク: 8a1b5e63eca24296b8ce5d0e06314c61
- 起動タイプ: FARGATE
- プラットフォームのバージョン: 1.3.0
- タスク定義: ecs-hands-on-for-web:1
- グループ: service:ecs-hands-on-laravel
- タスクロール: ECSTaskRole
- 前回のステータス: RUNNING
- 必要なステータス: RUNNING
- 作成時刻: 2020-10-18 22:53:14 +0900
- 開始時刻: 2020-10-18 22:53:57 +0900

**ネットワーク**

ネットワークモード	awsvpc
ENI ID	eni-0477ba5f87237269
サブネット ID	subnet-0255f3402d19bc3fb
プライベート IP	10.0.0.203
パブリック IP	13.113.59.79 (circled with red arrow)
Mac アドレス	06:d4:f6:c5:5e:92

**コンテナ**

名前	コンテナランタイム	ステータス	イメージ	イメージダイジェスト	CPU ユニット	ハードディスク	基本	リソース
nginx	9ff6f9e5558ff7046d3d...	RUNNING	[Image]	[Digest]	0	-/-	true	2bc74d...
laravel	0140420456a59ad67...	RUNNING	[Image]	[Digest]	0	-/-	false	3503b9...

▲図 6.9

### Web ページを表示する

- <http://パブリック IP/hello> にアクセスし、ページが表示されることを確認します (図 6.10)。

## Hello World

▲図 6.10

### 6.2.3 サービスの削除

動作確認ができたので、作成したサービスを削除してみましょう。サービスを使ってタスクを起動している場合、タスクを停止するためには、まずサービスを削除する必要があります。

#### サービスの詳細画面を開く

- クラスターの詳細画面にアクセスし、実行中のサービスのリンク（①）をクリックします（図 6.11）。

クラスター : ecs-hands-on

Cluster ARN : [REDACTED]

ステータス : ACTIVE

登録済みコンテナインスタンス : 0

保留中のタスクの数 : 0 個の Fargate、0 個の EC2

実行中のタスクの数 : 1 個の Fargate、0 個の EC2

アクティブサービス数 : 1 個の Fargate、0 個の EC2

ドレインングサービス数 : 0 個の Fargate、0 個の EC2

サービス	タスク	ECS インスタンス	メトリクス	タスクのスケジューリング	Tags	キャパシティーブロバイダー																
<a href="#">作成</a>	<a href="#">更新</a>	<a href="#">削除</a>	<a href="#">アクション</a>	最終更新日: 2020年10月18日 11:40:11 午後 (0 分前) <a href="#">戻る</a> <a href="#">?</a>																		
このページのフィルター <a href="#">起動タイプ</a> : ALL <a href="#">サービスタイプ</a> : ALL < 1-1 > <table border="1"> <thead> <tr> <th>サービス名</th> <th>ステータス</th> <th>サービ...</th> <th>タスク...</th> <th>必要な...</th> <th>実行中...</th> <th>起動タ...</th> <th>プラッ...</th> </tr> </thead> <tbody> <tr> <td><input type="checkbox"/> <a href="#">ecs-hands-on-laravel</a></td> <td>ACTIVE</td> <td>REPLICA</td> <td>ecs-han...</td> <td>1</td> <td>1</td> <td>FARGATE</td> <td>LATEST...</td> </tr> </tbody> </table>							サービス名	ステータス	サービ...	タスク...	必要な...	実行中...	起動タ...	プラッ...	<input type="checkbox"/> <a href="#">ecs-hands-on-laravel</a>	ACTIVE	REPLICA	ecs-han...	1	1	FARGATE	LATEST...
サービス名	ステータス	サービ...	タスク...	必要な...	実行中...	起動タ...	プラッ...															
<input type="checkbox"/> <a href="#">ecs-hands-on-laravel</a>	ACTIVE	REPLICA	ecs-han...	1	1	FARGATE	LATEST...															

▲図 6.11

### サービスを削除する

- サービスの詳細画面を開いたら、右上の削除ボタン（①）をクリックします（図 6.12）。

クラスター > ecs-hands-on > サービス: ecs-hands-on-laravel

サービス: ecs-hands-on-laravel

クラスター	ecs-hands-on	必要数	1
ステータス	ACTIVE	保留中の数	0
タスク定義	ecs-hands-on-for-web:3	実行中の数	1
サービスタイプ	REPLICAS		
起動タイプ	FARGATE		
サービスロール	AWSServiceRoleForECS		
Created By	[Redacted]		

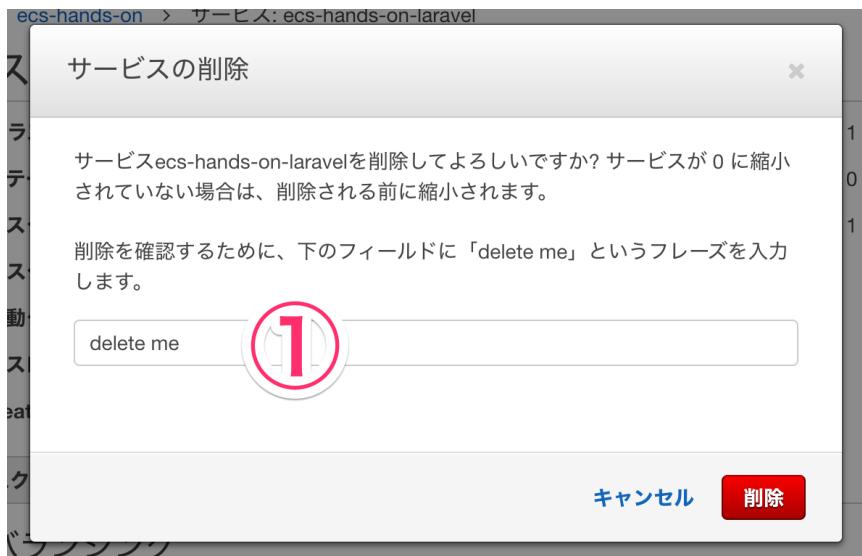
詳細 タスク イベント Auto Scaling デプロイメント メトリクス Tags ログ

ロードバランシング

ロードバランサー名	コンテナ名	コンテナポート
ロードバランサーがありません		

▲ 図 6.12

- 表示されたダイアログの指示に従い、テキストを入力し（①）、削除ボタンをクリックします（図 6.13）。



▲ 図 6.13

- サービスの一覧から、サービスが削除されていることを確認します（①）。

- 次にタスクタブ (②) をクリックします (図 6.14)。



▲ 図 6.14

### タスクの削除を確認する

- サービスを削除したあと、時間経過によってタスクが停止し、タスクの一覧からタスクが削除されます。
- もし時間が経過してもタスクが終了しない場合は、個別にタスクを停止させます (図 6.15)。



▲ 図 6.15

## 6.3 アプリケーションのデプロイ (ALB あり)

続いて、ECS サービスと ALB を紐付けて、アプリケーションをデプロイする方法について説明します。ALB を使用すると、ALB で受け付けたリクエストは ECS サービス経由でコンテナに割り振られます。

### 6.3.1 パブリックサブネットを追加する

ALB を作成するためには、同じアベイラビリティゾーンに存在しないサブ

ネットを2つ用意する必要があります。

一つは「4.2.1 サブネットを作成する」で作成済みのサブネットを使用します。ここではもう一つサブネットを作成します。作成するサブネットの内容を次に示します。

<b>名前タグ</b>	ecs-hands-on
<b>VPC</b>	「4.1 VPC を作成する」の VPC
<b>アベイラビリティーゾーン</b>	ap-northeast-1c
<b>IPv4 CIDR ブロック</b>	10.0.1.0/24

### サブネットの作成画面を開く

- VPC > サブネットのダッシュボード左上のサブネットの作成ボタン (①) を選択します (図 6.16)。



▲ 図 6.16

### サブネットの設定を入力する

- 名前タグ (①) に ecs-hands-on を入力します。
- VPC (②) には、「4.1 VPC を作成する」の VPC を選択します。
- アベイラビリティーゾーン (③) は、ap-northeast-1c (「4.2.1 サブネットを作成する」と別のアベイラビリティーゾーン) を設定します。
- IPv4 CIDR ブロック (④) は、10.0.1.0/24 を入力します。
- 各項目が入力できたら作成ボタン (⑤) をクリックします (図 6.17)。

## 6.3 アプリケーションのデプロイ (ALB あり)

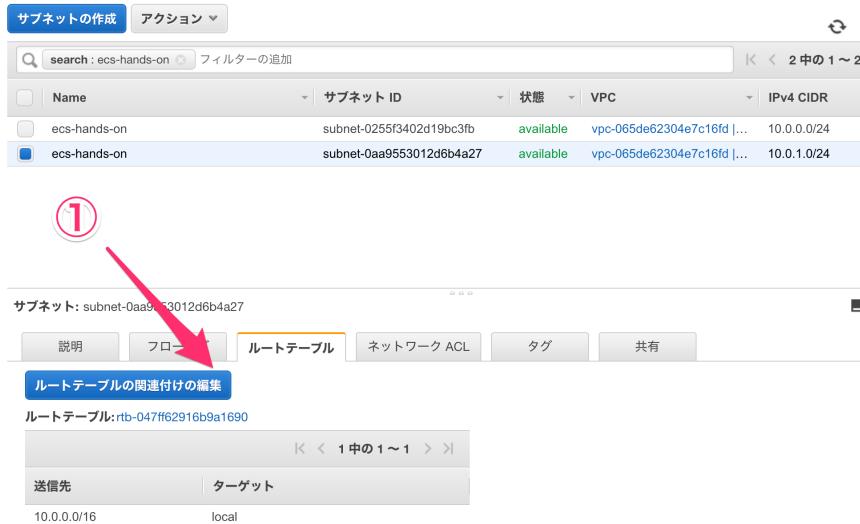


▲ 図 6.17

### 6.3.2 サブネットとルートテーブルを紐付ける

ルートテーブルの関連付けの編集画面を開く

- 作成したサブネットを選択し、画面下部のルートタブのルートの編集ボタンを押下します（図 6.18）。



▲ 図 6.18

### ルートテーブル ID を設定する

- ルートテーブルの関連付けの編集画面で、ルートテーブル ID に「4.3.1

「ルートテーブルを作成する」で作成したルートテーブルの ID を設定します。

- 設定ができたら保存ボタンを押下します（図 6.19）。



▲ 図 6.19

### 6.3.3 「パブリック IP アドレスの自動」を有効にする

「自動割り当て IP 設定の変更画面を開く」と同様に IP の自動割り当て設定を有効にします。

#### 自動割り当て IP 設定の変更画面を開く

- 作成したサブネットを右クリックします。表示されたメニューから、自動割り当て IP 設定の変更を選択します（図 6.20）。



▲ 図 6.20

#### IPv4 の自動割り当ての有効化

- IPv4 の自動割り当てにチェックを入れ（①）、保存ボタンをクリックします（図 6.21）。

## 6.3 アプリケーションのデプロイ (ALB あり)



▲ 図 6.21

### 6.3.4 ALB（ロードバランサー）を用意する

ECS のサービスと紐付ける ALB を作成しましょう。ALB の設定内容は次のとおりです。

名前	ecs-hands-on
スキーム	インターネット向け
VPC	「4.1 VPC を作成する」で作成した VPC
アベイラビリティーゾーン	作成済みのサブネット
セキュリティグループの設定	「4.4 セキュリティグループを設定する」のセキュリティグループ

#### ロードバランサーの作成画面を開く

- EC2 > ロードバランサーのダッシュボード左上のロードバランサー作成ボタン (①) を押下します (図 6.22)。



▲ 図 6.22

#### ALB の設定を選択・入力する

- Application Load Balancer の作成ボタン (①) をクリックします (図 6.23)。

## 第6章 Web アプリケーションをデプロイする

### ロードバランサーの種類の選択

Elastic Load Balancing は 3 種類のロードバランサー (Application Load Balancer、Network Load Balancer (新規)、および Classic Load Balancer) をサポートします。お客様のニーズに合うロードバランサーの種類を選択してください。

お客様に最も適なロードバランサーの詳細

Application Load Balancer	Network Load Balancer	Classic Load Balancer
 作成	 作成	以前の世代 HTTP, HTTPS, および TCP 作成

▲ 図 6.23

- 名前 (①) に ecs-hands-on と入力します。
- スキーム (②) はインターネット向けを選択します。
- VPC (③) は「4.1 VPC を作成する」で作成した VPC の ID を設定します (図 6.24)。

手順 1: ロードバランサーの設定

手順的基本的な設定

ロードバランサーを設定するには、名前を指定し、スキームを選択して、1つ以上のリスナーを指定し、ネットワークを選択します。デフォルトの設定は、ポート 80 で HTTP トрафィックを受信するリスナーを持つ、選択したネットワークのインターネット接続ロードバランサーです。

名前 (①)  ①

スキーム (②)  インターネット向け  内部

IP アドレスタイプ (③)

リスナー

リスナーとは、設定したプロトコルとポートを使用して接続リクエストをチェックするプロセスです。

ロードバランサーのプロトコル	ロードバランサーのポート
HTTP	80

アベイラビリティーゾーン

ロードバランサーのアベイラビリティーゾーンを指定します。ロードバランサーは、指定されたアベイラビリティーゾーンのみトraithックをルーティングします。アベイラビリティーゾーンごとに 1 つだけサブネットを指定できます。ロードバランサーの可用性を高めるには、2 つ以上のアベイラビリティーゾーンからサブネットを指定する必要があります。

VPC (①)  ③

キャンセル 次の手順: セキュリティ設定の構成

▲ 図 6.24

- アベイラビリティーゾーン (①、②) はそれぞれ作成済みのサブネットを選択します。
- 各項目を設定したら、次の手順ボタン (③) をクリックします (図 6.25)。

## 6.3 アプリケーションのデプロイ (ALB あり)

1. ロードバランサーの設定 2. セキュリティ設定の構成 3. セキュリティグループの設定 4. ルーティングの設定 5. ターゲットの登録 6. 確認

手順 1: ロードバランサーの設定  
つだけサブネットを指定できます。ロードバランサーの可用性を高めるには、2つ以上のアベイラビリティゾーンからサブネットを指定する必要があります。

VPC ①  vpc-065de62304e7c16fd [10.0.0.0/16] | ecs-hands-on

アベイラビリティゾーン  ap-northeast-1a  ap-northeast-1c

IPv4 アドレス ①  subnet-0255f3402d19bc3fb (ecs-hands-on) ① AWS によって割り当て済み

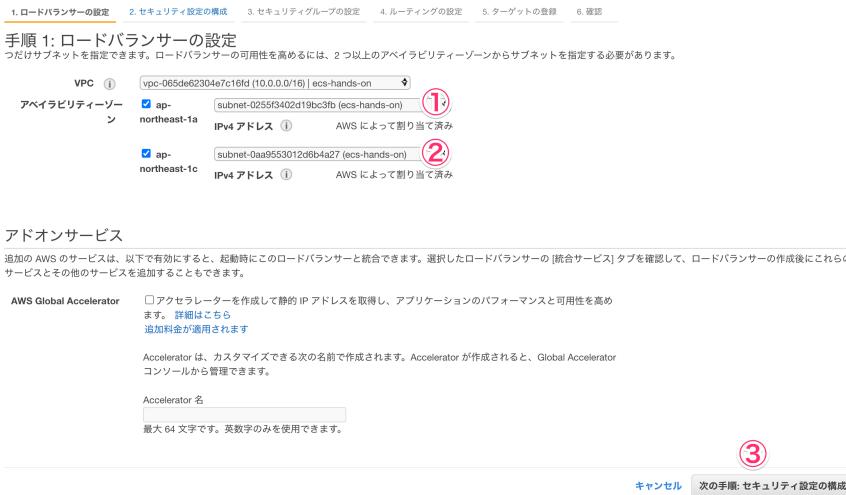
IPv4 アドレス ②  subnet-0aa9553012d6b4a27 (ecs-hands-on) ② AWS によって割り当て済み

アドオンサービス  
追加の AWS のサービスは、以下に有効になると、起動時にこのロードバランサーと結合できます。選択したロードバランサーの [統合サービス] タブを確認して、ロードバランサーの作成後にこれらのサービスとその他のサービスを追加することもできます。

AWS Global Accelerator  アクセラレーターを作成して静的 IP アドレスを取得し、アプリケーションのパフォーマンスと可用性を高めます。[詳細はこちら](#)  
[追加料金が適用されます](#)

Accelerator 名   
最大 64 文字です。英数字のみを使用できます。

キャンセル 次の手順: セキュリティ設定の構成 ③



▲ 図 6.25

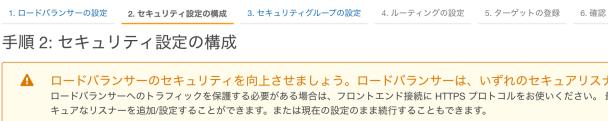
- 次の手順ボタン (①) をクリックします (図 6.26)。

1. ロードバランサーの設定 2. セキュリティ設定の構成 3. セキュリティグループの設定 4. ルーティングの設定 5. ターゲットの登録 6. 確認

手順 2: セキュリティ設定の構成

⚠ ロードバランサーのセキュリティを向上させましょう。ロードバランサーは、いずれのセキュアリスナーも使用していないません。  
ロードバランサーへのトラフィックを保護する必要がある場合は、フロントエンド接続に HTTPS プロトコルをお使いください。最初のステップに戻り、[基本的な設定](#) セクションでセキュアリスナーを追加/設定することができます。または現在の設定のまま続行することもできます。

キャンセル 戻る 次の手順: セキュリティグループの設定 ①



▲ 図 6.26

- セキュリティグループの設定で、「4.4 セキュリティグループを設定する」で作成したセキュリティグループの ID をチェックし (①)、  
次の手順ボタン (②) をクリックします (図 6.27)。

1. ロードバランサーの設定 2. セキュリティ設定の構成 3. セキュリティグループの設定 4. ルーティングの設定 5. ターゲットの登録 6. 確認

**手順 3: セキュリティグループの設定**  
セキュリティグループは、ロードバランサーへのトラフィックを制御するファイアウォールのルールセットです。このページで、特定のトラフィックに対してロードバランサーへの到達を許可するルールを追加できます。最初に、新しいセキュリティグループを作成するか、既存のセキュリティグループから選択するかを決定します。

セキュリティグループの割り当て: ○ 新しいセキュリティグループを作成する  
当て: ● 既存のセキュリティグループを選択する

セキュリティグループ ID	名前	説明	アクション
sg-00bed558e01959918	ecs-hands-on	ECS Sample	コピーして新規作成 コピーして新規作成

(1)

(2)

キャンセル 戻る 次の手順: ルーティングの設定

▲ 図 6.27

### ルーティングを設定する

次の内容でルーティングを設定します。

- 名前 (①) に ecs-hands-on を入力します。
- ターゲットの種類 (②) は IP を選択します。
- プロトコル (③) に HTTP を選び、ポートに 80 (④) を入力します。
- 入力後、次の手順ボタン⑤をクリックします (図 6.28)。

1. ロードバランサーの設定 2. セキュリティ設定の構成 3. セキュリティグループの設定 4. ルーティングの設定 5. ターゲットの登録 6. 確認

**手順 4: ルーティングの設定**  
ロードバランサーは、指定するプロトコルとポートを使用してこのターゲットグループのターゲットにリクエストをルーティングし、これらのヘルスチェック設定を使用してターゲットでヘルスチェックを実行します。各ターゲットグループは 1 つのロードバランサーのみに適用付けることができるることに注意してください。

ターゲットグループ

ターゲットグループ ①	新しいターゲットグループ
名前 ①	ecs-hands-on

ターゲットの種類

○ インスタンス
● IP (2)
○ Lambda 関数

プロトコル ①

HTTP (3)
----------

ポート ①

80 (4)
--------

ヘルスチェック

プロトコル ①	HTTP
パス ①	/

▶ ヘルスチェックの詳細設定

(5)

キャンセル 戻る 次の手順: ターゲットの登録

▲ 図 6.28

- ネットワークに「4.1 VPC を作成する」で作成した VPC を選択し (①)、  
次の手順ボタン (②) をクリックします (図 6.29)。

## 6.3 アプリケーションのデプロイ (ALB あり)

手順 5: ターゲットの登録  
ターゲットグループを登録します。有効にしたアベイラビリティゾーンでターゲットを登録する場合、登録処理が完了し、ターゲットが最初のヘルスチェックに合格するとすぐに、ロードバランサーはターゲットへのリクエストのルーティングを開始します。



▲ 図 6.29

- 設定内容を確認し、作成ボタン（①）を押下します。

手順 6: 確認  
続行する前にロードバランサーの詳細を確認してください。

ロードバランサー

名前 ecs-hands-on  
スキーLM Internal-facing  
リスナー ポート:80 - プロトコル:HTTP  
IP アドレスタイプ IPv4  
VPC vpc-065de62304e7c10fd (ecs-hands-on)  
サブネット subnet-0255f3402d19bc3fb (ecs-hands-on), subnet-0aa9553012d6b4a27 (ecs-hands-on)  
タグ

セキュリティグループ

セキュリティグループ sg-00bed558e01959918

ルーティング

ターゲットグループ 新しいターゲットグループ  
ターゲットグループ名 ecs-hands-on  
ポート 80  
ターゲットの種類 IP  
プロトコル HTTP  
ヘルスチェックプロトコル HTTP  
バス /  
ヘルスチェックポート traffic port  
正常応じきい値 5  
非正常応じきい値 2  
タイムアウト 5  
間隔 30  
成功コード 200

①

▲ 図 6.30

- ダッシュボードに作成した ALB が表示されていることを確認します。



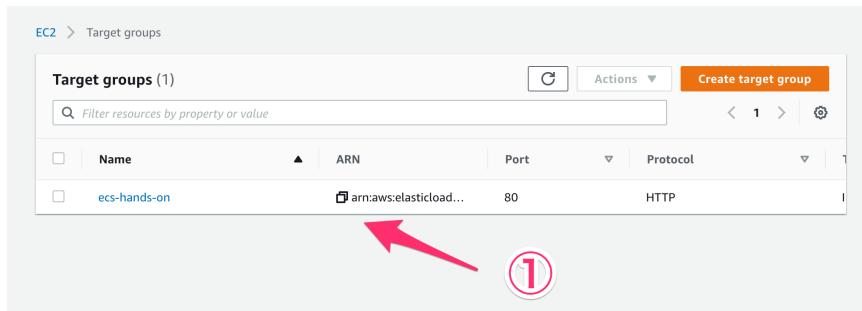
名前	DNS 名	状態	VPC ID	アベイラビリティーゾーン	種類
ecs-hands-on	██████████	provisioning	vpc-065de62304e7c16fd	ap-northeast-1a, ap-northeast-1c	app

▲ 図 6.31

### 6.3.5 ターゲットグループの ARN を確認

- EC2 > ターゲットグループにアクセスし、作成したロードバランサーの ARN をコピーします (①)。

この ARN は後述の「6.3.6 サービスの作成」で使用します (図 6.32)。



Name	ARN	Port	Protocol
ecs-hands-on	arn:aws:elasticload...	80	HTTP

▲ 図 6.32

### 6.3.6 サービスの作成

AWS CLI を使用して ECS サービスを作成します。ALB によってアクセスが振り分けられることを確認するため、--desired-count を 2 に設定し、タスクが複数起動するようにします。--load-balancers で Nginx のコンテナにリクエストを流すように指定します。使用するターゲットグループは「6.3.5 ターゲットグループの ARN を確認」で確認したターゲットグループ ARN を使用します。サブネット ID とセキュリティグループ ID も、事前に作成した ID を使用します。

```
$ aws ecs create-service \
--cluster ecs-hands-on \
--service-name ecs-hands-on-laravel \
--task-definition ecs-hands-on-for-web \
--launch-type FARGATE \
--load-balancers '[{"containerName": "nginx", "containerPort": 80, "targetGroupArn": "<ターゲットグループARN>"}]' \
```

```
--desired-count 2 \
--network-configuration "awsvpcConfiguration={subnets=[<サブネット
ID1>,<サブネットID2>],securityGroups=[<セキュリティグループID>],assignPu
>blicIp=ENABLED}"
```

#### ▼ 実行結果

```
{
  "service": {
    "serviceArn": "arn:aws:ecs:ap-northeast-1:<AWSアカウントの
ID>:service/ecs-hands-on/ecs-hands-on-laravel",
    "serviceName": "ecs-hands-on-laravel",
    "clusterArn": "arn:aws:ecs:ap-northeast-1:<AWSアカウントの
ID>:cluster/ecs-hands-on",
    "loadBalancers": [
      {
        "targetGroupArn": "arn:aws:elasticloadbalancing:>
ap-northeast-1:<AWSアカウントのID>:targetgroup/ecs-hands-on/20df3c>
>c11cf85f35",
        "containerName": "nginx",
        "containerPort": 80
      }
    ],
    "serviceRegistries": [],
    "status": "ACTIVE",
    "desiredCount": 2,
    "runningCount": 0,
    "pendingCount": 0,
    "launchType": "FARGATE",
    "platformVersion": "LATEST",
    "taskDefinition": "arn:aws:ecs:ap-northeast-1:<AWSアカウ
ントのID>:task-definition/ecs-hands-on-for-web:3",
    "deploymentConfiguration": {
      "maximumPercent": 200,
      "minimumHealthyPercent": 100
    },
    "deployments": [
      省略
    ],
    "roleArn": "arn:aws:iam:<AWSアカウントのID>:role/aws-serv
>ice-role/ecs.amazonaws.com/AWSServiceRoleForECS",
    "events": [],
    省略
    "enableECSManagedTags": false,
    "propagateTags": "NONE"
  }
}
```

#### 6.3.7 動作確認

ALB を使用してデプロイしたアプリケーションの動作を確認します。

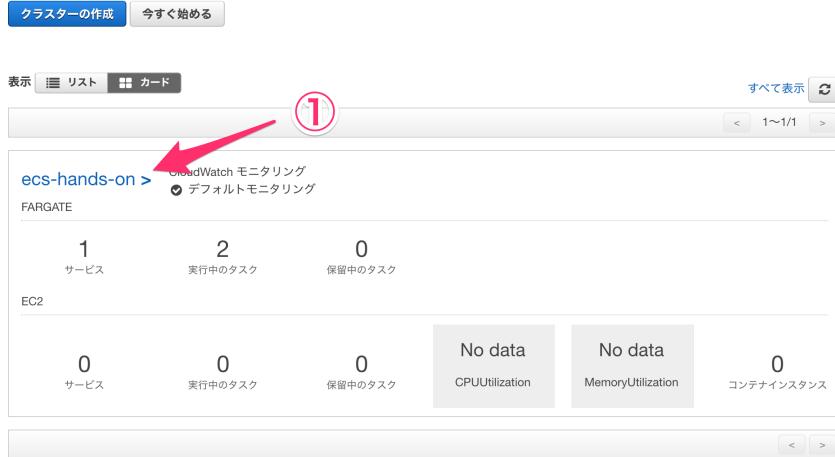
##### クラスターの詳細画面を開く

- ECS > クラスターのダッシュボードでクラスター名のリンク（①）をクリックします（図 6.33）。

## クラスター

Amazon ECS クラスターは、タスクリクエストを実行できる 1 つ以上のコンテナインスタンスのリージョングループです。各アカウントは、最初に Amazon ECS サービスを使用するときにデフォルトのクラスターを受け取ります。クラスターは、複数の Amazon EC2 インスタンスタイプを含むことができます。

詳細については、[ECS のドキュメント](#)を参照してください。



▲ 図 6.33

## サービスの詳細画面を開く

- サービスの一覧の実行中のサービス名のリンク（①）をクリックします（図 6.34）。

クラスター > ecs-hands-on

### クラスター : ecs-hands-on

クラスター上のリソースの詳細を取得します。

Cluster ARN: [REDACTED]  
ステータス: ACTIVE  
登録済みコンテナインスタンス: 0  
保留中のタスクの数: 0 個の Fargate、0 個の EC2  
実行中のタスクの数: 2 個の Fargate、0 個の EC2  
アクティブサービス数: 1 個の Fargate、0 個の EC2  
レイヤリングサービス数: 0 個の Fargate、0 個の EC2

サービス	タスク	ECS インスタンス	メトリクス	タスクのスケジューリング	Tags	キャバシティープロバイダー	
<a href="#">作成</a>	<a href="#">更新</a>	<a href="#">削除</a>	<a href="#">アクション ▾</a>	最終更新日: 2020年10月19日 12:55:09 午前 (0 分前) <a href="#">⟳</a> <a href="#">?</a>			
<a href="#">このページのフィルター</a>		起動タイプ: ALL	サービスタイプ: ALL	< 1-1 >			
<input type="checkbox"/> サービス名	ステータス	サービ... ス名	タスク... 数	必要な... ど	実行中... るタスク	起動タ... イプ	プラッ... ト
<input type="checkbox"/> <a href="#">ecs-hands-on-laravel</a>	ACTIVE	REPLICA	ecs-han... laravel	2	2	FARGATE	LATEST...

▲ 図 6.34

### タスクの一覧画面を開く

- サービスの詳細画面では、ロードバランシングにターゲットグループのコンテナが Nginx になっていることを確認します (①)。
- 次にタスクタブ (②) をクリックします (図 6.35)。

クラスター > [ecs-hands-on](#) > サービス: [ecs-hands-on-laravel](#)

### サービス: ecs-hands-on-laravel

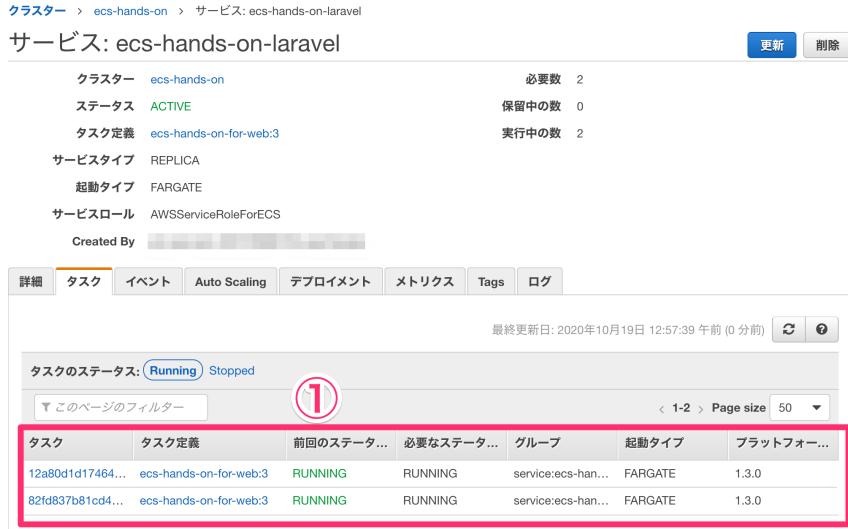
更新 削除

クラスター	ecs-hands-on	必要数	2
ステータス	ACTIVE	保留中の数	0
タスク定義	<a href="#">ecs-hands-on-for-web:3</a>	実行中の数	2
サービスタイプ	REPLICA		
起動タイプ	FARGATE		
サービスロール	AWSServiceRoleForECS		
Created By	[REDACTED]		

詳細	タスク	イベント	Auto Scaling	デプロイメント	メトリクス	Tags	ログ						
<b>ロードバランシング</b> <table border="1"> <tr> <th>ターゲットグループ名</th> <th>コンテナ名</th> <th>コンテナポート</th> </tr> <tr> <td>ecs-hands-on</td> <td>nginx</td> <td>80</td> </tr> </table> <span style="color: red;">①</span>								ターゲットグループ名	コンテナ名	コンテナポート	ecs-hands-on	nginx	80
ターゲットグループ名	コンテナ名	コンテナポート											
ecs-hands-on	nginx	80											
<b>ネットワークアクセス</b> <p>ヘルスチェックの猶予期間: 0</p> <p>許可された VPC: <a href="#">vpc-065de62304e7c16fd</a></p> <p>許可されたサブネット: <a href="#">subnet-0255f3402d19bc3fb, subnet-0aa9553012d6b4a27</a></p> <p>セキュリティグループ: <a href="#">sg-00bed558e01959918</a></p> <p>パブリック IP の自動割り当て: ENABLED</p>													

▲ 図 6.35

- タスク一覧でタスクが 2 つ起動していることを確認します (①) (図 6.36)。



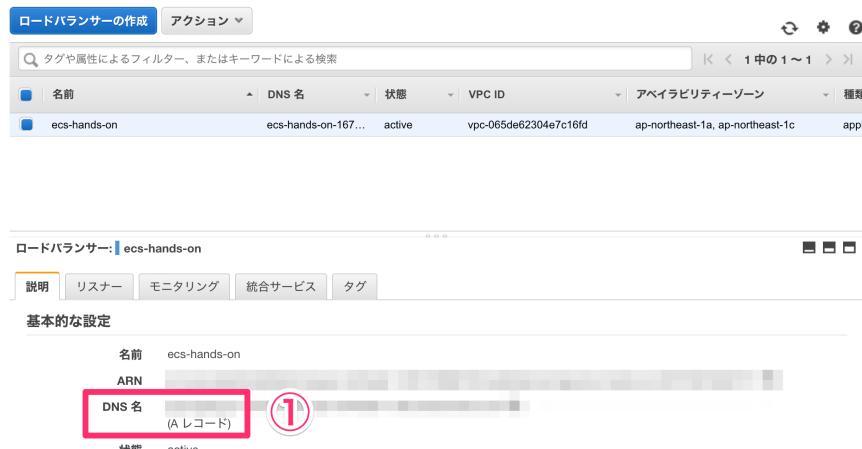
The screenshot shows the AWS CloudWatch Tasks console for the service 'ecs-hands-on-laravel'. It displays two tasks running under the task definition 'ecs-hands-on-for-web:3'. The tasks are labeled '12a80d1d17464...' and '82fd837b81cd4...', both in the 'RUNNING' state. A red box highlights the table containing these task details.

タスク	タスク定義	前回のステータス	必要なステータス	グループ	起動タイプ	プラットフォーム
12a80d1d17464...	ecs-hands-on-for-web:3	RUNNING	RUNNING	service:ecs-han...	FARGATE	1.3.0
82fd837b81cd4...	ecs-hands-on-for-web:3	RUNNING	RUNNING	service:ecs-han...	FARGATE	1.3.0

▲ 図 6.36

### ALB の DNS を確認する

- EC2 > ロードバランサーのダッシュボードで ALB の DNS 名 (①) を確認します (図 6.37)。



The screenshot shows the AWS Load Balancer console for the load balancer 'ecs-hands-on'. It displays the 'DNS 名' (DNS Name) field, which contains '(A レコード)' and is highlighted with a red box. A red circle with the number '1' is placed over the '(A レコード)' text.

▲ 図 6.37

### Web ページを表示する

- http://<DNS 名>/hello にアクセスし、CloudWatch にアクセスログが output されていることを確認します。

ALB によって、次の図のようにアクセスが振り分けられていることが確認できます（図 6.38、図 6.39）。

CloudWatch > CloudWatch Logs > Log groups > ecs-hands-on > nginx/nginx/12a80d1d174647a5acc29380765478c0

元のインターフェイスに切り替えます。

ログイベント	タイムスタンプ	メッセージ
▶	2020-10-19T01:00:46.744+09:00...	ロードする古いイベントがあります。 <a href="#">さらにロードします</a> 。
▶	2020-10-19T01:00:52.944+09:00...	10.0.0.226 - - [18/Oct/2020:16:00:46 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-Health...
▶	2020-10-19T01:01:16.744+09:00...	10.0.1.9 - - [18/Oct/2020:16:01:52 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-HealthCh...
▶	2020-10-19T01:01:22.943+09:00...	10.0.0.226 - - [18/Oct/2020:16:01:16 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-Health...
▶	2020-10-19T01:01:46.825+09:00...	10.0.1.9 - - [18/Oct/2020:16:01:22 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-Health...
▶	2020-10-19T01:01:52.944+09:00...	10.0.0.226 - - [18/Oct/2020:16:01:46 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-Health...
▶	2020-10-19T01:01:54.945+09:00...	10.0.1.9 - - [18/Oct/2020:16:01:54 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-HealthCh...
▶	2020-10-19T01:02:05.946+09:00...	10.0.0.226 - - [18/Oct/2020:16:02:05 +0000] "GET /hello HTTP/1.1" 200 21 "-" "Mozilla/5...
▶	2020-10-19T01:02:07.447+09:00...	10.0.0.226 - - [18/Oct/2020:16:02:07 +0000] "GET /hello HTTP/1.1" 200 21 "-" "Mozilla/...
▶	2020-10-19T01:02:08.643+09:00...	10.0.0.226 - - [18/Oct/2020:16:02:08 +0000] "GET /hello HTTP/1.1" 200 21 "-" "Mozilla/...
▶	2020-10-19T01:02:09.645+09:00...	10.0.0.226 - - [18/Oct/2020:16:02:09 +0000] "GET /hello HTTP/1.1" 200 21 "-" "Mozilla/...
▶	2020-10-19T01:02:10.940+09:00...	10.0.0.226 - - [18/Oct/2020:16:02:10 +0000] "GET /hello HTTP/1.1" 200 21 "-" "Mozilla/...
▶	2020-10-19T01:02:12.442+09:00...	10.0.0.226 - - [18/Oct/2020:16:02:12 +0000] "GET /hello HTTP/1.1" 200 21 "-" "Mozilla/...
▶	2020-10-19T01:02:16.843+09:00...	10.0.0.226 - - [18/Oct/2020:16:02:16 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-Health...

▲ 図 6.38

CloudWatch > CloudWatch Logs > Log groups > ecs-hands-on > nginx/nginx/82fd837b81cd4685ba6567c6195d0b7

元のインターフェイスに切り替えます。

ログイベント	タイムスタンプ	メッセージ
▶	2020-10-19T01:00:46.828+09:00...	ロードする古いイベントがあります。 <a href="#">さらにロードします</a> 。
▶	2020-10-19T01:00:52.928+09:00...	10.0.0.226 - - [18/Oct/2020:16:00:52 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-Health...
▶	2020-10-19T01:01:16.827+09:00...	10.0.1.9 - - [18/Oct/2020:16:01:16 +0000] "GET / HTTP/1.1" 200 17470 "-" "ELB-Health...
▶	2020-10-19T01:01:22.928+09:00...	10.0.0.226 - - [18/Oct/2020:16:01:22 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-Health...
▶	2020-10-19T01:01:46.908+09:00...	10.0.0.226 - - [18/Oct/2020:16:01:46 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-Health...
▶	2020-10-19T01:01:52.927+09:00...	10.0.1.9 - - [18/Oct/2020:16:01:52 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-HealthCh...
▶	2020-10-19T01:02:04.028+09:00...	10.0.0.226 - - [18/Oct/2020:16:02:04 +0000] "GET /hello HTTP/1.1" 200 21 "-" "Mozilla/...
▶	2020-10-19T01:02:16.929+09:00...	10.0.0.226 - - [18/Oct/2020:16:02:16 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-Health...
▶	2020-10-19T01:02:23.018+09:00...	10.0.1.9 - - [18/Oct/2020:16:02:23 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-HealthCh...
▶	2020-10-19T01:02:46.927+09:00...	10.0.0.226 - - [18/Oct/2020:16:02:46 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-Health...
▶	2020-10-19T01:02:53.025+09:00...	10.0.1.9 - - [18/Oct/2020:16:02:53 +0000] "GET / HTTP/1.1" 200 17485 "-" "ELB-HealthCh...

▲ 図 6.39

### 6.3.8 サービスの削除

「6.2.3 サービスの削除」と同様の手順で、サービスを削除し、タスクを停止

させます。

### 6.3.9 AWS リソースの削除

本書で作成した ALB や VPC やサブネット、ECS クラスターなどのリソースは不要になった段階で削除することを推奨します。

# あとがき

@aiiro29 (寺田 晃大) / サーバサイドエンジニア

(同) セイランソフト代表。

SIer、受託開発企業を経験したのち、フリーランスに転向しました。普段は Laravel で開発することが多いです。

Twitter: @aiiro29

# 使って学ぶ AWS ECS 入門

Fargate と Laravel でコンテナデプロイを体験

---

2020 年 12 月 26 日 ver 1.0

著 者 寺田晃大

---

© 2020 Project Airo

(powered by Re:VIEW Starter)