

株式会社デュークシ

Think only pleasant thoughts
Survive just by having fun

ECSとCodePipelineでウェブアプリ ケーションを構築する

著者: 丸山浩之

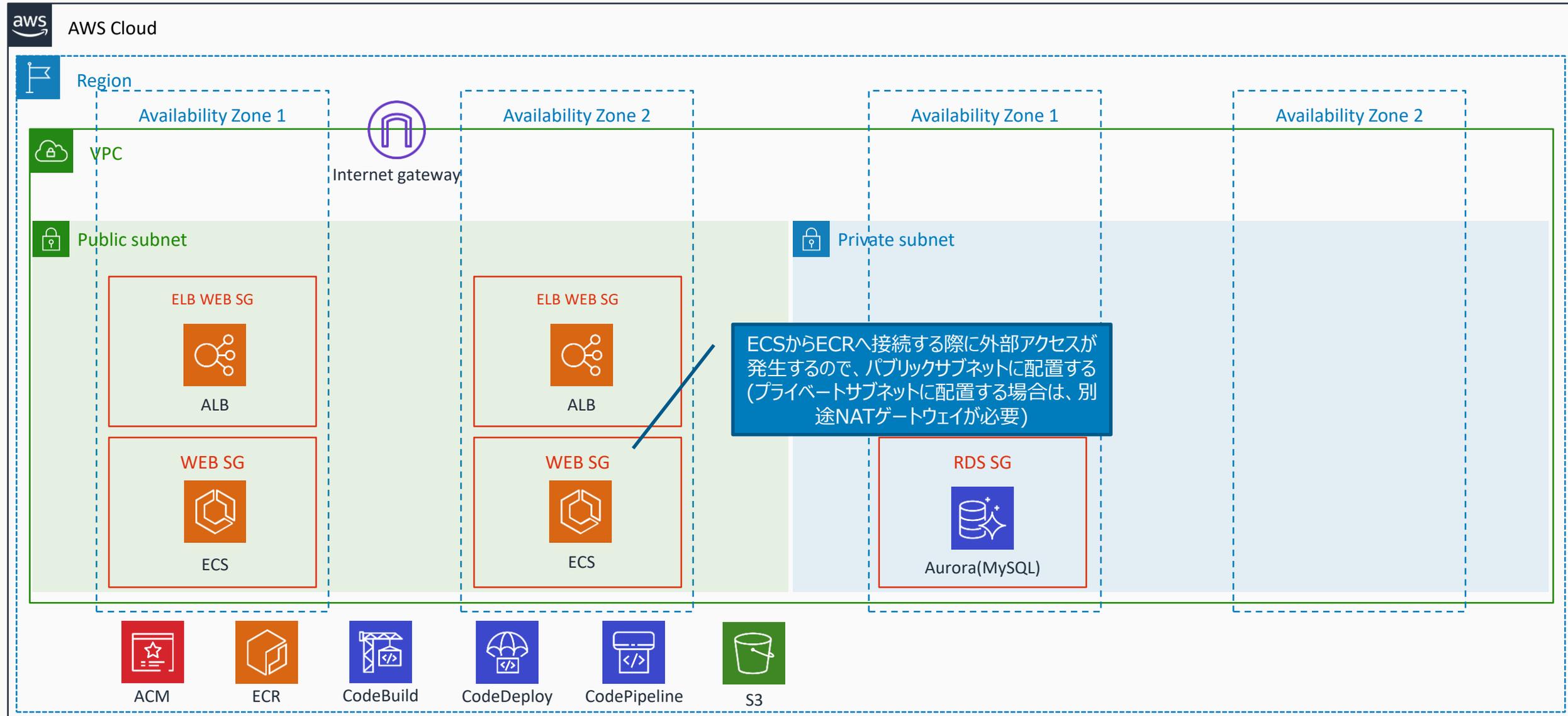
目次

- [目的](#)
- [全体構成図](#)
- [VPC編](#)
 - [VPC作成](#)
 - [サブネット作成](#)
 - [インターネットゲートウェイ作成](#)
 - [インターネットゲートウェイをVPCにアタッチ](#)
 - [パブリックサブネット用のルートテーブル作成](#)
 - [サブネット関連付け](#)
 - [ルート編集](#)
 - [セキュリティグループ作成](#)
 - [セキュリティグループのインバウンドルール編集](#)
- [RDS編](#)
 - [サブネットグループ作成](#)
 - [データベース作成](#)
- [SSL証明書発行編](#)
 - [証明書リクエスト](#)
- [ELB\(ALB\)編](#)
 - [ロードバランサー作成](#)
 - [リスナー編集](#)
- [コンテナ編](#)
 - [リポジトリ作成](#)
 - [リポジトリにプッシュ](#)
 - [タスク定義](#)
 - [クラスター作成](#)
 - [サービス作成](#)
- [デプロイ編](#)
 - [ビルドプロジェクト作成](#)
 - [パイプライン作成](#)
- [APPENDIX編](#)
 - [プロジェクト構成](#)
 - [appspec.ymlのサンプル](#)
 - [buildspec.ymlのサンプル](#)
 - [taskdef.jsonのサンプル](#)
 - [サンプルコード](#)
 - [サンプルコードをローカル環境で利用する](#)

目的

1. CakePHP 3.8.10で制作したウェブアプリケーションを、ECS(Fargate)とCodePipeline を利用して公開する
2. 公開する際は、ACMを利用してSSL対応する

全体構成図



VPC編

VPC作成

VPC > VPC の作成

VPC の作成

VPC は、Amazon EC2 インスタンスなどの AWS オブジェクトによって使用される AWS クラウドの分離された部分です。VPC の IPv4 アドレス範囲を指定する必要があります。クラスレスドメイン間ルーティング (CIDR) のブロックとして、IPv4 アドレス範囲 (例: 10.0.0.0/16) を指定します。/16 より大きな IPv4 CIDR ブロックを指定することはできません。オプションで、Amazon が提供した IPv6 CIDR ブロックを VPC に関連付けることができます。

名前タグ

myapp



判別しやすい名称をつける

IPv4 CIDR ブロック*

172.16.0.0/21



サブネットマスク「/21」なので、使用可能IPアドレスは下記の通り。
172.16.0.1 ~ 172.16.7.254
(あらかじめ必要なIPレンジを設計するとよい)

IPv6 CIDR ブロック

- IPv6 CIDR ブロックなし
- Amazon が提供した IPv6 CIDR ブロック
- IPv6 CIDR owned by me



テナンシー

デフォルト



* 必須

キャンセル

作成

サブネット作成

1. サブネットは、同一の役割で使用するものを、異なるアベイラビリティゾーンで2つ以上作成する(リージョンごとに使用できるAZは異なる)
2. 今回は、ALB向けとECSやDB向けを作成するため、下記の6つを作成する
 1. myapp-subnet-public-1 / myapp-subnet-public-2
 2. myapp-subnet-ecs-1 / myapp-subnet-ecs-2
 3. myapp-subnet-rds-1 / myapp-subnet-rds-2

サブネット > サブネットの作成

サブネットの作成

CIDR 形式でサブネットの IP アドレスブロックを指定します (例: 10.0.0.0/24)。IPv4 ブロックサイズは、/16 ネットマスクから /28 ネットマスクの間で、VPC と同じサイズとすることができます。IPv6 CIDR ブロックは /64 CIDR ブロックである必要があります。

名前タグ ⓘ

VPC* ⓘ

アベイラビリティゾーン ⓘ

VPC CIDR

CIDR	Status	Status Reason
172.16.0.0/21	associated	

IPv4 CIDR ブロック* ⓘ

判別しやすい名称をつける

作成したVPCを選択

適当に、aから順に選択

172.16.[0-7].0の範囲で指定する。

* 必須

キャンセル 作成

インターネットゲートウェイ作成

インターネットゲートウェイ > インターネットゲートウェイの作成

インターネットゲートウェイの作成

インターネットゲートウェイは、VPC をインターネットに接続する仮想ルーターです。新しく作成するには、ゲートウェイの名前を以下から指定します。

名前タグ

myapp-igw



判別しやすい名称をつける

* 必須

キャンセル

作成

インターネットゲートウェイをVPCにアタッチ

インターネットゲートウェイ > VPC にアタッチ

VPC にアタッチ

インターネットゲートウェイを VPC にアタッチし、インターネットとの通信を有効にします。以下に添付す

VPC*

作成したVPCを選択

▶ AWS コマンドラインインターフェイスコマンド

* 必須

キャンセル

アタッチ

パブリックサブネット用のルートテーブル作成

- 1. VPC作成と同時に作られたメインルートテーブルとは別に、パブリックサブネット用のルートテーブルを作成する

ルートテーブル > ルートテーブルの作成

ルートテーブルの作成

ルートテーブルは、VPC、インターネット、および VPN 接続内のサブネット間でパケットがどのように転送

名前タグ ⓘ

VPC* ⓘ

判別しやすい名称をつける

作成したVPCを選択

* 必須

キャンセル

サブネット関連付け

1. パブリックサブネット用のルートテーブルに、サブネットの関連付けを行う
2. 外部公開するサブネットを選択する(下記の4つを外部公開向けサブネットとして使用する)
 1. myapp-subnet-public-1 / myapp-subnet-public-2
 2. myapp-subnet-ecs-1 / myapp-subnet-ecs-2

ルートテーブル > サブネットの関連付けの編集

サブネットの関連付けの編集

ルートテーブル rtb-06a80cc3424ed2c67 (myapp-public-rt)

関連付けられたサブネット subnet-056eed1a1abfbae54 subnet-070837b4572304329

検索 属性によるフィルター、またはキーワードによる検索

4 中の 1 ~ 4

<input type="checkbox"/>	サブネット ID	IPv4 CIDR	現在のルートテーブル
<input type="checkbox"/>	subnet-004d647c5e97986b5 myapp-subnet-rds-2	172.16.3.0/24	メイン
<input checked="" type="checkbox"/>	subnet-070837b4572304329 myapp-subnet-public-1	172.16.0.0/24	メイン
<input checked="" type="checkbox"/>	subnet-056eed1a1abfbae54 myapp-subnet-public-2	172.16.1.0/24	メイン
<input type="checkbox"/>	subnet-0630430a709a9a339 myapp-subnet-rds-1	172.16.2.0/24	メイン

* 必須

キャンセル

保存

ルート編集

1. パブリックサブネット用のルートテーブルを外部公開するために、ルートにインターネットゲートウェイを追加する

ルートテーブル > ルートの編集

ルートの編集

送信先	ターゲット	ステータス	伝播済み
172.16.0.0/21	local	active	いいえ
0.0.0.0/0	igw-0257b7d41bbb67aac		いいえ

作成したインターネットゲートウェイを選択

ルートの追加

* 必須

キャンセル **ルートの保存**

セキュリティグループ作成

1. セキュリティグループは、ELB用、コンテナ用、RDS用という具合に細分化したほうが、アクセス制限に対応しやすい
2. 今回は、下記の3つを作成する
 1. myapp-elb-sg
 2. myapp-web-sg
 3. myapp-rds-sg

セキュリティグループ > セキュリティグループの作成

セキュリティグループの作成

セキュリティグループは、インスタンスの仮想ファイアウォールとして機能し、インバウンドトラフィックとアウトバウンドトラフィックをコントロールします。新しいセキュリティグループを作成するには、以下のフィールドに入力します。

セキュリティグループ名* myapp-elb-sg

説明* ELB

VPC vpc-0b705cab945b494dd

判別しやすい名称をつける

適当な説明をつける

作成したVPCを選択

* 必須

キャンセル

作成

セキュリティグループのインバウンドルール編集

セキュリティグループ > インバウンドのルールの編集

インバウンドのルールの編集

インバウンドのルールはインスタンスへの入力を許可された受信トラフィックを制御します。

タイプ ⓘ	プロトコル ⓘ	ポート範囲 ⓘ	ソース ⓘ	説明 ⓘ	
HTTP ▼	TCP	80	カスタム ▼ 0.0.0.0/0, ::/0	HTTP for ECS	✕
HTTPS ▼	TCP	443	カスタム ▼ 0.0.0.0/0, ::/0	HTTPS for ECS	✕
カスタム TCP ルー... ▼	TCP	8888	カスタム ▼ 0.0.0.0/0, ::/0	HTTP for ECS Test	✕
カスタム TCP ルー... ▼	TCP	8443	カスタム ▼ 0.0.0.0/0, ::/0	HTTPS for ECS Test	✕

各セキュリティグループのインバウンドルールは次ページ

ルールの追加

注意: 既存のルールを編集すると、編集したルールが削除されて、新しい詳細を含む新しいルールが作成されます。これにより、そのルールに依存するトラフィックは、新しいルールが作成されるまで非常に短時間切断されます。

* 必須

キャンセル **ルールの保存**

セキュリティグループのインバウンドルール編集(設定内容)

1. 外部からの接続を制限したい場合は、myapp-elb-sgのソースに接続元IPアドレスを指定する

セキュリティグループ	プロトコル	ポート	ソース	説明
myapp-elb-sg	TCP	80	0.0.0.0/0, ::/0	HTTP
myapp-elb-sg	TCP	443	0.0.0.0/0, ::/0	HTTPS
myapp-elb-sg	TCP	8888	0.0.0.0/0, ::/0	HTTPのテストポート(CodeDeployで必要)
myapp-elb-sg	TCP	8443	0.0.0.0/0, ::/0	HTTPSのテストポート(CodeDeployで必要)
myapp-web-sg	TCP	80	グループID	myapp-elb-sgのグループID
myapp-web-sg	TCP	443	グループID	myapp-elb-sgのグループID
myapp-web-sg	TCP	8888	グループID	myapp-elb-sgのグループID
myapp-web-sg	TCP	8443	グループID	myapp-elb-sgのグループID
myapp-rds-sg	TCP	3306	グループID	myapp-web-sgのグループID

RDS編

サブネットグループ作成(サブネットグループの詳細)

1. RDSインスタンスに設定するサブネットグループを作成する(myapp-rds-subnet)
2. 画面が長かったので2ページに渡って説明

RDS > サブネットグループ > DB サブネットグループの作成

DB サブネットグループの作成

新しいサブネットグループを作成するには、名前と説明を入力し、既存の VPC を選択します。次に、その VPC に関連するサブネットを追加できます。

サブネットグループの詳細

名前
サブネットグループの作成後に名前を変更することはできません。

1 ~ 255 文字を含める必要があります。英数字、スペース、ハイフン、アンダースコア、ピリオドを使用できます。

説明

VPC
DB サブネットグループに使用するサブネットに対応する VPC 識別子を選択します。サブネットを選択することはできません。

判別しやすい名称をつける

適切な説明をつける

作成したVPCを選択

サブネットグループ作成(サブネットの追加)

加することもできます。このグループの作成後、追加/編集ができます。最低で2つのサブネットが必要です。

アベイラビリティゾーン

サブネット

各アベイラビリティゾーンから、RDS用のサブネットを選択して追加

このサブネットグループのサブネット (2)

アベイラビリティゾーン	サブネット ID	CIDR ブロック	アクション
us-west-2b	subnet-004d647c5e97986b5	172.16.3.0/24	<input type="button" value="削除"/>
us-west-2a	subnet-0630430a709a9a339	172.16.2.0/24	<input type="button" value="削除"/>

データベース作成

1. RDSはアプリケーションの要件に合わせて適当なものを選択して作成する
2. 今回はAurora(MySQL)を選択して作成する
3. 設定内容はアプリケーションの要件にあわせて設定する
 1. セキュリティグループはRDS用に作成したものを使用する(ここではmyapp-rds-sgを使用する)

RDS > データベースの作成

データベースの作成

データベース作成方法を選択 情報

標準作成
可用性、セキュリティ、バックアップ、メンテナンスといったすべての設定オプションを設定します。

簡単作成
推奨されるベストプラクティス設定を使用します。一部の設定オプションは、データベースの作成後に変更できます。

簡単作成は設定できる項目が少ないので標準作成を選択

エンジンのオプション

エンジンのタイプ 情報

Amazon Aurora

MySQL

MariaDB

今回はAuroraを選択

SSL証明書発行編

証明書リクエスト

1. パブリック証明書のリクエストを行う
2. ドメインは事前に用意する
3. 検証方法はDNS検証を利用する
 1. 確定するとDNSレコードに設定する値が表示(またはCSVダウンロード)されるので、ドメインを登録しているDNSサーバーに設定する
4. DNSレコード登録後、早ければ10分ぐらいで証明書が発行される
5. なお、DNSサーバーはRoute53でなくてもよい(弊社はPointDNSを使用)

ELB(ALB)編

ロードバランサー作成(基本的な設定)

1. Application Load Balancer(ALB)を作成する

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

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 トラフ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 ⓘ 判別しやすい名称をつける

スキーム ⓘ インターネット向け 内部

IP アドレスタイプ ⓘ

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
----------------	--------------

[キャンセル](#) [次の手順: セキュリティ設定の構成](#)

ロードバランサー作成(リスナー)

1. 通常のHTTP/HTTPSの他に、テスト用のポートを追加する(CodeDeployでデプロイする際のテストで利用する)

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

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	HTTPのテストポート
HTTP	8888	
HTTPS (セキュア HTTP)	443	HTTPSのテストポート
HTTPS (セキュア HTTP)	8443	

リスナーの追加

キャンセル 次の手順: セキュリティ設定の構成

ロードバランサー作成(アベイラビリティゾーン)

1. パブリックサブネット用のルートテーブルに登録したサブネットを指定する

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

アベイラビリティゾーン

ロードバランサーのアベイラビリティゾーンを指定します。ロードバランサーは、これらのアベイラビリティゾーンにのみトラフィックをルーティングします。アベイラビリティゾーンごとに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ゾーンからサブネットを指定する必要があります。

VPC ⓘ	vpc-0b705cab945b494dd (172.16.0.0/21) myapp
アベイラビリティゾーン	<input checked="" type="checkbox"/> us-west-2a subnet-070837b4572304329 (myapp-subnet-public-1) ▼
	IPv4 アドレス ⓘ AWS によって割り当て済み
	<input checked="" type="checkbox"/> us-west-2b subnet-056eed1a1abfbae54 (myapp-subnet-public-2) ▼
	IPv4 アドレス ⓘ AWS によって割り当て済み

パブリックサブネットのルートテーブルに登録したサブネット

パブリックサブネットのルートテーブルに登録したサブネット

キャンセル 次の手順: セキュリティ設定の構成

ロードバランサー作成(デフォルトの証明書の選択)

1. 事前にACMで作成した証明書を選択する

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

にデプロイできます。HTTPS リスナーおよび証明書の管理については、[詳細はこちら](#)。

証明書タイプ ⓘ

- ACM から証明書を選択する (推奨)
- 証明書を ACM にアップロードする (推奨)
- IAM から証明書を選択する
- IAM に証明書をアップロードする

ACM から新しい証明書をリクエスト
AWS Certificate Manager は AWS プラットフォーム上での SSL 証明書のプロビジョニング、管理、デプロイ、および更新を簡単にします。ACM によって、証明書の更新が行われます。 [詳細はこちら](#)

証明書の名前 ⓘ myapp.duxi.co.jp (arn:aws:acm:us-west-2:188495475513:certificate/8c7) 🔄

ACMで作成した証明書

セキュリティポリシーの選択

セキュリティポリシー ⓘ ELBSecurityPolicy-2016-08 ▼

キャンセル 戻る 次の手順: セキュリティグループの設定

ロードバランサー作成(セキュリティグループの設定)

1. ELB用のセキュリティグループを選択する

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 既存のセキュリティグループを選択する

フィルタ: VPC セキュリティグループ ▼

セキュリティグループ ID	名前	説明	アクション
<input type="checkbox"/> sg-017435ac43afc0bf5	default		コピーして新規作成
<input checked="" type="checkbox"/> sg-0a975653375de8583	myapp-elb-sg	ELB	コピーして新規作成
<input type="checkbox"/> sg-039dcc2bbf4448e63	myapp-rds-sg	RDS	コピーして新規作成
<input type="checkbox"/> sg-0fb0e2eaba673d5cd	myapp-web-sg	WEB	コピーして新規作成

キャンセル 戻る 次の手順: ルーティングの設定

ロードバランサー作成(ルーティングの設定、ターゲットの登録)

1. CodeDeployの設定を行うときにターゲットグループは改めて作成するため、ここでは適当な設定でターゲットグループを作成する
 1. ターゲットグループの名前だけ適当に入力して、あとはデフォルトのままでも問題ない(インスタンス追加などしなくてもよい)
 2. このターゲットグループは不要で削除するため、説明は割愛する

リスナー編集

1. 作成したロードバランサーのリスナーを、下表の設定になるよう編集する
2. HTTPは全部対応するHTTPSへ転送し、HTTPSは固定レスポンスを返すようにしておく
 1. HTTPSは、CodeDeployのターゲット作成を行うときにルールが追加される予定
 2. 固定レスポンスの「レスポンス本文」は空白でも問題ない(アプリケーションの要件で決める)
3. **終わったら、ロードバランサー作成時にできたターゲットグループは削除しておく**
4. **SSL証明書を発行した際に指定したドメインのDNSレコードに、ロードバランサーのDNS名(Aレコード)を登録しておく**
 1. **myapp.duxi.co.jp** で発行した場合、**myapp.duxi.co.jp** のCNAMEレコードを作り、そこへDNS名を設定する

リスナーID	ルール
HTTP : 80	デフォルト: 次の宛先にリダイレクト中: HTTPS://#{host}:443/#{path}?#{query}
HTTPS : 443	デフォルト: 固定レスポンスを返しています 503
HTTPS : 8443	デフォルト: 固定レスポンスを返しています 503
HTTP : 8888	デフォルト: 次の宛先にリダイレクト中: HTTPS://#{host}:8443/#{path}?#{query}

コンテナ編

リポジトリ作成

1. ECRでアプリケーションのコンテナイメージを格納するリポジトリを作成する
2. 今回はCakePHP 3.8.10で構築することを前提としているため、下記の2リポジトリを作成する
 1. nginx
 2. php-fpm

ECR > リポジトリ > リポジトリを作成

リポジトリを作成

リポジトリの設定

リポジトリ名

188495475513.dkr.ecr.us-west-2.amazonaws.com/

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イメージタグ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。

無効にする

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。

有効にする

キャンセル

リポジトリにプッシュ

1. 作成したリポジトリにイメージをプッシュする
2. 事前にECRへプッシュするための、IAMユーザーを用意しておく
 1. ポリシー「AmazonEC2ContainerRegistryFullAccess」があればよい
 2. ユーザーを作成する際、アクセスキーIDとシークレットアクセスキーを取得しておく
3. 作成したリポジトリを選択するとプッシュコマンドを見ることができる(詳細は次ページ)

ECR > リポジトリ > nginx

nginx

プッシュコマンドの表示

イメージ (0) 🔄 削除 スキャン

🔍 イメージを検索 < 1 > ⚙️

<input type="checkbox"/>	Image タグ	イメージの URI	プッシュされた日時 ▼	ダイジェスト	サイズ (MB) ▼	ステータス
イメージがありません 表示するイメージがありません						

リポジトリにプッシュ(プッシュコマンド)

1. 「プッシュコマンドの表示」をクリックして表示された内容をもとにすすめる
 1. WSLの利用を前提とするため、コマンドは「macOS / Linux」を使用する
 2. AWS CLIが必要になるので、別途インストールを済ませておく(執筆時点のバージョンは「1.18.26」)
 3. Dockerビルドを行うので、Docker Desktopなどをインストールしてdockerコマンドを使用できるようにしておく
2. CakePHP 3.8.10で作成したプロジェクトのカレントディレクトリに移動する
 1. プロジェクトの構成についてはAPPENDIXを参照
3. `aws ecr get-login-password --region [リポジトリを作成したリージョン] --profile [事前作成したIAMユーザーのプロファイル] | docker login --username AWS --password-stdin [アカウントID].dkr.ecr.[リポジトリを作成したリージョン].amazonaws.com/nginx`
4. nginxのビルドとプッシュ
 1. `docker build -t nginx -f docker/nginx/Dockerfile .`
 2. `docker tag nginx:latest [アカウントID].dkr.ecr.[リポジトリを作成したリージョン].amazonaws.com/nginx:latest`
 3. `docker push [アカウントID].dkr.ecr.[リポジトリを作成したリージョン].amazonaws.com/nginx:latest`
5. php-fpmのビルドとプッシュ
 1. `docker build -t php-fpm -f docker/php-fpm/Dockerfile .`
 2. `docker tag nginx:latest [アカウントID].dkr.ecr.[リポジトリを作成したリージョン].amazonaws.com/nginx:latest`
 3. `docker push [アカウントID].dkr.ecr.[リポジトリを作成したリージョン].amazonaws.com/nginx:latest`

タスク定義

1. 起動タイプの互換性は「FARGATE」を選択する

タスクとコンテナの定義の設定

タスク定義では、タスクに含めるコンテナとコンテナ間のやり取りの方法を指定します。また、コンテナに使用するデータボリュームを指定することもできます。[詳細はこちら](#)

タスク定義名* myapp-web 判別しやすい名称をつける

互換性が必要* FARGATE FARGATEを選択

タスクロール
権限付与された AWS サービスへの API リクエストを行うためにタスクで使用できるオプションの IAM ロール。Amazon Elastic Container Service タスクロールは [IAM コンソール](#) で作成します。🔗

ネットワークモード awsvpc ⓘ
<デフォルト> を選択すると、ECS は Docker のデフォルトネットワークモード (Linux on Bridge と Windows の NAT) を使用してコンテナを起動します。<デフォルト> は Windows で唯一サポートされ

タスク定義

1. タスク実行ロールは、はじめてタスク定義を行うときは「新しいロールの作成」が選択されているが、これはそのままにしておく(タスク作成と同時に実行ロールも作成される)
2. 2回目以降にタスク定義を行うときは、自動作成した「ecsTaskExecutionRole」が選択されている

るモードです。

⚠ ネットワークモード : awsvpc
タスク中のコンテナは共通のネットワークスタックを使用して ENI を共有します。ポートマッピングではコンテナポートのみを指定できます (既存のホストポートの指定は削除されます)。

タスクの実行 IAM ロール

このロールは、お客様に代わってコンテナイメージをプルし、コンテナログを Amazon CloudWatch に発行するタスクに必要です。まだ ecsTaskExecutionRole を持っていない場合は、お客様のためにそれを作成することが

タスク実行ロール **自動作成した実行ロール**

タスクサイズ

タスクサイズにより、タスクの固定サイズを指定できます。Fargate 起動タイプを使用したタスクにはタスクサイズが必須で、EC2 起動タイプではオプションです。タスクサイズが設定されている場合、コンテナレベルのメモリ設定はオプションです。タスクサイズは Windows コンテナではサポートされません。

タスク定義

1. タスクサイズのタスクメモリとタスク CPU(vCPU)は、アプリケーションの要件にあわせて設定する(今回は最小の構成を選択した)

タスク実行ロール ⓘ

タスクサイズ ⓘ

タスクサイズにより、タスクの固定サイズを指定できます。Fargate 起動タイプを使用したタスクにはタスクサイズが必須で、EC2 起動タイプではオプションです。タスクサイズが設定されている場合、コンテナレベルのメモリ設定はオプションです。タスクサイズは Windows コンテナではサポートされません。

タスクメモリ (GB) ▼
0.25 vCPU の有効なメモリ範囲: 0.5GB - 2GB。

タスク CPU (vCPU) ▼
0.5 GB メモリの有効な CPU: 0.25 vCPU

コンテナメモリの予約用のタスクメモリの最大割り当て

0 512 共有 512 MiB

コンテナへの CPU の最大割り当て

0 256 共有 256 CPU ユニット数

タスク定義

1. コンテナは、nginxとphp-fpmの2つを追加する
2. nginx : コンテナ名「nginx」、イメージ「[アカウントID].dkr.ecr.[リポジトリを作成したリージョン].amazonaws.com/nginx:latest」、ポートマッピング : 80
3. php-fpm : コンテナ名「php-fpm」、イメージ「[アカウントID].dkr.ecr.[リポジトリを作成したリージョン].amazonaws.com/php-fpm:latest」、ポートマッピング : 9000

0 256 共有 256 CPU ユニット数

コンテナの定義 ?

[コンテナの追加](#)

コンテナ名 ...	イメージ	ハード/ソフ...	CPU ユニ...	GPU	Inference ア...	基本	
 nginx	1884954755...	--/--				true	
 php-fpm	1884954755...	--/--				true	

サービス統合

AWS App Mesh は、マイクロサービスの監視と操作を容易にするための Envoy プロキシに基づいたサービスマッシュです。App Mesh は、マイクロサービスの通信方法を標準化して、エンドツーエンドを可視化し、アプリケーションの高可用性維持に役立てます。App Mesh 統合を有効にするには、次のフィールドに入力してから **[適用]** を選択します。これによりプロキシ設定が自動設定されます。 [詳細はこちら](#)

App Mesh 統合の有効化

プロキシ設定

タスク定義

1. その他の触れていない項目についてはデフォルトの状態でよい

FireLens の統合を有効にする

ボリューム

ボリューム設定を使用して、タスク内のコンテナが使用するボリュームを追加します。ボリュームを追加するには、**[ボリュームを追加]**を選択して、フィールドに入力し、**[追加]**を選択します。[詳細はこちら](#)

+ ボリュームの追加

JSON による設定

Tags

キー	値
<input type="text" value="キーの追加"/>	<input type="text" value="値を追加してください"/>

*必須

キャンセル

クラスター作成

1. テンプレートは「AWS Fargate を使用」を選択する

クラスターの設定

クラスター名* 判別しやすい名称をつける

ネットワーキング

クラスターで使用する新しい VPC を作成します。VPC は、Fargate タスクなどの AWS オブジェクトによって取得される AWS クラウドの分離された部分です。

VPC の作成 このクラスター用の新しい VPC を作成する

Tags

キー	値
<input type="text" value="キーの追加"/>	<input type="text" value="値を追加してください"/>

CloudWatch Container Insights

CloudWatch Container Insights はコンテナ化されたアプリケーションとマイクロサービス用のモニタリングおよびトラブルシューティングソリューションです。CPU、メモリ、ディスク、ネットワークなどの計算使用率や、コンテナ再起動失敗などの診断情報を、収集、集約、要約してクラスターの問題を切り分け、迅速に解決します。 [詳細はこちら](#)

CloudWatch Container Insights Container Insights を有効にする

*必須

キャンセル 戻る 作成

サービス作成(設定)

1. CodeDeploy用のロールが必要になるため、事前にユースケース「CodeDeploy - ECS」から作成したロールを準備しておく

サービスの設定

サービスでは、クラスターで実行して維持するタスク定義のコピー数を指定できます。オプションで Elastic Load Balancing ロードバランサーを使用して、受信トラフィックをサービス内のコンテナに分散させることができます。Amazon ECS はタスクの数を維持し、ロードバランサーを使用してタスクのスケジュールを調整します。オプションで Service Auto Scaling を使用して、サービス内のタスクの数を調整することもできます。

起動タイプ FARGATE EC2

FARGATEを選択

タスク定義 ファミリー

myapp-web

作成したタスク

値を入力

リビジョン

1 (latest)

プラットフォームのバージョン

LATEST

i

クラスター

myapp-cluster

作成したクラスター

i

サービス作成(設定)

サービス名	<input type="text" value="myapp-web"/>		判別しやすい名称をつける
サービスタイプ*	REPLICA		
タスクの数	<input type="text" value="1"/>		要件で数を決める

デプロイメント

サービスのデプロイメントオプションを選択してください。

デプロイメントタイプ* ローリングアップデート  Blue/Green デプロイメント (AWS CodeDeploy を使用) 

これは AWS CodeDeploy をサービスのデプロイメントコントローラーとして設定します。CodeDeploy アプリケーションとデプロイメントグループは、サービス用に自動的に作成されます。default settings サービスの作成後にローリングアップデートのデプロイメントタイプに変更するには、サービスを再作成して「ローリングアップデート」のデプロイメントタイプを選択する必要があります。

サービス作成(設定)

1. ロールを事前に作成していない状態について確認することができなかつたので、事前にロールを作成していなかった場合は、空白かまたはAWSが自動作成したロールが選択されるかもしれない

Deployment configuration* CodeDeployDefault.ECSAllAtOnce ▼

The deployment configuration specifies how traffic is shifted to the updated Amazon ECS task set. [Learn more](#)

CodeDeploy のサービスロール* ecs-codedeploy ▼

サービスが承認済み AWS サービスへの API リクエストを行うために使用する IAM ロール。IAM コンソールで CodeDeploy のサービスロールを作成します。 [詳細はこちら](#)

i タグ付けするには、新しい ARN とリソース ID の形式をオプトインする必要があります。IAM ユーザーまたはロールが新しい ARN 形式をオプトインしていません。新しい形式をオプトインしてこの機能を使用します。 [オプトイン設定を管理します。](#)

*必須

キャンセル **次のステップ**

事前に作成したロールが
選択されている

サービス作成(ネットワーク構成)

1. サブネットはECS用に作成したものを選択する

ネットワーク構成

VPC とセキュリティグループ

VPC とセキュリティグループは、タスク定義のネットワークモードが `awsvpc` であるときに設定可能です。

クラスター VPC* ⓘ

サブネット*

subnet-0d06c760edb534b96 ⓘ
(172.16.4.0/24) | myapp-subnet-ecs-1 - us-west-2a
作成時に ipv6 を割り当てます。 Disabled

subnet-0da04f1ba029b9ad0 ⓘ
(172.16.5.0/24) | myapp-subnet-ecs-2 - us-west-2b
作成時に ipv6 を割り当てます。 Disabled

ⓘ **ECS用サブネット**

サービス作成(ネットワーク構成)

1. WEB用に作成したセキュリティグループを選択する(本資料では「myapp-web-sg」が該当する)

セキュリティグループ* sg-0fb0e2eaba673d5cd 編集

パブリック IP の自動割り当て ENABLED

ヘルスチェックの猶予期間

サービスのタスクで、ELB ヘルスチェックを開始して応答するまでに時間がかかる場合は、ヘルスチェックの猶予期間として最大 2,147,483,647 秒まで指定できます。この間は、ECS サービススケジューラは ELB ヘルスチェックのステータスを無視します。この猶予期間により、ECS サービススケジューラがタスクを異常とマークして時間より前に停止することがなくなります。これが有効であるのは、ロードバランサーを使用するようにサービスが設定されている場合のみです。

ヘルスチェックの猶予期間 0

ロードバランシング

Elastic Load Balancing ロードバランサーはサービス内で実行中のタスク間に受信トラフィックを分散させます。既存のロードバランサーを選択するか、[Amazon EC2 コンソール](#)でロードバランサーを作成してください。

WEB用セキュリティグループ

サービス作成(ネットワーク構成)

1. 作成したロードバランサーを選択する

ロードバランサーの種類*

Application Load Balancer
コンテナで動的ホストポートマッピングを有効にします (コンテナインスタンスごとに複数のタスクを許可)。ルールベースのルーティングとパスを使用することで、1つのロードバランサーの同じリスナーポートを複数のサービスで共有できます。

Network Load Balancer
Network Load Balancer は、Open Systems Interconnection (OSI) モデルの第 4 層で機能します。ロードバランサーが、リクエストを受信した後、フローハッシュルーティングアルゴリズムを使用して、デフォルトルールターゲットグループからターゲットを選択します。

サービス用の IAM ロールの選択

awsvpc ネットワークモードを使用するタスク定義は、自動的に作成される AWSServiceRoleForECS サービスリンクロールを使用します。 [詳細はこちら](#)

ロードバランサー名

ロードバランサー名 myapp-alb-web

ロードバランサー用のコンテナ

nginx : 80 削除 ✕

ALBを選択

複数のALBを作成していた場合は、CodeDeployで使うものを選択

サービス作成(ネットワーク構成)

1. ロードバランス用のコンテナとして「nginx」を選択し、ロードバランサーに追加する
2. 追加後は、下記の図のとおりとなる
3. プロダクションリスナーポートは、アプリケーションが通常のリクエストを受け付けるポートになるため、443ポートを指定する

プロダクションリスナーポート* 443:HTTPS ⓘ

プロダクションリスナープロトコル* HTTPS

テストリスナー

トラフィックをルーティングする前に、オプションのテストリスナーを使用して新しいアプリケーションリビジョンをテストします。

テストリスナーポート* 8443:HTTPS ⓘ

テストリスナープロトコル* HTTPS

▼ Additional configuration

AWS CodeDeploy で Blue/Green デプロイメントを容易にするために、2つのターゲットグループが必要です。各ターゲットグループは、デプロイメント内の個別のタスクセットにバインドします。 [Learn more](#)

443を選択

デプロイ時のテストに使われるポート

サービス作成(ネットワーク構成)

1. ターゲットグループを2つ設定する
2. プロトコルはHTTPのままよい(コンテナが80で待ち受けているため)
3. パスパターンは、ここではひとつしか設定できないが、あとからロードバランサーのリスナー設定で追加することができる

ターゲットグループ 1 の名前* ⓘ

ターゲットグループ 1 プロトコル*

ターゲットの種類* ⓘ

パスパターン* 評価順

パスパターン: リスナーの最初のパスパターンは、デフォルトのパス (/) であり、別のルールに一致しないすべてのトラフィックに一致します。他のサービス用に、このリスナーにパターンと優先順位の値を後で追加できます。

評価順: ルールは優先順位の低いものから高いものへと評価されます。パスパターンルールに一致したら、他のすべてのルールは無視されます。サービスごとにパスを作成することで、このリスナーからのトラフィックを複数のサービスにルーティングできます。

このリスナーで使用されている既存のパス

パスには、アプリケーションで使用する可能性があるすべてのパスを含める必要があります。たとえば、パス /webapp1* が含まれたサービスは、このリスナーで

判別しやすい名称をつける

パスパターンはあとからリスナーの設定で追加可能

サービス作成(ネットワーク構成)

1. ヘルスチェックパスは、通常時にHTTPステータスコード200を返すパスであればよい
2. 例えば特定のHTMLファイルをリクエストして200を返すことがテストになるようであれば、そのHTMLファイルのパスを指定する(/healthy.htmlなど)

位の値を後で追加できます。

評価順: ルールは優先順位の低いものから高いものへと評価されます。パスパターンルールに一致したら、他のすべてのルールは無視されます。サービスごとにパスを作成することで、このリスナーからのトラフィックを複数のサービスにルーティングできます。

このリスナーで使用されている既存のパス

パスには、アプリケーションで使用する可能性があるすべてのパスを含める必要があります。たとえば、パス /webapp1* が含まれたサービスは、このリスナーで /webapp1 and /webapp1/page.html に送信されたトラフィックを受信します。一意のパスを選択することをお勧めします。また、評価の順序を低くすると、複数の競合するパス間でトラフィックをルーティングできます。

評価順	ルールパス	ターゲットグループ
default	/	

ヘルスチェックパス*

専用のパスがあればそれを指定する

追加のヘルスチェックオプションは、サービスの作成後、ELB コンソールで設定できます。

サービス作成(ネットワーク構成)

サービスの検出 (オプション)

サービスの検出では、Amazon Route 53 を使用してサービスの名前空間を作成します。これにより、サービスは DNS を介して検出可能になります。

サービスの検出の統合の有効化

名前空間*

local | プライベート



サービスの検出サービスの設定

- 新しいサービスの検出サービスの作成
- 既存のサービスの検出サービスの選択

サービスの検出名*

myapp-web



英数字とアンダースコアの文字列 (中にピリオドを含む) は有効です。詳細については、[Route 53 Auto Naming](#) のドキュメントを参照してください。

サービス作成(ネットワーク構成)

サービスの検索名	myapp-web
----------	-----------

英数字とアンダースコアの文字列 (中にピリオドを含む) は有効です。詳細については、[Route 53 Auto Naming](#) のドキュメントを参照してください。

ECS タスク状態の伝達の有効化

このフィールドを有効にすると、ECS はタスク状態を Route 53 に伝え、異常なタスクを DNS から削除するためにかかる時間を減らします。

Docker ヘルスチェック Docker ヘルスチェックは、タスク定義で定義されます。これにより、必要不可欠な集約コンテナの複数のヘルスチェックで、サービスの状態を確認できます。

パブリック DNS ヘルスチェックの有効化 プライベート名前空間の DNS ヘルスチェックは、現在サポートされていません。

サービス作成(ネットワーク構成)

パブリック DNS ヘルスチェックの有効化 プライベート名前空間の DNS ヘルスチェックは、現在サポートされていません。

サービスの検出の DNS レコード

DNS レコード型*

A



TTL*

60

秒



[+ DNS レコードの追加](#)

*必須

キャンセル

戻る

次のステップ

サービス作成(Auto Scaling)

1. 必要であれば設定するが、今回は使用しない

サービス作成(確認)

1. 最終的な設定内容は下図の通り

確認

サービスを作成する前に、設定を確認してください。

サービス

編集

サービス名	myapp-web
クラスター	myapp-cluster
起動タイプ	FARGATE
タスク定義	myapp-web:1
タスクの数	1

ネットワーク

編集

サービス作成(確認)

VPC ID vpc-0b705cab945b494dd

サブネット subnet-0d06c760edb534b96, subnet-0da04f1ba029b9ad0

選択したセキュリティグループ sg-0fb0e2eaba673d5cd

IP の自動割り当て ENABLED

ロードバランシング

ロードバランサーの種類 Application Load Balancer (ALB)

ロードバランサー名 myapp-alb-web

ロードバランス用のコンテナ nginx

コンテナポート 80

サービス作成(確認)

ターゲットグループ 1 の名前 myapp-web-1

プロダクションリスナーのパスパターン /*

プロダクションリスナーのヘルスチェックパス /

テストリスナーポート 8443

ターゲットグループ 2 の名前 myapp-web-2

テストリスナーのパスパターン /*

テストリスナーのヘルスチェックパス /

サービスの検出の設定

編集

サービス作成(確認)

サービスの検出名の設定

名前空間 ns-ctzxpymvn3bt3eav

サービスの検出名 myapp-web

ECS タスク状態の伝達の有効化 true

DNS レコード型および TTL A 60

編集

Auto Scaling (オプション)

not configured

編集

キャンセル **戻る** **サービスの作成**

デプロイ編

ビルドプロジェクト作成

1. .envを利用している場合は、s3のバケットに事前に.envをアップロードしておき、buildspec.ymlのpre_buildでコピーすると良い
 1. aws s3 cp s3://bucket_name/.env path/to/.env
 2. ビルドプロジェクトに設定するロールに、「bucket_name」へのアクセス権限を付与しておく

開発者用ツール > CodeBuild > ビルドプロジェクト > ビルドプロジェクトを作成する

ビルドプロジェクトを作成する

プロジェクトの設定

プロジェクト名

プロジェクト名は 2~255 文字にする必要があります。アルファベット (A~Z、a~z)、番号 (0~9)、特殊文字 (- と _) を含めることができます。

説明 - オプション

ビルドバッジ - オプション

ビルドバッジを有効にする

判別しやすい名称をつける

ビルドプロジェクト作成

1. ソースコードはGitHubのリポジトリを使用する
 1. 自分のアカウントから参照できるリポジトリを利用する場合は、事前にGitHubのパーソナルアクセストークンを取得しておく

The screenshot shows a configuration panel for a build project. At the top, there is a section for '追加設定' (Additional Settings) with a 'タグ' (Tag) field. Below this is the '送信元' (Source) section, which includes a 'ソースの追加' (Add Source) button. The 'ソース 1 - プライマリ' (Source 1 - Primary) section is active, showing 'ソースプロバイダ' (Source Provider) set to 'GitHub'. Under 'リポジトリ' (Repository), 'パブリックリポジトリ' (Public Repository) is unselected, and 'GitHub アカウントのリポジトリ' (GitHub Account Repository) is selected. A blue callout box labeled 'リポジトリ選択' (Repository Selection) points to the selected option. Below, the 'GitHub リポジトリ' (GitHub Repository) field contains the URL 'https://github.com/jupitris/cakephp3-ecs-demo.git', with a search icon, a clear button, and a refresh button. A template URL 'https://github.com/<user-name>/<repository-name>' is shown at the bottom.

ビルドプロジェクト作成

1. オペレーティングシステムは「Amazon Linux 2」を選択し、あとは図の通りに選択する

環境

環境イメージ

マネージド型イメージ
AWS CodeBuild によって管理されたイメージの使用

カスタムイメージ
Docker イメージの指定

オペレーティングシステム

Amazon Linux 2

Amazon Linux 2を選択

i プログラミング言語のランタイムが Ubuntu 18.04 の標準イメージに含まれるようになりました。これは、コンソールで作成される新しい CodeBuild プロジェクトに推奨されています。詳細については、CodeBuild で提供される Docker イメージ [🔗](#) を参照してください。

ランタイム

Standard

イメージ

aws/codebuild/amazonlinux2-x86_64-standard:3.0

ビルドプロジェクト作成

1. サービスロールについては、CodeBuild用のロールを作っていない場合は新規に作成する
2. 利用するロールに「AmazonEC2ContainerRegistryFullAccess」を付与しておく(ビルド実行時に権限エラーが発生するため)

イメージのバージョン
aws/codebuild/amazonlinux2-x86_64-standard:3.0-20.03.13

環境タイプ
Linux

特権付与
 Docker イメージを構築するか、ビルドで昇格されたアクセス権限を取得するには、このフラグを有効にします

サービスロール
 新しいサービスロール
アカウントでサービスロールを作成

既存のサービスロール
アカウントから既存のサービスロールを選択

ロール名
myapp-codebuild-service-role
サービスロール名の入力

▼ 追加設定

特権付与はチェック

ロールがなければ新規作成

ビルドプロジェクト作成

1. buildspec.ymlで使用する環境変数を定義する
2. リポジトリに登録できないような定義は、環境変数を使うと良い

▼

コンピューティング

3 GB メモリ、2 vCPU

7 GB メモリ、4 vCPU

15 GB メモリ、8 vCPU

145 GB メモリ、72 vCPU

環境変数

名前	値	入力	
<input type="text" value="APP_NAME"/>	<input type="text" value="myapp"/>	<input type="text" value="プレーンテキスト"/> ▼	<input type="button" value="削除"/>
<input type="text" value="AWS_ACCOUNT_ID"/>	<input type="text" value="188495475513"/>	<input type="text" value="プレーンテキスト"/> ▼	<input type="button" value="削除"/>
<input type="text" value="AWS_DEFAULT_REGION"/>	<input type="text" value="us-west-2"/>	<input type="text" value="プレーンテキスト"/> ▼	<input type="button" value="削除"/>

ビルドプロジェクト作成

1. リポジトリに登録したbuildspec.ymlを指定する(別名の場合は、そのファイル名を指定する)

ファイルシステムを追加

Buildspec

ビルド仕様

buildspec ファイルを使用する
build コマンドを YAML 形式の buildspec ファイルに保存

ビルドコマンドの挿入
ビルドプロジェクト設定としてビルドコマンドを保存

Buildspec 名 - オプション
CodeBuild のデフォルトでは、buildspec.yml という名前のファイルがソースコードルートディレクトリで検索され、ビルドプロジェクト設定としてビルドコマンドを保存します。別の名前または場所を使用している場合は、ここにソースルートからのパス (buildspec-two.yml や configuration/buildspec.yml) を入力します。

buildspec.yml

ファイル名入力

アーティファクト

アーティファクトの追加

ビルドプロジェクト作成

1. ビルドプロジェクト作成後、一度ビルドを実行してすべて正常に完了するか確認しておくが良い

ログ

CloudWatch

CloudWatch Logs - オプション
このオプションをチェックすると、ビルド出力ログが CloudWatch にアップロードされます。

グループ名

ストリーム名

S3

S3 ログ - オプション
このオプションをチェックすると、ビルド出力ログが S3 にアップロードされます。

キャンセル **ビルドプロジェクトを作成する**

パイプライン作成

開発者用ツール > CodePipeline > パイプライン > 新規のパイプラインを作成する

Step 1
パイプラインの設定を選択する

Step 2
ソースステージを追加する

Step 3
ビルドステージを追加する

Step 4
デプロイステージを追加する

Step 5
レビュー

パイプラインの設定を選択する

パイプラインの設定

パイプライン名
パイプライン名を入力します。作成後にパイプライン名を編集することはできません。

myapp-pipeline

100 文字を超えることはできません

サービスロール

新しいサービスロール
アカウントでサービスロールを作成

既存のサービスロール
アカウントから既存のサービスロールを選択

ロール名

myapp-pipeline-service-role

サービスロール名の入力

AWS CodePipeline がサービスロールを作成できるようになるため、この新しいパイプラインでの使用が可能になります。

▶ 高度な設定

キャンセル 次

パイプライン作成

1. CodeBuildのビルドプロジェクト作成のときに指定したリポジトリを選択する

開発者用ツール > CodePipeline > パイプライン > 新規のパイプラインを作成する

Step 1
パイプラインの設定を選択する

Step 2
ソースステージを追加する

Step 3
ビルドステージを追加する

Step 4
デプロイステージを追加する

Step 5
レビュー

ソースステージを追加する

ソース

ソースプロバイダー
ここでは、パイプラインの入カアティファクトを保存します。プロバイダ接続の詳細を指定します。

GitHub

AWS CodePipeline に GitHub リポジトリへのアクセスを許可します。これにより、AWS CodePipeline を使って GitHub からパイプラインへコミットをアップロードできます。

Connected

リポジトリ
jupitris/cakephp3-ecs-demo

ブランチ
master

検出オプションを変更する
検出モードを選択して、ソースコードに変更が発生したときにパイプラインを自動的に開始させます。

GitHub ウェブフック (推奨)
GitHub でウェブフックを使用して、変更が発生したときにパイプラインを自動的に開始

AWS CodePipeline
AWS CodePipeline を使用して、変更を定期的に確認する

キャンセル 戻る 次に

リポジトリ選択

パイプライン作成

1. CodeBuildで作成したビルドプロジェクトを選択する

開発者用ツール > CodePipeline > パイプライン > 新規のパイプラインを作成する

Step 1
パイプラインの設定を選択する

Step 2
ソースステージを追加する

Step 3
ビルドステージを追加する

Step 4
デプロイステージを追加する

Step 5
レビュー

ビルドステージを追加する

構築する - オプション

プロバイダーを構築する
これはビルドプロジェクトのツールです。オペレーティングシステム、ビルドスペックファイル、および出力ファイル名のようなビルドアーティファクトの詳細を提供してください。

AWS CodeBuild ▼

リージョン
米国西部 (オレゴン) ▼

プロジェクト名
AWS CodeBuild コンソールで既に作成したビルドプロジェクトを選択します。または AWS CodeBuild を作成してこのタスクに戻ります。

Q myapp-codebuild X または **ビルドプロジェクト選択** プロジェクトを作成する ↗

環境変数 - オプション
CodeBuild 環境変数のキー、値、タイプを選択します。[value] フィールドで、CodePipeline によって生成された変数を参照できます。詳細 ↗

環境変数の追加

キャンセル 戻る **ビルドステージをスキップ** 次に

パイプライン作成

1. デプロイプロバイダーに「Amazon ECS(ブルー/グリーン)」を選択する
2. CodeDeployのアプリケーション名とデプロイグループは、ECSサービス作成時に自動的に作成されたものがあるので、それを選択する

開発者用ツール > CodePipeline > パイプライン > 新規のパイプラインを作成する

Step 1
パイプラインの設定を選択する

Step 2
ソースステージを追加する

Step 3
ビルドステージを追加する

Step 4
デプロイステージを追加する

Step 5
レビュー

デプロイステージを追加する

デプロイ - オプション

デプロイプロバイダー
インスタンスにデプロイする方法を選択します。プロバイダを選択し、プロバイダの設定の詳細を入力します。

Amazon ECS(ブルー/グリーン)

リージョン
米国西部 (オレゴン)

AWS CodeDeploy アプリケーション名
既存のアプリケーションを一つ選択します。または AWS CodeD

AppECS-myapp-cluster-myapp-wet

アプリケーションを作成する

自動作成のものを選択

AWS CodeDeploy デプロイグループ
既存のデプロイグループを一つ選択します。または AWS CodeD

DgpECS-myapp-cluster-myapp-web

自動作成のものを選択

パイプライン作成

1. ECS タスク定義およびCodeDeploy AppSpecは、ともにリポジトリに入っているファイル名を入力する

Amazon ECS タスク定義
Amazon ECS タスク定義ファイルが保存されている入力アーティファクトを選択します。デフォルトは taskdef.json です。タスク定義ファイルのパスとファイル名を指定します。

BuildArtifact ▼ taskdef.json

デフォルトのパスは taskdef.json です。

AWS CodeDeploy AppSpec ファイル
AWS CodeDeploy AppSpecファイルが保存されている入力アーティファクトを選択します。デフォルトは appspec.yml です。AppSpecファイルのパスとファイル名を指定します。

BuildArtifact ▼ appspec.yml

タスク定義の動的な更新イメージ - オプション
プレースホルダーおよび入力アーティファクトを指定することができ、コンテナ定義に使用するイメージの名前を動的にタスク定義を更新します。複数の入力アーティファクトおよびプレースホルダーを指定できます。

入力アーティファクトを持つイメージの詳細

入力アーティファクトを選択する ▼

タスク定義のプレースホルダー文字

画像

キャンセル 戻る 導入段階をスキップ 次に

パイプライン作成(レビュー)

1. 最終的な設定内容は下図の通り

開発者用ツール > CodePipeline > パイプライン > 新規のパイプラインを作成する

Step 1
パイプラインの設定を選択する

Step 2
ソースステージを追加する

Step 3
ビルドステージを追加する

Step 4
デプロイステージを追加する

Step 5
レビュー

レビュー

ステップ 1: パイプラインの設定を選択する

パイプラインの設定

パイプライン名
myapp-pipeline
Artifact の場所
codepipeline-us-west-2-834179868216
サービスロール名
myapp-pipeline-service-role

ステップ 2: ソースステージを追加する

ソースアクションプロバイダー

パイプライン作成(レビュー)

Source アクションプロバイダー

ThirdParty GitHub
PollForSourceChanges
false
Repo
cakephp3-ecs-demo
Owner
jupitris
Branch
master

ステップ 3: ビルドステージを追加する

アクションプロバイダーを構築する

Build アクションプロバイダー

AWS CodeBuild
ProjectName
myapp-codebuild

パイプライン作成(レビュー)

1. 作成したロールには「AmazonEC2ContainerRegistryFullAccess」を付与する
2. 作成後、パイプラインを実行してみて、正常に完了するか確認しておくが良い
3. ソースコードをリポジトリにpushすると、自動的にパイプラインを実行してデプロイする

ステップ 4: デプロイステージを追加する

アクションプロバイダーをデプロイする

Deploy アクションプロバイダー

AWS CodeDeployToECS

ApplicationName

AppECS-myapp-cluster-myapp-web

DeploymentGroupName

DgpECS-myapp-cluster-myapp-web

TaskDefinitionTemplateArtifact

BuildArtifact

TaskDefinitionTemplatePath

taskdef.json

AppSpecTemplateArtifact

BuildArtifact

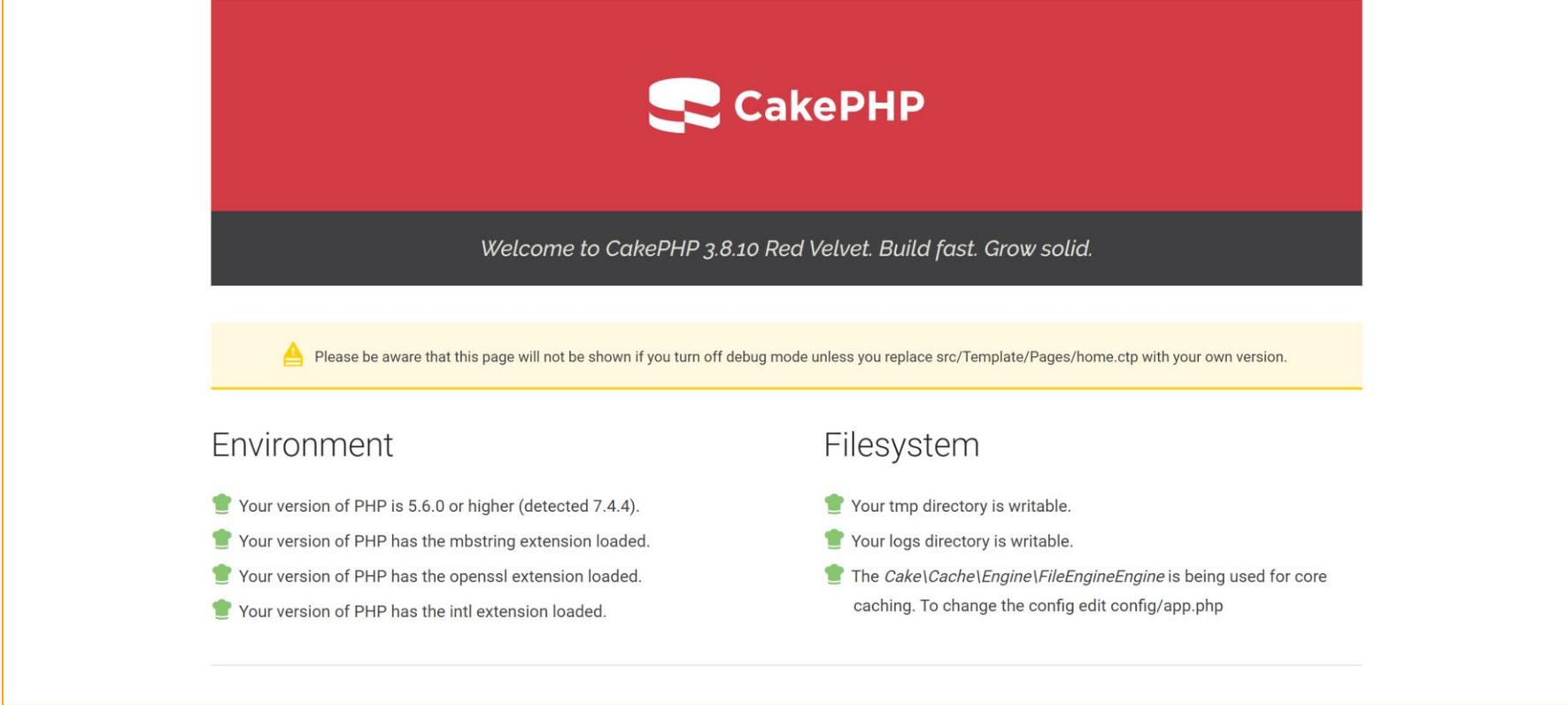
AppSpecTemplatePath

appspec.yml

キャンセル

動作確認

1. 最後に、アプリケーションにアクセスして動作を確認する



The screenshot shows the CakePHP 3.8.10 Red Velvet installation success page. At the top, there is a red banner with the CakePHP logo and the text "CakePHP". Below this is a dark grey banner with the text "Welcome to CakePHP 3.8.10 Red Velvet. Build fast. Grow solid." A yellow warning box contains a warning icon and the text: "Please be aware that this page will not be shown if you turn off debug mode unless you replace src/Template/Pages/home.ctp with your own version." The page is divided into two columns: "Environment" and "Filesystem".

Environment

- ✔ Your version of PHP is 5.6.0 or higher (detected 7.4.4).
- ✔ Your version of PHP has the mbstring extension loaded.
- ✔ Your version of PHP has the openssl extension loaded.
- ✔ Your version of PHP has the intl extension loaded.

Filesystem

- ✔ Your tmp directory is writable.
- ✔ Your logs directory is writable.
- ✔ The `Cake\Cache\Engine\FileEngineEngine` is being used for core caching. To change the config edit `config/app.php`

APPENDIX編

プロジェクト構成

1. プロジェクトの構成は、CakePHP 3.8.10とほぼ同様
 1. 公式ドキュメント内「[クイックスタートガイド](#)」で説明しているcomposerコマンドを利用し、「composer self-update && composer create-project --prefer-dist cakephp/app:^3.8 cakephp3-ecs-demo」でプロジェクトを作成している
 2. 詳細は公式ドキュメント内「[CakePHP のフォルダー構成](#)」を参照
2. ローカルのDockerやECS利用のために、いくつかのファイルやフォルダを追加している

CakePHP 3.8.10に標準で含まれるファイルやフォルダ

```
/cakephp3-ecs-demo
/bin
/config
/logs
/plugins
/src
/tests
/tmp
/vendor
/webroot
.editorconfig
.gitignore
.htaccess
.travis.yml
composer.json
index.php
phpunit.xml.dist
README.md
```

追加したファイルやフォルダ

```
/cakephp3-ecs-demo
/config
  app_local.php
docker-compose.yml
appspec.yml
buildspec.yml
taskdef.json
/docker
  /php-fpm
    Dockerfile
    php.ini
  /nginx
    Dockerfile
    default.conf.production
    default.conf.local
```

appspec.ymlのサンプル

```
version: 0.0
Resources:
  - TargetService:
    Type: AWS::ECS::Service
    Properties:
      TaskDefinition: <TASK_DEFINITION>
      LoadBalancerInfo: # ContainerNameはECRに登録したリポジトリ名を指定する
        ContainerName: "nginx"
        ContainerPort: 80
```

buildspec.ymlのサンプル

```
version: 0.2

phases:
  install:
    runtime-versions:
      docker: 18
    pre_build:
      commands: # ${APP_NAME}や${AWS_DEFAULT_REGION}、${AWS_ACCOUNT_ID}はCodeBuild環境変数
        - aws --version
        - $(aws ecr get-login --no-include-email --region ${AWS_DEFAULT_REGION})
        - APP_ENV=production
        - BUCKET=${APP_NAME}.duxi.co.jp/${APP_ENV}
        - aws s3 cp s3://${BUCKET}/app.php config/app.php
        - IMAGE_NAME_PHP_FPM=php-fpm
        - IMAGE_NAME_NGINX=nginx
        -
      REPOSITORY_URI_PHP_FPM=${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_DEFAULT_REGION}.amazonaws.com/${IMAGE_NAME_PHP_FPM}
        -
      REPOSITORY_URI_NGINX=${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_DEFAULT_REGION}.amazonaws.com/${IMAGE_NAME_NGINX}
        - COMMIT_HASH=$(echo ${CODEBUILD_RESOLVED_SOURCE_VERSION} | cut -c 1-7)
        - IMAGE_TAG=${COMMIT_HASH:=latest}
    build:
      commands: # --build-argで渡した値はDockerfile内で利用する
        - echo Build started on `date`
        - echo Building the Docker image...
```

```
- docker build -t ${REPOSITORY_URI_PHP_FPM}:latest -f docker/php-fpm/Dockerfile .
- docker build --build-arg APP_ENV=${APP_ENV} -t
  ${REPOSITORY_URI_NGINX}:latest -f docker/nginx/Dockerfile .
- docker tag ${REPOSITORY_URI_PHP_FPM}:latest
  ${REPOSITORY_URI_PHP_FPM}:${IMAGE_TAG}
- docker tag ${REPOSITORY_URI_NGINX}:latest
  ${REPOSITORY_URI_NGINX}:${IMAGE_TAG}
post_build:
  commands:
    - echo Build completed on `date`
    - echo Pushing the Docker images...
    - docker push ${REPOSITORY_URI_PHP_FPM}:latest
    - docker push ${REPOSITORY_URI_PHP_FPM}:${IMAGE_TAG}
    - docker push ${REPOSITORY_URI_NGINX}:latest
    - docker push ${REPOSITORY_URI_NGINX}:${IMAGE_TAG}
    - echo Writing image definitions file...
    - cat taskdef.json | sed -e
      s@¥<IMAGE_NAME_PHP_FPM¥>@${REPOSITORY_URI_PHP_FPM}:${IMAGE_TAG}@ -e
      s@¥<IMAGE_NAME_NGINX¥>@${REPOSITORY_URI_NGINX}:${IMAGE_TAG}@ -e
      s@¥<AWS_ACCOUNT_ID¥>@${AWS_ACCOUNT_ID}@ -e
      s@¥<APP_NAME¥>@${APP_NAME}@ -e
      s@¥<APP_REGION¥>@${AWS_DEFAULT_REGION}@ > taskdef.json
    - cat appspec.yml | sed -e s@¥<APP_NAME¥>@${APP_NAME}@ > appspec.yml

artifacts:
  files:
    - appspec.yml
    - taskdef.json
```

taskdef.jsonのサンプル

```
{
  "executionRoleArn": "arn:aws:iam::<AWS_ACCOUNT_ID>:role/ecsTaskExecutionRole",
  "containerDefinitions": [
    {
      "name": "php-fpm",
      "image": "<IMAGE_NAME_PHP_FPM>",
      "logConfiguration": {
        "logDriver": "awslogs",
        "secretOptions": null,
        "options": {
          "awslogs-group": "/ecs/<APP_NAME>-web",
          "awslogs-region": "<APP_REGION>",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "portMappings": [
        {
          "containerPort": 9000,
          "hostPort": 9000,
          "protocol": "tcp"
        }
      ],
      "essential": true
    }
  ],
  "essential": true
},
```

```
{
  "name": "nginx",
  "image": "<IMAGE_NAME_NGINX>",
  "logConfiguration": {
    "logDriver": "awslogs",
    "secretOptions": null,
    "options": {
      "awslogs-group": "/ecs/<APP_NAME>-web",
      "awslogs-region": "<APP_REGION>",
      "awslogs-stream-prefix": "ecs"
    }
  },
  "portMappings": [
    {
      "containerPort": 80,
      "hostPort": 80,
      "protocol": "tcp"
    }
  ],
  "essential": true
},
"requiresCompatibilities": ["FARGATE"],
"networkMode": "awsvpc",
"cpu": "256",
"memory": "512",
"family": "<APP_NAME>-web"
}
```

サンプルコード

1. サンプルコードは「<https://github.com/jupitris/cakephp3-ecs-demo>」からクローンできる
2. サンプルコードのリポジトリにconfig/app.phpは含まれていないので、別途自身で作成してS3にアップロードしておく
 1. 本資料でのアップロード先は「s3://myapp.duxi.co.jp/production」を想定しているため、適当な設定値へ変更する

サンプルコードをローカル環境で利用する

1. DockerとDocker Composeが必要
2. クローンしたリポジトリに移動し、「docker-compose up -d --build」を実行
3. ビルドが完了したら、「<http://localhost:40080>」にアクセスし、CakePHPのページが表示できれば完成