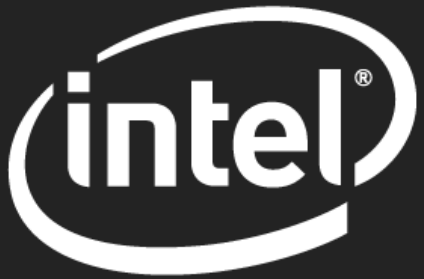

aws
DEVDAY
GLOBAL SERIES

THANKS TO OUR FRIENDS AT:



C#開発者必見、Dockerコンテナへの継続的デプロイメント on AWS

～CodeCommit, CodeBuild, CodePipeline, CloudFormation, ECR, ECS を活用した CI/CD ～

アマゾン ウェブ サービス ジャパン株式会社

ソリューション アーキテクト　福井 厚

2017/05/31

amazon web services

本セッションのFeedbackをお願いします

受付でお配りしたアンケートに本セッションの満足度やご感想などをご記入ください
アンケートをご提出いただきました方には、もれなく素敵なAWSオリジナルグッズをプレゼントさせていただきます

アンケートは各会場出口、パミール3FのEXPO展示会場内にて回収させて頂きます

自己紹介

❖名前

- ❖ 福井 厚（ふくい あつし）fatsushi@

❖所属

- ❖ アマゾン ウェブ サービス ジャパン株式会社
- ❖ 技術統括本部エンタープライズ ソリューション部
- ❖ ソリューション アーキテクト

❖前職

- ❖ エンタープライズ アプリケーション開発コンサルタント

❖好きなAWSサービス

- ❖ AWS Code シリーズ、AWS IoT、AWS Lambda（C#）

アジェンダ

- エンタープライズを取り巻く環境
- なぜ継続的デプロイメントなのか？
- C#と.NET Core
- Dockerコンテナの活用
- コンテナと共にCI/CDを実現するAWSサービス
- ASP.NET Core アプリの継続的デプロイメント on AWS
- まとめ


エンタープライズを取り巻く環境

15年

大型株銘柄の企業の平均寿命
1920年代の67年から現在は15年に低下してきている

2/3

2/3以上の**IT予算**は現在の運用を維持することに使われている

77%

のCEOはここ数年でセキュリティリスクが増加しており、65%のCEOはリスク管理の能力が遅れを取っていると感じている

これらがどのように影響するか

必要なリソースもなく競合優位性を維持するためのクリティカルなビジネス イニシアチブを追求ことが要求される

伝統的なITモデルは革新的なスタートアップと同じペースを維持するために必要な**アジリティに欠けている**

不十分なセキュリティ、コンプライアンス、可用性は不正な攻撃に対する洗練された対策を実施する能力を妨げる

要求に対応する新しいモデル

企業としての差別化にフォーカス

スタートアップのようなスピードで革新する

リスクを低減する

差別化にフォーカスする

かつ技術的負債を低減する

コア ミッションにフォーカスする

インフラストラクチャに
割く時間を低下させる

新しいビジネスの主導
権獲得に集中する

革新に対してより多くのリ
ソースを投入する

かつてないほど速く革新する

ソフトウェア開発のアジリティーを高める

ソフトウェアの動きは加速している

ソフトウェアの作成と配布はかつてないほど簡単で高速になっている：

- ほとんどあるいはまったく資金調達せずに中小企業が巨大企業に対抗できる
- ダウンロードひとつで数百万人のユーザーにすぐにソフトウェアを配布できる
- 混乱を抑制するには機敏性が最も重要

ソフトウェア配布モデルは大きく様変わりしている

かつてのソフトウェア配布モデル

新しいソフトウェア配布モデル

エンタープライズ アプリケーションでも**頻繁なアップデート**が求められる

インフラストラクチャのリスクを削減

Move
Fast

OR

Stay
Secure

Move
Fast

AND

Stay
Secure


なぜ継続的デプロイメントなのか？

ビジネス アプリケーションもサービス指向に

- 複雑でモノリシックなシステムに対するメンテナンスの限界
- 機能要求に対するリリース期間の短縮が命題
- 業務システムへのシングル ページ アプリケーション（SPA）の導入
  - Web APIによるサービス化
  - マイクロサービス志向
- サービス単位に並列でチーム開発

アジャイルな開発に必要なツールとは？

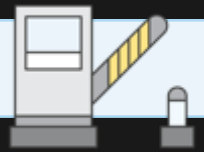
- この新しいソフトウェア駆動の世界でソフトウェアをリリースするのに必要なツール
  - ソフトウェア開発のリリース プロセスの流れを管理するツール
  - コードの不具合や潜在的な問題をテスト／検査するツール
  - アプリケーションをデプロイするツール

リリース プロセスのレベル

ソース → ビルド → テスト → 運用

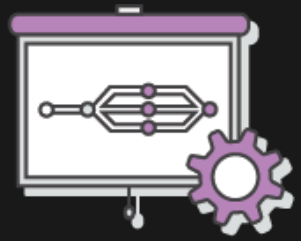
継続的インテグレーション

継続的デリバリ

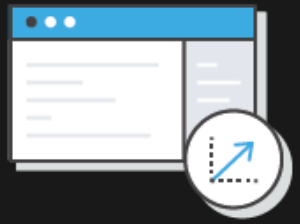


継続的デプロイメント

継続的デプロイメントのメリット



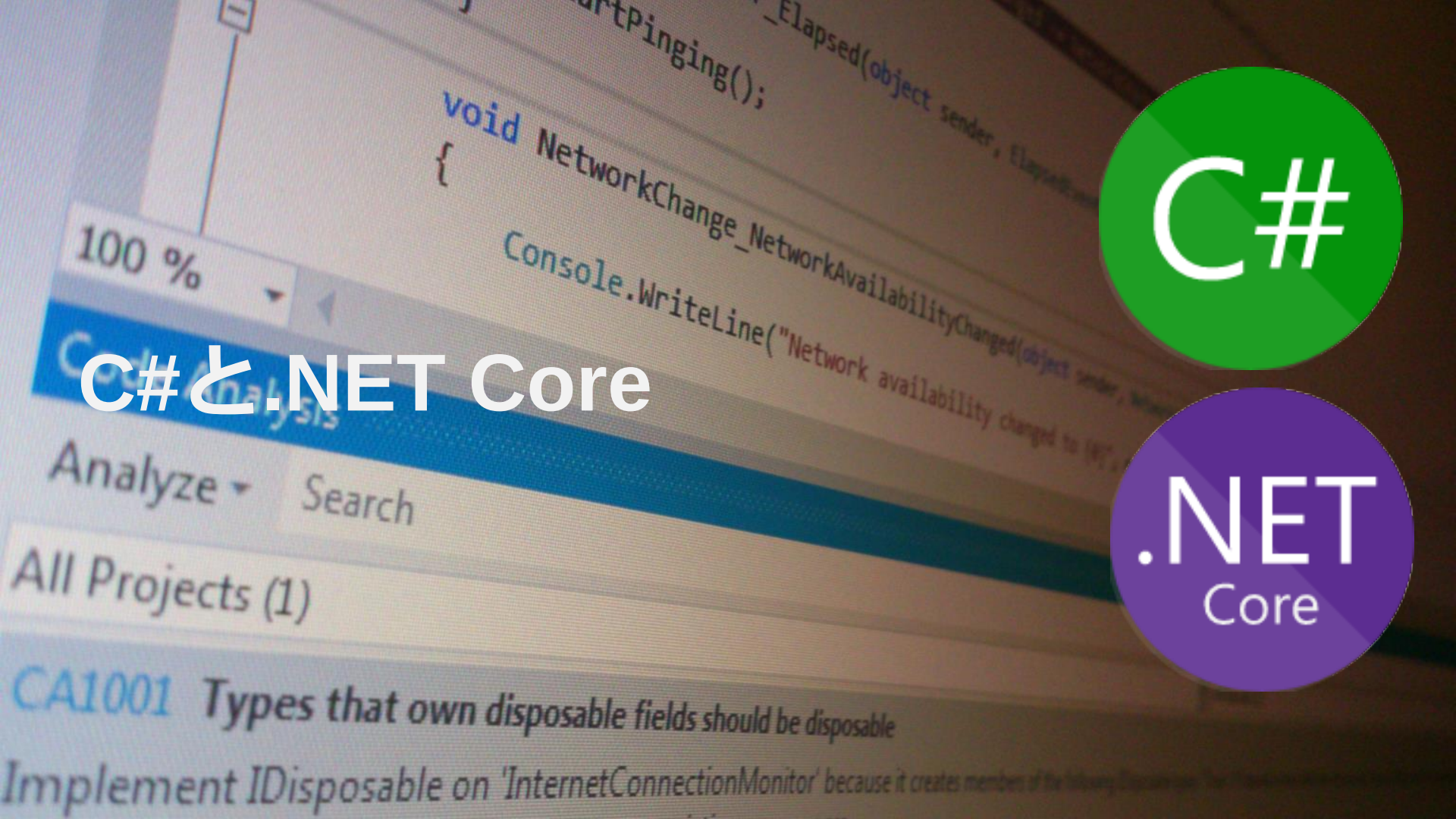
ソフトウェアの
リリースプロセスを
自動化



開発者の
生産性を改善

バグをすばやく
検出して対処

アップデートの
配信を高速化


C#と.NET Core
C#
.NET
Core
void NetworkChange_NetworkAvailabilityChanged(
{
Console.WriteLine("Network availability
100 %
Analyze
Search
All Projects (1)
CA1001 Types that own disposable fields should be disposable
Implement IDisposable on 'InternetConnectionMonitor'

C#

（ウィキペディアより）

- マルチパラダイム プログラミング言語
- 強い型付け、命令型、宣言型、手続き型、関数型、ジェネリック、オブジェクト指向の要素を持つ
- .NET Frameworkとともに作られ、Ecma Internationalおよび国際標準化機構 (ISO)によって標準化
- 日本においても日本工業規格 (JIS)によって採択

- 多くの魅力的な言語仕様
- .NETの豊富なクラス ライブラリ
- エンタープライズ系アプリケーションでの利用
  - 特にWindows開発者の多くが利用

C# - 多くの魅力的な言語仕様

- プロパティ、デリゲート、属性
- Generics、イテレータ、パーシャル型、Nullable型
- 暗黙的型付け、拡張メソッド、ラムダ式、初期化子、匿名型、暗黙的型付け配列、LINQ、自動プロパティ、パーシャル メソッド
- 動的型付け変数、オプション引数、名前付き引数
- 非同期処理
- Null条件演算子、文字列挿入、nameof演算子、インデックス初期化子
- タプル、型スイッチ

.NET Core

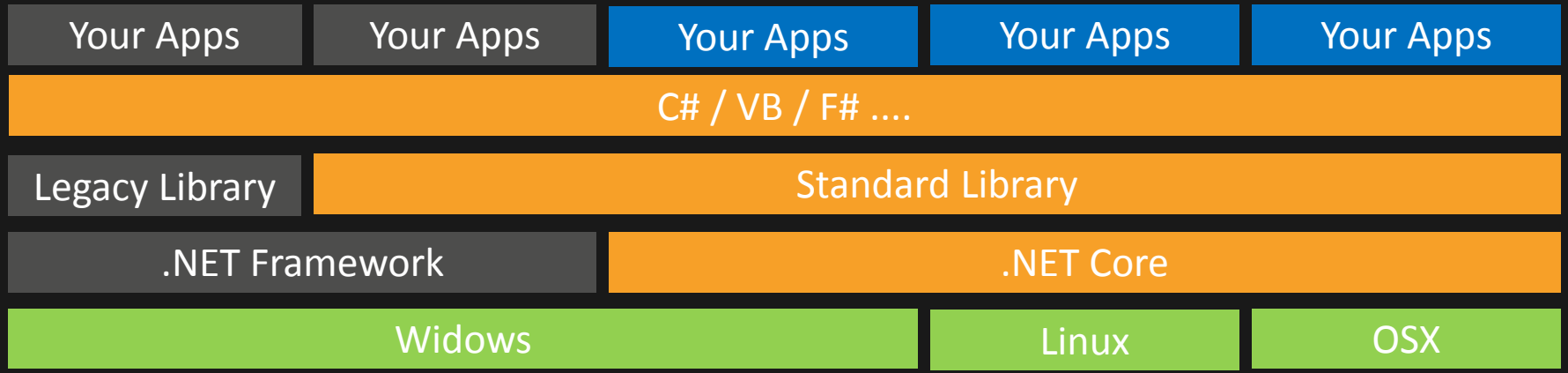
- クロスプラットフォーム
  - Windows、Linux、Macで実行可能
  - Linux上のDocker コンテナで動作可能
- 統合標準ライブラリ
- 高速
- 軽量
  - Docker イメージのビルドも高速
- モダン
- オープンソース

.NET Coreの便利なライブラリ

- .NET Standard Library
- ASP.NET Core MVC
- ASP.NET Core Web API
- Entity Framework
- …More

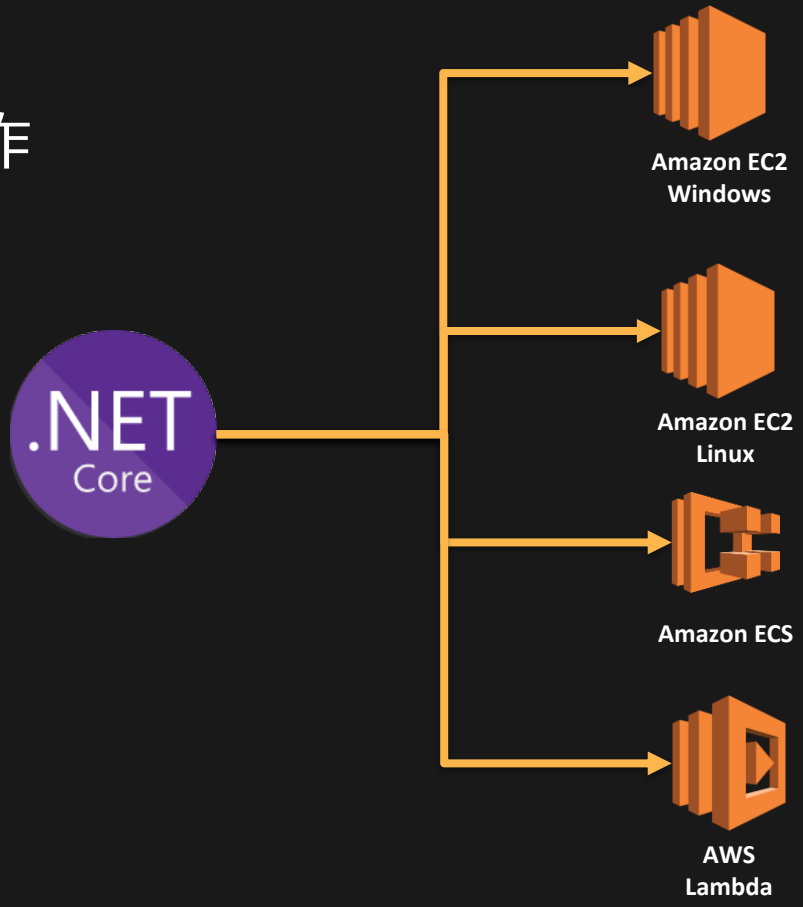
C#と.NET Core とプラットフォームの関係

.NET Core 上のアプリはマルチプラットフォームで動作



AWS と .NET Core

- 様々なプラットフォームで動作
  - Amazon EC2 Windows
  - Amazon EC2 Linux
  - Amazon ECS
  - AWS Lambda



Dockerコンテナの活用

なぜDockerコンテナなのか？

- 詳細は去年のAWS Developer Conferenceの下記のセッションをご参照ください。

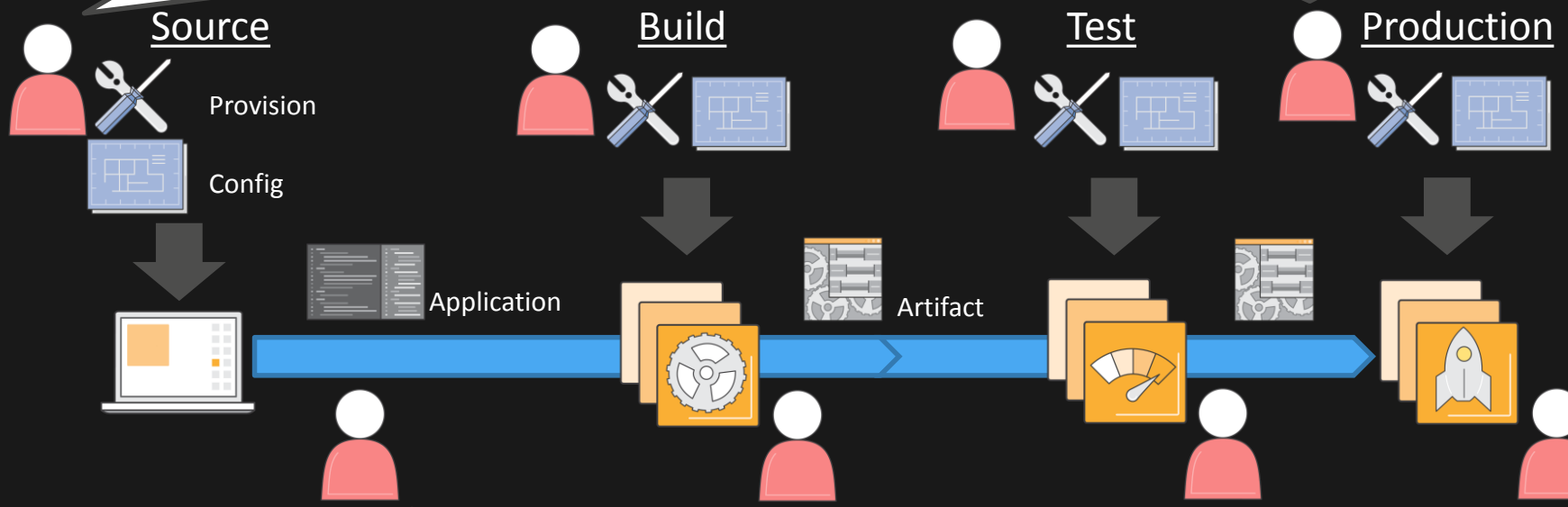
  「Docker と Amazon ECS で DevOps を進化させる」

  動画：
  https://www.youtube.com/watch?v=3oC98Vt-uy0
- **スライド：**
  http://media.amazonwebservices.com/jp/summit2016/3Dev-T04.pdf

デプロイメントの課題

開発環境の構成の
メンテナンスが必要

開発、テスト、本番で
環境に差異がある。。。

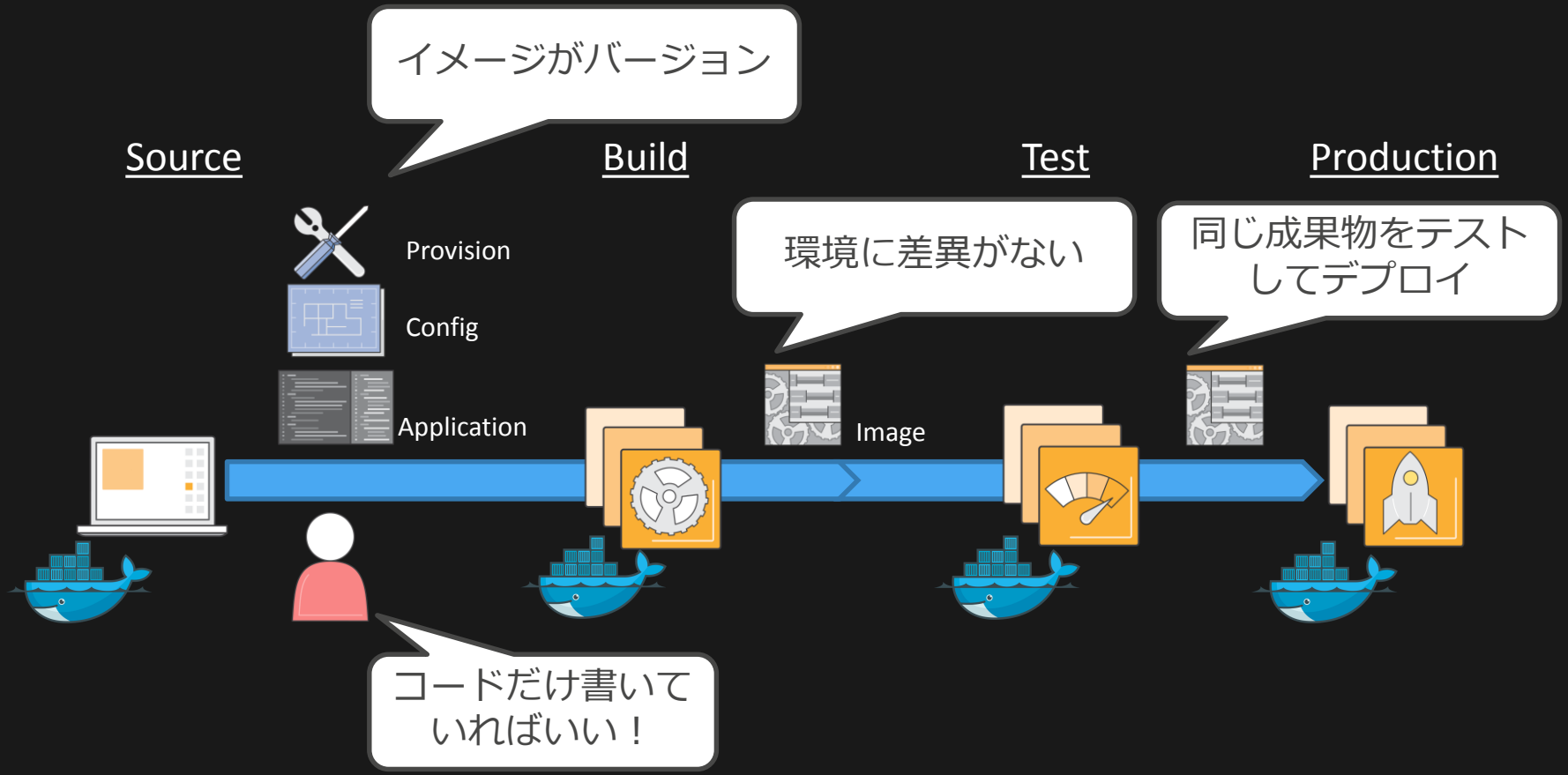


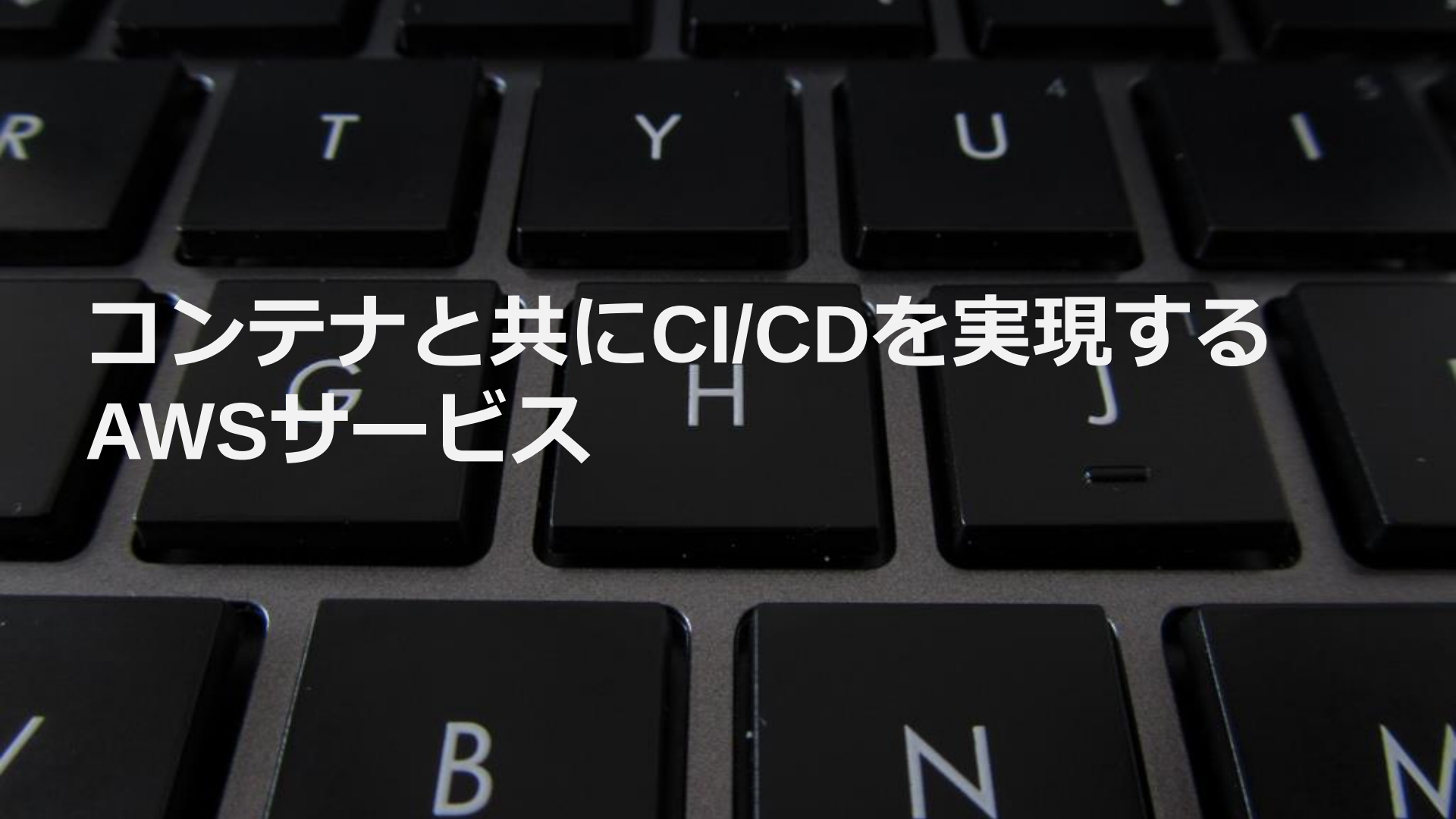
なるほど、
全てが必要なんですね。。。

テストの需要がバラバラ
で管理が大変。。。。

オートスケールや
ノード障害対応。。。

Dockerを取り入れたデプロイメント





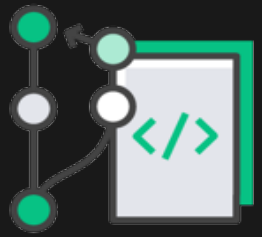
コンテナと共にCI/CDを実現するAWSサービス

AWS Code シリーズ

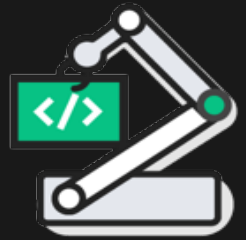
AWS CodeStar

AWS CodePipeline

AWS CodeDeploy



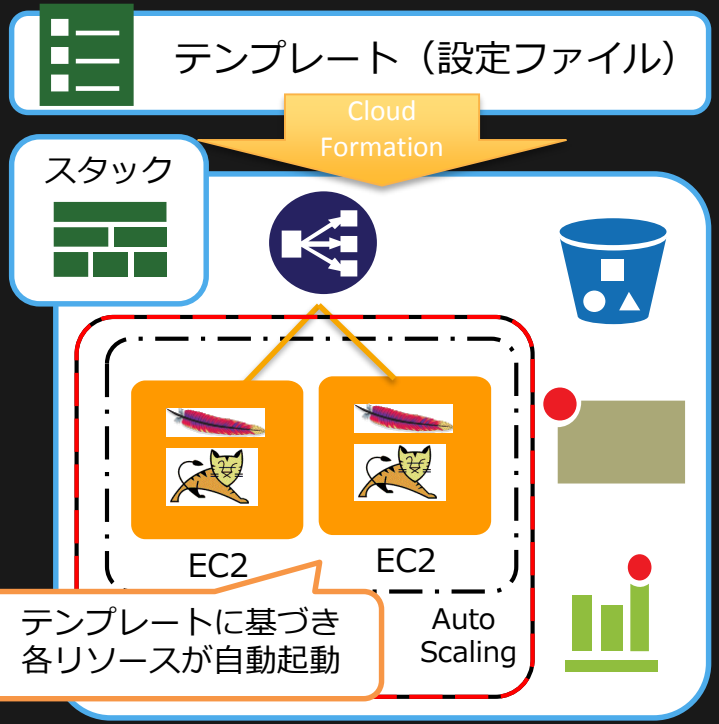
AWS CodeCommit



AWS CodeBuild

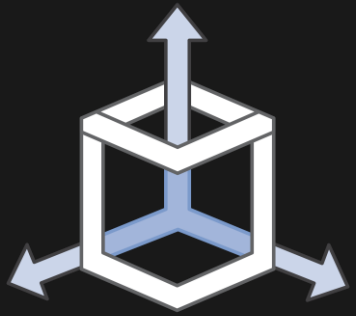
Amazon CloudFormation

設定管理 & クラウドのオーケストレーション サービス

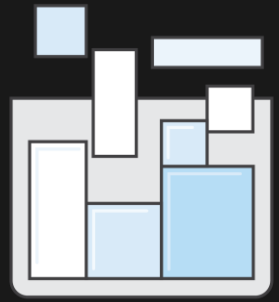


- 環境を自動構築
  - テンプレートを元に、EC2やELBといったAWSリソースの環境構築を自動化
  - JSONまたはYAMLのテキストでテンプレートを自由に記述可能
- 豊富なリファレンス
  - Microsoft Windows Server や SAP HANA などのリファレンス実装を用意

Amazon EC2 Container Service



コンテナ管理を
あらゆるスケールで



柔軟なコンテナの配置

AWSの基盤との連携

Amazon EC2 Container Service

- **フルマネージドで使えるDockerレジストリサービス**

完全マネージド型

安全性

高い可用性

シンプルなワークフロー

ASP.NET Core アプリの継続的デプロイメント on AWS

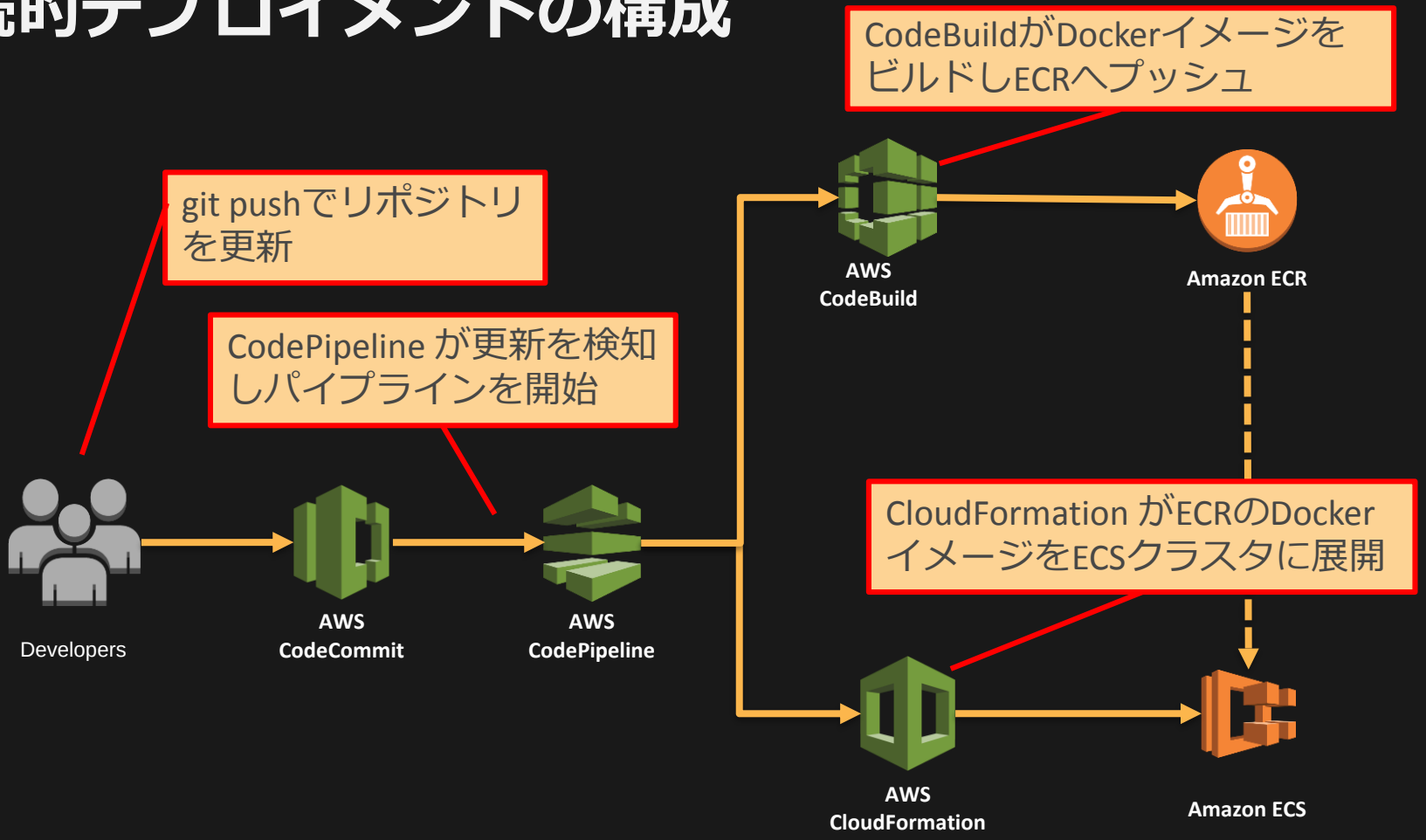
アジェンダ

- ECS Reference Architecture: Continuous Deployment
- 継続的デプロイメントの構成
- パイプラインの設定
- Demo: 継続的デプロイメント
- まとめ

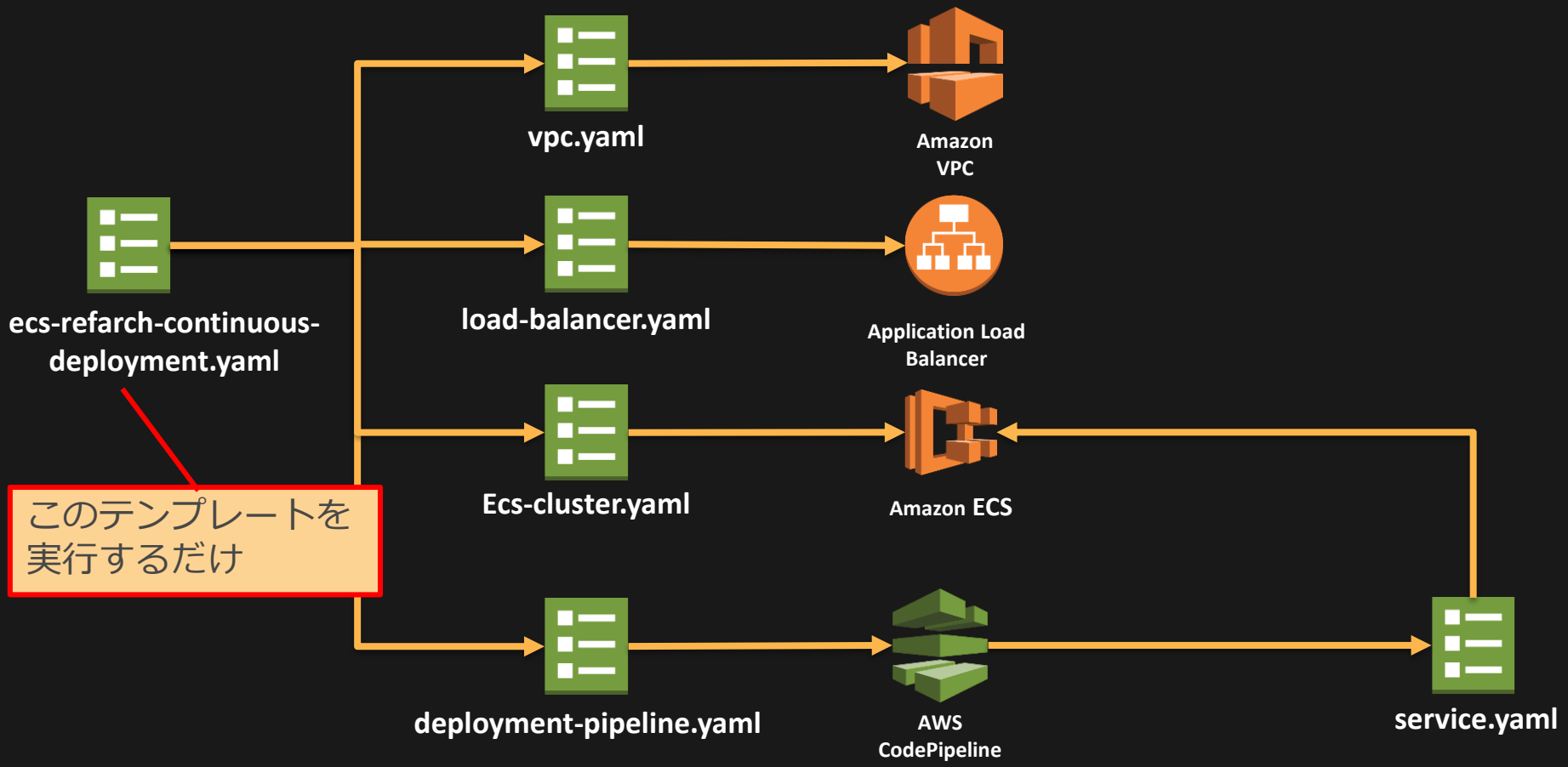
ECS Reference Architecture: Continuous Deployment

- 「 AWS CodePipeline, AWS CodeBuild, Amazon ECR, AWS CloudFormationを利用したAmazon ECSへの継続的デプロイメント」を参照
  https://aws.amazon.com/jp/blogs/news/continuous-deployment-to-amazon-ecs-using-aws-codepipeline-aws-codebuild-amazon-ecr-and-aws-cloudformation/
- 上記の記事をベースにASP.NET Coreアプリに対して継続的デプロイメント環境を構築

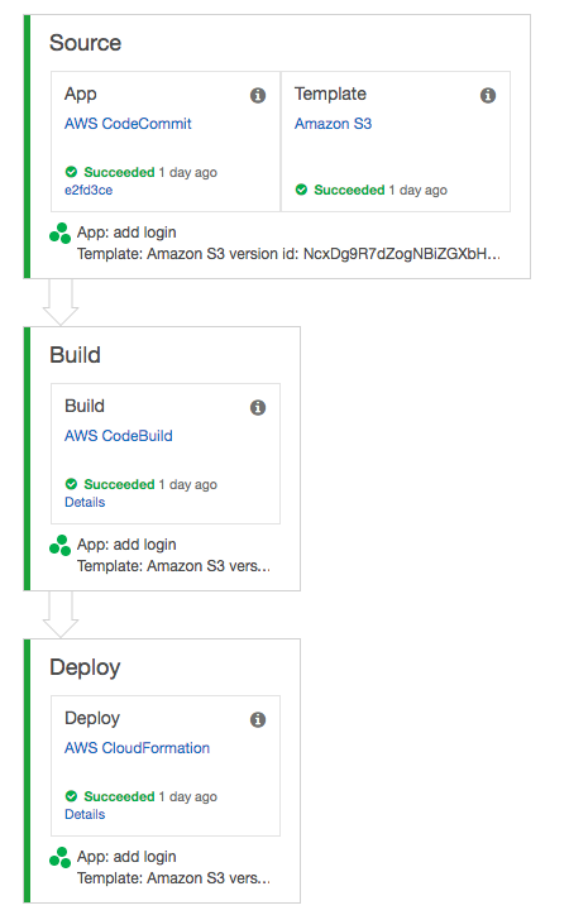
継続的デプロイメントの構成



環境はすべてCloudFormationで構築



CodePipeline パイプライン



■ソース

- AWS CodeCommit をリポジトリとして利用し、git push の実行でソースの更新を検知しパイプラインの実行を開始
- Amazon ECSのサービスを作成、更新する AWS CloudFormation テンプレートをS3 バケットに保存

■ビルド

- AWS CodeBuild を利用し ASP.NET Core の Docker イメージを作成、Amazon ECR に登録

■デプロイ

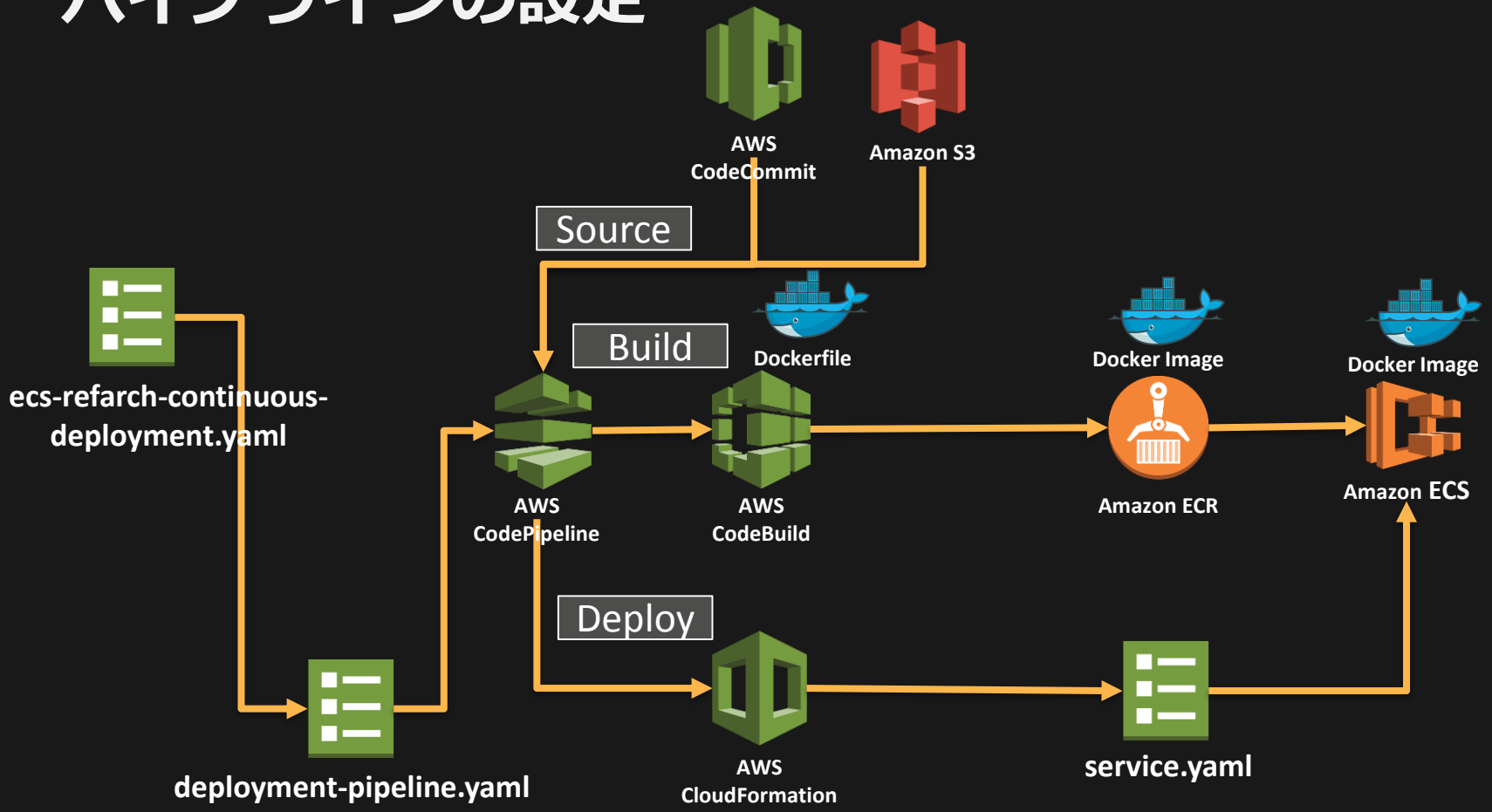
- AWS CloudFormation を利用し Amazon ECS のサービスを作成、更新

Tips: CodeCommit のHTTPS接続と認証の手順

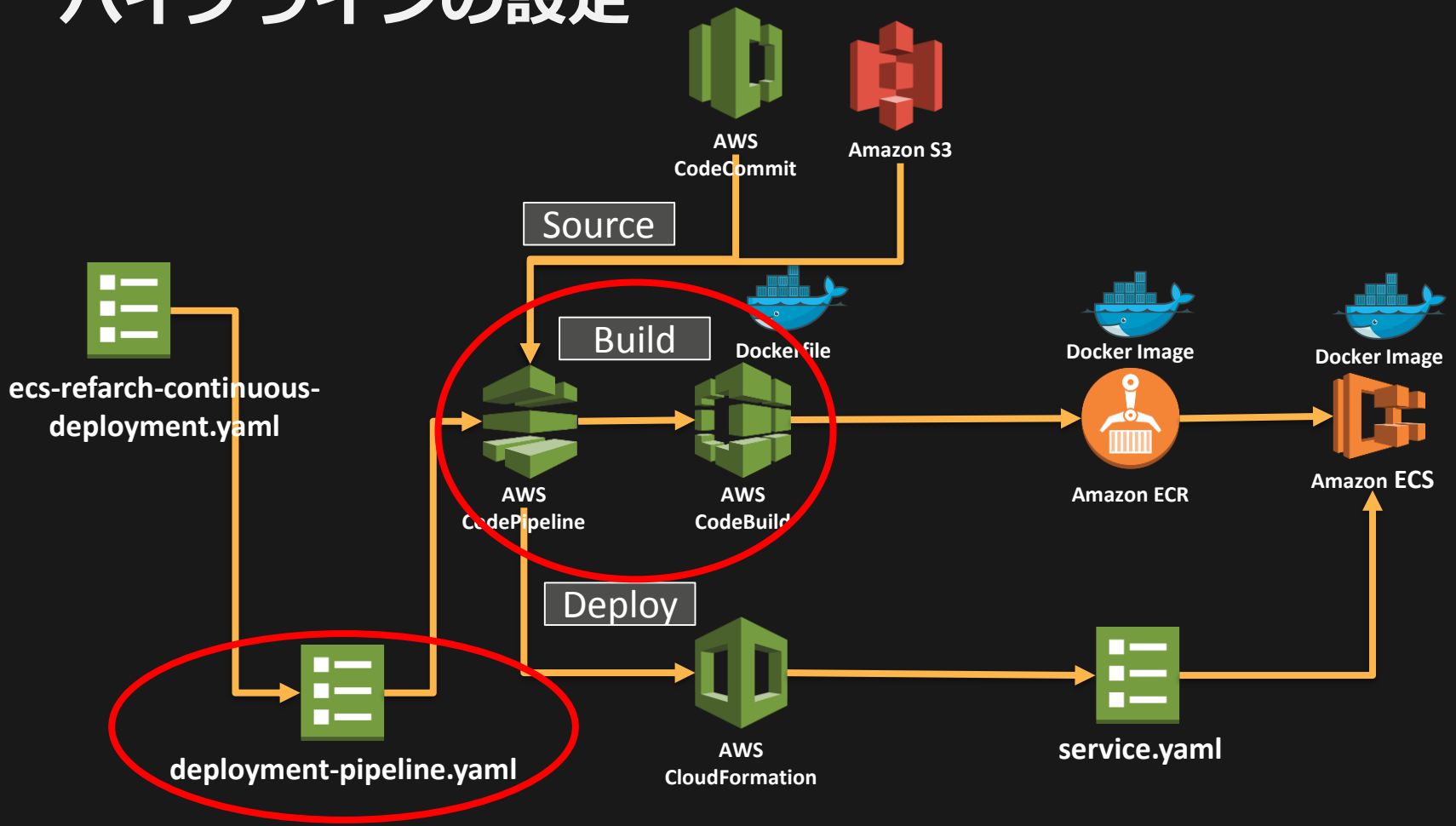
1. AWS CodeCommit にアクセスするIAM Userを作成
2. IAM UserにCodeCommit用のユーザー名とパスワードを生成
3. 生成した認証情報を Git のHTTPS接続時にユーザー名、パスワード認証で利用
4. IDEからの接続も同様

❖AWS CodeCommit はGit version 1.7.9以上をサポート

パイプラインの設定


AWS
CodeCommit
Amazon S3
Source
ecs-refarch-continuous-
deployment.yaml
Build
Dockerfile
Docker Image
Docker Image
AWS
CodePipeline
AWS
CodeBuild
Amazon ECR
Amazon ECS
Deploy
deployment-pipeline.yaml
AWS
CloudFormation
service.yaml

パイプラインの設定


AWS
CodeCommit
Amazon S3
Source
ecs-refarch-continuous-
deployment.yaml
Build
Docker File
Docker Image
Docker Image
AWS
CodePipeline
AWS
CodeBuild
Amazon ECR
Amazon ECS
Deploy
deployment-pipeline.yaml
AWS
CloudFormation
service.yaml

CodePipeline Buildの設定

```
CodeBuildProject:
  Type: AWS::CodeBuild::Project
  Properties:
    Artifacts:
      Location: !Ref ArtifactBucket
      Type: "S3"
    Source:
      Location: !Sub ${ArtifactBucket}/source.zip
      Type: "S3"
      BuildSpec: |
        version: 0.1
        phases:
          pre_build:
          commands:
            - echo -n "$CODEBUILD_BUILD_ID" | sed "s/.*:¥([[:xdigit:]]¥{7¥}¥).*/¥1/" > /tmp/build_id.out
            - printf "%s:%s" "$REPOSITORY_URI" "$(cat /tmp/build_id.out)" > /tmp/build_tag.out
            - printf '{"tag":"%s"}' "$(cat /tmp/build_id.out)" > /tmp/build.json
            - $(aws ecr get-login)
```

SourceステージでCodeCommitリポジトリから取得したS3バケット上のZipを指定

CodeBuild環境で実行するコマンドを指定
CodeBuildがセットする$CODEBUILD_BUILD_ID環境変数でビルドIDを取得ここではビルドIDから7桁の16進数を抜き出してタグとして利用
$ REPOSITORY_URIにはECRのURL
aws ecr get-login の戻り値を実行することでECRへのアクセスが可能

... 続く

CodePipeline Buildの設定

... 続き

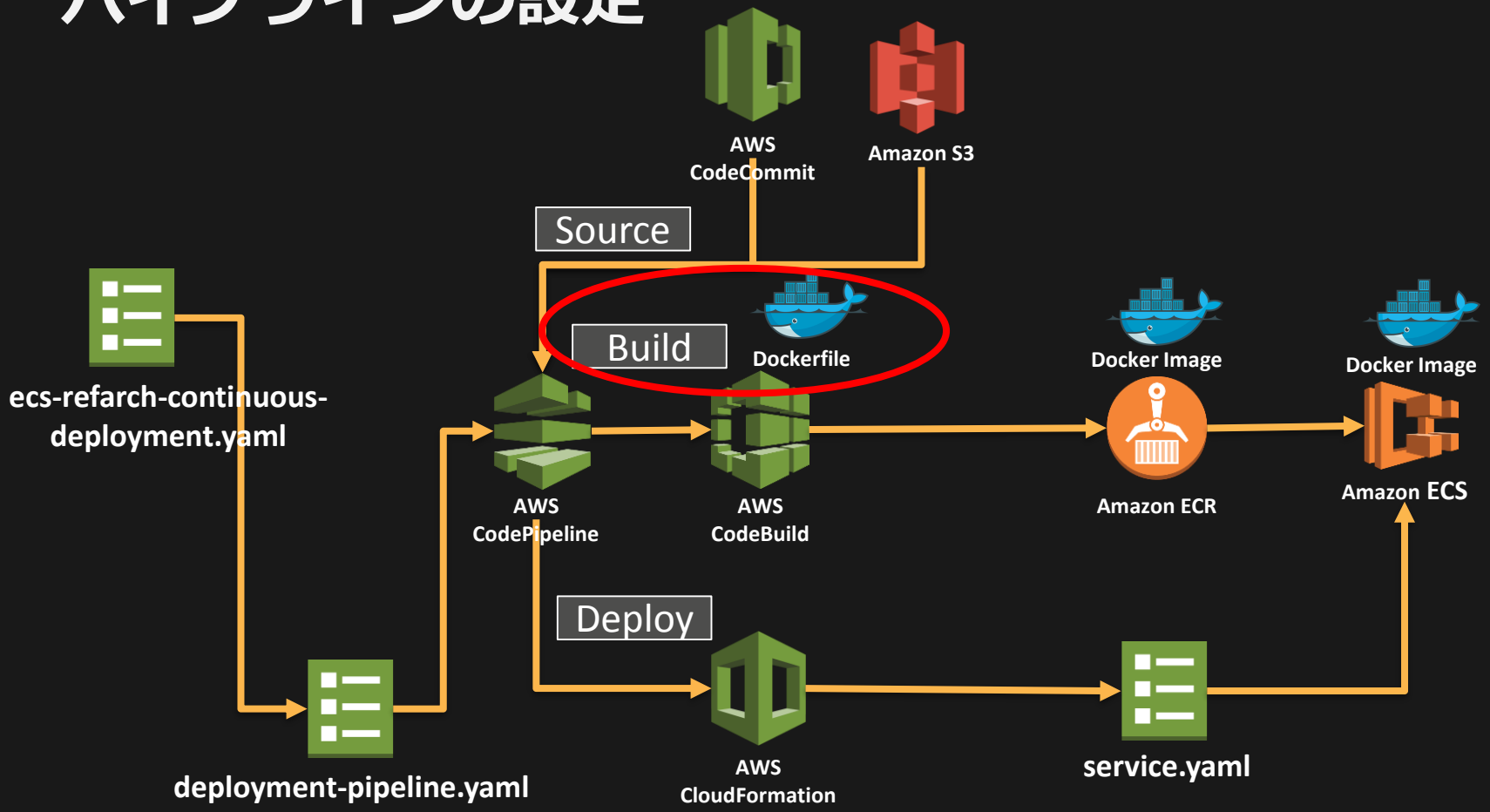
```
      build:
        commands:
          - docker build --tag "$(cat /tmp/build_tag.out)" .
      post_build:
        commands:
          - docker push "$(cat /tmp/build_tag.out)"
    artifacts:
      files: /tmp/build.json
      discard-paths: yes
  Environment:
    ComputeType: "BUILD_GENERAL1_SMALL"
    Image: "aws/codebuild/docker:1.12.1"
    Type: "LINUX_CONTAINER"
    EnvironmentVariables:
      - Name: AWS_DEFAULT_REGION
        Value: !Ref AWS::Region
      - Name: REPOSITORY_URI
        Value: !Sub ${AWS::AccountId}.dkr.ecr.${AWS::Region}.amazonaws.com/${Repository}
  Name: !Ref AWS::StackName
    ServiceRole: !Ref CodeBuildServiceRole
```

CodeBuildのbuildフェーズでdocker buildを実行

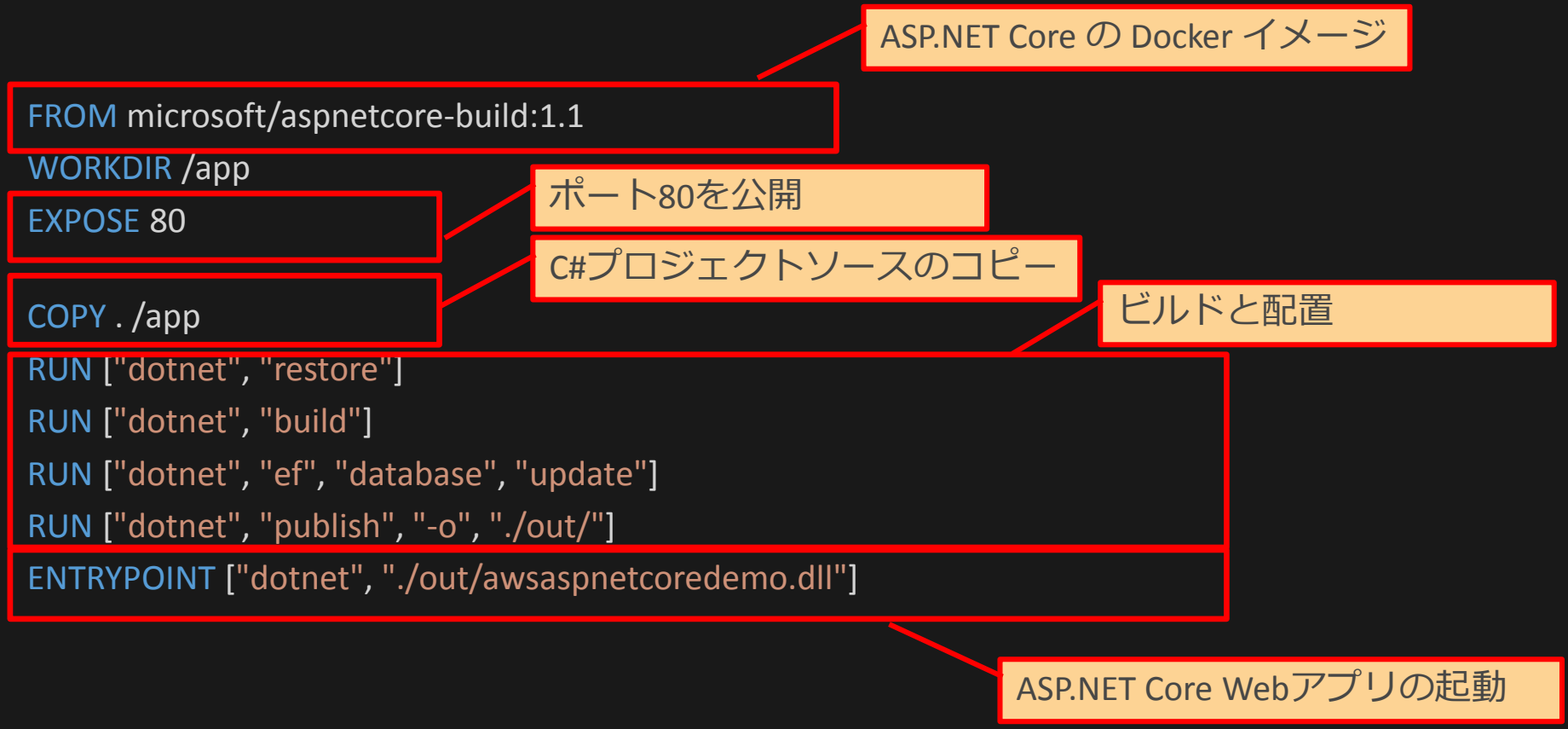
CodeBuildのpost_buildフェーズでECRにpush

CodeBuildへ渡す環境変数を指定
$REPOSITORY_URIにはECRのURLを指定

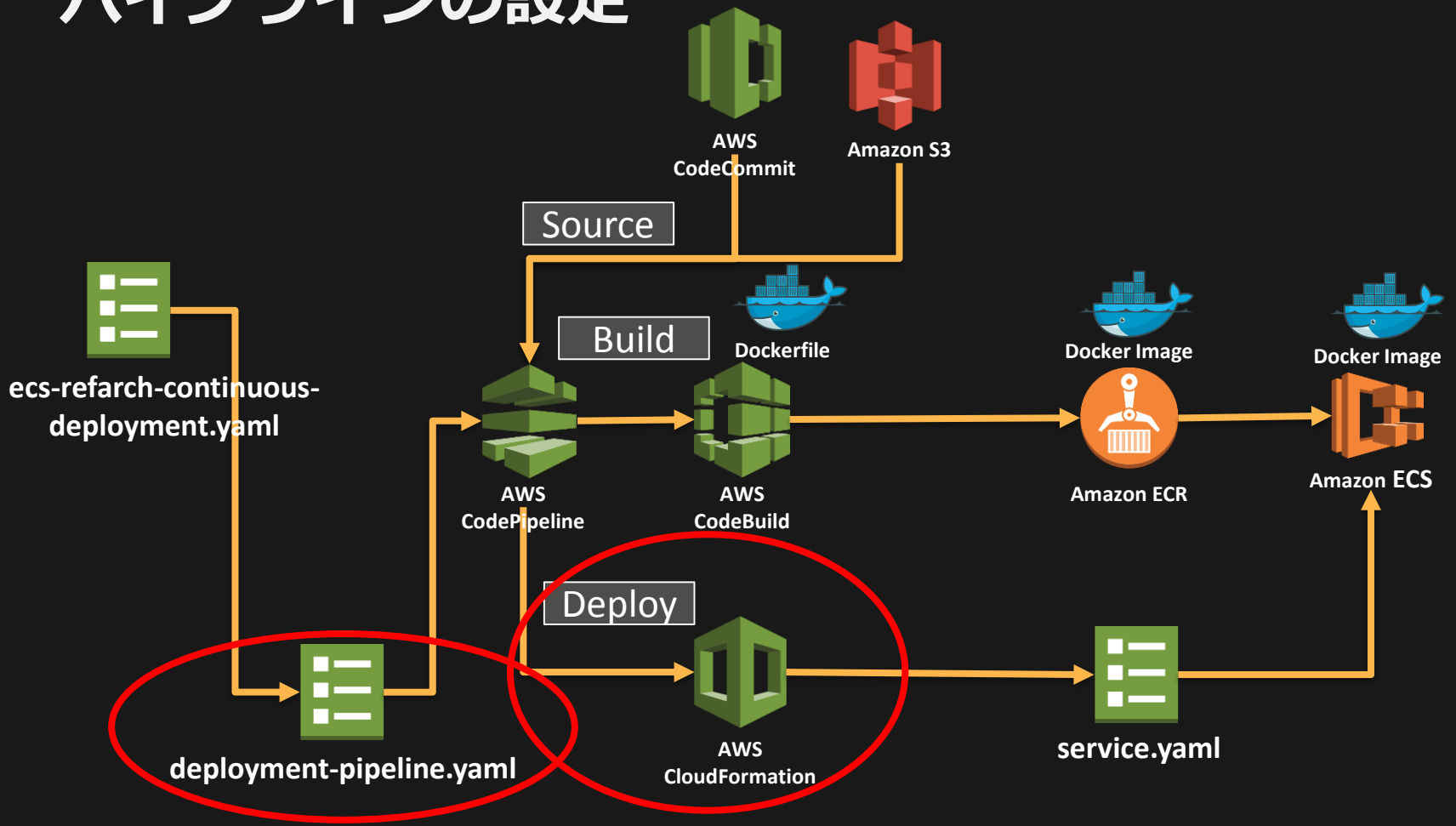
パイプラインの設定


AWS
CodeCommit
Amazon S3
Source
ecs-refarch-continuous-
deployment.yaml
Build
Dockerfile
Docker Image
Docker Image
AWS
CodePipeline
AWS
CodeBuild
Amazon ECR
Amazon ECS
Deploy
deployment-pipeline.yaml
AWS
CloudFormation
service.yaml

DockerFile


ASP.NET Core の Docker イメージ
FROM microsoft/aspnetcore-build:1.1
WORKDIR /app
EXPOSE 80
ポート80を公開
COPY . /app
C#プロジェクトソースのコピー
ビルドと配置
RUN ["dotnet", "restore"]
RUN ["dotnet", "build"]
RUN ["dotnet", "ef", "database", "update"]
RUN ["dotnet", "publish", "-o", "./out/"]
ENTRYPOINT ["dotnet", "./out/awsaspnetcoredemo.dll"]
ASP.NET Core Webアプリの起動

パイプラインの設定


AWS
CodeCommit
Amazon S3
Source
ecs-refarch-continuous-
deployment.yaml
Build
Dockerfile
Docker Image
Docker Image
AWS
CodePipeline
AWS
CodeBuild
Amazon ECR
Amazon ECS
Deploy
deployment-pipeline.yaml
AWS
CloudFormation
service.yaml

CodePipeline Deployの設定

```
- Name: Deploy
    Actions:
        - Name: Deploy
            ActionTypeId:
                Category: Deploy
                Owner: AWS
                Version: 1
            Provider: CloudFormation
            Configuration:
                ChangeSetName: Deploy
                ActionMode: CREATE_UPDATE
                StackName: !Sub "${AWS::StackName}-Service"
                Capabilities: CAPABILITY_NAMED_IAM
                TemplatePath: Template::templates/service.yaml
                RoleArn: !GetAtt CloudFormationExecutionRole.Arn
```

...続く

DeployステージではCloudFormationを利用

CloudFormationスタックの作成または更新を指定

CloudFormationスタックテンプレートの指定

CodePipeline Deployの設定

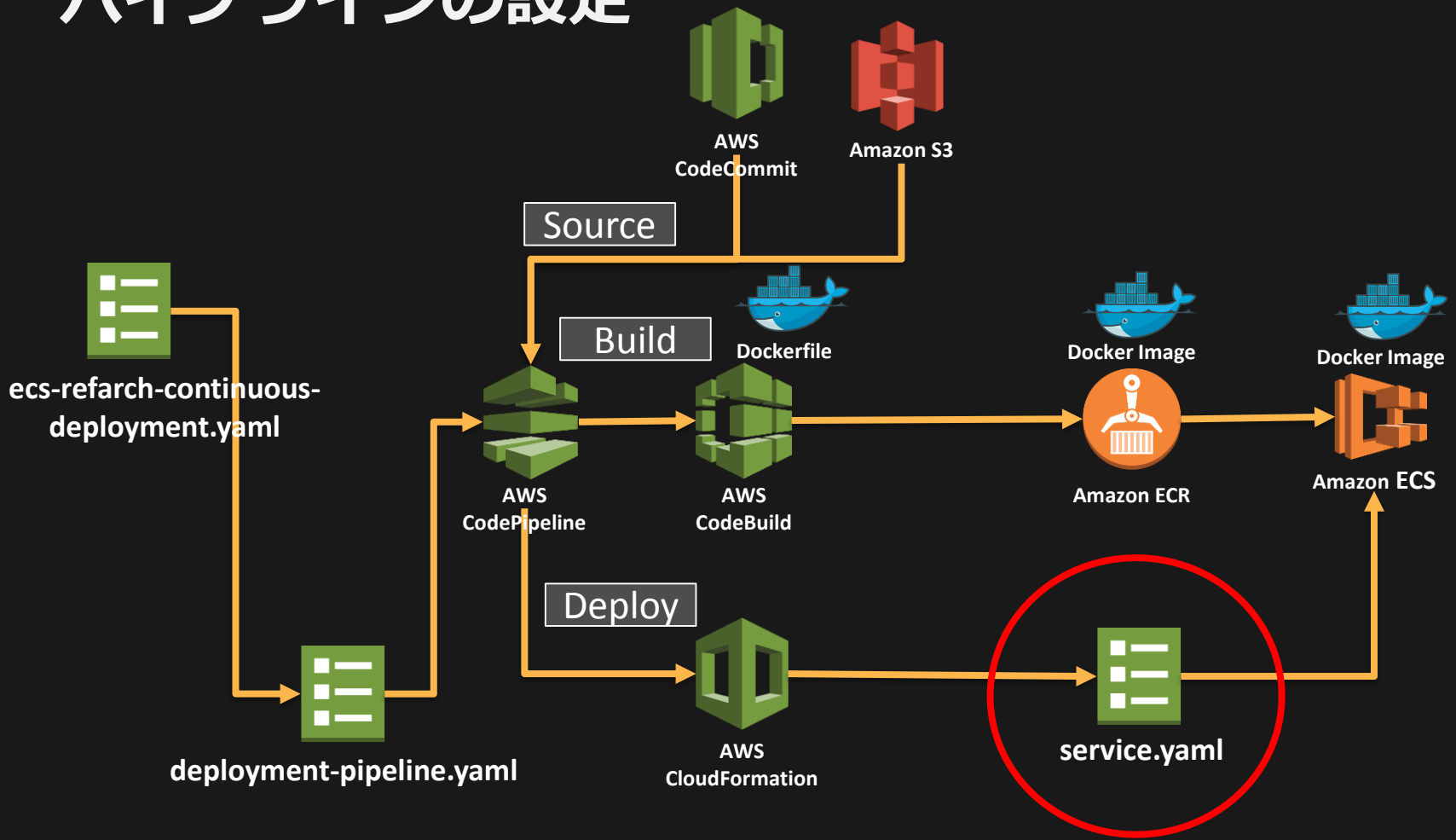
...続き

CloudFormationテンプレート パラメータのオーバーライド

Buildステージの成果物から値を取得

```
ParameterOverrides: !Sub |
    {
      "Tag" : { "Fn::GetParam" : [ "BuildOutput", "build.json", "tag" ] },
      "DesiredCount": "1",
      "Cluster": "${Cluster}",
      "TargetGroup": "${TargetGroup}",
      "Repository": "${Repository}"
    }
InputArtifacts:
  - Name: Template
  - Name: BuildOutput
RunOrder: 1
```

パイプラインの設定


AWS
CodeCommit
Amazon S3
Source
ecs-refarch-continuous-
deployment.yaml
Build
Dockerfile
Docker Image
Docker Image
AWS
CodePipeline
AWS
CodeBuild
Amazon ECR
Amazon ECS
Deploy
deployment-pipeline.yaml
AWS
CloudFormation
service.yaml

CloudFormation: Service.yaml

```
Service:
  Type: AWS::ECS::Service
  Properties:
    Cluster: !Ref Cluster
    Role: !Ref ECSServiceRole
    DesiredCount: !Ref DesiredCount
    TaskDefinition: !Ref TaskDefinition
    LoadBalancers:
      - ContainerName: aspnetcore-app
        ContainerPort: 80
        TargetGroupArn: !Ref TargetGroup
```

ECS タスク定義

コンテナのポートとターゲットグループの指定

... 続く

CloudFormation: Service.yaml

```
...続き
TaskDefinition:
  Type: AWS::ECS::TaskDefinition
  Properties:
    Family: !Sub ${AWS::StackName}-aspnetcore-app
    ContainerDefinitions:
      - Name: aspnetcore-app
        Image: !Sub ${AWS::AccountId}.dkr.ecr.${AWS::Region}.amazonaws.com/${Repository}:${Tag}
        Cpu: 512
        Essential: true
        Memory: 512
        PortMappings:
          - ContainerPort: 80
        Environment:
          - Name: Tag
            Value: !Ref Tag
```

Dockerイメージの指定

リソースの指定

Demo: 継続的デプロイメント

まとめ

まとめ

- C#はエンタープライズ開発に最適な開発言語
- .NET Core は軽量、高速、マルチプラットフォーム
- 継続的インテグレーション/継続的デプロイメントで開発生産性を向上
- Dockerの導入で開発環境から本番環境まで一貫性を保つ
- CI/CDを実現するAWSの各サービスをうまく利用することで価値ある作業に集中する

関連セッション

- 2017/5/31　17:20 ～ 18:00　プリンスホール
  D2T7-6（Dev D2T7-6Day トラック 1）
  Amazon ECS の進化、DevOps と Microservices の実践

- 2017/6/1　13:20 ～ 14:00　プリンスホール
  D3T7-2（Dev Day トラック 1）
  DevSecOps on AWS - Policy in Code

- 2017/6/2　17:20 ～ 18:00　国際館パミール 3F
  D4T2-6（AWS Techトラック 2）
  AWS マネージドサービスで実現する CI/CD パイプライン

Don’t Forget Evaluations!

アンケートにご記入をお願いします。

Thank You!

ご清聴ありがとうございました。