AWSの継続的インテグレーション/デリバリー総まとめ！
モダンアプリケーション構築のためのCI/CDベストプラクティス！

福井 厚

シニアソリューションアーキテクト
アマゾンウェブサービスジャパン株式会社

aws SUMMIT ONLINE

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
  - 技術統括本部レディネスソリューション本部
  - シニアソリューションアーキテクト
  - サーバーレス スペシャリスト

- **関心領域**
  - ソフトウェア アーキテクチャ、オブジェクト指向設計、アジャイル開発
- **好きなAWSサービス**
  - サーバーレステクノロジー全般、 AWS Code シリーズ、AWS Amplify

本セッションの目的

- AWS上でモダンアプリケーション開発を行う方を対象に継続的インテグレーション/継続的デリバリー・デプロイメント(CI/CD)に利用可能なAWSの各サービスについて理解し、有効活用して頂く

本セッションで扱う内容

- CI/CDおよびInfrastructure as Codeに関連するAWSサービスの解説

本セッションで扱わない内容

- CI/CDとは何か？ プロジェクトにCI/CDを採用するメリットや理由(Appendixをご参照ください)

Agenda

- CI/CDに関連するAWSサービス
- 継続的インテグレーション
- 継続的デプロイメント
- Infrastructure as code
- まとめ

CI/CDに関連するAWSサービス

aws SUMMIT ONLINE

モダンアプリケーション開発のアプローチ

- 環境の管理をシンプルに
- コードの変更の影響を低減
- 新しい高品質のサービスのデリバリーを促進
- 自動化されたオペレーション
- リソースとアプリケーションに対する洞察力を高める
- 顧客とビジネスを保護

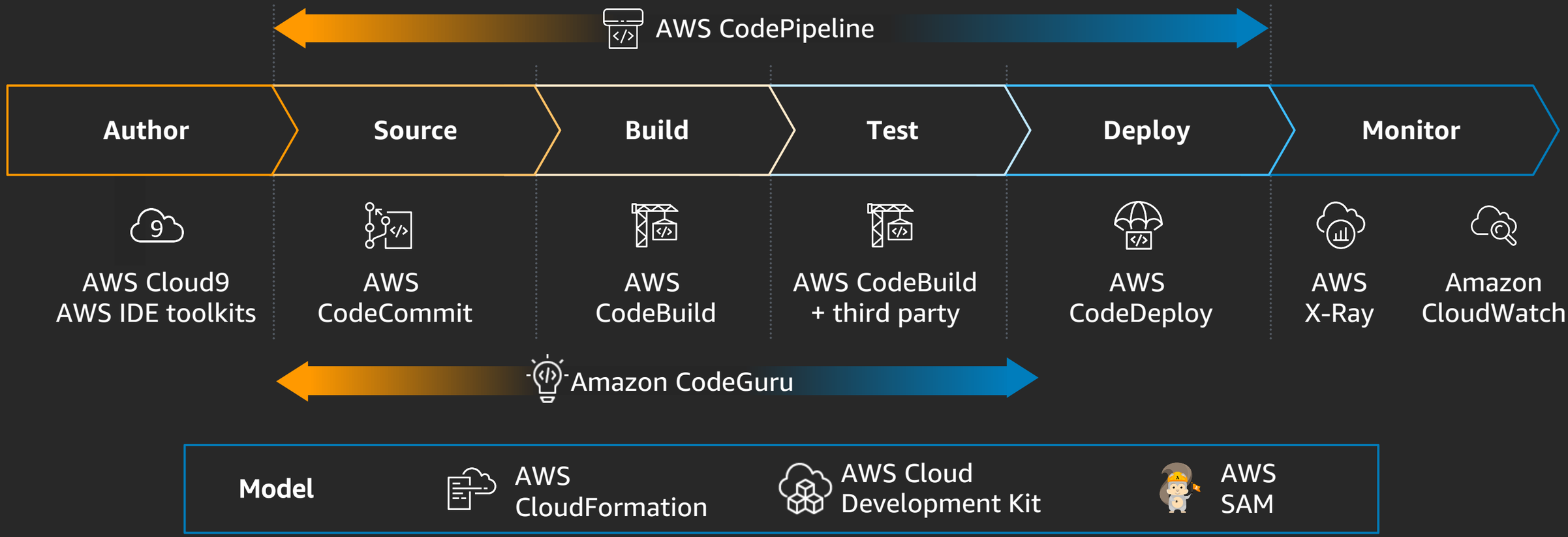
モダンアプリケーション開発のアプローチ

- サーバーレステクノロジーで環境の管理をシンプルに
- マイクロサービスアーキテクチャでコードの変更の影響を低減
- CI/CDで新しい高品質のサービスのデリバリーを促進
- Infrastructure as Codeによる自動化されたオペレーション
- 観測性向上でリソースとアプリケーションに対する洞察力を高める
- セキュリティとコンプライアンスで顧客とビジネスを保護

モダンアプリケーション開発のアプローチ

- サーバーレステクノロジーで環境の管理をシンプルに
- マイクロサービスアーキテクチャでコードの変更の影響を低減
- CI/CDで新しい高品質のサービスのデリバリーを促進
- Infrastructure as Codeによる自動化されたオペレーション
- 観測性向上でリソースとアプリケーションに対する洞察力を高める
- セキュリティとコンプライアンスで顧客とビジネスを保護

CI/CDに関連するAWSサービス



継続的インテグレーション

aws SUMMIT ONLINE

継続的インテグレーションのゴール

1. コードがチェックインされた時に自動的にリリースを開始
2. 一貫性のある繰り返し可能な環境でコードのビルドとテストを実施
3. デプロイの準備ができているアーティファクトを常に保持
4. ビルドが失敗した時のフィードバックループが常に回せる

AWS CodeBuild

- フルマネージドなビルドサービスでソースコードのコンパイル、テストの実行、ソフトウェアパッケージの作成を実行
- 継続的なスケールと同時複数ビルドプロセス
- サーバーの管理は不要
- 利用した分のみの支払い(分単位の課金)
- Amazon CloudWatchによるモニタ可能

AWS CodeBuild

- 一貫したイミュータブルな環境のために個々のビルドを新規Dockerコンテナで実行
- すべてのオフィシャルなAWS CodeBuildイメージにDockerとAWS CLIをインストール済み
- ニーズに応じてDockerイメージを作成することによってカスタムなビルド環境を提供可能

AWS CodeBuild: AWS Lambda buildspec

```
version: 0.2

phases:
  build:
    commands:
      - npm ci
      - npm test
      - >
          aws cloudformation package
              --template-file template.yml
            --output-template template-output.yml
            --s3_bucket $BUCKET

artifacts:
  type: zip
  files:
    - template-output.yml
```

依存モジュールの取得と
単体テストの実施

CloudFormation
Packageコマンドを実行

AWS CodeBuild: Docker buildspec

```
version: 0.2

phases:
  build:
    commands:
      - $(aws ecr get-login --no-include-email)
      - docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG .
      - docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $ECR_REPO:$IMAGE_TAG
      - docker push $ECR_REPO:$IMAGE_TAG
```

Amazon ECRへのログイン
Dokcerイメージの作成
Dockerイメージへのタグ付け
DockerイメージをECRへPush

継続的デプロイメント

aws SUMMIT ONLINE

継続的デプロイメントのゴール

1. 更新された変更をテスト用のステージ環境に自動的にデプロイする
2. 顧客に影響を与えることなく安全に本番環境にデプロイする
3. より迅速に顧客に展開：デプロイメントの頻度の向上、変更によるリードタイムと失敗率の削減

AWS CodePipeline

- 素早く信頼性の高いアプリケーション更新のための継続的デリバリーサービスを提供
- ソフトウェアリリースプロセスをモデル化することにより視覚化
- コードが変更されるとコードのビルド、テスト、デプロイを実施
- サードパーティのツールやAWSサービスと統合

AWS CodePipeline: サポートしているソース

自動的な最新のソースコード取得とリリースの開始

ブランチからの取得

AWS CodeCommit
GitHub

フォルダのオブジェクトからの取得

Amazon Simple Storage Service (Amazon S3)

Dokerレポジトリからの取得

Amazon ECR

AWS CodePipeline: サポートしているターゲット※

様々なデプロイメント先をサポート

Amazon EC2

AWS CodeDeploy
AWS Elastic Beanstalk
AWS OpsWorks stacks

コンテナ

AWS CodeDeploy
Amazon ECS/Fargate
Amazon ECS /Fargate(blue/green)

サーバーレス

AWS CloudFormation (SAM)
AWS Lambda (AWS CodeDeploy)
Amazon S3
Amazon Service Catalog
Alexa Skills Kit

※ その他、オンプレミスのインスタンス、**XebiaLabs** へのデプロイもサポート

AWS CodePipeline: サポートするトリガー

自動的なリリースの実行

Amazon CloudWatch Events

- スケジュール (夜間リリース)
- AWSサービスのイベント

CloudWatch Eventsコンソール、
API、SDK、CLI、
AWS CloudFormationで利用可能

Webhooks

- GitHub
- Quay
- Artifactory

AWS CodePipeline API、SDK、
CLI、AWS CloudFormationで
利用可能

AWS CodeDeploy

- サーバーのインスタンスやAWS Lambdaへの自動的なデプロイメント
- アプリケーションの複雑なアップデートの実施
- アプリケーションのデプロイ中のダウンタイムを削減
- エラーを検知すると自動的にロールバックを実行
- Amazon EC2、AWS Lambda、オンプレミスサーバー、コンテナへのデプロイ

本セッションで紹介するデプロイターゲット
サーバーレスアプリケーションの実行環境

AWS Lambda

Serverless functions

イベント駆動
多数の言語ランタイム
データソースとの統合
サーバーの管理不要

AWS Fargate

Serverless containers

ロングランニング
OSの抽象化
フルマネージドなオーケストレーション
フルマネージドなクラスタスケーリング

CodeDeploy Lambdaデプロイメント

- AWS Lambdaの関数重み付けエイリアスを利用したトラフィックのシフト
- カナリアデプロイ(10分間10%のトラフィックをシフト、その後残り全部もシフト)やリニアデプロイ(毎10分ごとに10%ずつシフト)を選択可能
- Validation Hookは各ステージへのデプロイ時のテストを有効化
- Hookが失敗した場合やAmazon CloudWatchアラームを検知した場合は数秒で迅速にロールバック
- コンソール、API、Amazon SNSの通知、CloudWatchイベントによるデプロイメントステータスのモニタが可能

CodeDeploy Lambdaデプロイメント(SAM)

サーバーレスアプリケーションのテンプレート

```
Resources:
  GetFunction:
    Type: AWS::Serverless::Function
    Properties:
      DeploymentPreference:
        Type: Canary10Percent10Minutes
        Alarms:
          - !Ref ErrorsAlarm
        Hooks:
          PreTraffic: !Ref PreTrafficHook
```

SAMによるLambda関数の定義

カナリアデプロイの定義

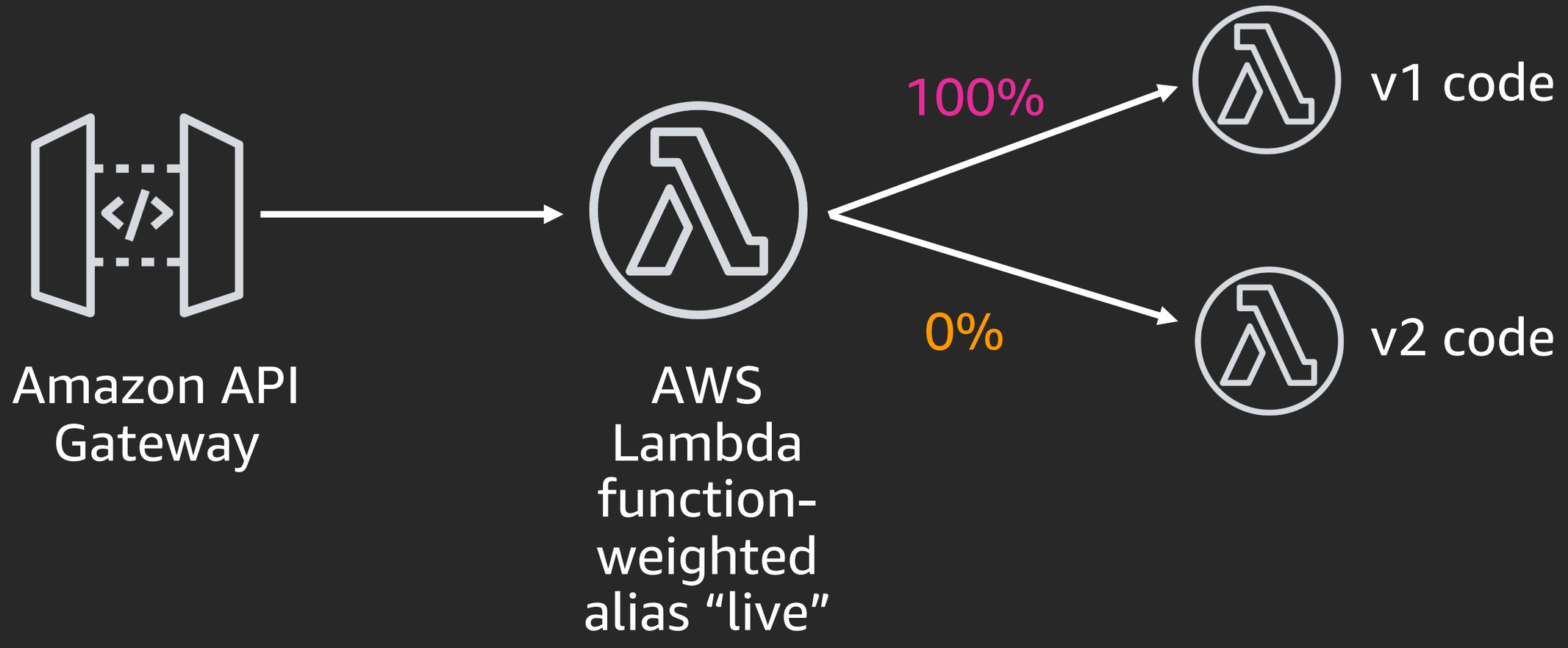
CloudWatchアラームの定義

Hook用Lambda関数の指定

CodeDeploy Lambda カナリア デプロイメント

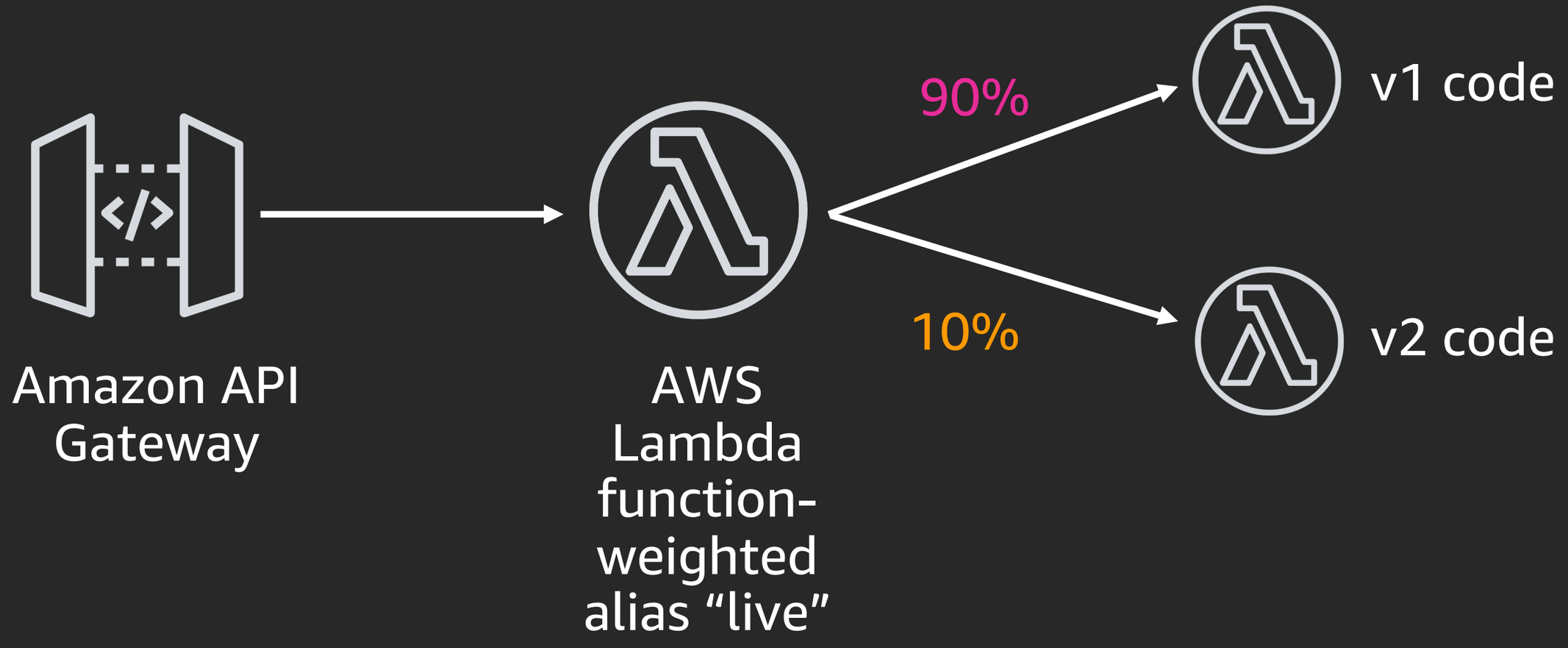
CodeDeploy Lambda カナリア デプロイメント

V2コードがトラフィックを受け取る前にHook関数が実行される

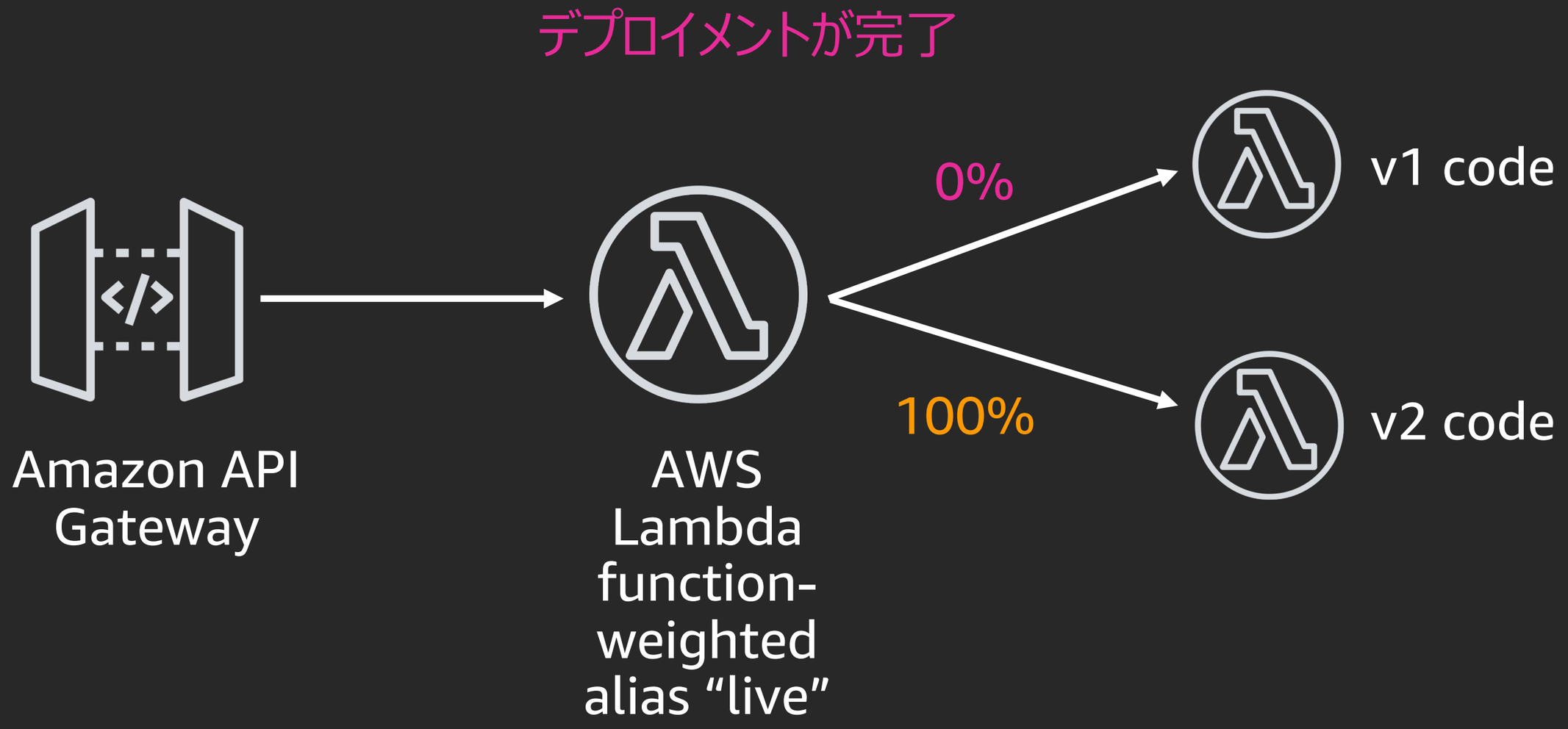


CodeDeploy Lambda カナリア デプロイメント

警告によるロールバックに備えて10分間待機



CodeDeploy Lambda カナリア デプロイメント



CodeDeploy ECS blue/greenデプロイメント

- Greenタスクをプロビジョニングし、ロードバランサーのトラフィックを切り替え
- 検証Hookによって各ステージのデプロイメントでテストを有効化
- Hookが失敗した場合やAmazon CloudWatchアラームを検知した場合は数秒でBlueタスクに迅速にロールバック
- コンソール、API、Amazon SNSの通知、CloudWatchイベントによるデプロイメントステータスのモニタが可能

CodeDeploy ECS appspec

```
version: 1.0
Resources:
    - TargetService:
        Type: AWS::ECS::Service
        Properties:
            - TaskDefinition: "my_task_definition:8"
              LoadBalancerInfos:
                    - ContainerName: "SampleApp"
                      ContainerPort: 80
```

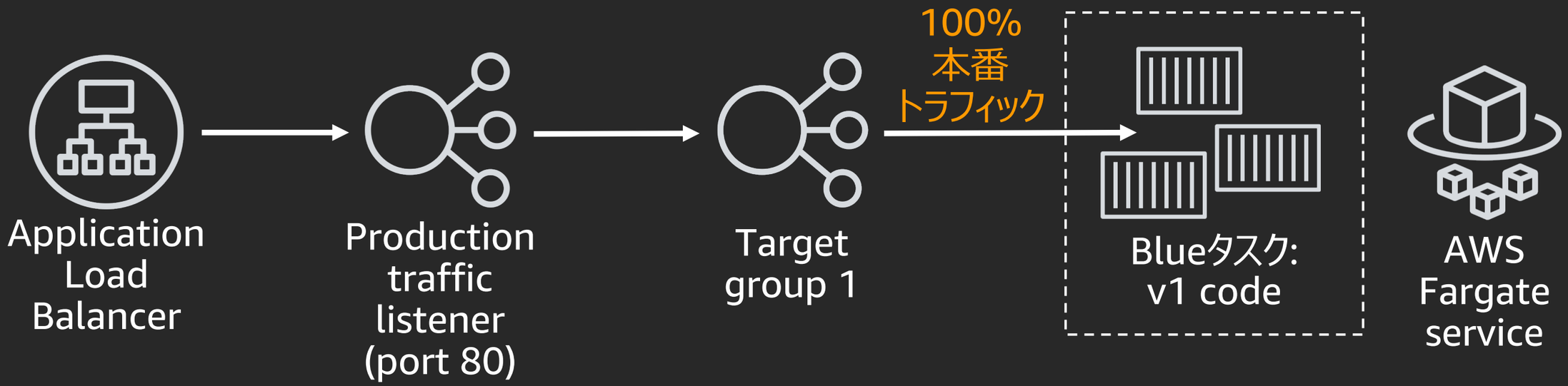
ターゲットタイプの指定
タスク定義
ロードバランサの定義

```
Hooks:
    - BeforeInstall:
"LambdaFunctionToExecuteAnythingBeforeNewRevisionInstalltion"
    - AfterInstall:
"LambdaFunctionToExecuteAnythingAfterNewRevisionInstallation"
    - AfterAllowTestTraffic: "LambdaFunctionToValidateAfterTestTrafficShift"
    - BeforeAllowTraffic: "LambdaFunctionToValidateBeforeTrafficShift"
    - AfterAllowTraffic: "LambdaFunctionToValidateAfterTrafficShift"
```

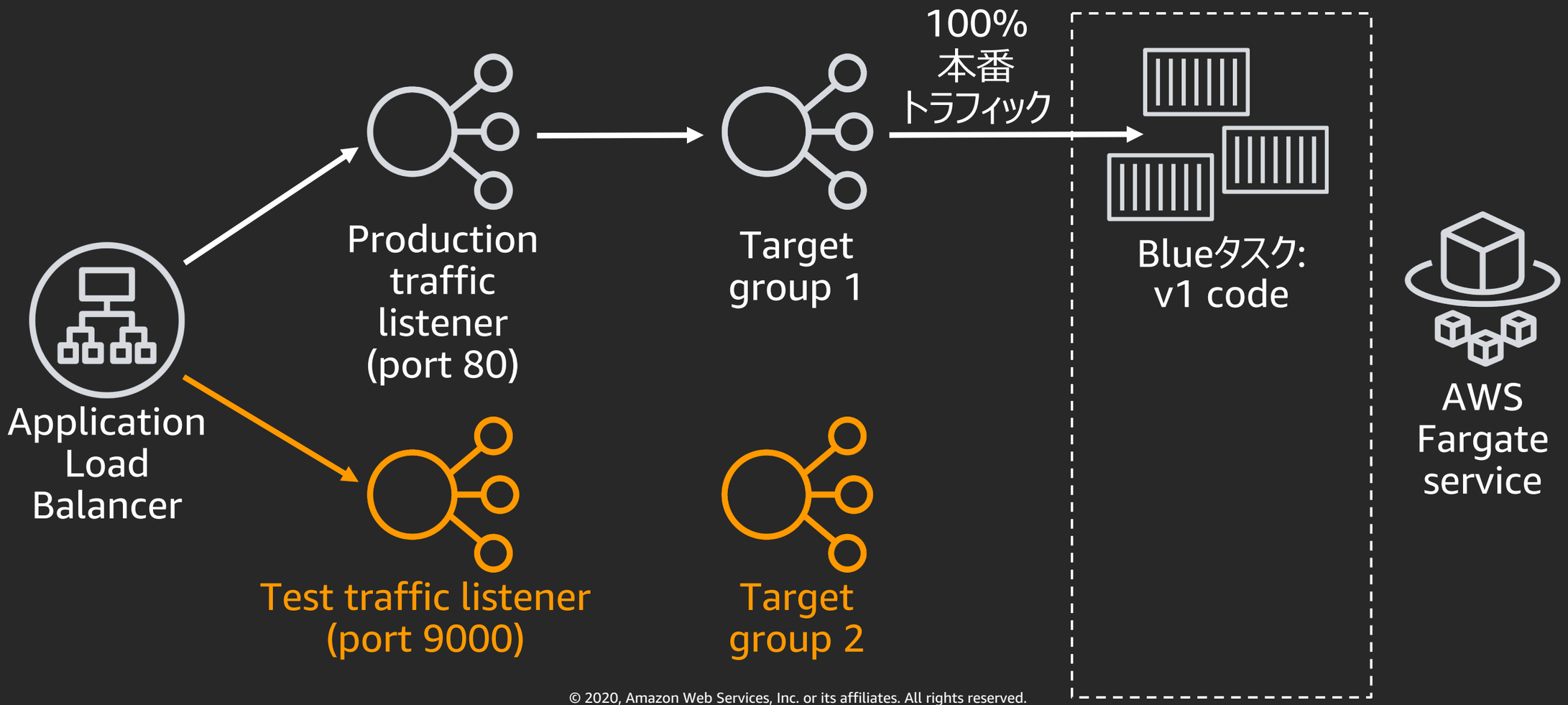
フック関数の定義 ※

※実際にはLambdaのARNを指定

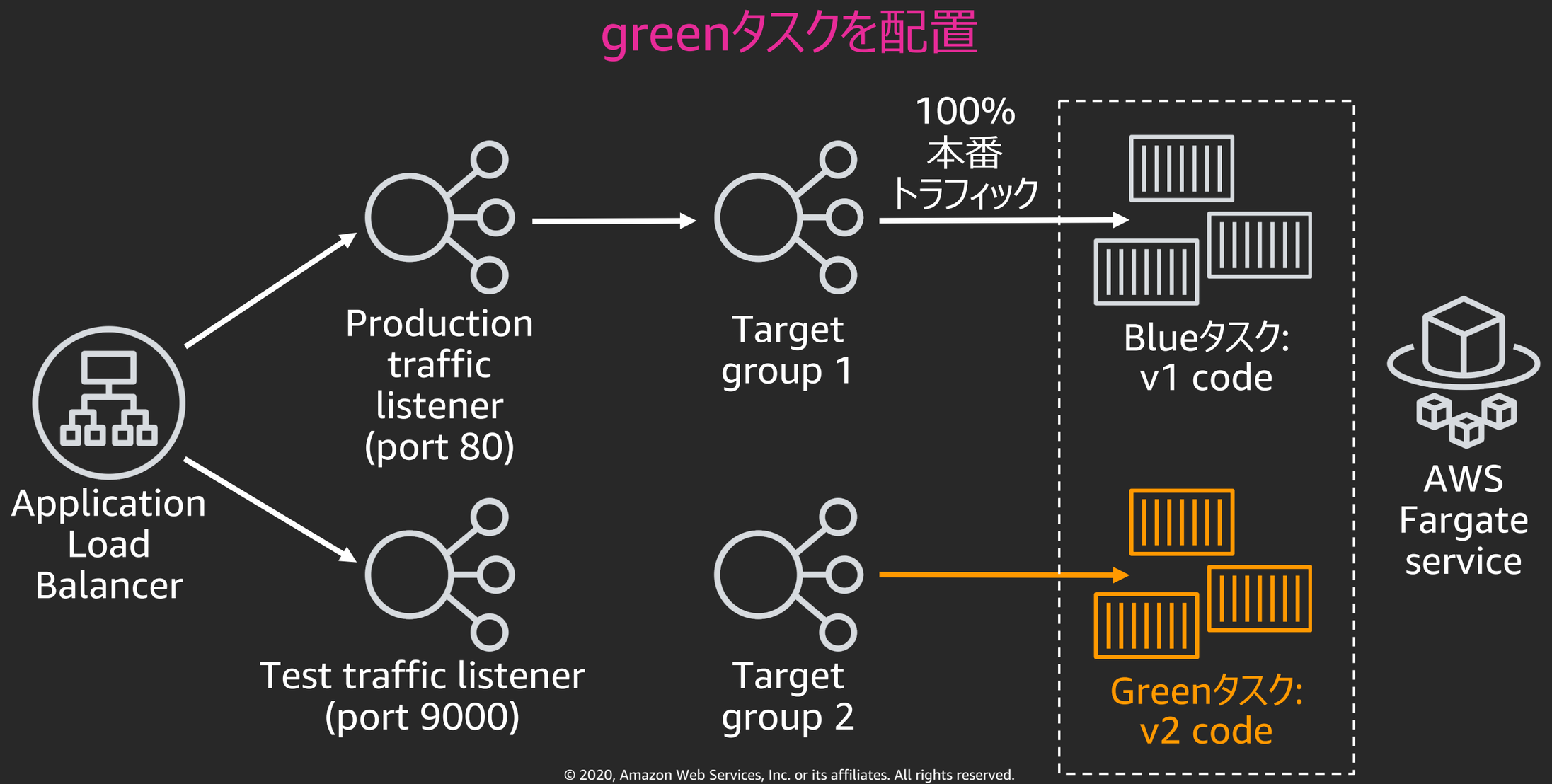
CodeDeploy ECS blue/greenデプロイメント



CodeDeploy ECS blue/greenデプロイメント

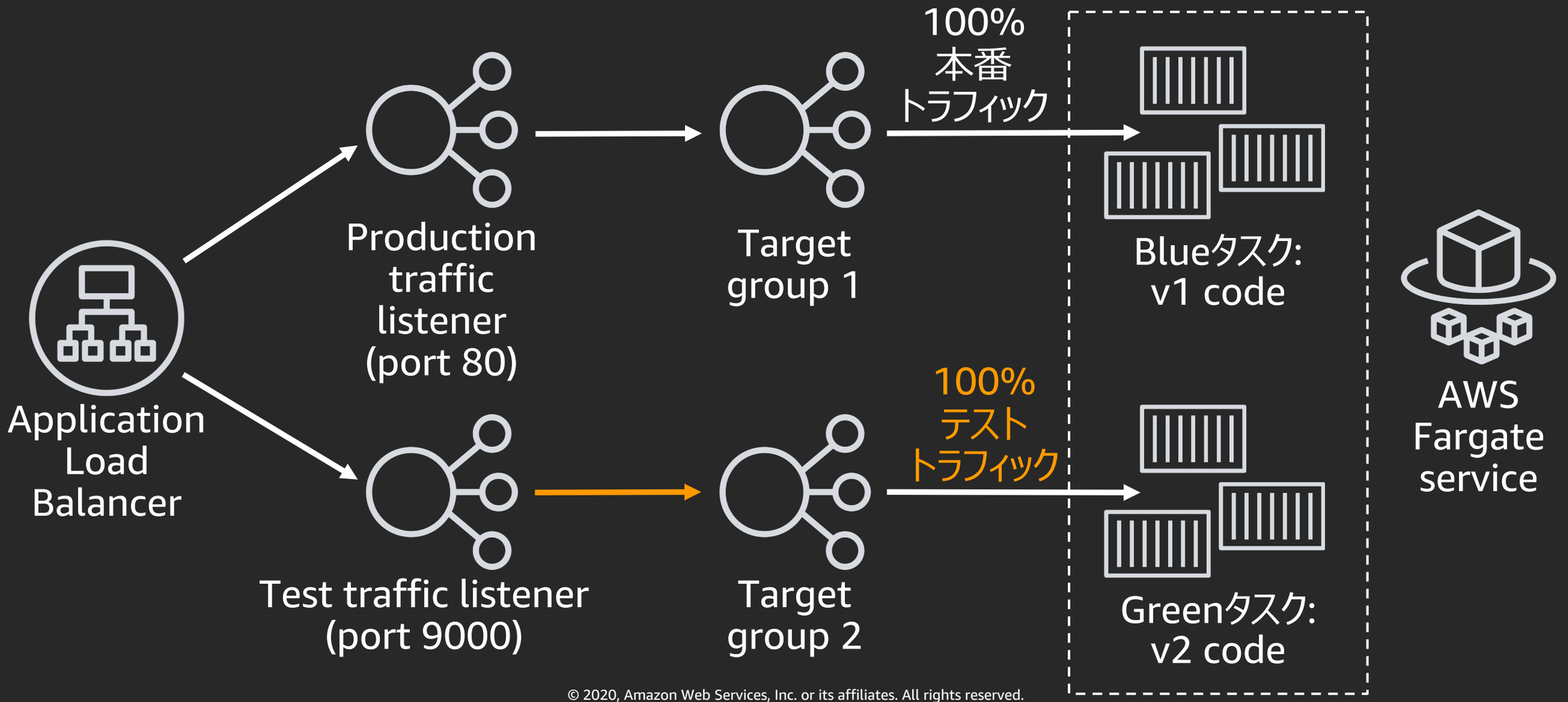


CodeDeploy ECS blue/greenデプロイメント



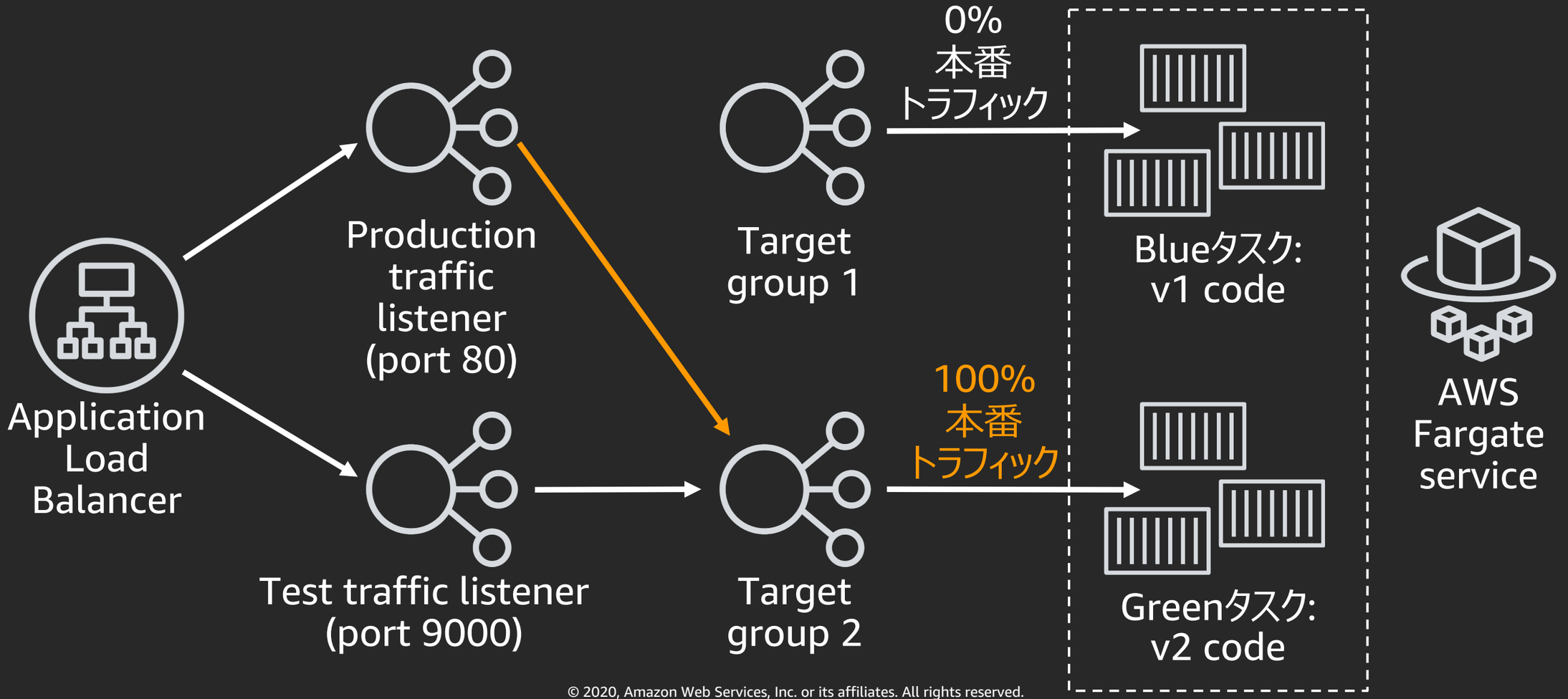
CodeDeploy ECS blue/greenデプロイメント

greenタスクが本番用トラフィックを受け取る前にテストエンドポイントでhookが実行される

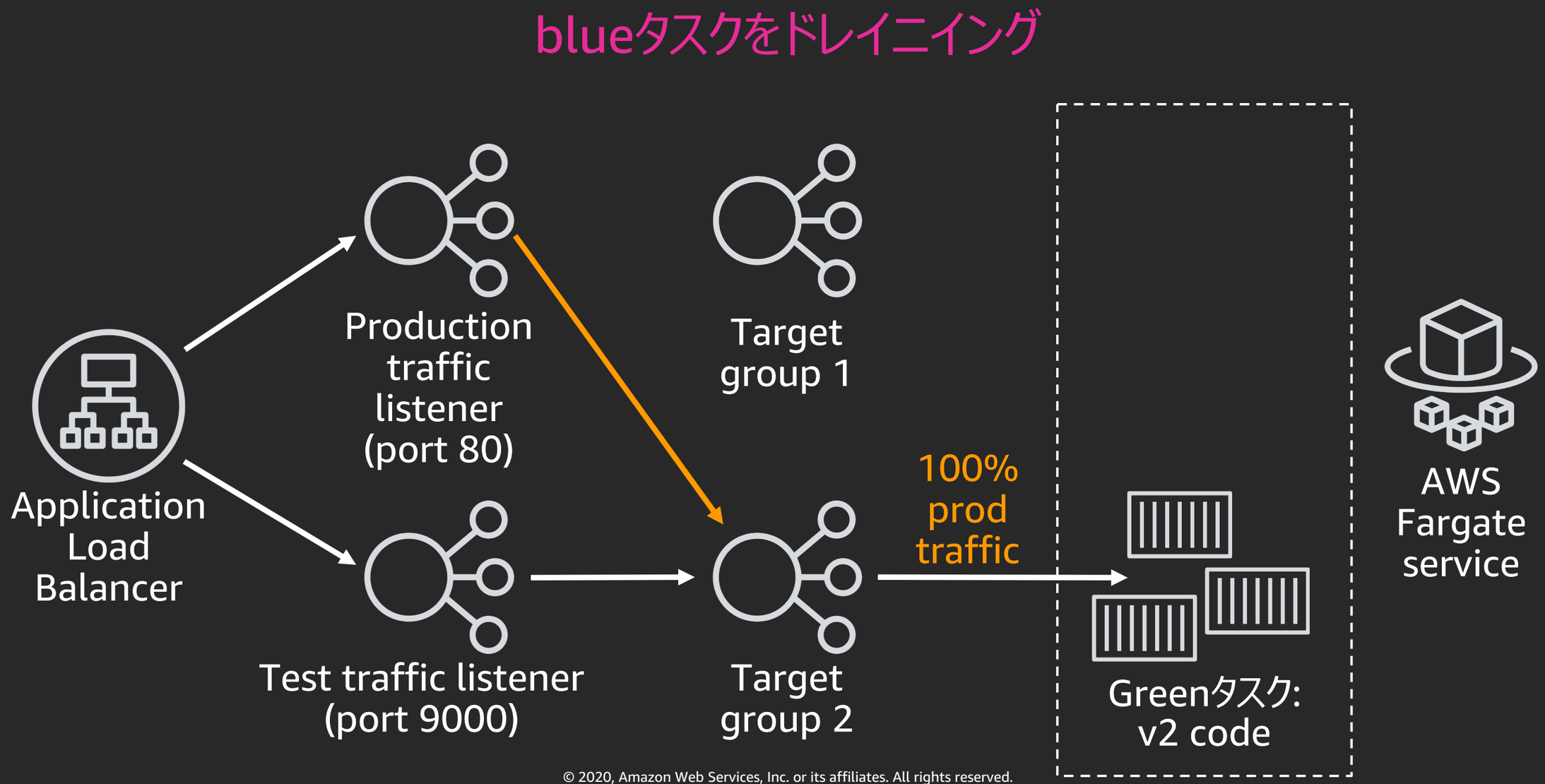


CodeDeploy ECS blue/greenデプロイメント

Greenタスクへのトラフィックを切り替え。もし失敗を検知した場合はロールバック



CodeDeploy ECS blue/greenデプロイメント



重要：デプロイメントのためのコンテナイメージのタグ付け

- Dockerタグはデプロイ時のみではなく個々のコンテナの開始時にも利用

- latest や prod タグはスケールアウトイベント発生時に未テストのコードを本番環境へデプロイする結果を招く

  デプロイメントにはユニークでイミュータブルなタグを利用すべき

デプロイメントのためのコンテナイメージのタグ付け

AWS
Fargate
service

Image: sha256@11111... (“latest”)

Amazon
ECR
repository

デプロイメントのためのコンテナイメージのタグ付け

ビルドが新しい latest イメージをプッシュ

AWS
Fargate
service

Image: sha256@11111...

Image: sha256@22222... (“latest”)

Amazon
ECR
repository

デプロイメントのためのコンテナイメージのタグ付け

サービスがスケールアウトし新しいタスクを実行開始

AWS
Fargate
service

Image: sha256@11111...

Image: sha256@22222... (“latest”)

Amazon
ECR
repository

デプロイメントのためのコンテナイメージのタグ付け

イミュータブルなタグをデプロイ

SHA256 digestを利用

```
{
    "name": "sample-app",
    "image": "amazon/amazon-ecs-
        sample@sha256:3e39d933b1d948c92309bb583b5a1f3d28f0119e1551ca1fe538ba414a41af48d"
}
```

Build IDを利用

```
{
    "name": "sample-app",
    "image": "amazon/amazon-ecs-sample:build-b2085490-359f-4eaf-8970-6d1e26c354f0"
}
```

デプロイメントのためのコンテナイメージのタグ付け

ビルド時にイミュータブルなタグを生成

SHA256 digest

```
export IMAGE_URI=`docker inspect --format='{{index .RepoDigests 0}}' my_image:$IMAGE_TAG
```

サンプル結果：

```
amazon/amazon-ecs-sample@sha256:3e39d933b...
```

Build ID

```
export IMAGE_TAG=build-`echo $CODEBUILD_BUILD_ID | awk -F":" ‘{print $2}'`
```

サンプル結果:

```
build-b2085490-359f-4eaf-8970-6d1e26c354f0
```

デプロイメントのためのコンテナイメージのタグ付け

AWS
Fargate
service

Image: sha256@11111... (“build-11111”)

Amazon
ECR
repository

デプロイメントのためのコンテナイメージのタグ付け

ビルドが新しいビルドIDのイメージタグをプッシュ

AWS Fargate service

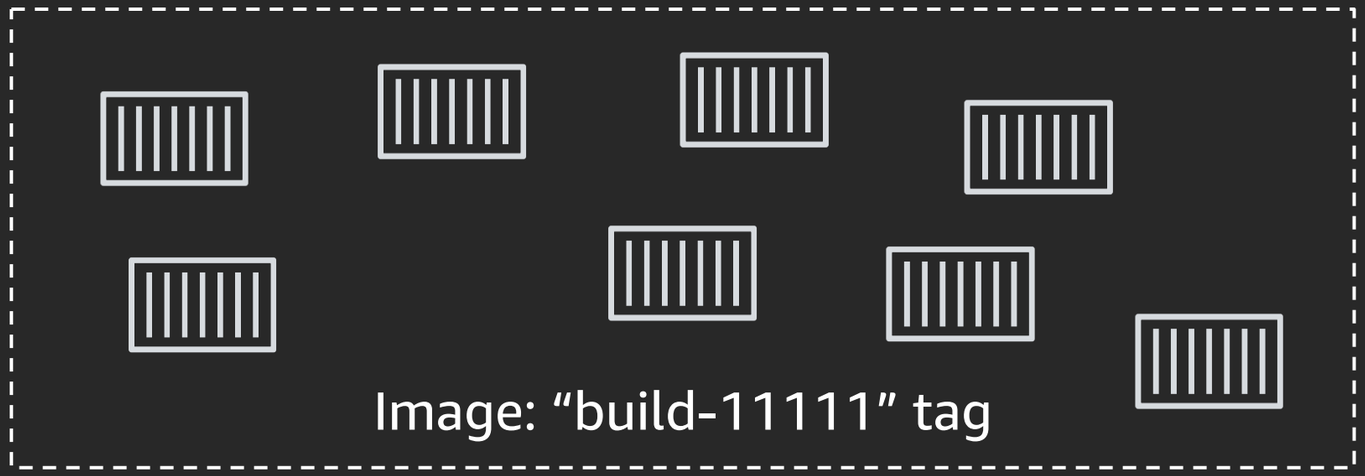
Image: sha256@11111... (“build-11111”)

Image: sha256@22222... (“build-22222”)

Amazon ECR repository

デプロイメントのためのコンテナイメージのタグ付け

サービスがスケールアウトし、新しいタスクを実行しようとする



AWS Fargate service

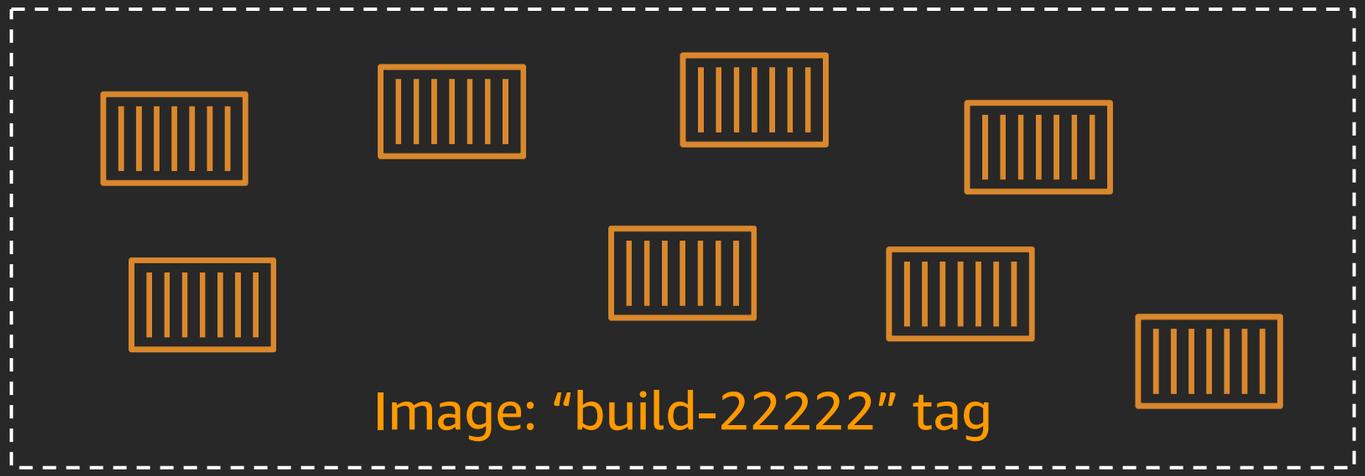
Image: sha256@11111... (“build-11111”)

Image: sha256@22222... (“build-22222”)

Amazon ECR repository

デプロイメントのためのコンテナイメージのタグ付け

デプロイメントはサービス定義を更新し、タスクを置き換え



AWS Fargate service

Image: sha256@11111... (“build-11111”)

Image: sha256@22222... (“build-22222”)

Amazon ECR repository

モダンアプリケーション開発のアプローチ

- サーバーレステクノロジーで環境の管理を単純化
- マイクロサービスアーキテクチャでコードの変更の影響を低減
- CI/CDで新しい高品質のサービスのデリバリーを促進
- Infrastructure as Codeによる自動化されたオペレーション
- 観測性向上でリソースとアプリケーションに対する洞察力を高める
- セキュリティとコンプライアンスで顧客とビジネスを保護

Infrastructure as code

aws SUMMIT ONLINE

Infrastructure as codeのゴール

1. インフラストラクチャの変更を繰り返し、予測可能にする
2. インフラストラクチャの変更のリリースにコードの変更と同じツールを利用する
3. 本番環境をステージング環境に複製して継続的テストを有効化する

infrastructure as codeによる継続的テスト

成果物の検証

(ビルドステージ)

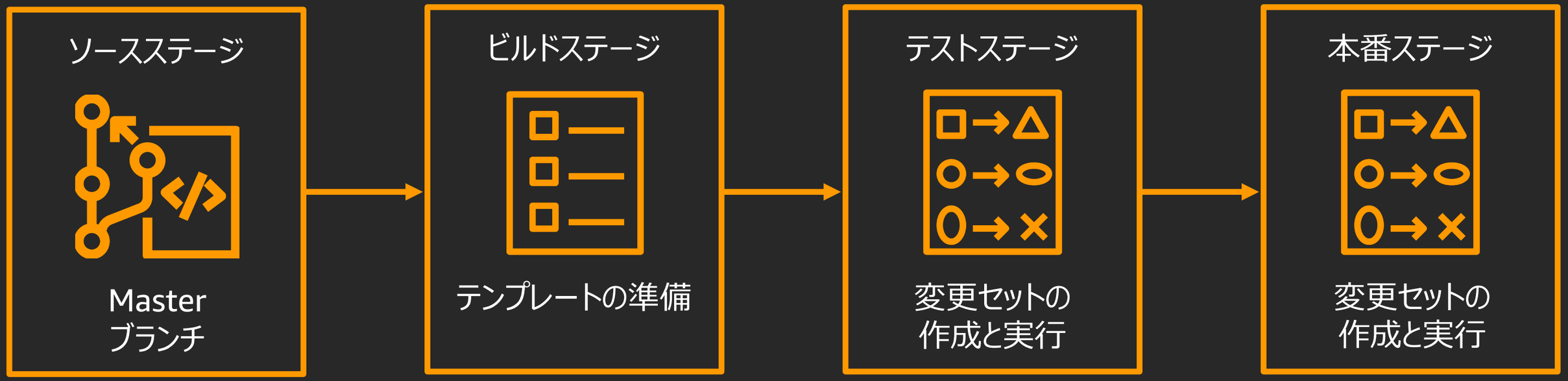
- ユニットテスト
- 静的解析
- 依存ライブラリと環境のモック化
- 脆弱性イメージスキャン

環境の検証

(テストステージ)

- テストエージェントとの統合
  リアルな依存ライブラリと環境
- 負荷テスト
- 侵入テスト
- 環境にデプロイした成果物に対するテストの影響をモニタリング

infrastructure as codeのリリース



LambdaのデプロイにはAWS Serverless Application Model (SAM)が利用可能

- AWS上のサーバーレスアプリケーションを構築するためのオープンソース フレームワーク
- 関数、API、データベース、イベントソースマッピングを表現する簡易な文法を使用
- デプロイ時にSAMの文法をAWS CloudFormationの文法に変換、展開
- すべてのAWS CloudFormationリソースタイプをサポート

https://aws.amazon.com/serverless/sam/

SAMテンプレート

- 同一テンプレート内にSAM以外のAWS CloudFormationリソースを混在させることが可能(例：Amazon S3、Amazon Kinesis、AWS Step Functionsなど)
- パタメータ、マッピング、出力などをサポート
- 組み込み関数をサポート(Fn::GetAtt、Fn::Base64など)
- ImportValueを利用可能(RestApi-id、Policies、StageNme属性は除く)

SAM テンプレート

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: app.lambda_handler
      Runtime: python3.8
      CodeUri: hello_world/
      Policies: AmazonDynamoDBReadOnlyAccess
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /resource/{resourceId}
            Method: get
  Table:
    Type: AWS::Serverless::SimpleTable
```

- リソースタイプ
- ここではLambda関数を指定

- イベントソースの指定
- ここではAPI Gatewway

- データストアの指定
- ここではDynamoDB

SAMテンプレート

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: app.lambda_handler
      Runtime: python3.8
      CodeUri: hello_world/
      Policies: AmazonDynamoDBReadOnlyAccess
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /resource/{resourceId}
            Method: get
  Table:
    Type: AWS::Serverless::SimpleTable
```

SAMシンタックスがたった18行で多くのリソースを作成

- AWS Lambda関数
- Amazon API Gateway
- Amazon DynamoDB テーブル
- AWS Identity and Access Management (IAM) ロール

SAM CLIでSAMテンプレートのパッケージとデプロイ

```
$ pip install --user aws-sam-cli
$ sam init
$ sam build
$ sam deploy --guided
```

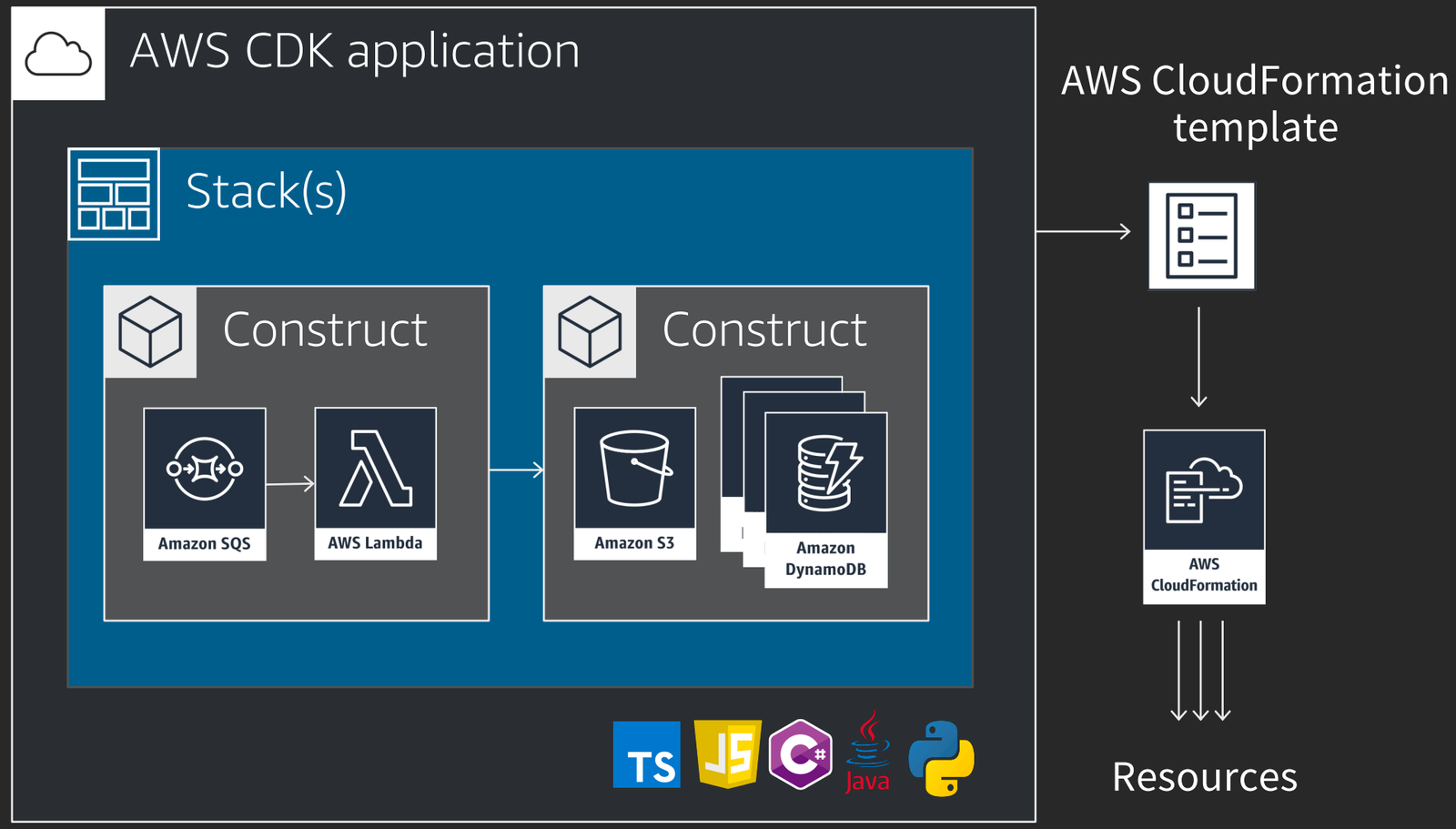
CodePipeline

SAMアプリケーションは
CloudFormation
デプロイメント アクションを
利用してデプロイ

Jenkins

SAM CLI プラグインを利用

FargateのデプロイにはAWS CDKが利用可能



AWS CloudDevelopment Kit(CDK)でモデル化

AWS Cloud Development Kit

- クラウドインフラストラクチャをTypeScriptで定義するためのオープンソースフレームワーク
- デフォルトでAWSのベストプラクティスを組み込んだハイレベルリソースタイプ(コンストラクトクラス)を提供 (npmモジュールとしてパッケージ)
- AWS Cloudformationでリソースをプロビジョニング
- すべてのCloudFormationリソースタイプをサポート

https://awslabs.github.io/aws-cdk

AWS CDK template

```
import ec2 = require('@aws-cdk/aws-ec2');
import ecs = require('@aws-cdk/aws-ecs');
import cdk = require('@aws-cdk/cdk');

class BonjourFargate extends cdk.Stack {
  constructor(parent: cdk.App, name: string, props?: cdk.StackProps) {
    super(parent, name, props);

    const vpc = new ec2.VpcNetwork(this, 'MyVpc', { maxAZs: 2 });
    const cluster = new ecs.Cluster(this, 'Cluster', { vpc });

    new ecs.LoadBalancedFargateService(
      this, "FargateService", {
        cluster,
        image: ecs.DockerHub.image("amazon/amazon-ecs-sample"),
    });
  }
}

const app = new cdk.App();
new BonjourFargate(app, 'Bonjour');
app.run();
```

VPC、サブネット、
セキュリティグループ、
インターネットゲートウェイ、
NATゲートウェイ、ルートテーブルを
含むハイレベルVPCクラス

AWS CDK template

```
import ec2 = require('@aws-cdk/aws-ec2');
import ecs = require('@aws-cdk/aws-ecs');
import cdk = require('@aws-cdk/cdk');

class BonjourFargate extends cdk.Stack {
  constructor(parent: cdk.App, name: string, props?: cdk.StackProps) {
    super(parent, name, props);

    const vpc = new ec2.VpcNetwork(this, 'MyVpc', { maxAZs: 2 });
    const cluster = new ecs.Cluster(this, 'Cluster', { vpc });

    new ecs.LoadBalancedFargateService(
      this, "FargateService", {
        cluster,
        image: ecs.DockerHub.image("amazon/amazon-ecs-sample"),
    });
  }
}

const app = new cdk.App();
new BonjourFargate(app, 'Bonjour');
app.run();
```

Amazon ECSサービス、ECSタスク定義、アプリケーションロードバランサ、リスナールール、ターゲットグループ、オプションのAmazon Route 53エイリアスレコードを含むハイレベルAWS Fargateクラス

AWS CDK template

```
import ec2 = require('@aws-cdk/aws-ec2');
import ecs = require('@aws-cdk/aws-ecs');
import cdk = require('@aws-cdk/cdk');

class BonjourFargate extends cdk.Stack {
  constructor(parent: cdk.App, name: string, props?: cdk.StackProps) {
    super(parent, name, props);

    const vpc = new ec2.VpcNetwork(this, 'MyVpc', { maxAZs: 2 });
    const cluster = new ecs.Cluster(this, 'Cluster', { vpc });

    new ecs.LoadBalancedFargateService(
      this, "FargateService", {
        cluster,
        image: ecs.DockerHub.image("amazon/amazon-ecs-sample"),
    });
  }
}

const app = new cdk.App();
new BonjourFargate(app, 'Bonjour');
app.run();
```

22行のTypeScriptコードが
400行以上の
AWS CloudForamation
テンプレートを生成

CDK CLIを利用したCDKテンプレートのsynthとdeploy

```
$ npm install -g aws-cdk
$ cdk init app --language typescript
$ cdk synth
$ cdk deploy
```

CodePipeline

CDKアプリケーションのsynth実行結果をAWS Cloudformationのデプロイメントアクションを利用してデプロイ

Jenkins

AWS CDK CLIを利用

まとめ

aws SUMMIT ONLINE

まとめ

- モダンアプリケーション開発のアプローチにはCI/CDは必須
- AWS Codeシリーズを活用することで付加価値を生まない重労働をAWSにオフロード
- AWS CodeBuild、AWS CodePipeline、AWS CodeDeployをフルに活用
- 価値を生むビジネスロジックに開発リソースを集中する！

Thank you!

aws SUMMIT ONLINE

Appendix

組織がCI/CDの原則を適用する主な理由

- 新しいハイクオリティーのサービスのデリバリーを加速するため
- 変更のインパクトを軽減するため
- リソースとアプリケーションに対する洞察を得るため
- 顧客とビジネスを守るため

CI/CDによってハイクオリティな機能を迅速にリリース

CI/CDを実践するチームは、より信頼性が向上し、より早くコードをリリースする

5x
変更失敗のレートの低下

440x
コミットからデプロイまでの速度

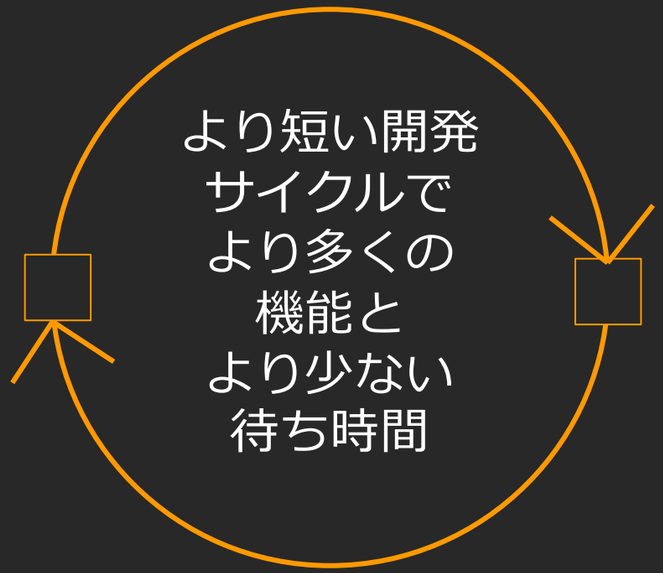
46x
より頻繁なデプロイメント

44%
新しい機能やコードに費やす時間

2017 State of DevOps Report (Puppet)

自動化の価値

マニュアルプロセスの
削減による
ソフトウェア実行
基盤の改善



チーム流動化と
採用の容易さ

リリースプロセスのステージ

リリースプロセスのステージ

- ソースコードのチェックイン
- コードのピアレビュー

リリースプロセスのステージ

- ソースコードのチェックイン
- コードのピアレビュー

- コードのコンパイル
- ユニットテスト
- スタイルチェッカー
- コンテナイメージ、関数デプロイパッケージの作成

リリースプロセスのステージ

- ソースコードのチェックイン
- コードのピアレビュー

- コードのコンパイル
- ユニットテスト
- スタイルチェッカー
- コンテナイメージ、関数デプロイパッケージの作成

- 周辺システムとの統合テスト
- 負荷テスト
- UIテスト
- セキュリティテスト

リリースプロセスのステージ

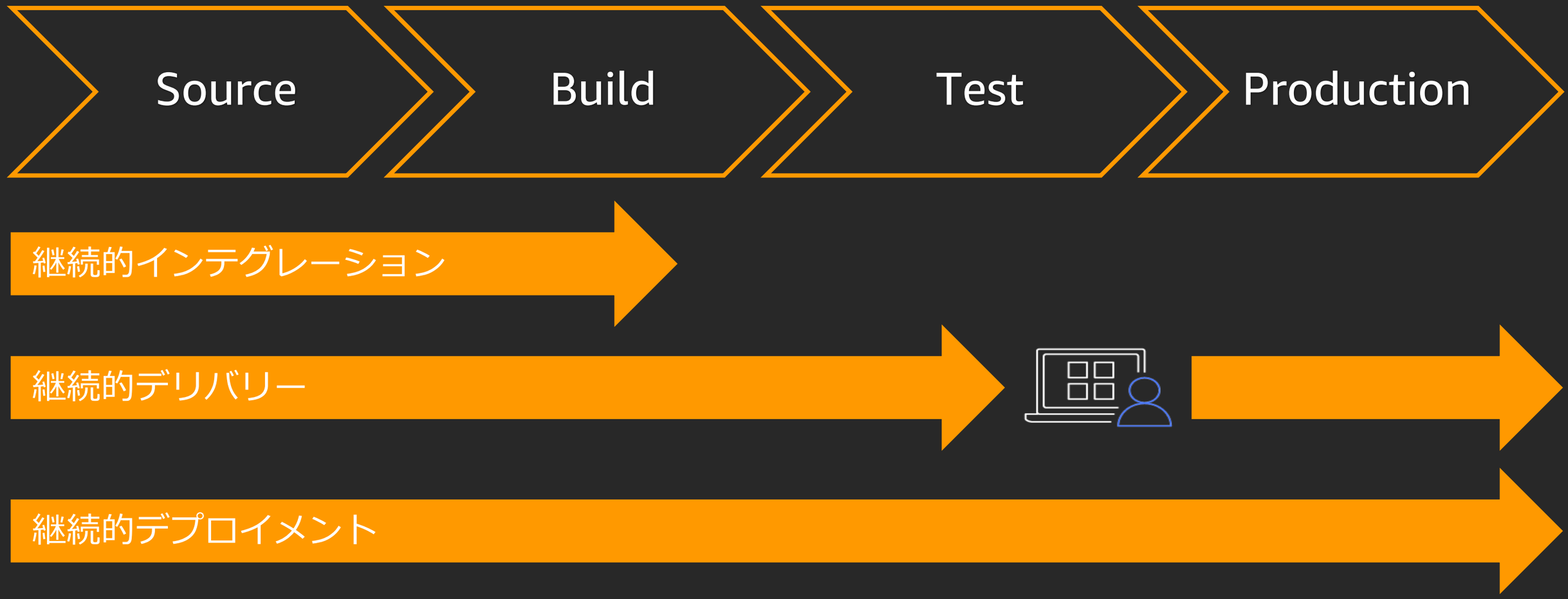
- ソースコードのチェックイン
- コードのピアレビュー

- コードのコンパイル
- ユニットテスト
- スタイルチェッカー
- コンテナイメージ、関数デプロイパッケージの作成

- 周辺システムとの統合テスト
- 負荷テスト
- UIテスト
- セキュリティテスト

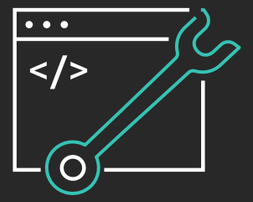
- 本番環境へのデプロイ
- エラーを素早く検知するための本番環境のモニタリング

リリースプロセスのステージ



AWS上のモダンアプリケーション構築を学ぶ

デベロッパーのスキルを構築、検証するために役立つAWSのエキスパートによって作成されたリソース



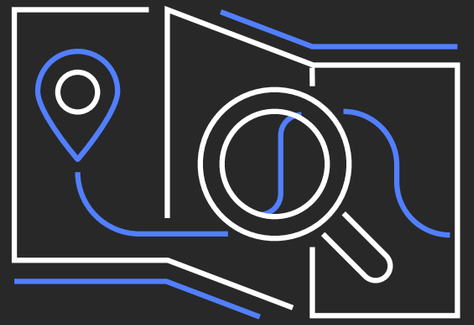
モダンアプリケーションの設計、構築、管理のスキルを開発することによって迅速なイノベーションを可能にする

無料のデジタルトレーニングとArchitecting on AWS、Developing on AWS、DevOps Engineering on AWSの有償クラスルームトレーニングによってアプリケーションのモダン化を学ぶ

AWS認定DevOps – ProfessionalまたはAWS認定Developer – Associate試験によって専門性を検証

デベロッパー学習パスはこちら： aws.amazon.com/training/path-developing

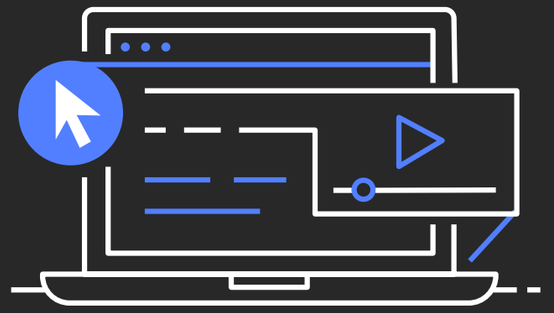
AWS トレーニングと認定



クラウド人材の育成

AWS トレーニングを活用し、ビジネスを牽引する人材の育成と組織作りを促進する

AWS トレーニング活用事例 »



自習コンテンツの活用

ウェビナーやのデジタルトレーニングを受講して、個人のスキルアップを目指す

AWS デジタルトレーニング »

AWS 認定取得を目指す

認定取得を目指して知識を底上げし、AWS の経験とスキルを証明する

AWS 認定の詳細 »

学習パスをお探しの方に

日本語版ランプアップガイドを公開しました。AWS ウェブページ、無料のデジタルトレーニング、クラスルームコース、動画、ホワイトペーパー、認定等を含んだ、9 種の役割別学習ガイドをご覧いただけます。 詳細を見る »

aws.amazon.com/training