EC2からECSへ移行を始めたお話

Takayuki Yoshioka
FAREND Technologies co., Ltd. / JAWS-UG Shimane

自己紹介

名前

吉岡隆行

所属

ファーエンドテクノロジー株式会社
JAWS-UG Shimane
Matsue.rb

業務

元 フロントエンドエンジニア
元 アプリケーションエンジニア
現 インフラエンジニア

好きなサービス

Cloud9

アジェンダ

～　EC2からECSへ移行を始めたお話　～

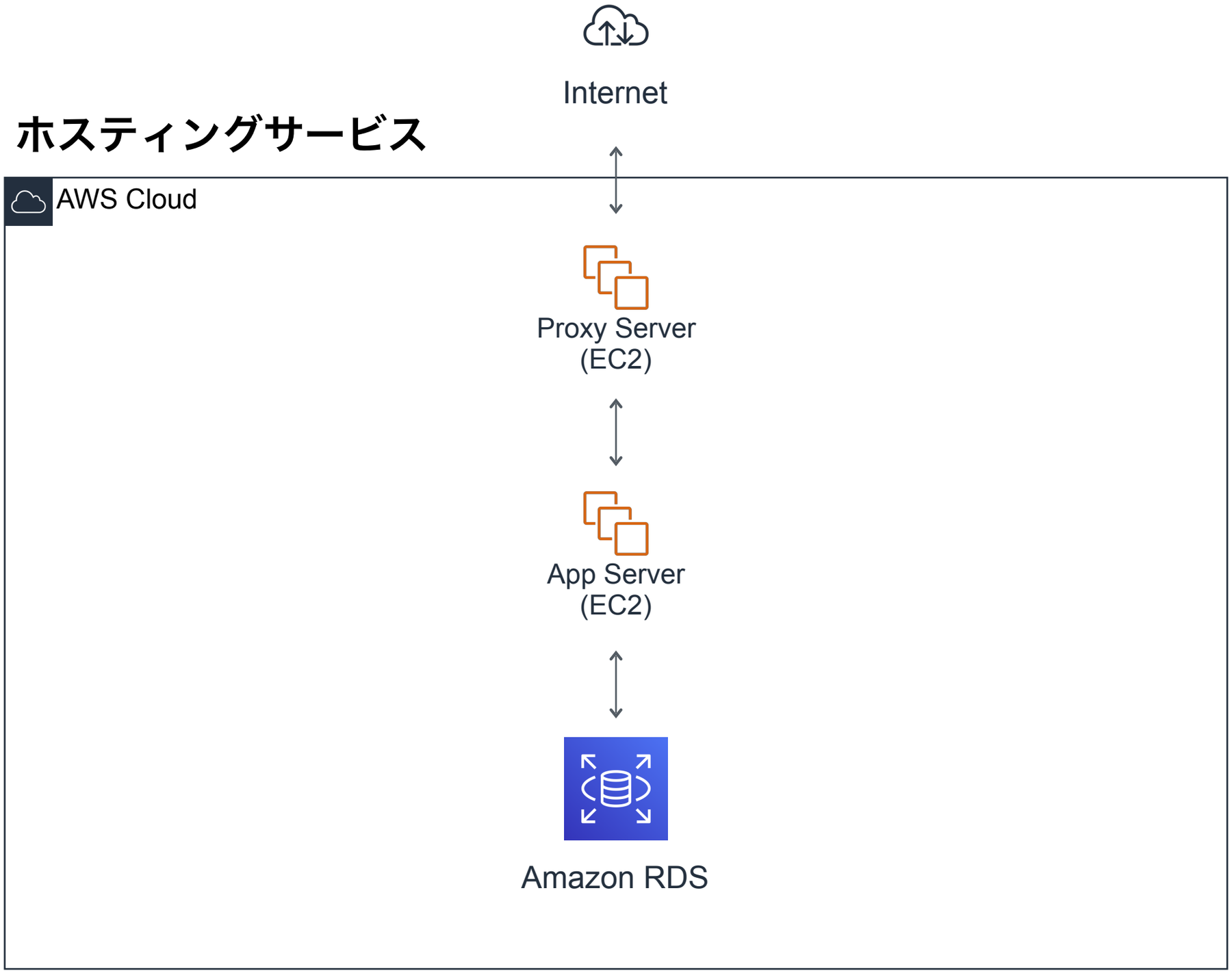
- 経緯（背景）
- 古い構成
- 新しい構成
  - Point
- 課題
- まとめ

経緯（背景）

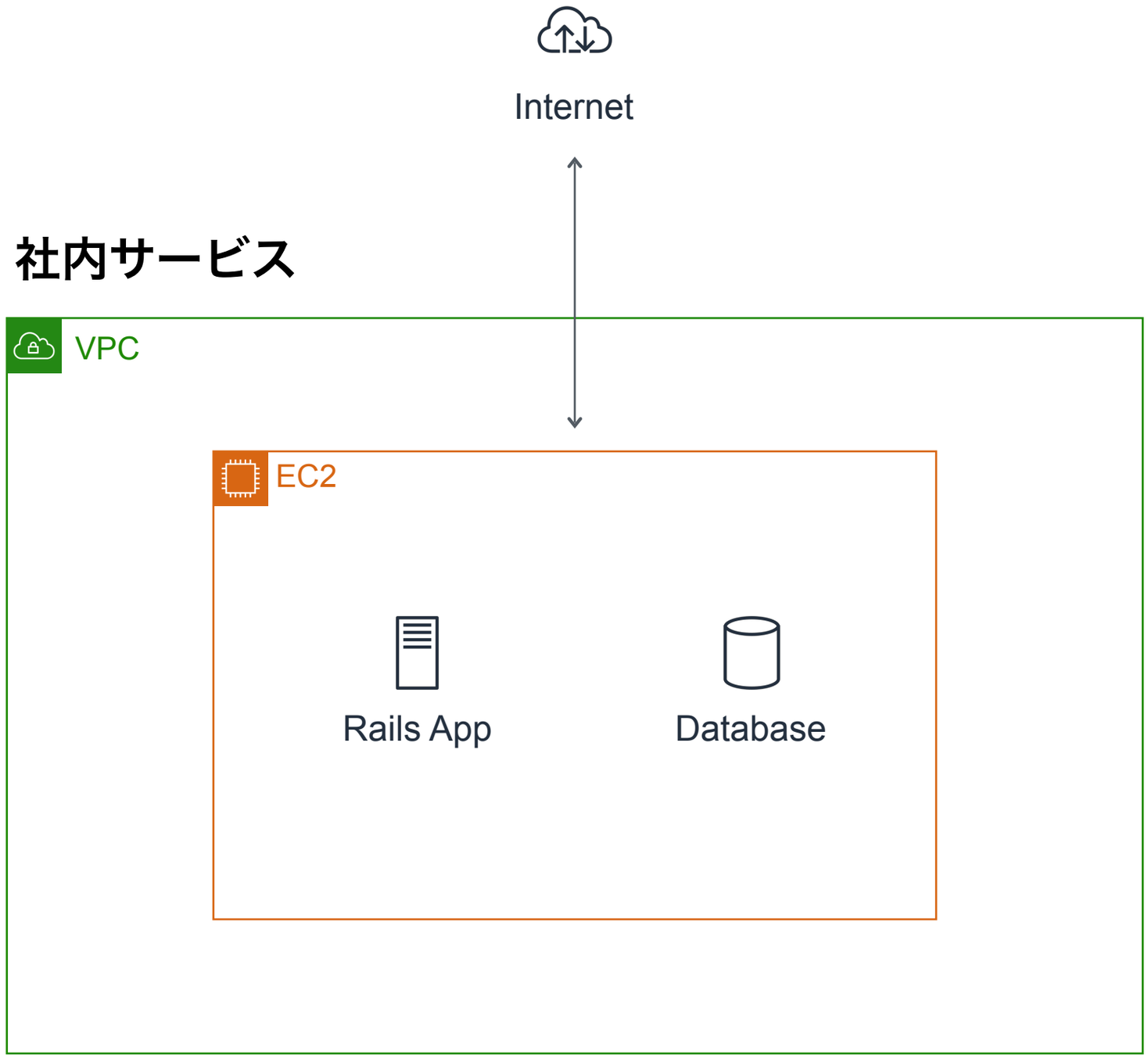
- 社内サービス、ホスティングサービスのインフラを今風のクラウドにしたい
- 新しいサービスの基盤を考える

古い構成

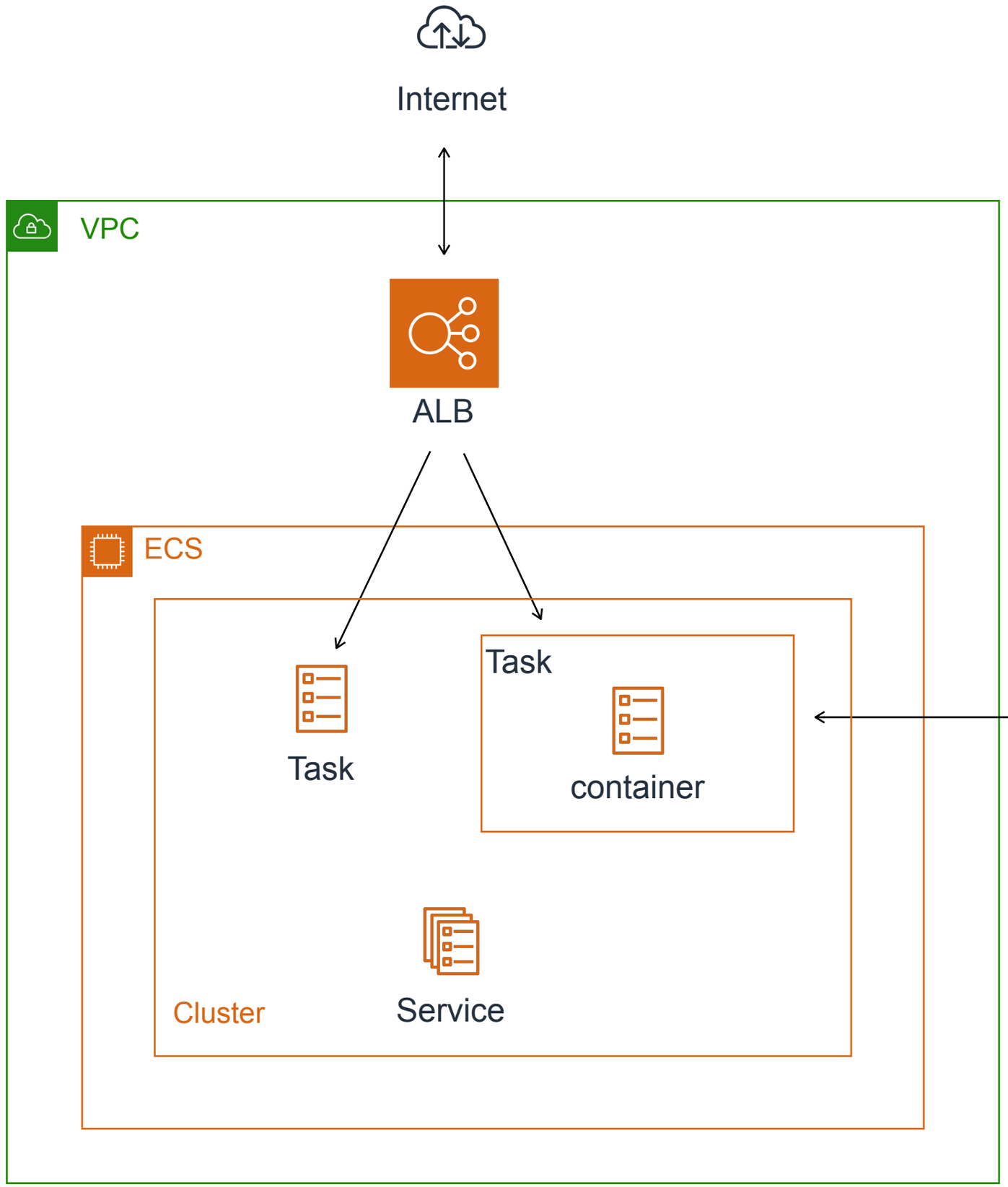
ホスティングサービス



サービス提供中のシステムなので詳細ぼやかし気味…


Internet
社内サービス
VPC
EC2
Rails App
Database

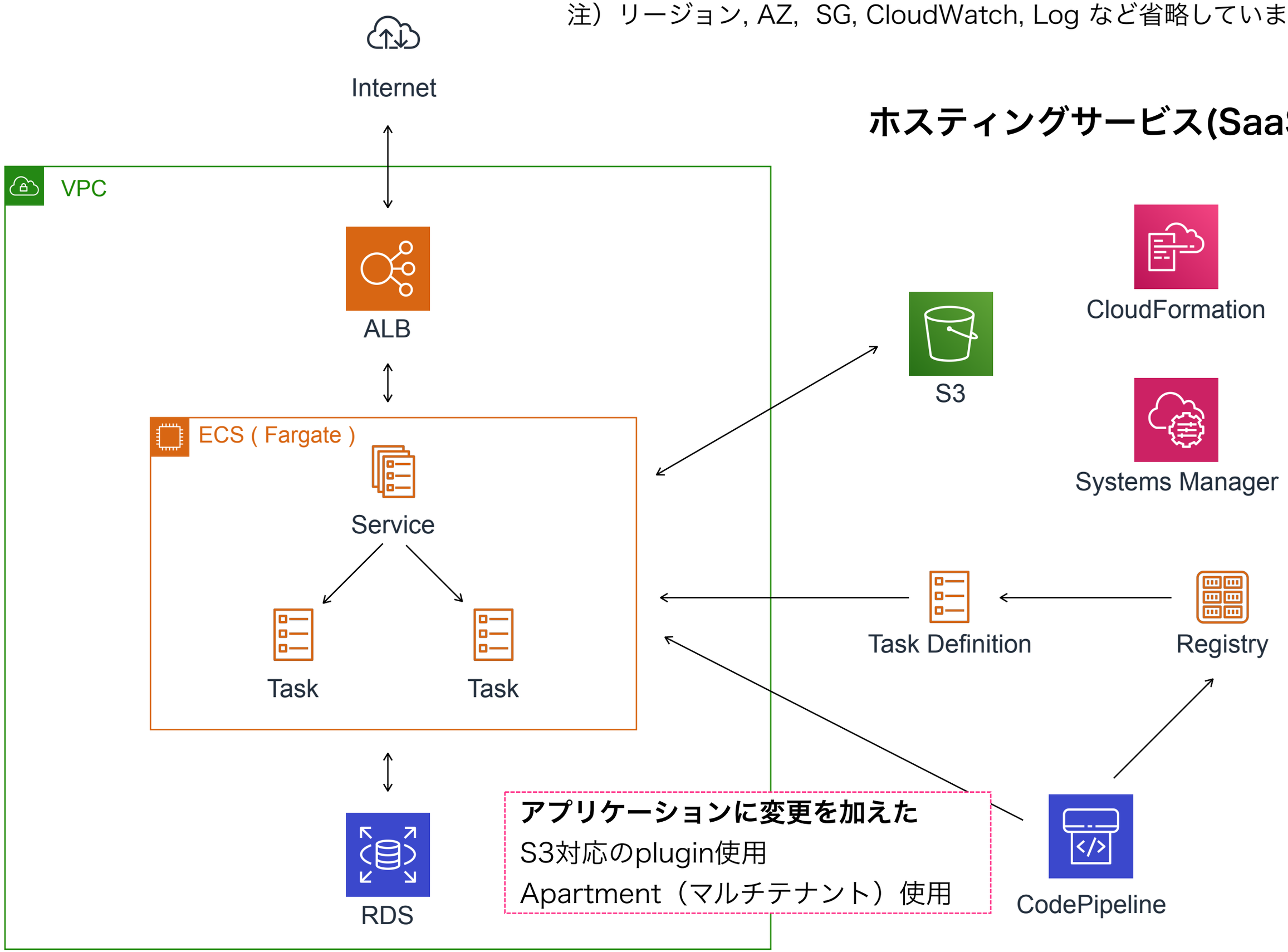
新しい構成



- EC2
  値段安い
  docker volume plugin使える
- Fargate
  フルマネージド
  ファイル永続化できない
  EC2にくらべ値段高め

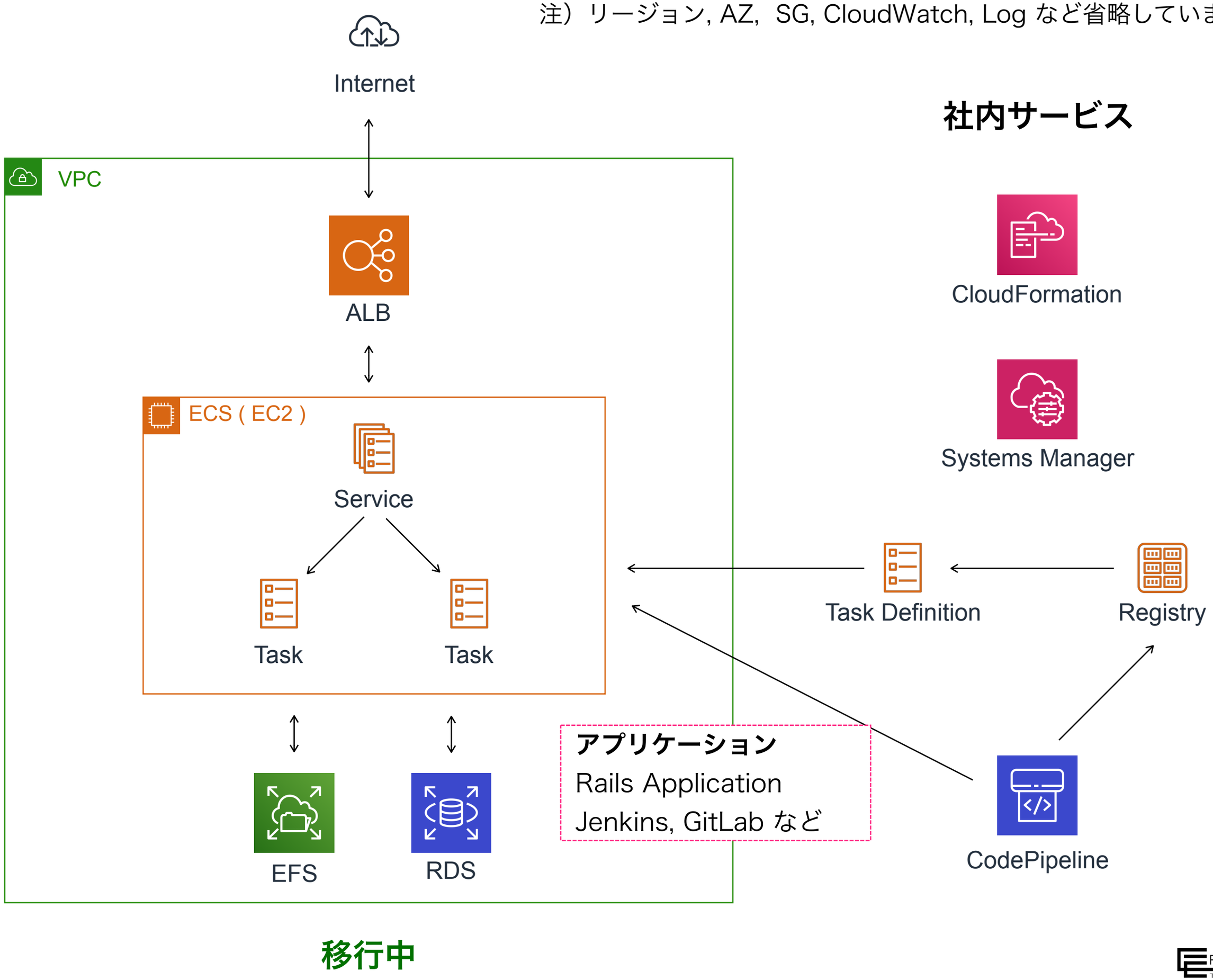
注）リージョン, AZ, SG, CloudWatch, Log など省略しています。

ホスティングサービス(SaaS)



※ 訳あって現在検証中

注）リージョン, AZ, SG, CloudWatch, Log など省略しています。



移行中

Point

1. ECSを選択（Dockerで本番環境）

- ECS + Pipelineの組み合わせで開発もサーバ管理もコスト軽減（心理的・時間的）

2. Pipelineの導入

- Pipeline便利！ AWSで完結できる！！

3. CloudFormationで環境構築

- 本番、ステージング、テスト環境の構築が簡単になった

4. Systems Manager利用

- Systems Manager 無料で便利！

5. Ruby(Rails) + コンテナ

1. ECSを選択
(コンテナで本番環境)

よかったこと
(ECS / コンテナ)

デプロイ時の心理的負担の軽減

- Blue/Green deployment
- すぐにロールバックできる安心感
- 本番環境と同じ環境で開発ができる

よかったこと
(ECS / コンテナ)

サーバ管理の負担軽減

- 冗長化、高可用性が簡単
    - AMIの管理も特に不要（EC2のAutoScaleのような
- 運用コストを軽減したい（EC2・デプロイの管理)
    - ミドルウェアの管理が容易にできる

よかったこと
(ECS / コンテナ)

Docker volume pluginに対応

(ストレージの永続化)

その他の選択について

EKSを利用しなかった理由

東京リージョン、技術面（学習コスト）、金額

2. Pipelineの導入

よかったこと

（デプロイ・テストの自動化）

1. デプロイのために、SSHでサーバに入らなくて良くなった。（さよならオレオレデプロイ）
2. テスト実行忘れ防止
3. Docker Imageを使うと簡単だった

通知

docker-compose

User

git push

CodeCommit

Jenkins
Test

Code
Pipeline

Source

buildspec.yml

実行

```yaml
version: 0.2

phases:
  set:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - $(aws ecr get-login --region ${AWS_DEFAULT_REGION} --no-include-email)
      - REPOSITORY_URI=${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_DEFAULT_REGION}.amazonaws.
      - COMMIT_HASH=$(echo ${CODEBUILD_RESOLVED_SOURCE_VERSION} | cut -c 1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
  build:
    commands:
      - echo Building the Docker image...
      - docker build -t ${REPOSITORY_URI}:latest .
      - docker tag ${REPOSITORY_URI}:latest ${REPOSITORY_URI}:${IMAGE_TAG}
  test:
    commands:
      - echo Run test
      - docker-compose run web bundle exec rspec
  post:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push ${REPOSITORY_URI}:latest
      - docker push ${REPOSITORY_URI}:${IMAGE_TAG}
      - echo Writing image definitions file...
      - echo "[{\"name\":\"${TASK_DEFINITION}\",\"imageUri\":\"${REPOSITORY_URI}:
artifacts:
    files: imagedefinitions.json
```

Build

CodeBuild

pre_build → build
→ post_build

ECR

TaskDefinition

Cluster

Service

Deploy

3. CloudFormationで環境構築

CloudFormation

AWSリソースを自動で構築できるサービス

テンプレート
設定ファイル

スタック
テンプレートを元に作成されるリソースを管理する単位

目的・理由

- 手順書とか書きたくない
- 環境の再現性の確保（別アカウントでも）
- aws-cli, ansible 組み合わせちょっとツライ
- YAMLで書けるようになってた
- 初期構築はこれで全てできる（Dockerの恩恵）

注意点

- スタックのネストについて
- テンプレートの再利用
- CFn以外（GUI, CLIなど手動）で更新すると壊れるかも
- 学習コスト高め

CloudFormation

VPC

VPC, Subnet, SG, etc…

ECS

ALB(rule, target, listener), Cluster,
AutoScale, EFS, IAM, Role, SG, etc…

主に追加・変更するスタック

Service1, 2 …

DNS, TaskDefinition, Service, AutoScale,
Target, Rule, Role, IAM,

RDS

RDS, Subnet, SG

4. Systems Managerの利用

目的・理由

- パスワード等の管理

- SSHログイン

5. Ruby(Rails) + コンテナ

- コンテナの起動は早いけど、Railsの起動は遅い
  - 起動時（entrypoint.sh）の処理をなるべく軽く
    - bundle install, migrationのタイミング
- メモリーの消費量
  - Rails マルチテナント化

課題

1. Rails アプリのデプロイ
2. 社内でのDockerの布教活動
3. 環境変数増やしすぎに注意
4. ロギング
5. セキュリティー

まとめ

Point

1. ECSを選択（Dockerで本番環境）

- ECS + Pipelineの組み合わせで開発もサーバ管理もコスト軽減（心理的・時間的）

2. Pipelineの導入

- Pipeline便利！ AWSで完結できる！！

3. CloudFormationで環境構築

Docker（コンテナ）の恩恵

- 本番、ステージング、テスト環境の構築が簡単になった

4. Systems Managerで運用管理

- Systems Manager 無料で便利！

5. Ruby(Rails) + コンテナ

- … そもそもコンテナに向いてないアプリ（システム）もある。

ご静聴ありがとうございました。