
# AKB48のモバイルサイトを 他社DCからAWSにECSを用いて移管した話

株式会社シーエー・モバイル　庭木 勝也　坂本 佳久

C.A.MOBILE

## AWS移管やECSの利用を検討する方向け

【庭木】

- 移管にあたっての技術の紹介とその選定理由
- DBの移管方法
- エンジニアの意識の変化

【坂本】

- ECSの概念とその活用
- コンテナ使用時のログ管理
- 弊社での具体的な設定例

# 目次

## 3ヶ月でECSを用いてAWS移管した方法をまるっと紹介


会社紹介


AKB48
モバイルサイトの移管


利用技術の選定


DBの移管


Webサービスの
ECS利用


まとめ

# INTRODUCTION
# 自己紹介

**Software Engineer Manager**
**Corporate Development Manager**

**庭木 勝也 @にわっち**

2012年 新卒入社

DC運用サービスのインフラ担当として
OpenStack導入や、DC間のサービスの
移管にも従事

2017年7月以降は、社内システムも合わせ
てエクセルからAWSまで管理
現在は、AWS移管に注力

1

**会社紹介**

2

AKB48
モバイルサイトの移管

3

利用技術の選定

4

DBの移管

5

Webサービスの
ECS利用

6

まとめ

# シーエー・モバイルが展開する事業


インターネット広告・動画広告事業

DSPではなくてSRPの広告事業を展開


占い・エンタメ・ライフスタイル

分野のWebサービスを提供

# 技術に明るい取締役とCTOが技術を引っ張る

取締役
エンジニアの母 (VP of Engineering)
**齋藤 匠**

CTO
**船ヶ山 慶**

# エンジニアブログと採用サイトの紹介

C.A.MOBILE
TECH BLOG

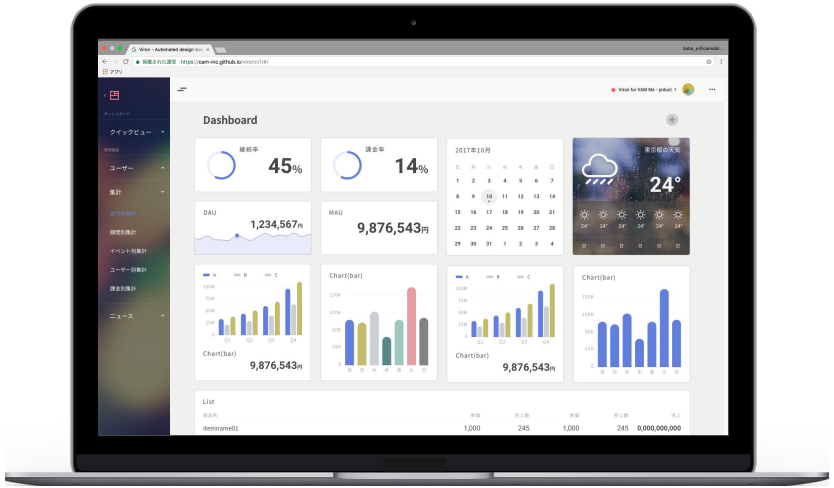

## エンジニアブログ

http://tech.camobile.com/

## 採用サイト

https://hr.camobile.com/

# シーエー・モバイルの大型OSS




Viron

統一した操作性・ダッシュボードを備えた

**管理ツールプロダクト**

## 参考資料

GitHub : https://github.com/cam-inc/viron

エンジニアブログ : http://tech.camobile.com/entry/viron_20180201

## シーエー・モバイルの開発スタイルのコンセプト

# いつでもどこでも
# 同じ環境でサービスの開発ができる

## シーエー・モバイルの開発スタイルのコンセプトを実現するために


# 自社DCからパブリッククラウドへ移行中

# 会社紹介

## 現在シーエー・モバイルで利用している技術



















1 会社紹介

2 AKB48
モバイルサイトの移管

3 利用技術の選定

4 DBの移管

5 Webサービスの
ECS利用

6 まとめ

# AKB48モバイルサイトの移管

## AKB48のモバイルサイトの移管について


**期間**

3ヶ月

**メンテナンス時間**

8時間 (24:00-8:00)を2回

**ステークホルダー**

AKS様・ドメインSSL管理会社・他社システム会社

**付属サービス**

FP・SP・ネイティブアプリ・メールサービス・検閲

**CAMOBILE**

インフラ2名・サーバー6名・アプリ2名

1 会社紹介

2 AKB48
モバイルサイトの移管

3 利用技術の選定

4 DBの移管

5 Webサービスの
ECS利用

6 まとめ

# 利用技術の選定

## AKB48のモバイルサイト移管後の利用技術


独自フレームワーク

アプリケーション



開発環境・コード管理


環境構築・構成管理

CI・バッチ

ログの管理


メール送受信・配信


監視・外形監視

bot ・通知・コミュニケーション


AWS上のサービス

Amazon RDS

Amazon S3

Amazon ECS


Memcached

# 利用技術の選定

## アプリケーション



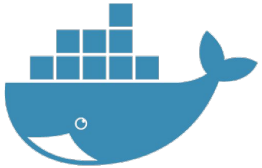
独自フレームワーク

3ヶ月という時間の制約のため、ミドルウェア自体は変更せず、バージョンアップのみ実施

- apache 2.4系
- php 5.4系

# 利用技術の選定

## 開発環境・コード管理



Docker

- 本番と同一の構成でローカル環境を作れる
- コンテナなので起動がはやい
- DockerComposeでグループ化してコンテナを起動可能


- 外部サービスとの連携可能
- Github Enterprise のようにサーバーの管理・運用コストがない

# 利用技術の選定

## 環境構築と構成管理

HashiCorp Terraform


- 社内で利用実績があった
- コードドリブンで構成管理可能
- 手動構築を最小限に
- YAML形式の設定を作成するだけで構成管理が可能
- githubで管理可能

# 利用技術の選定

## CI・バッジ

Jenkins

- 社内で利用実績が豊富
- デプロイの内容をgithubで管理可能
- botからの実行も可能
- オペレーションの統一化が可能
- デプロイ・バッチの一元管理が可能

# 利用技術の選定

## ログの管理


- コンテナの利用が可能
- ログの集約が可能
- ログのフォーマットやタグの指定が可能
- S3やBigQueryなど出力先を柔軟に指定可能


- ログの保存に利用
- BIツールと連携し可視化可能
- SQLをキャッシュしてくれる
- 低コスト
- google analyticsと連携可能

# 利用技術の選定

## メール送受信・配信


- 会員登録などの空メール送信に利用
- WEB API を利用したメール送信
- メール受信をフックしてメールの内容を POST する機能
- Freeプランでもメール受信可能


- オンプレ環境での利用実績あり
- 国内でのキャリアへの到達率がよい
- メルマガの大量配信に利用

## 外形監視


https://www.statuscake.com/

## Statuscakeを利用した理由

- タグをベースで監視設定を一括変更可能
- 東京リージョンもある
- プロトコルごとに監視可能(HTTP,HTTPS,TCP,UDP等)
- 死活監視やパフォーマンス監視が可能
- 1アカウントで複数のサイトを監視可能
- モニタリングするサーバーを構築、運用するコストが不要
- slack インテグレーションでslackにアラート
- リクエストページのString Matchが可能
- 費用が安い(Business plan $80)

## 監視


https://www.datadoghq.com

## DataDogを利用した理由

- モニタリングするサーバーを構築、運用するコストが不要
- タグで管理が可能
- AWS インテグレーションで CloudWatch のメトリクスでアラート設定が可能
- Datadog のエージェント (dd-agent) がコンテナで提供されている
- AutoDiscovery で動的ポートで起動しているコンテナも監視可能
- グラフのみやすさ
- dogpushでmonitor、dogshell で dashboard を生成
- slack インテグレーションでslackにアラート
- EC2 コンテナインスタンスにエージェントを入れていると10コンテナまでの監視が無料 (https://docs.datadoghq.com/ja/guides/billing/)

# 利用技術の選定

## bot・通知・コミュニケーション


- 容易に導入が可能
- 様々なチャットツールに対応

- インテグレーションが豊富
  - datadog
  - statuscake
  - hubot
  - jenkins


会社紹介


AKB48
モバイルサイトの移管

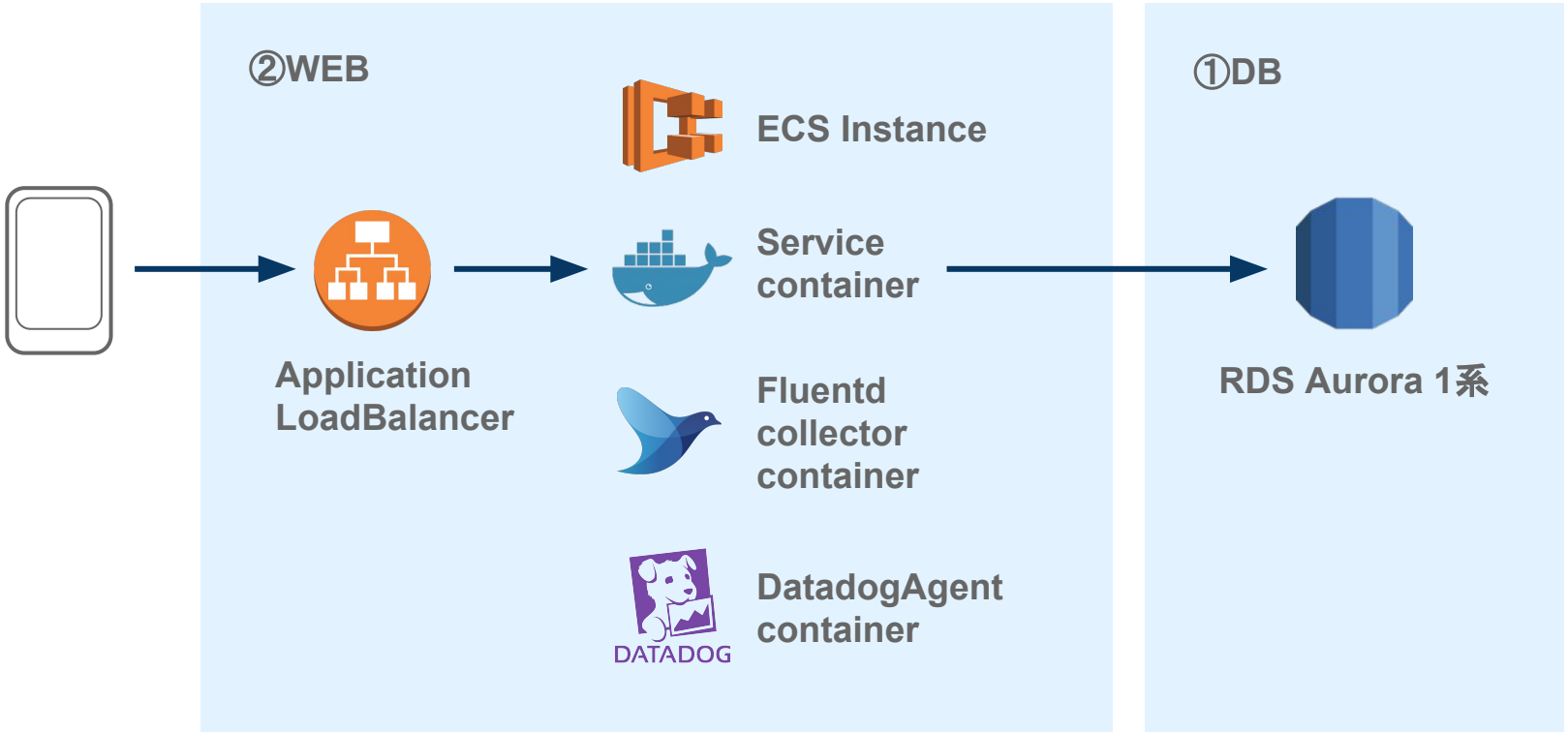

利用技術の選定


**DBの移管**


Webサービスの
ECS利用


まとめ

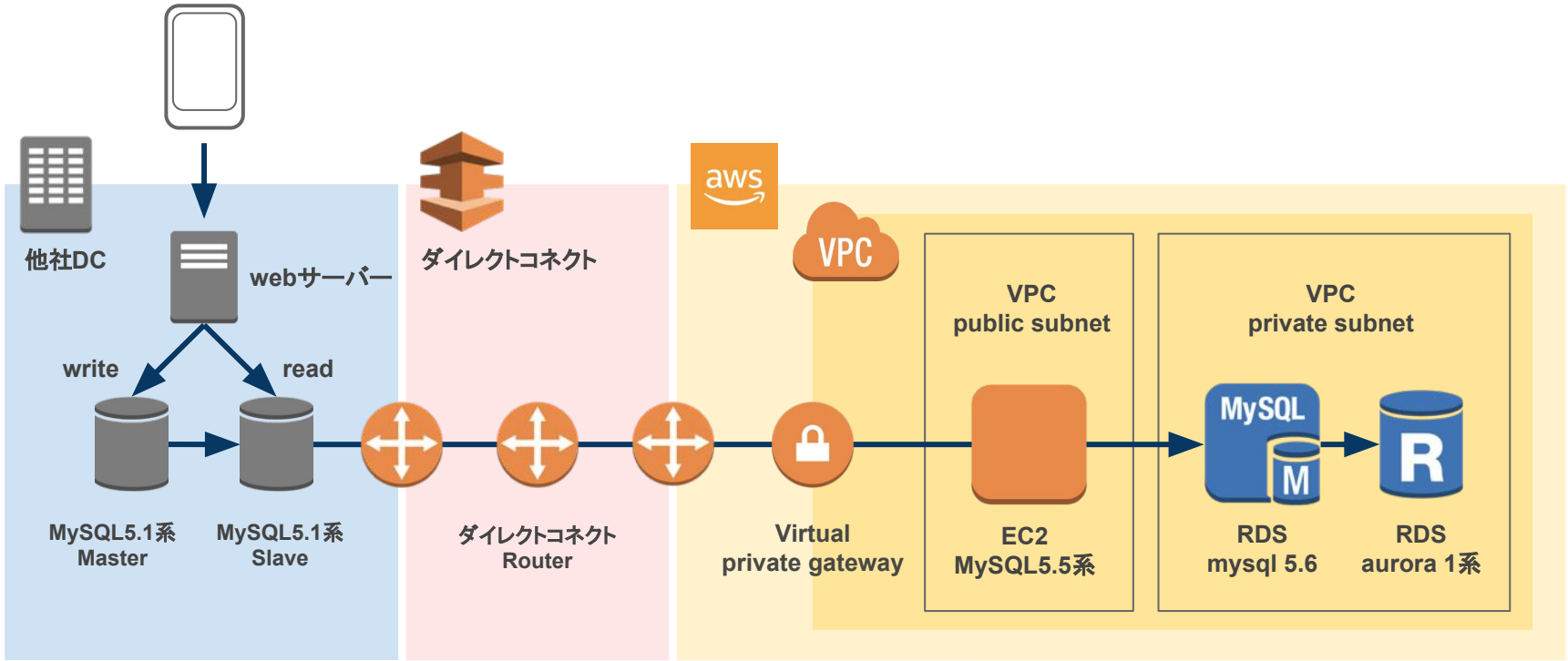
# 全体構成図


②WEB
ECS Instance
Service
container
Application
LoadBalancer
Fluentd
collector
container
DatadogAgent
container
DATADOG
①DB
RDS Aurora 1系

## 3つのフェーズに分けて移管を実施

1. DB先行移管の準備
2. DBのみ先行で移管
3. WEB、DBすべて移管

# 1.DB先行移管の準備


他社DC
webサーバー
write
read
MySQL5.1系
Master
MySQL5.1系
Slave
ダイレクトコネクト
ダイレクトコネクト
Router
aws
VPC
Virtual
private gateway
VPC
public subnet
EC2
MySQL5.5系
VPC
private subnet
MySQL
M
R
RDS
mysql 5.6
RDS
aurora 1系

## ダイレクトコネクトを利用した理由

- mysqlの5.1系からaurora 1系まで多段でレプリケーションを組むことで、メンテナンス時間を短縮したい
- DBのみを先行で移管したい

### よかったこと

- オンプレミスの環境よりも構築とバックアップが容易
- インスタンスタイプを一時的に大きくすることで、データの転送、レプリケーションの作成時間を短縮
- EC2 → EC2 c4.large 2000iops (10GB を7mでimport)
- EC2 → aurora r4.large（10GBを45分でimport）

# ダイレクトコネクト以外の移管方法と検証

**【ケース1】他社DCにmysql proxy サーバーの構築と利用**

- privateサブネットにAuroraを立てるため、nginx等をパブリックサブネットに立てる必要あり
- 暗号化する必要あり（phpのバージョンが非対応））

**【ケース2】他社DCインスタンスとEC2インスタンス間でSSH トンネルを用いたレプリケーション**

- インターネット経由は不安定で再レプリケーション設定の作業が発生する場合がある

**【ケース3】AWS DMS の検証**

- auto incrementがなくなる等の制約あり

**参考資料**

https://docs.aws.amazon.com/ja_jp/dms/latest/userguide/CHAP_Source.MySQL.html

# EC2上のmysqlとAuroraの設定

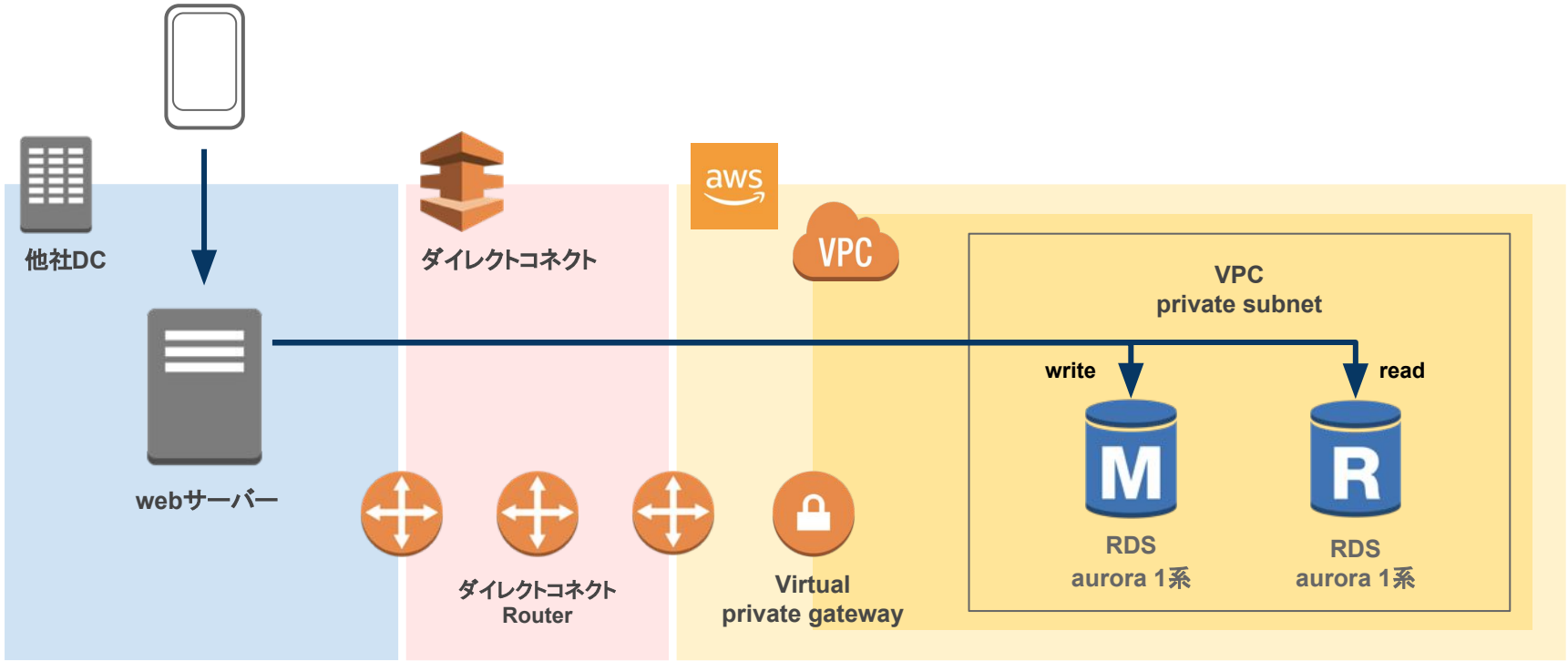
## EC2 mysql

- max_allow_packetを大きく128MB
- old_passwordsを無効
  - レプリケーション用のユーザー作成に影響
- replicate-ignore-db = mysql, replicate-wild-ignore-table = mysql.%
  - レプリケーションが停止する可能性があるため、ユーザーデータのレプリケーションは実施しない

## Aurora

- max_allow_packetを大きく128MB
- 文字コードを変更 EC2と同様utf8に設定
- log_output =File
- 拡張モニタリングを有効

# 2. DBのみ先行で移管


他社DC
webサーバー
ダイレクトコネクト
ダイレクトコネクト
Router
aws
VPC
Virtual
private gateway
VPC
private subnet
write
read
M
R
RDS
aurora 1系
RDS
aurora 1系

## DBの先行移管の際に実施したこと

- 旧マスターDB(MySQL 5.1系)とAuroraの間で差分が発生しないように、iptablesで、webサーバーから旧マスターDBに対して3306 portを利用したTCP接続を拒否

- テーブルの文字コード、テーブル数、テーブルの行数を、確認するスクリプトを作成し、差分確認を実施

# 他社DCからAuroraへのアクセス

- Auroraのcluster(writer)とreaderのエンドポイントに対して、
  他社DCのwebサーバーから引けるように、R53のpublic dns側にttlを短めに設定
- 接続の切り分けとセキュリティを考慮し、ユーザーも3つ新規に作成
  - reader（selectのみの権限）
  - user（insert,alter等の権限）
  - admin（すべての権限）
- 事前に、他社DC側のwebサーバーから接続テストを実施

**DBの先行移管時は、他社DC内のwebサーバーに登録されているDBの接続情報(akb48db)をR53に登録したAuroraの接続情報（aws-akb48db）へ変更**

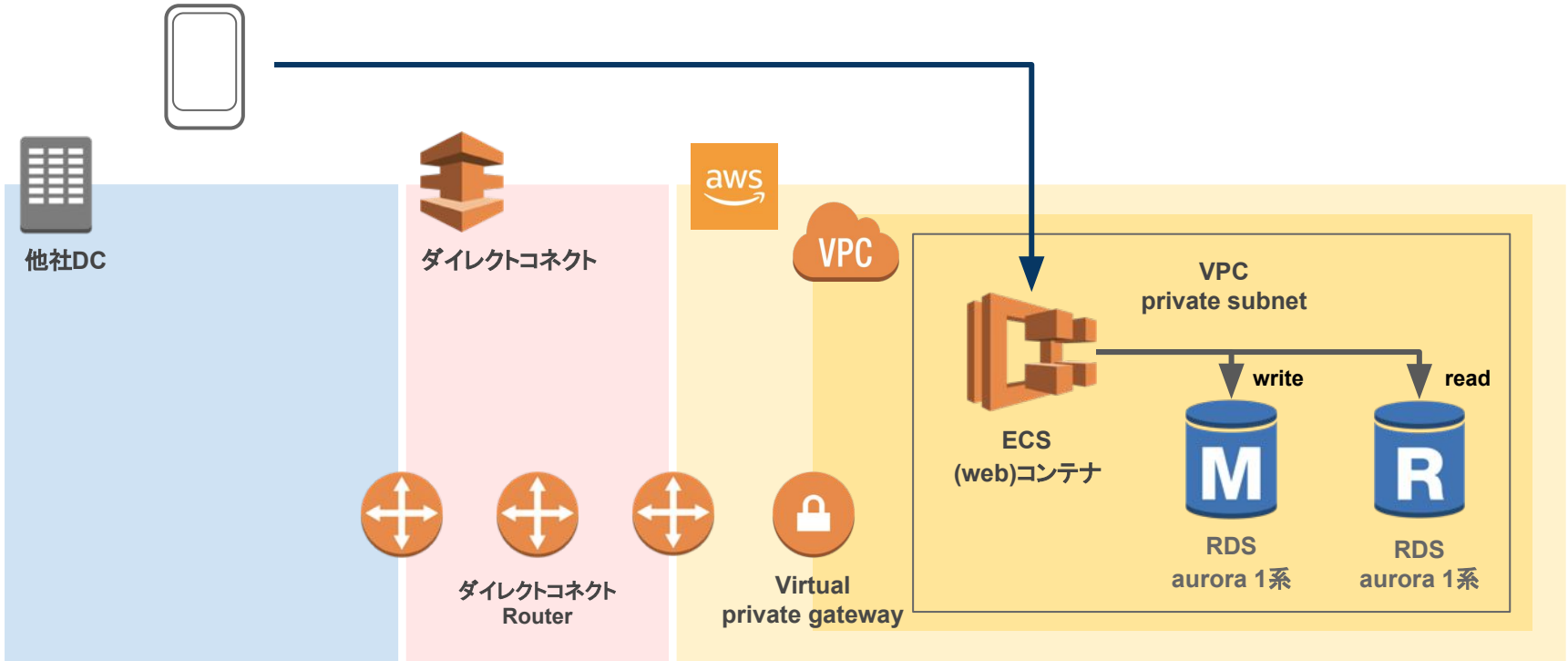
## Auroraへの接続サーバーの特定方法

- VPCのサブネットの設定でフローログを出力
- 対象のネットワークインターフェースのログをCloudWatchで確認
- 他社DCのネットワークサブネットでフィルター

### よかったこと

移管対象外と思われていた他社DC内共通サーバーの発見と、必要可否の精査ができた

# 3. WEB、DBすべて移管



# Aurora の問題発生時の対処方法

**問題のあるインスタンスを削除し、再作成すると即時復旧が可能**

問題の解決に必要な情報を、バッチでS3に定期的に保存する

- SHOW PROCESSLIST
- エラーログ、もしあればスロークエリログ
- 正常時と事象発生時の SHOW ENGINE INNODB STATUS\G の結果
- information_schema の innodb_trx, innodb_locks, innodb_lock_waits 等の情報
- （オプション）拡張モニタリングの有効化

※再起動、インスタンスタイプ変更をするとエラーログ等は消失する

# INTRODUCTION
# 自己紹介

## Site Reliability Engineer
## 坂本　佳久

2015年 中途入社

ネットワークやサーバーの知識を活かした環境設計や構築、トラブルシューティングが強み。

現在はサーバーサイドエンジニアとして他のサービス開発に従事。

## AWS移管やECSの利用を検討する方向け

【庭木】

- 移管にあたっての技術の紹介とその選定理由
- DBの移管方法
- エンジニアの意識の変化

【坂本】

- ECSの概念とその活用
- コンテナ使用時のログ管理
- 弊社での具体的な設定例

1

会社紹介

2

AKB48
モバイルサイトの移管

3

利用技術の選定

4

DBの移管

5

**Webサービスの
ECS利用**

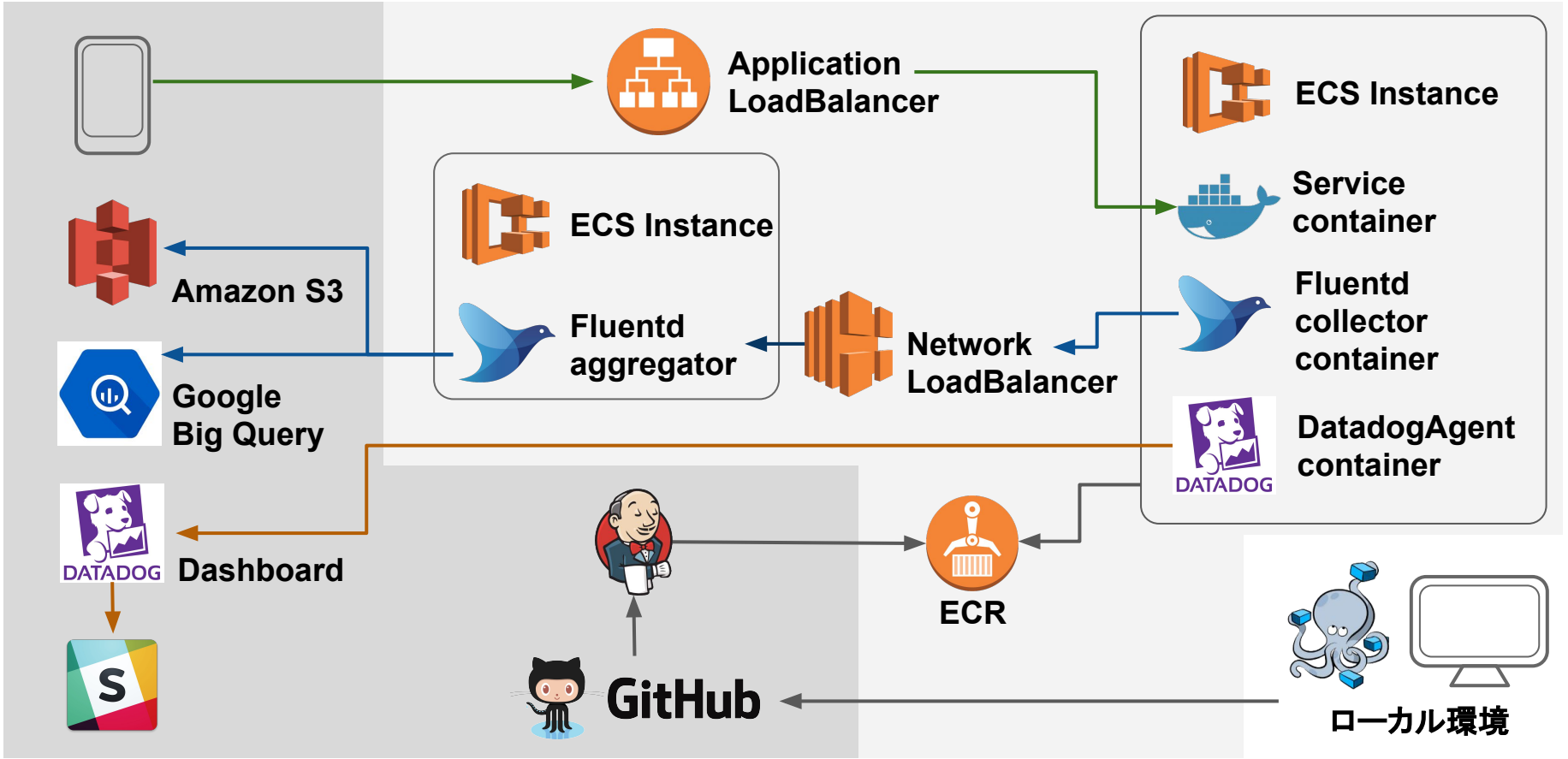
6

まとめ

# 目次

# 1 システム構成

# 1 システム構成


Application
LoadBalancer
ECS Instance
Service
container
Fluentd
collector
container
DatadogAgent
container
ECS Instance
Fluentd
aggregator
Network
LoadBalancer
Amazon S3
Google
Big Query
DATADOG
Dashboard
ECR
GitHub
ローカル環境

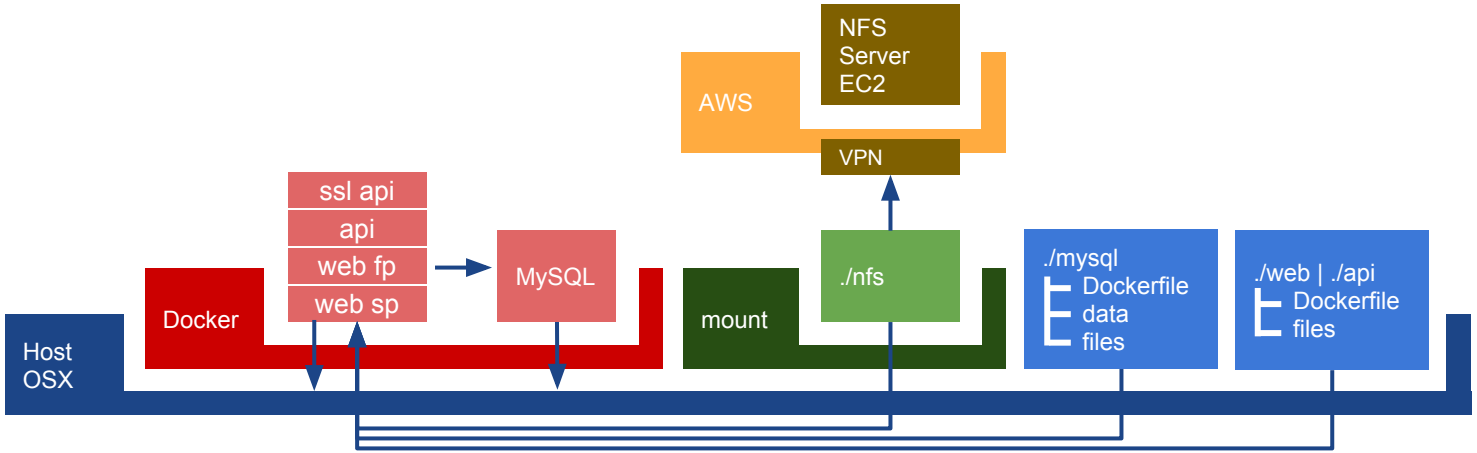
# 2 ローカル環境

## DockerとDockerComposeを使用

### 機能ごとに15個のコンテナを用意

- FP
- SP
- メールサービス
- 検閲
- MySQL
- etc...

# ローカル環境構成




- mac OS 上に各コンテナを起動させて、開発中のソースコードを
  ボリュームマウントして、変更したソースコードを即時反映させる。
- Docker composeを利用して、開発に必要なコンテナを複数起動する。
- NFSで開発に必要なデータをVPN越しにマウントして利用。

# 3 コンテナイメージ

## ソースコードを実行する際の OS は AmazonLinux

- AWS で管理しているパッケージを使用できるなど、AWS の恩恵を受けることができる
- Alpine や Debian は慣れていない人が多い
- コンパイルのみを実施する場合やgolang製のアプリなど、用途に合わせてalpineやscratchなどを使用

## AmazonLinux のイメージ

**DockerHub / OFFICIAL REPOSITORY amazonlinux**
https://hub.docker.com/_/amazonlinux/

## Alpine linux

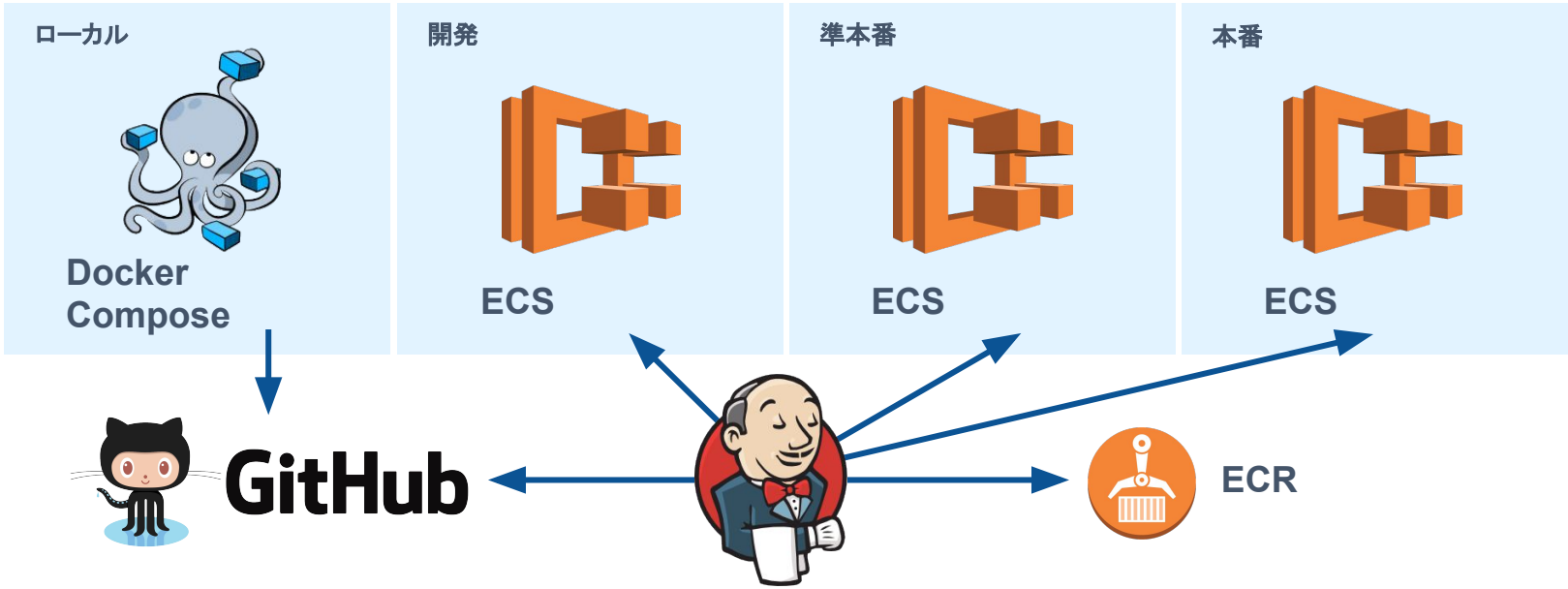
https://alpinelinux.org/

## scratch

**docker docs / Create a base image**
https://docs.docker.com/develop/develop-images/baseimages/

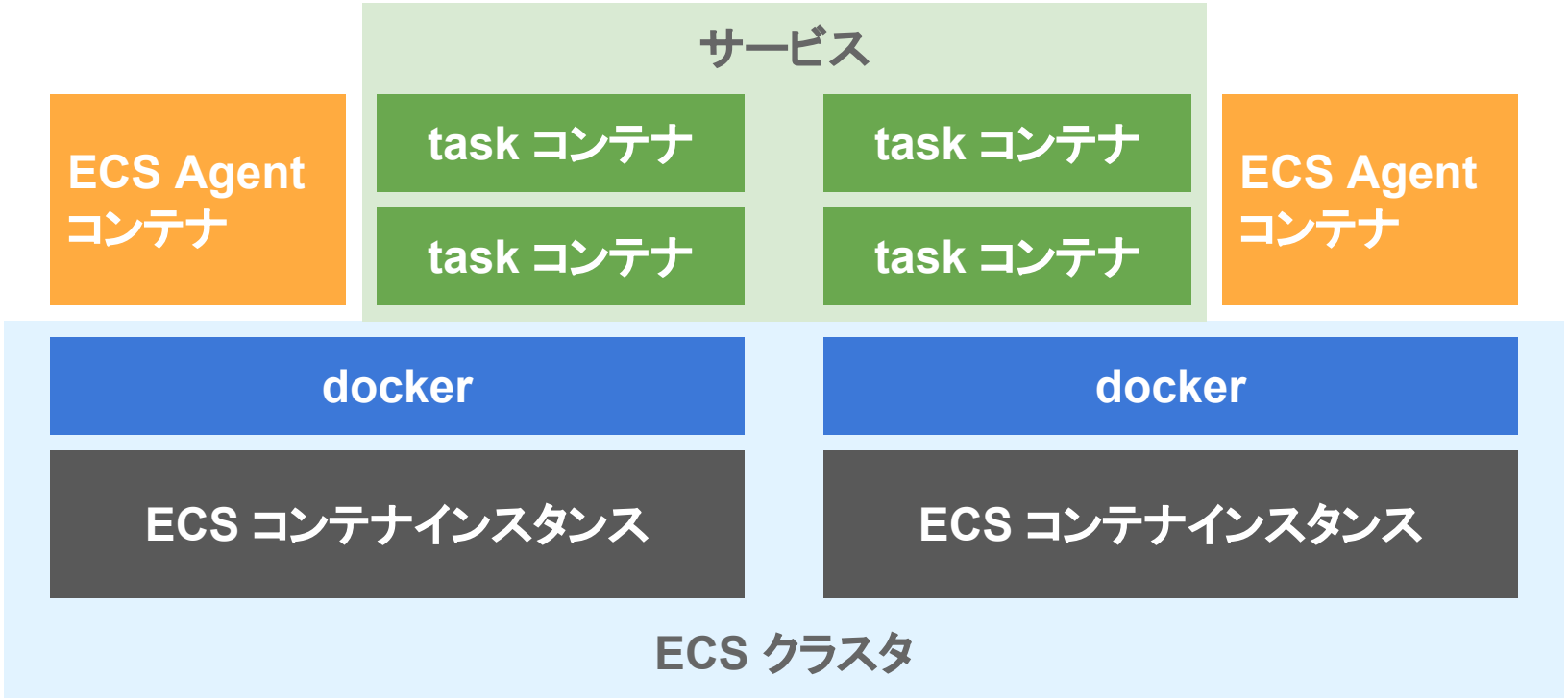
# 4 アプリケーションデプロイ

# 各環境へリリース

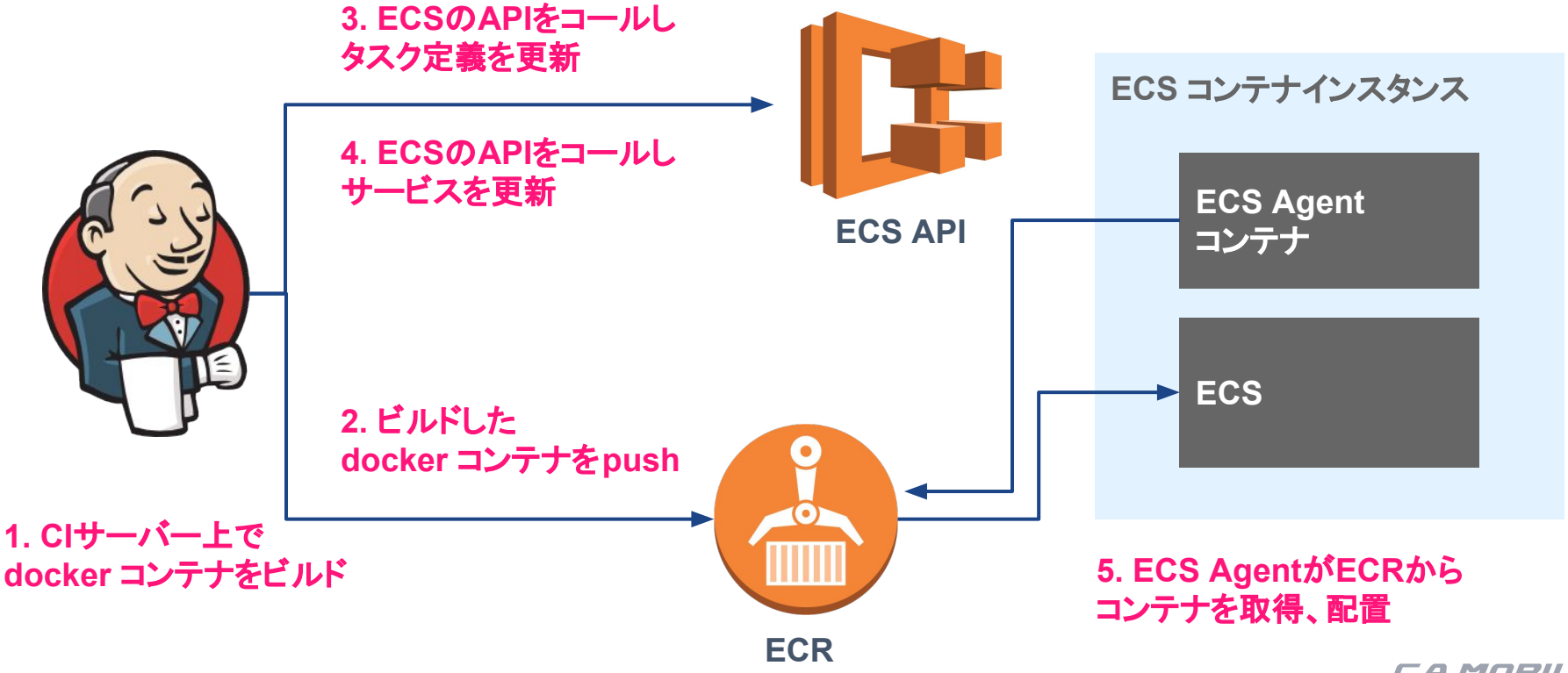



- コンテナ化することで、ローカル環境をそのままリリース

## ECSの構成


サービス
ECS Agent コンテナ
task コンテナ
task コンテナ
task コンテナ
task コンテナ
ECS Agent コンテナ
docker
docker
ECS コンテナインスタンス
ECS コンテナインスタンス
ECS クラスタ

# デプロイ時の動作


3. ECSのAPIをコールし
タスク定義を更新
4. ECSのAPIをコールし
サービスを更新
ECS API
ECS コンテナインスタンス
ECS Agent
コンテナ
ECS
2. ビルドした
docker コンテナをpush
1. CIサーバー上で
docker コンテナをビルド
ECR
5. ECS AgentがECRから
コンテナを取得、配置

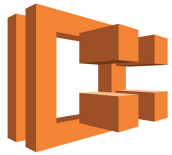
# 4 アプリケーションデプロイ

## slackを使用してdeploy

slack


hubot

jenkins



ECS

## 実行例

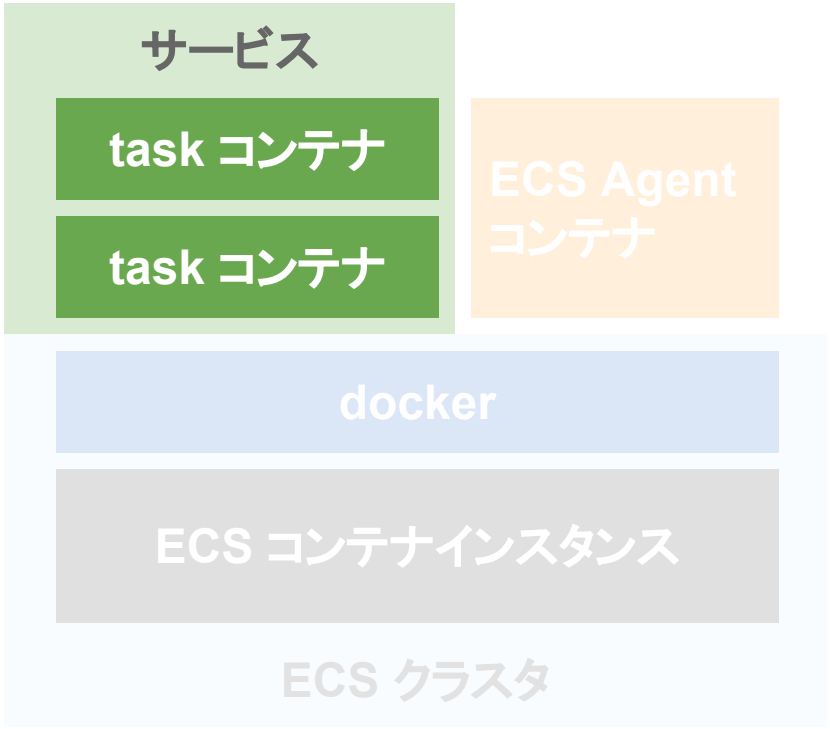
## タスクの配置とデプロイオプション

### 戦略

- spread
  - ecs.availability-zone
  - instanceId

### デプロイオプション

- 最大率 130%
- 最小率 60%



## アプリケーションデプロイ時の事象

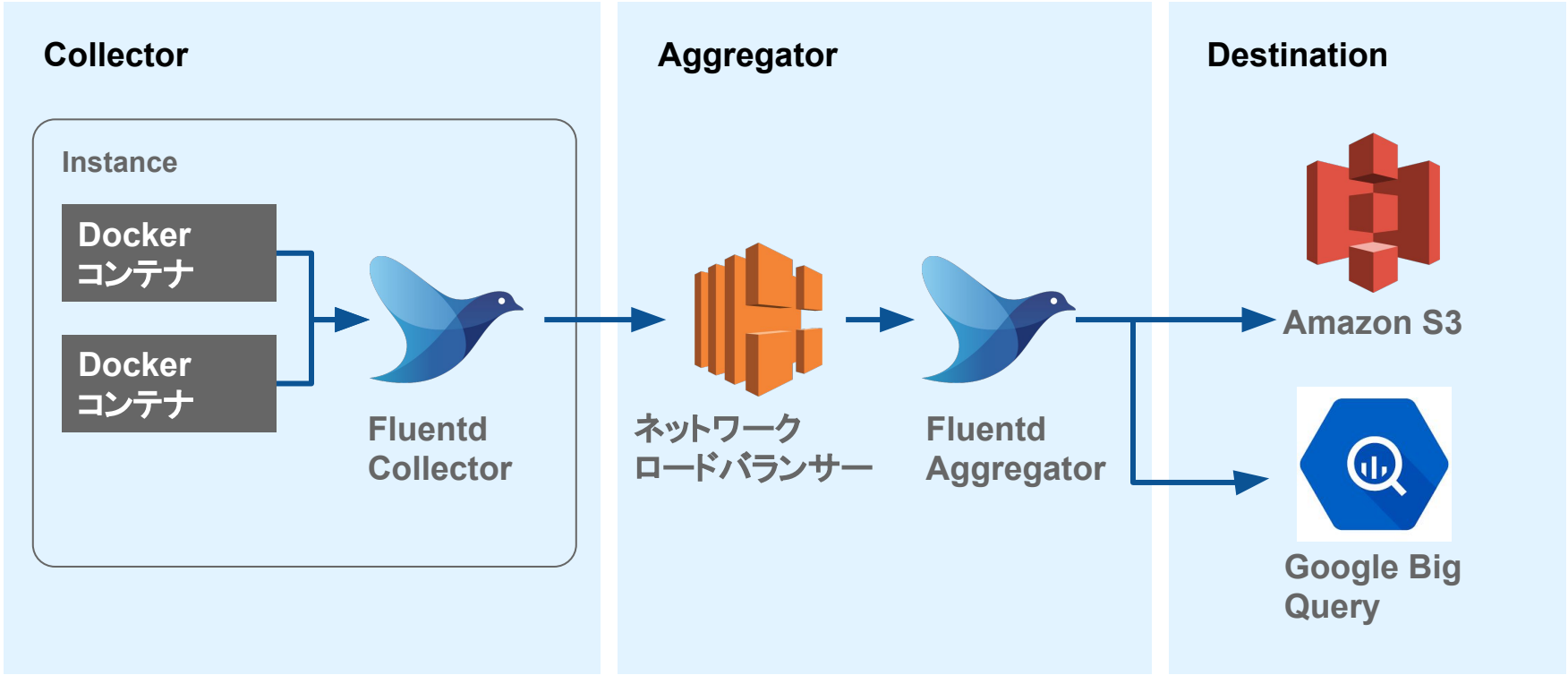
- ローリングアップデートに時間がかかる
- アプリケーションロードバランサーのターゲットグループの登録解除の遅延時間が長い
- 登録解除の遅延時間を10秒へ変更
（デフォルト300秒）

参考資料

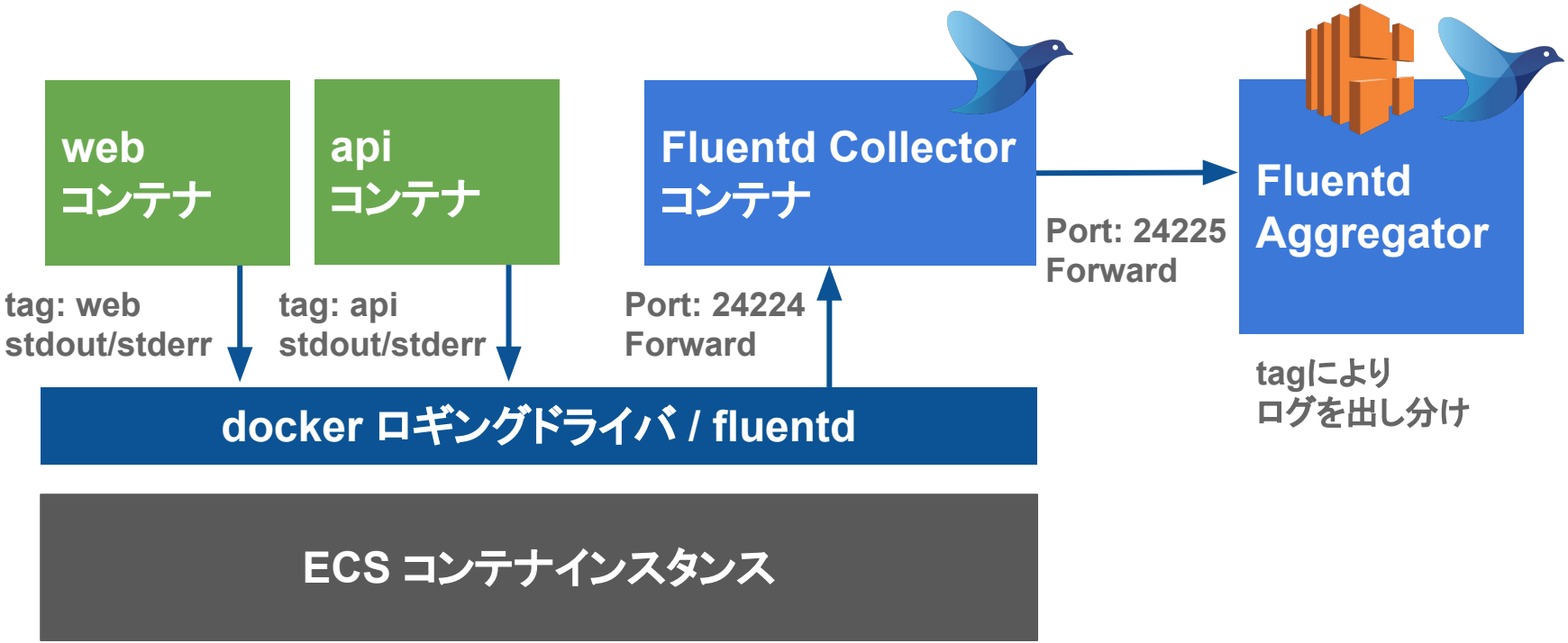
https://docs.aws.amazon.com/ja_jp/elasticloadbalancing/latest/application/load-balancer-target-groups.html#target-group-attributes

# 5 ログ管理

# Fluentdの構成


Collector
Aggregator
Destination
Instance
Docker
コンテナ
Docker
コンテナ
Fluentd
Collector
ネットワーク
ロードバランサー
Fluentd
Aggregator
Amazon S3
Google Big
Query

# Fluentd collectorがログを転送する流れ



# コンテナインスタンスの中にFluentd collectorを立てている理由

- Aggregatorとの通信時に問題があった場合、
  ログを確認できる。
- Fluentd Collectorが使用しているCPU，メモリのリソース、ログ転送状況が監視できる。
- ログフォーマットやタグなどの設定を変更したいことがあった場合、対応可能。

参考資料

The Patterns of Distributed Logging and Containers / SATOSHI TAGOMORI
https://www.slideshare.net/tagomoris/the-patterns-of-distributed-logging-and-containers

## 6 ECSでFluentdを起動

## Fluentd collectorの要件

- コンテナインスタンス1台につき、fluentdのコンテナを一つだけ起動する必要がある
- Fluentd Aggregator にログ転送をさせる

## 解決方法

- AutoScalingグループの起動設定のuser-dataを使用
- user-data 内でコンテナインスタンスの/etc/rc.localにコンテナ起動コマンドを記載するよう設定

コンテナインスタンス起動時に、必ず1つの Fluentdコンテナが起動する

# 起動設定抜粋 / CloudFormation

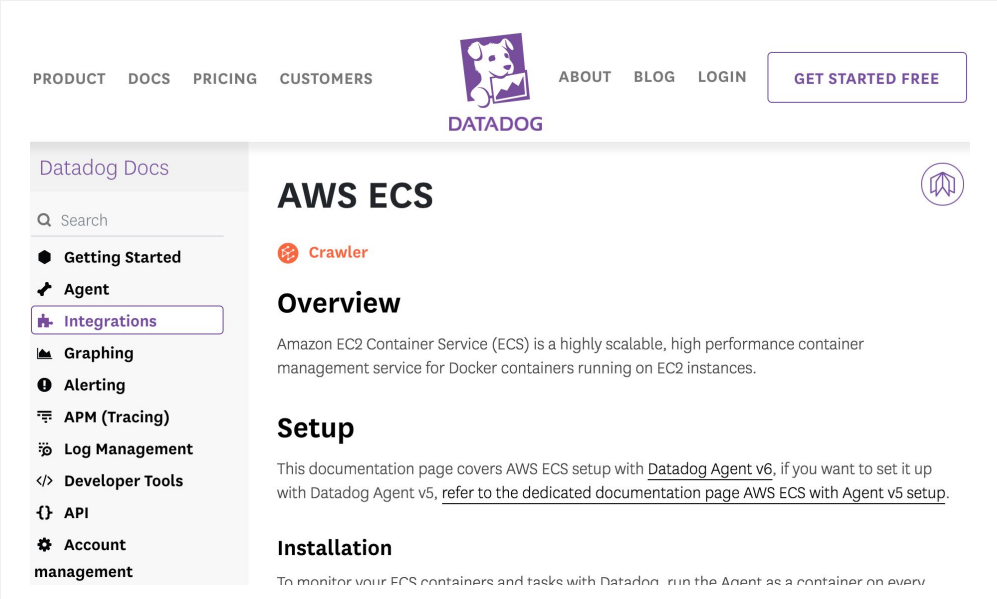
```
UserData:
  -
    Fn::Base64: !Sub |
      #!/bin/bash
      echo ECS_CLUSTER=${EcsCluster} >> /etc/ecs/ecs.config
      echo 'ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs","fluentd"]' >> /etc/ecs/ecs.config
      export PATH=/usr/local/bin:$PATH

      cluster=${EcsCluster}
      td_agent_task_def=${CollectorName}
      start ecs
      yum install -y aws-cli jq curl
      aws configure set default.region ap-northeast-1
      function get_instance_arn {
        curl -s http://localhost:51678/v1/metadata | jq -r '. | .ContainerInstanceArn' | awk -F/ '{print $NF}'
      }
      function get_az {
        curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone
      }

      instance_arn=$(get_instance_arn)
      az=$(get_az)
      region=${!az:0:${!#az} - 1}

      echo "
      cluster=${!cluster}
      az=${!az}
      region=${!region}
      aws ecs start-task --cluster ${!cluster} --task-definition ${!td_agent_task_def} --container-instances ${!instance_arn} --region ${!region}" >> /etc/rc.local
```

ログドライバに fluentd を指定

user-dataからインスタンスARNなどの必要な情報を取得

rc.localにコンテナ起動コマンドを書き込み

# 6 ECSでFluentdを起動




## 参考資料

**Datadog / AWS ECS Integration**
https://docs.datadoghq.com/integrations/amazon_ecs/

## タスク定義抜粋

アプリケーションコンテナのlogDriver設定

```
"logConfiguration": {
    "logDriver": "fluentd",
    "options": {
        "fluentd-address": "localhost:24224",
        "tag": "xxxx"
    }
},
```

## Fluentd Aggregator の要件

- S3などへ転送するだけでなく、ローカルディスクにもログを書き込む
- 負荷によってスケールイン、スケールアウトさせたい
- Fluentd Collectorの設定はなるべく変更せずFluentd Aggregatorの設定で吸収したい

## 解決方法

- ECSインスタンスをボリュームマウントし、ローカルディスクに書き込めるようにする
- ECSのサービスとしてFluentd Aggregatorを設定しロードバランサーで分散
- ローカルディスクに書き込むため、タスクの配置制約とデプロイオプションを、1コンテナインスタンスに、1コンテナが起動するように設定

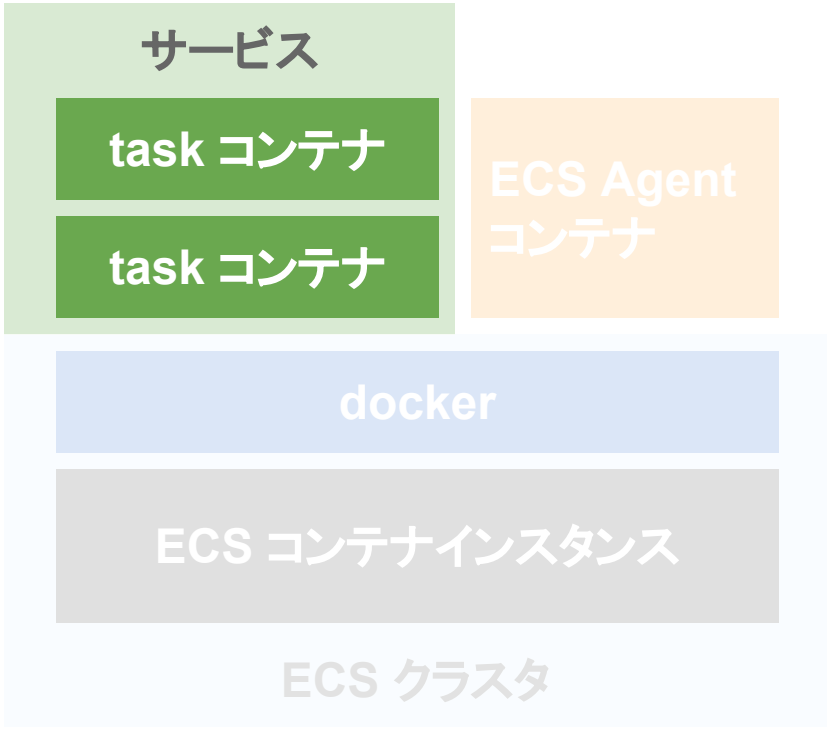
## タスクの配置とデプロイオプション

**制約**

- distinctInstance

**デプロイオプション**

- 最大率 100%
- 最小率 50%



## モニタリング

### Fluentd

- Monitoring Agent Input Plugin

### Datadog

- Autodiscovery

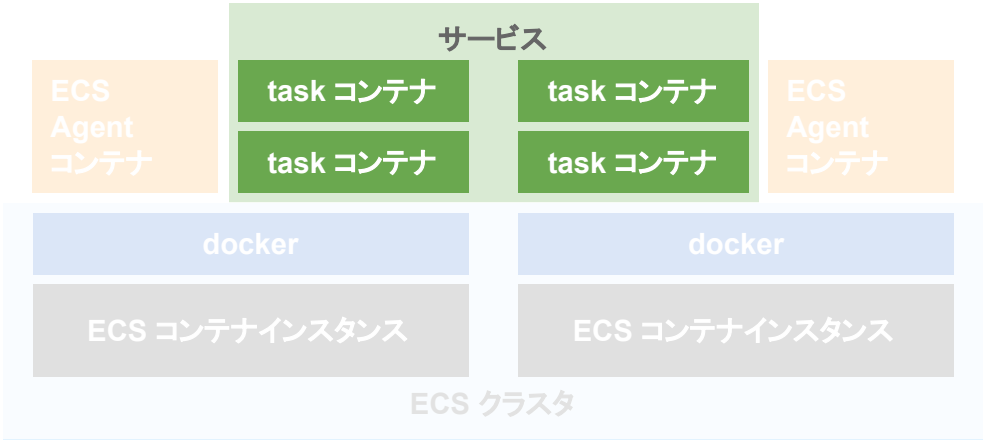
**参考資料**

**Datadog / Fluentd**
https://docs.datadoghq.com/integrations/fluentd/

# 7 ECSのAutoScaling

# コンテナのAutoScaling

- コンテナサービスのCPUの使用率を使用

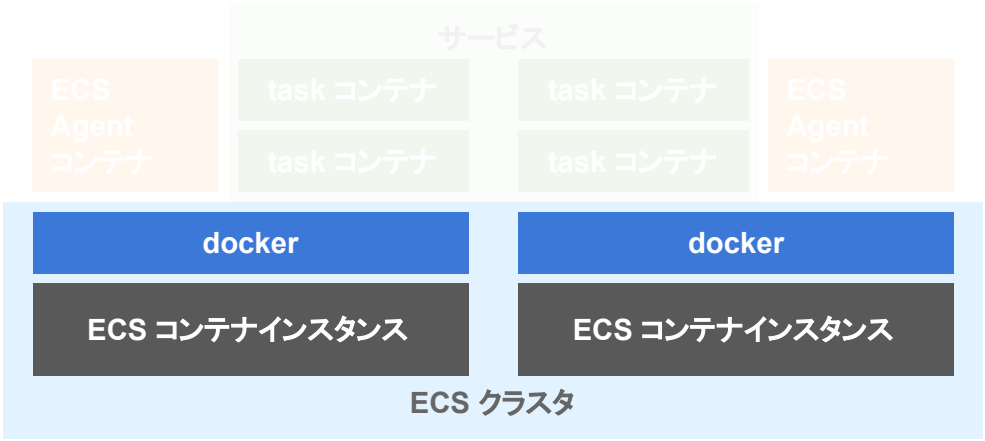



**参考資料**

https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/service-auto-scaling.html

# コンテナインスタンスのAutoScaling

- ECSクラスターのメモリとCPUの予約率を使用




**参考資料**

https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/cloudwatch_alarm_autoscaling.html

# 8 コンテナからNFSを利用

## NFSを利用した理由

- 移行前の構成ですでにNFSを利用
- 限られた時間で、S3などに移行することが難しい

# 8 コンテナからNFSを利用

## タスク定義の抜粋

ボリュームセクション

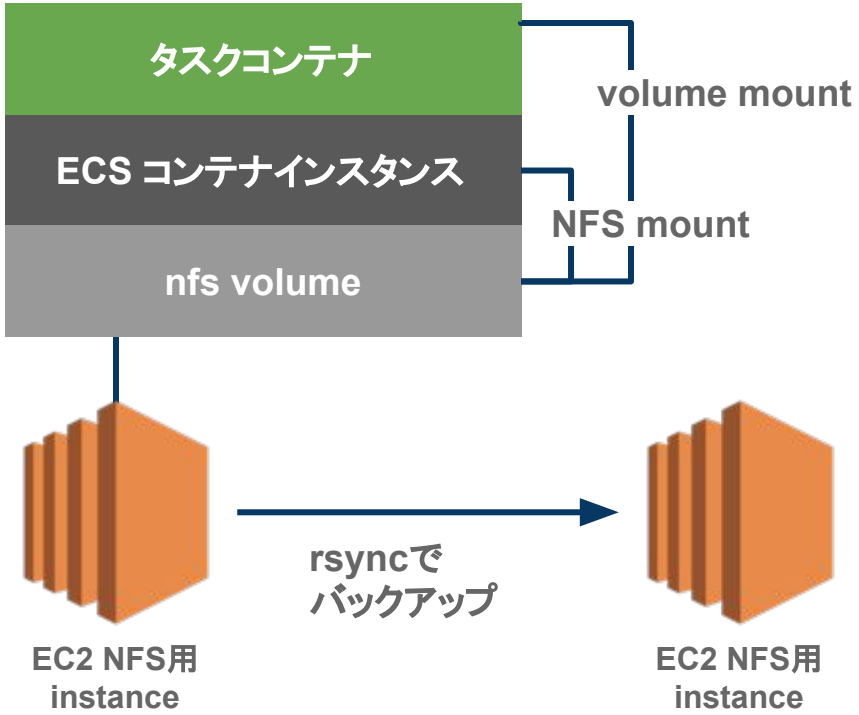
```
{
  "host": {
    "sourcePath": "/path/to/nfsvolume"
  },
  "name": "nfs-data"
}
```

コンテナ定義セクション

```
"mountPoints": [
  {
    "containerPath": "/path/to/nfsvolume",
    "sourceVolume": "nfs-data"
  }
],
```

## User-Data設定の抜粋

```
# bootcmd:
cat << EOT >> $USER_DATA_OUTPUT_FILE_PATH
bootcmd:
- yum install -y nfs-utils
- mkdir -p /nfs
- mount -t nfs -o rw,exec,dev,suid,intr,vers=4,retrans=3,timeo=10,soft ${_NFS_SERVER}:/data/nfs
```



# 9 CIサーバーの移行

wercker

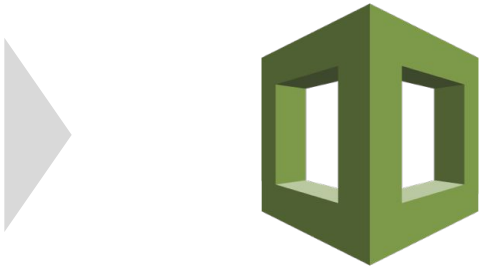
jenkinsからwerckerへ移行

## 当時werckerにした理由

- コンテナベースで CI が可能だった
- 他の CI サービスにはない ECS / ECR(aws が用意している docker リポジトリ) のインテグレーションがあった
- steps と言われる特定の動作を自動化したタスクが数多く存在しており、それを使用することが可能だった
- 欲しいstepsを自身で作成し共有することも可能だった
- 環境変数を利用でき、環境別のデプロイが容易だった
- githubのプライベートリポジトリのCIも無料で始めることができた

# 10 環境セットアップツールの移行

HashiCorp
Terraform



terraformからcloudformationへ移行

## terraform使用時の苦労

- ステートファイルの管理
- クラスターのローリングアップデートなど
  対応していない機能がある
- 失敗時にきれいに戻らない場合がある
- バージョンアップが頻繁

## cloudformationにした理由

- 設定ファイルなので複雑にならない
- GUIからも作成・実行できる
- 失敗時はきれいに切り戻しされる

## cloudformation使用時の苦労

- プログラマブルではないので、記述量が多くなってしまう場合がある

**terraformとcloudformationは、**
**それぞれいいところがある**

1

会社紹介

2

AKB48
モバイルサイトの移管

3

利用技術の選定

4

DBの移管

5

Webサービスの
ECS利用

6

まとめ

# まとめ

## エンジニアの意識の変化

### プラットフォーム

- 物理サーバー
- VMware
- OpenStack
- Vagrant

### エンジニアの意識

- コンテナは難しそう
- インフラエンジニアがある程度準備してくれることを期待
- パブリッククラウドの新しい技術も利用したい

### プラットフォーム

- ECS,ECR
- コンテナ管理用CIツール
- docker

### エンジニアの意識

- ローカル環境と開発環境が同一になって開発がやりやすくなった
- 容易にデプロイが可能になった
- よりクラウドを活用していきたい

## シーエー・モバイルの開発スタイルのコンセプト

**いつでもどこでも同じ環境でサービスの開発ができる**

- ローカル環境は起動が軽いdockerを採用
- ローカル環境で使用しているdocker コンテナをサービスでも利用するために、ECSを採用

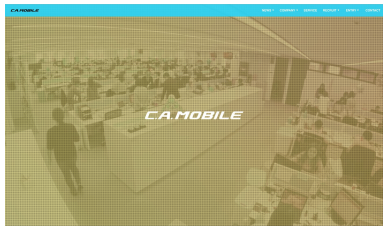
# ECSを利用することで要件が満たされた

- ローカル環境と開発環境が同一になった
- 容易にデプロイとデプロイの実行が可能になった

# 今後の展望と採用紹介

## すべてのサービスを
## パブリッククラウドで開発・運用

- AWSやGCP、ECSやKubernetesの知見・開発経験のある方
- クラウドを活用したいフロントやサーバーサイドエンジニアの方
  どんどん新しい技術も取り入れてますので、ぜひご応募ください！




### 採用サイト

https://hr.camobile.com/

# CAグループの担当のお二方

辻本さん

成田さん

- 毎週定例の実施
- AWS移設に関する技術情報のサポート

引き続き、柔軟かつ迅速なサポートをよろしくお願い致します。