NAVITIME

「乗換NAVITIME」での移行事例

ECSを活用したシステム移行

株式会社ナビタイムジャパン　萱島 克英

アジェンダ

- NAVITIMEサービスのご紹介
- AWS移行に至った経緯
- インフラ構築手順のCode化
- Amazon ECS、Auroraの活用事例
- 検証/切り替え
- クラウド移行とコンテナ化がもたらした効果
- 今後の展望

NAVITIMEサービスのご紹介

NAVITIME サービス紹介

月間ユーザ数　約3500万UU（2016年9月末時点）

有料会員数　約450万UU（2016年9月末時点）

NAVITIME サービス紹介

NAVITIME

NAVITIME サービス紹介

NAVITIME

ほぼ全サービスのバックエンドシステムをオンプレミスで運用

サーバー台数　約3500台（VM含む）

サービス紹介

乗換NAVITIME

公共交通機関での移動に特化したナビゲーションアプリ

- 乗換案内、時刻表、鉄道運行情報
- 乗換に最適な乗車位置の案内
- 混雑を避けたルートも選べる！

サービス紹介

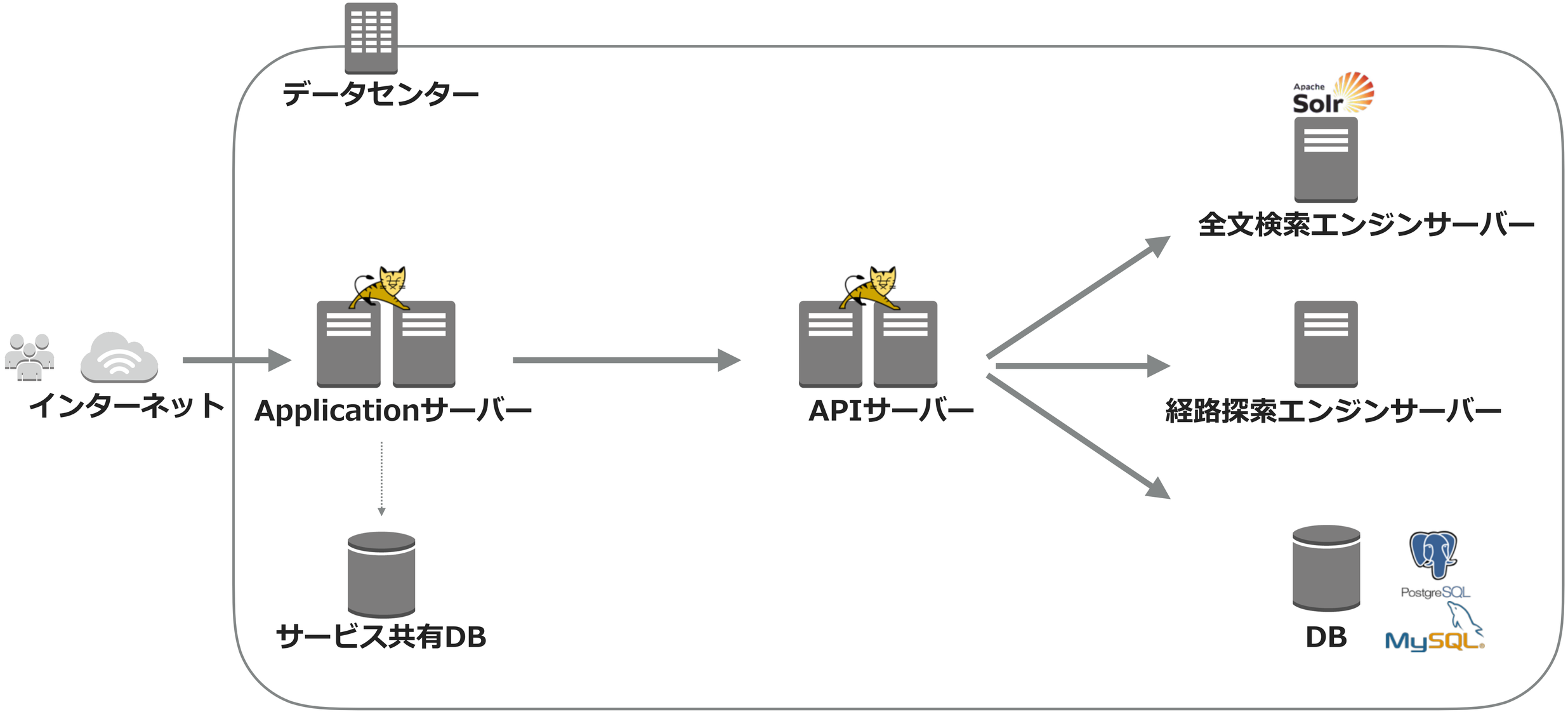
乗換NAVITIME

- 2012年にサービス提供開始
- サービス開始～2017年3月まではオンプレでサービスを提供
- 現状はAWSとオンプレの**ハイブリッド構成**（割合は9：1）

AWS移行に至った経緯

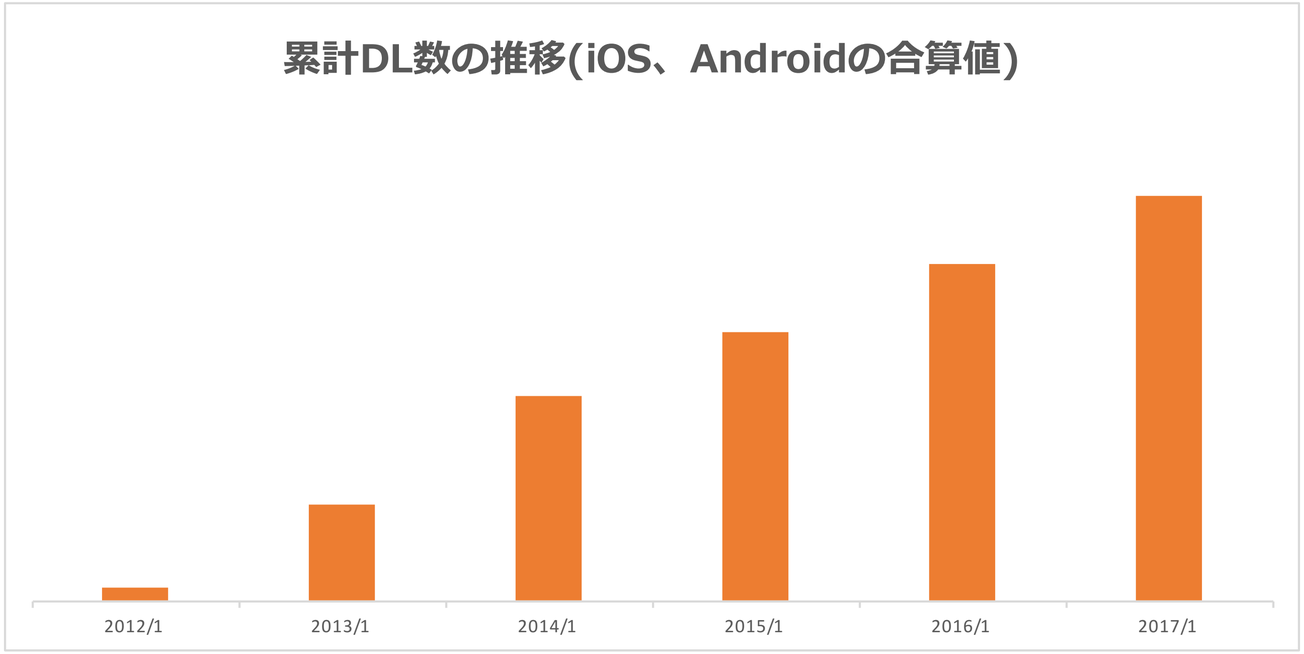
NAVITIMEのインフラ

一般的なWEB 3階層+αでOSSをベースにした構成



AWS移行に至った背景 - ①乗換NAVITIMEの利用者数増加

- 乗換NAVITIMEサービスは非常に多くのお客様にご利用いただいているサービス
- 繁忙期のアクセス数を予測しながらのサーバー調達・運用は**インフラエンジニアの人的コストを増加**



AWS移行に至った背景 - ②インフラの課題

課題

- アクセス数の増加時に**自動でスケール**できていない。
- インフラ構築作業のCode化が進んでいない(部分的なCode化のみ)

オンプレにおけるインフラ構築/運用フロー

1 
インフラエンジニアがベンターとの調整/契約

2 
インフラエンジニアがデータセンターにてサーバーをラックに格納

3 
インフラエンジニアがOS、ミドルウェアをインストール

4 
サービス開発担当がサーバーにアプリケーションをDeploy

5 
インフラエンジニアがサーバーを死活監視、障害発生時に
サーバー追加、再起動作業を実施

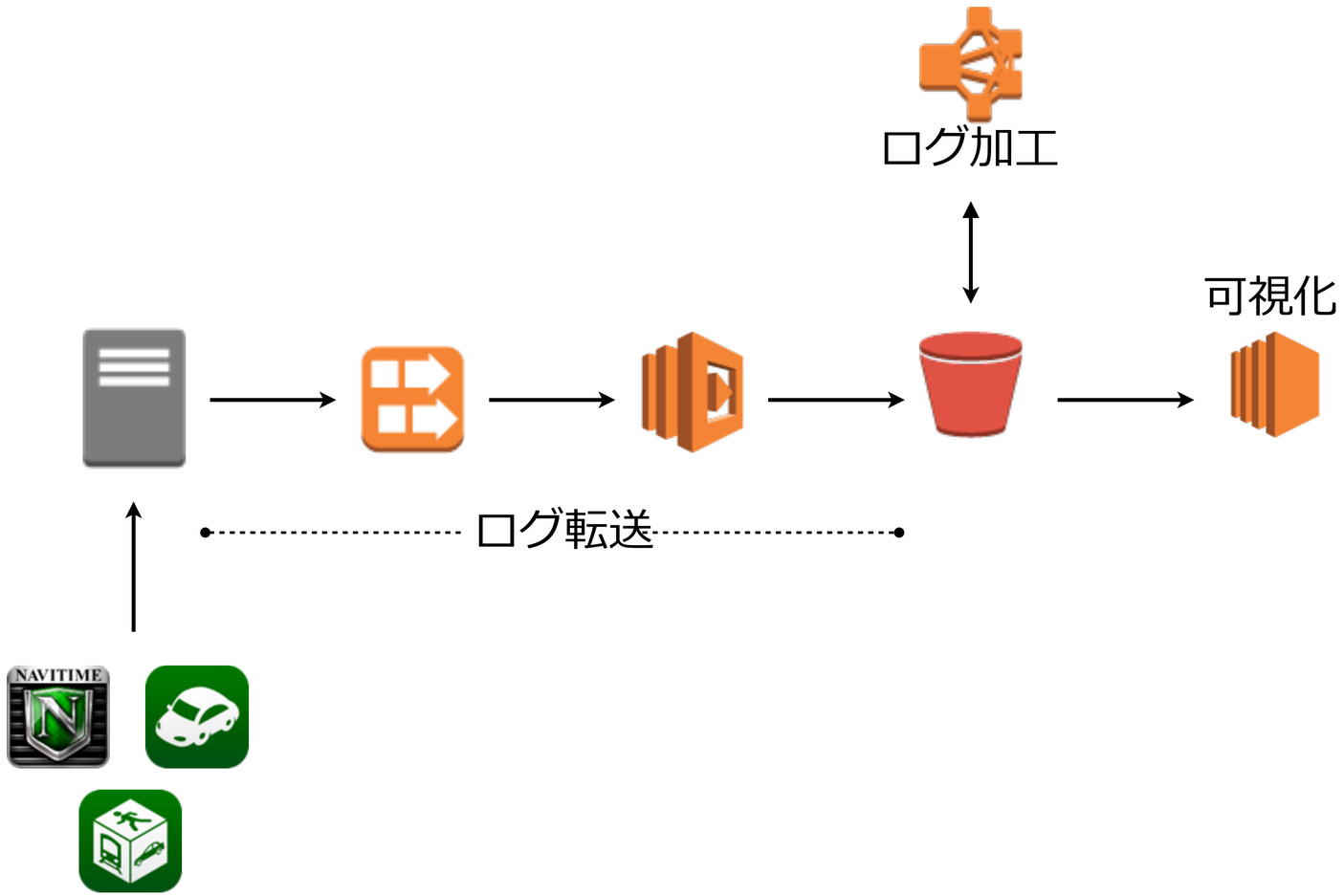
AWS移行に至った背景 - ③部分的に利用していたAWS活用事例から効果が見えてきた

- 2014年～2016年当時、一部のシステムでAWSを利用
- 徐々にAWSの活用メリットが見えてきた
  - AWSのマネージドサービスを利用することによる**開発効率**の向上
  - オンデマンドによる**リードタイムの短縮**といった効果が見えてきた。

AWS移行に至った背景 - ③社内に部分的なAWS活用事例があった

交通コンサルティング (2014年～)

- ログの転送、加工、可視化をAWS上で実施

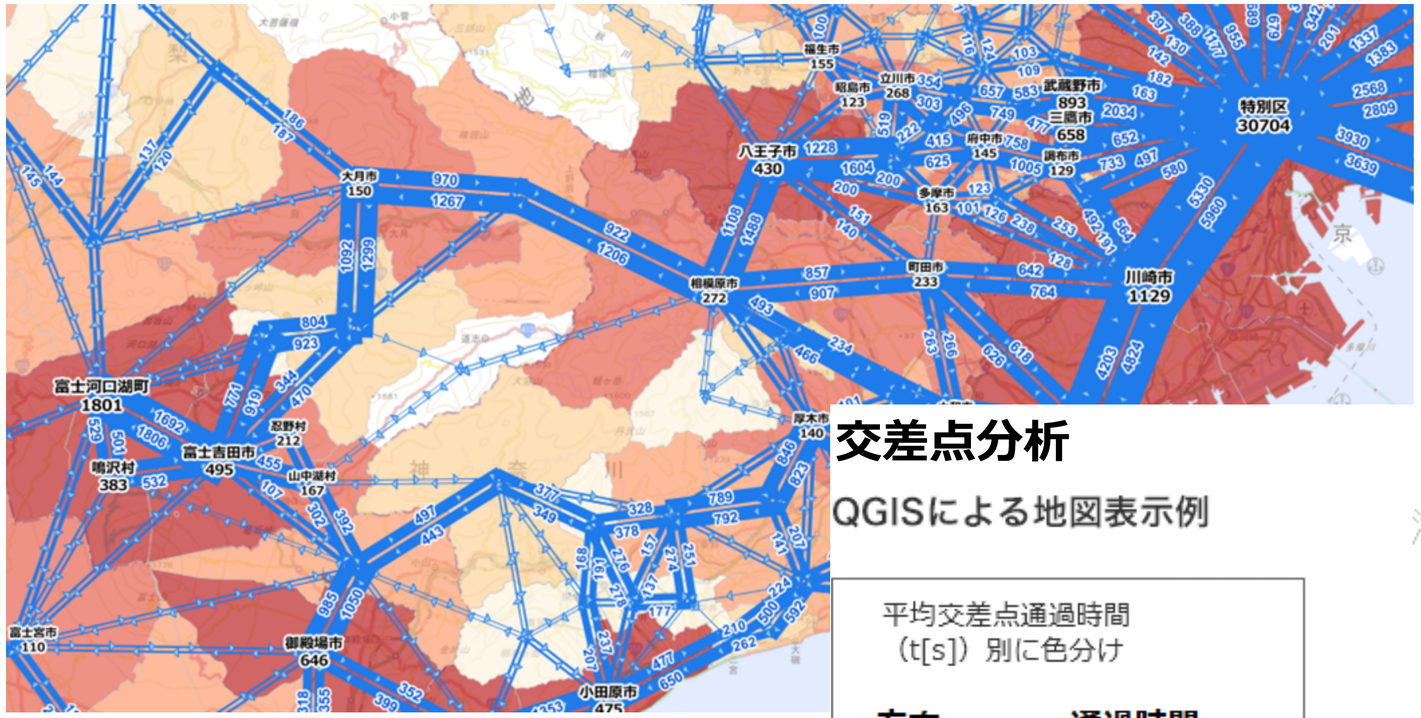


AWS移行に至った背景 - ③社内に部分的なAWS活用事例があった

交通コンサルティング (2014年～) 可視化の例

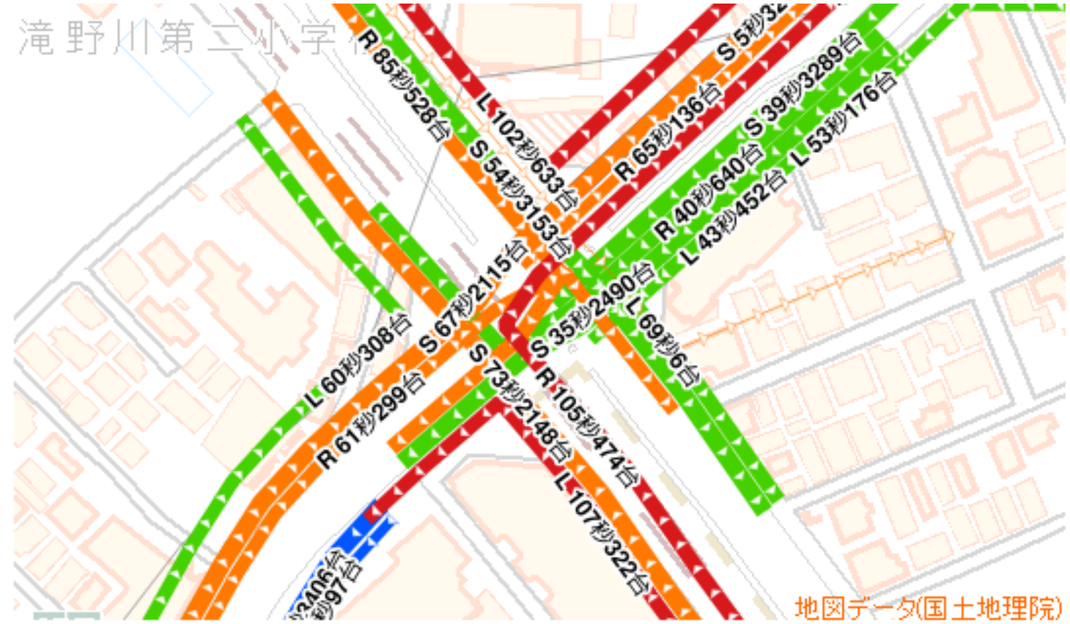
インバウンド旅行者流動分析

詳細は http://consulting.navitime.biz/



交差点分析

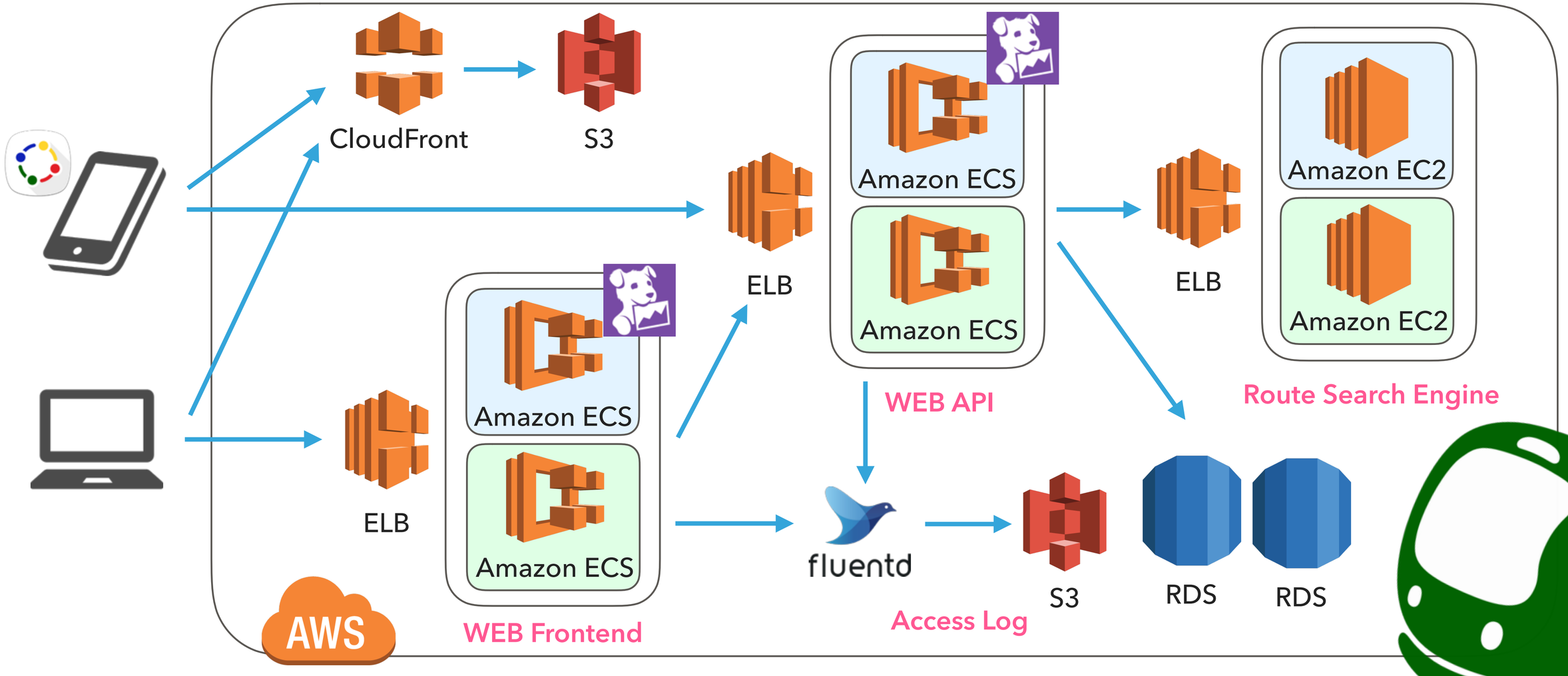
QGISによる地図表示例



AWS移行に至った背景 - ③社内に部分的なAWS活用事例があった

NAVITIME Transit ～18ヶ国, 30以上の都市に対応している乗換検索アプリ・サイト～

- 2016/02 ：EC2 中心の構成から ECS, Cloudformation を利用した構成へ移行



AWS移行に至った背景 - ③部分的に利用していたAWS活用事例から効果が見えてきた

- AWSを活用する事によるメリットが見えてきた
  - AWSのマネージドサービスを利用することによる**開発効率**の向上
  - オンデマンドによる**リードタイムの短縮**といった効果が見えてきた。

AWSに乗換NAVITIMEのバックエンドシステムを移行する事を決断

AWSへの移行方針検討

２つの移行案

- 案１：VM Import/Exportを使った移行案
- 案２：アプリケーションをコンテナ化し、ECSで運用する案

AWSへの移行方針検討

社内で検討した移行の流れ

- **案１：VM Import/Exportを使った移行案**
- 案２：アプリケーションをコンテナ化し、ECSで運用する案

AWSへの移行方針検討

案１：VM Import/Exportを使った移行案

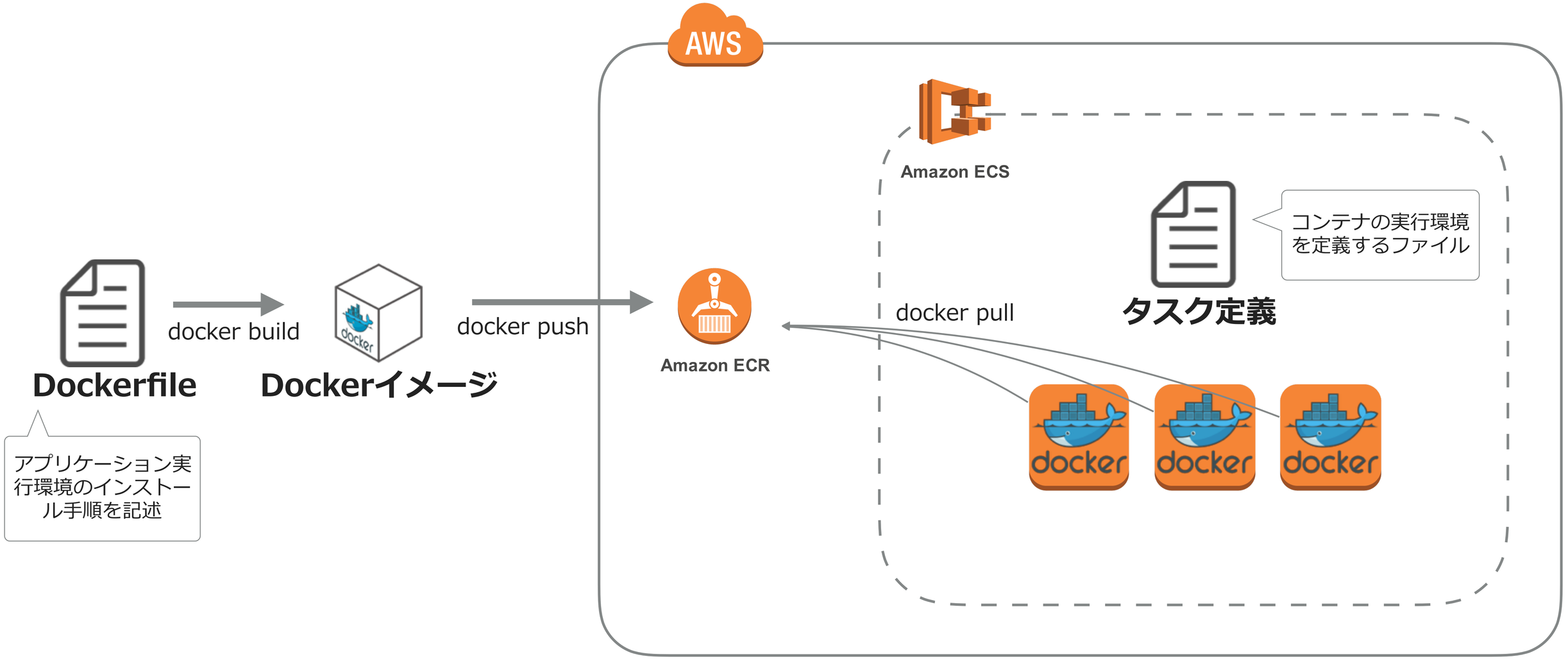
AWSへの移行方針検討

社内で検討した移行の流れ

- 案１：VM Import/Exportを使った移行案
- **案２：アプリケーションをコンテナ化し、ECSで運用する案**

AWSへの移行方針検討

案２：アプリケーションをコンテナ化し、ECSで運用



AWSへの移行方針検討

どっちがよいか？

運用方式	メリット	デメリット
案１：VM Import・Exportを使ってAMIで運用	・AWSへの移行コストが低い	・運用に関する課題が残ったままになる。 ・運用コストは高い（ECS利用方式と比較）
案２：コンテナ化しECSで運用	・アプリケーション環境構築がCode化される。 ・ECSがインフラエンジニアの作業を代替してくれる ・アプリケーションのポータビリティが上がる ・他サービスのAWS移行でも流用が可能	・初期環境構築に少し時間がかかる。(AMI利用方式と比較)

AWSへの移行方針検討

どっちがよいか？

運用方式	メリット	デメリット
案１：VM Import・Exportを使ってAMIで運用	▸ AWSへの移行コストが低い	▸ 運用に関する課題が残ったままになる。 ▸ 運用コストは高い（ECS利用方式と比較）
案２：コンテナ化しECSで運用	▸ アプリケーション環境構築がCode化される。 ▸ ECSがインフラエンジニアの作業を代替してくれる ▸ アプリケーションのポータビリティが上がる ▸ 他サービスのAWS移行でも流用が可能	▸ 初期環境構築に少し時間がかかる。(AMI利用方式と比較)

採用

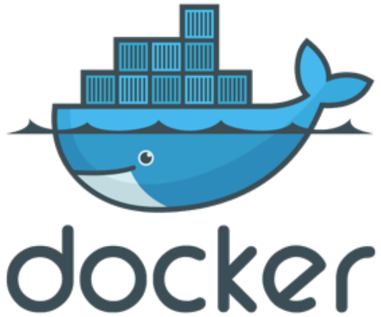
移行するにあたり実現したかった事

APサーバーの
オートスケーリング

インフラエンジニア
の**運用コスト削減**

リードタイムの短縮

と



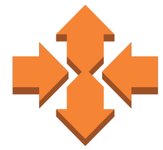
の有効活用で実現させたい！

移行するにあたり実現したかった事

APサーバーの
オートスケーリング

インフラエンジニアの
運用コスト削減

リードタイムの短縮



Auto Scaling

Amazon ECS

alarm

移行するにあたり実現したかった事

APサーバーの
オートスケーリング

インフラエンジニア
の**運用コスト削減**

リードタイムの短縮

成果物生成、インフラ構築フローをCode化、コンテナマネジメントシステムの活用

Amazon ECS

AWS
CloudFormation

alarm

移行前の懸念

- そもそもECSコンテナ上で乗換NAVITIMEのバックエンドシステムは動くのか？
- ECS上で経路探索エンジンサーバーを稼働させた場合、スループットが下がらないか？

インフラ構築手順のCode化

インフラ構築手順のCode化

- APサーバーのコンテナ化
- CloudFormationを使ったAWS環境構築手順のCode化

APサーバーのコンテナ化

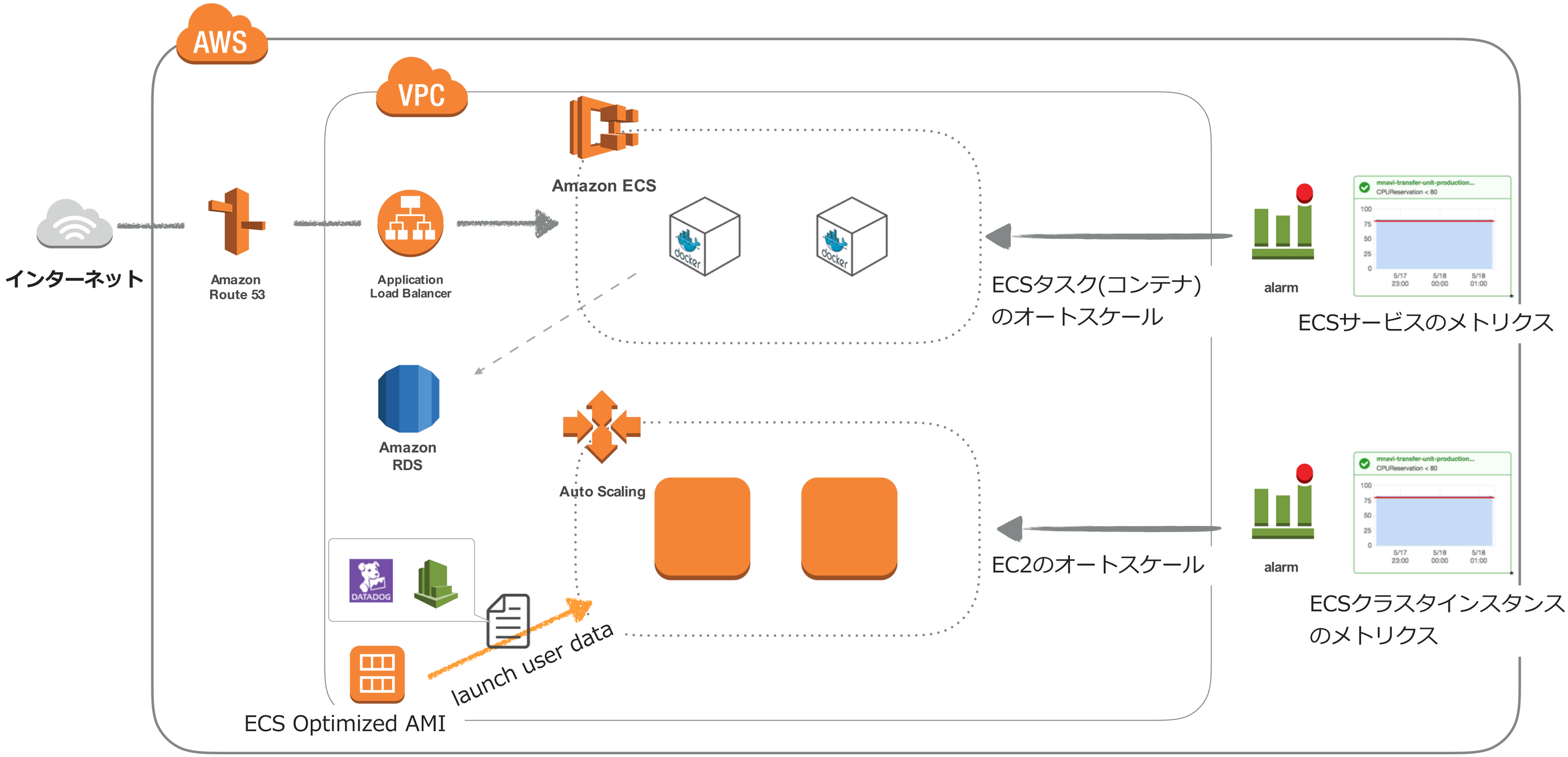
- 一部のオンプレ環境でansibleを試験的に利用していた為、ansible-containerを使ってDockerコンテナを作成
- コンテナの構成チェックはServerspecを利用

CloudFormationを使ったAWS環境構築手順のCode化

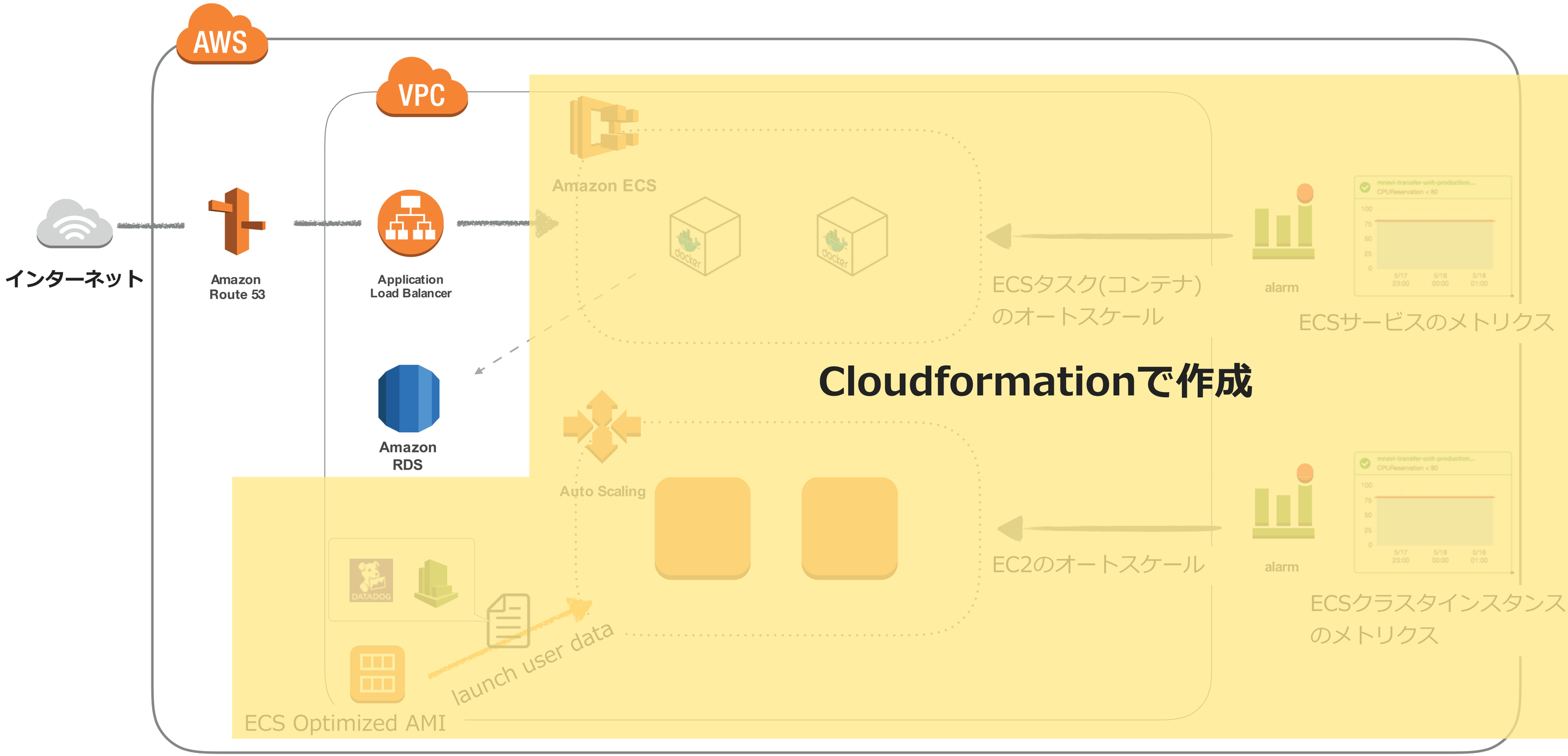
- Cloud Formationとは？
  - AWSの環境構築手順をファイルに定義することができる。
  - 定義ファイルを元にAWS環境を作成・更新・削除する事ができる。

```
ECSServiceAutoScalingTarget:
  Type: AWS::ApplicationAutoScaling::ScalableTarget
  Properties:
    MaxCapacity:
      Ref: ECSServiceMaxCapacity
    MinCapacity:
      Ref: ECSServiceMinCapacity
    ResourceId: !Sub
      - service/${ECSClusterName}/${ECSServiceName}
      - { ECSServiceName: !GetAtt ECSService.Name }
    RoleARN:
      Ref: ECSServiceRoleARN
    ScalableDimension: "ecs:service:DesiredCount"
```

乗換NAVIITMEのAWS概要構成図 - AWS環境構築手順のCode化



Cloudformationの利用範囲 - AWS環境構築手順のCode化



Cloudformationの良かった点

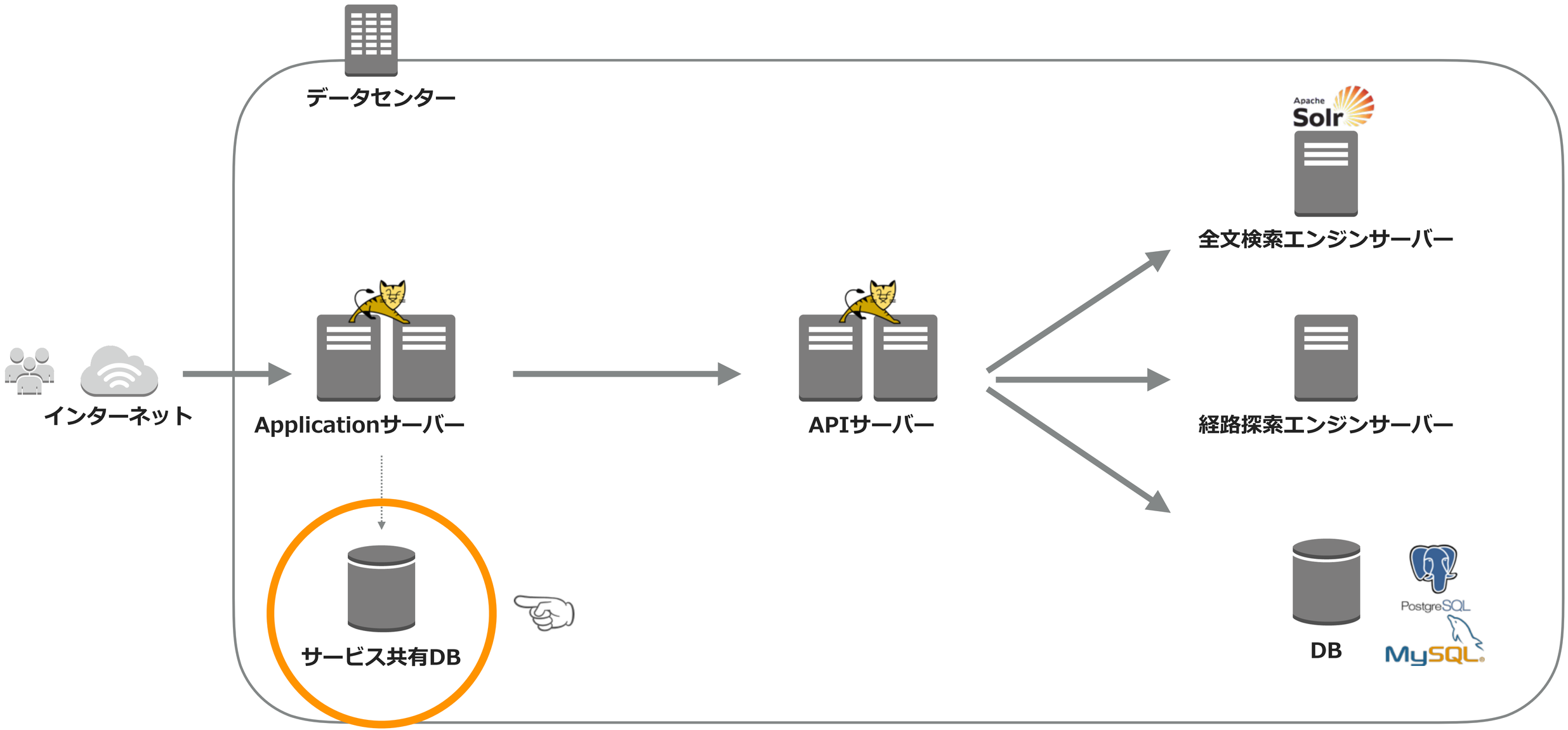
- AWS環境手順がCode化され、環境の作成/更新/削除が簡単にできる
  - 例１）環境構築後にインスタンスタイプを変更したい
  - 例２）性能比較を行う為、EBSボリュームの種類が違う２つのECSクラスターを生成したい

Cloudformationの良かった点 - 台湾プランニングでの流用事例

- NAVITIMEトラベルで提供している台湾プランニングのバックエンドシステム作成時に乗換NAVITIME向けに作成していたCloudFormationを流用

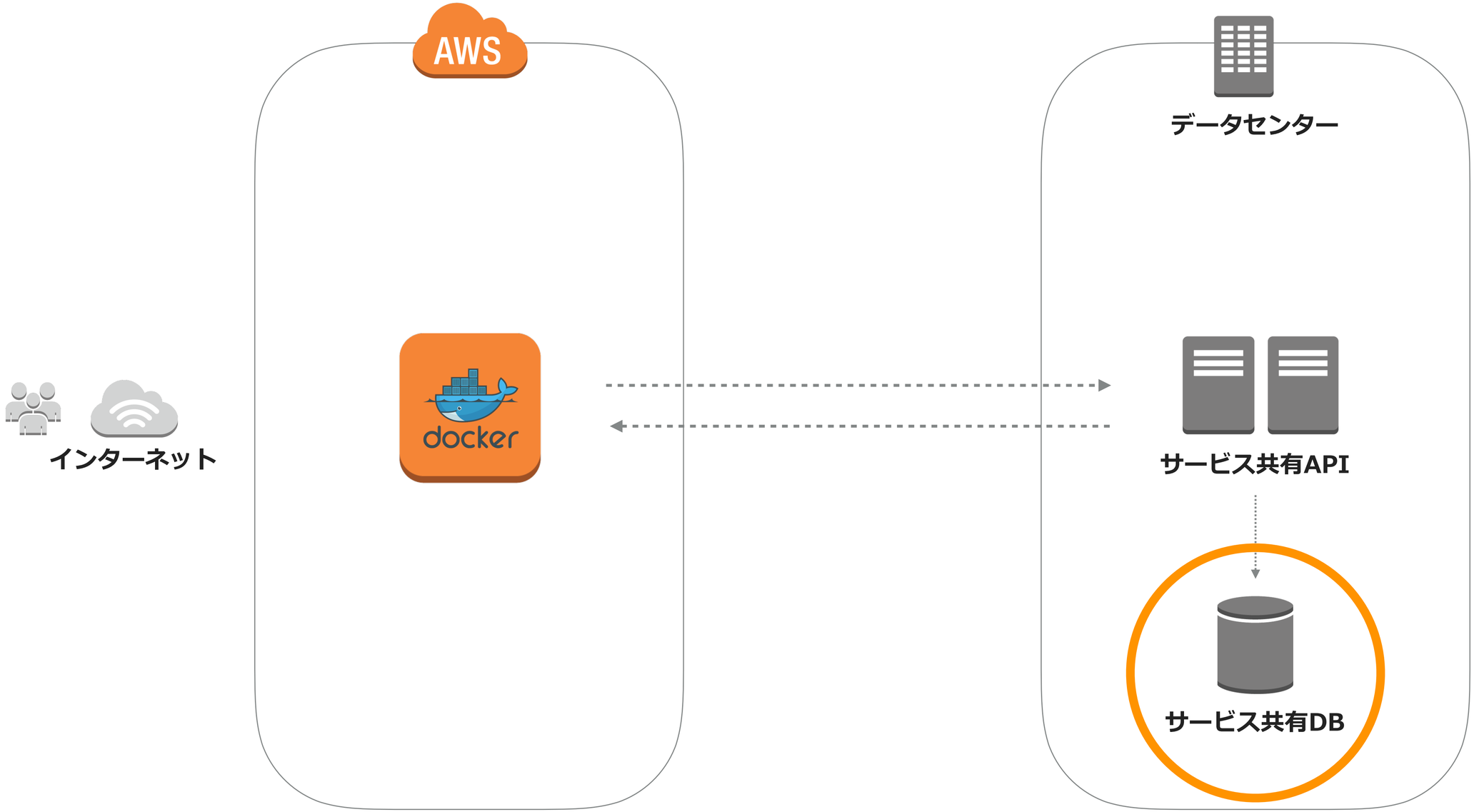
AWS環境構築、検証、サービスリリースまでを **2週間** で対応

AWS移行しなかった部分



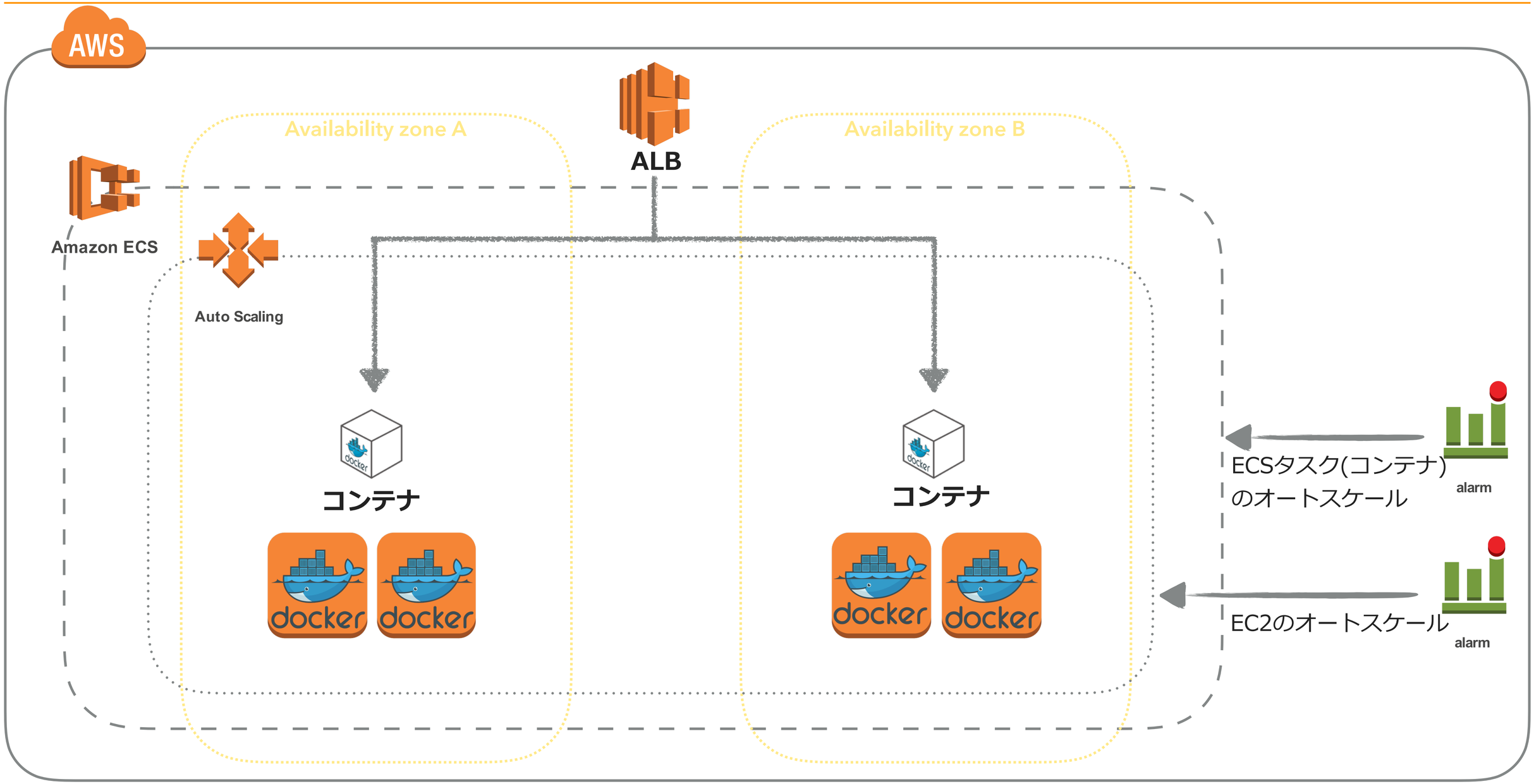
AWS移行しなかった部分

移行しない機能はRest API化し、AWS環境からHTTPアクセスさせる方針に



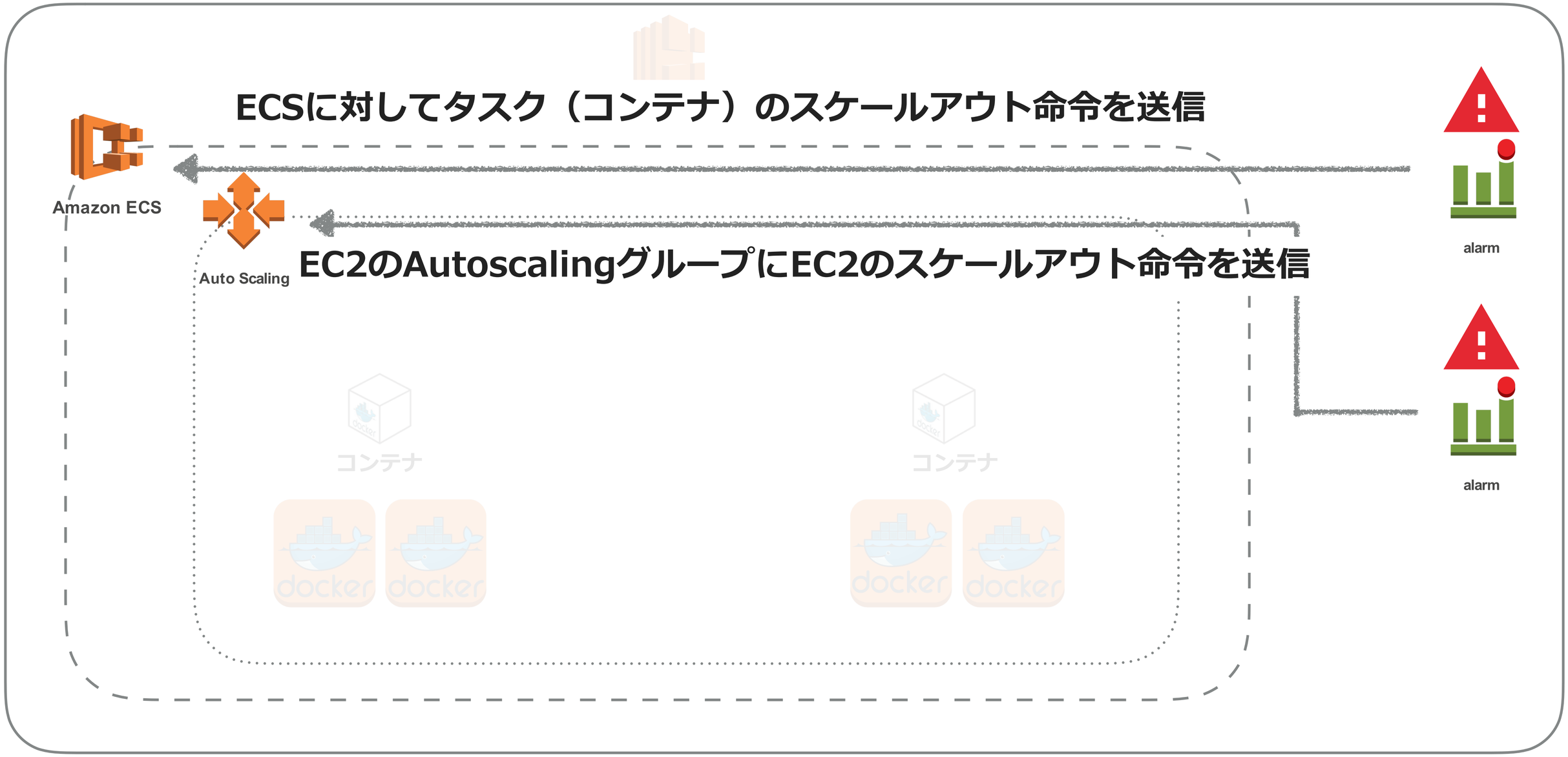
Amazon ECS、Auroraの活用事例

Amazon ECSの活用事例

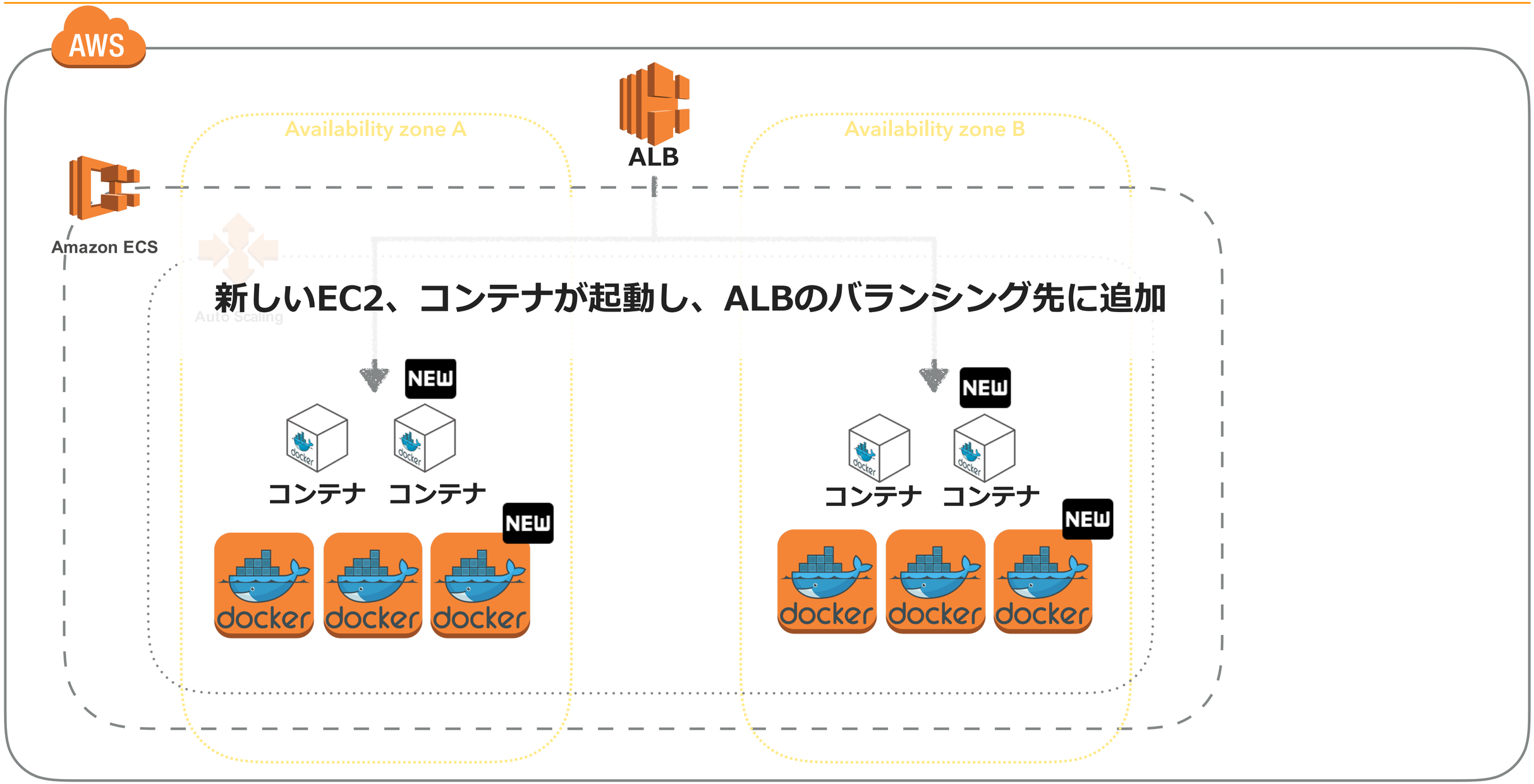


ECSを利用したオートスケールの流れ

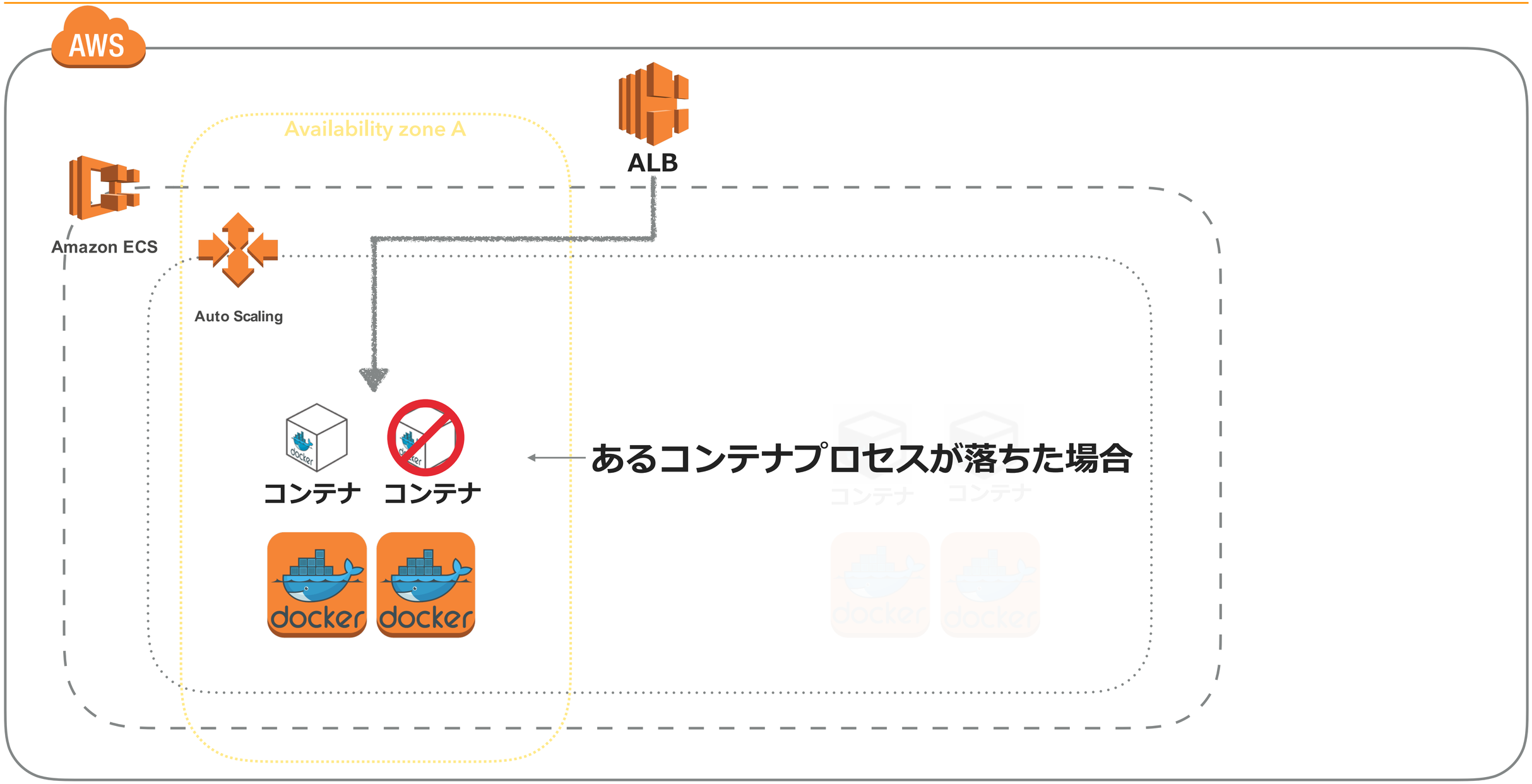
ECSを利用したオートスケールの流れ



ECSを利用したオートスケールの流れ



コンテナの死活監視

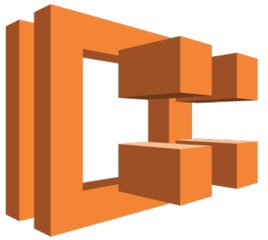

AWS
Availability zone A
ALB
Amazon ECS
Auto Scaling
コンテナ
コンテナ
あるコンテナプロセスが落ちた場合

コンテナの死活監視


AWS
Availability zone A
ALB
Amazon ECS
Auto Scaling
NEW
コンテナ
コンテナ
落ちたコンテナをALBのバランシング先から外し、
新規にコンテナを起動してくれる

コンテナの管理はECSまかせ

便利!



Amazon ECS

- Deploy方法が柔軟
  - RollingUpdate
  - Blue&GreenUpdate
- コンテナの死活監視、再起動はECSがやってくれる
- CloudWatch Alermと連動したECS Task(コンテナ)のScaleOut

Amazon Auroraの活用事例

スポットデータの更新仕様

- 1日数回、スポット情報データの全件更新を実施
- 大量な検索クエリを捌く必要がある為、データ更新処理は**Blue/Green方式**で実行

Amazon Auroraの活用事例

Auroraを採用した理由

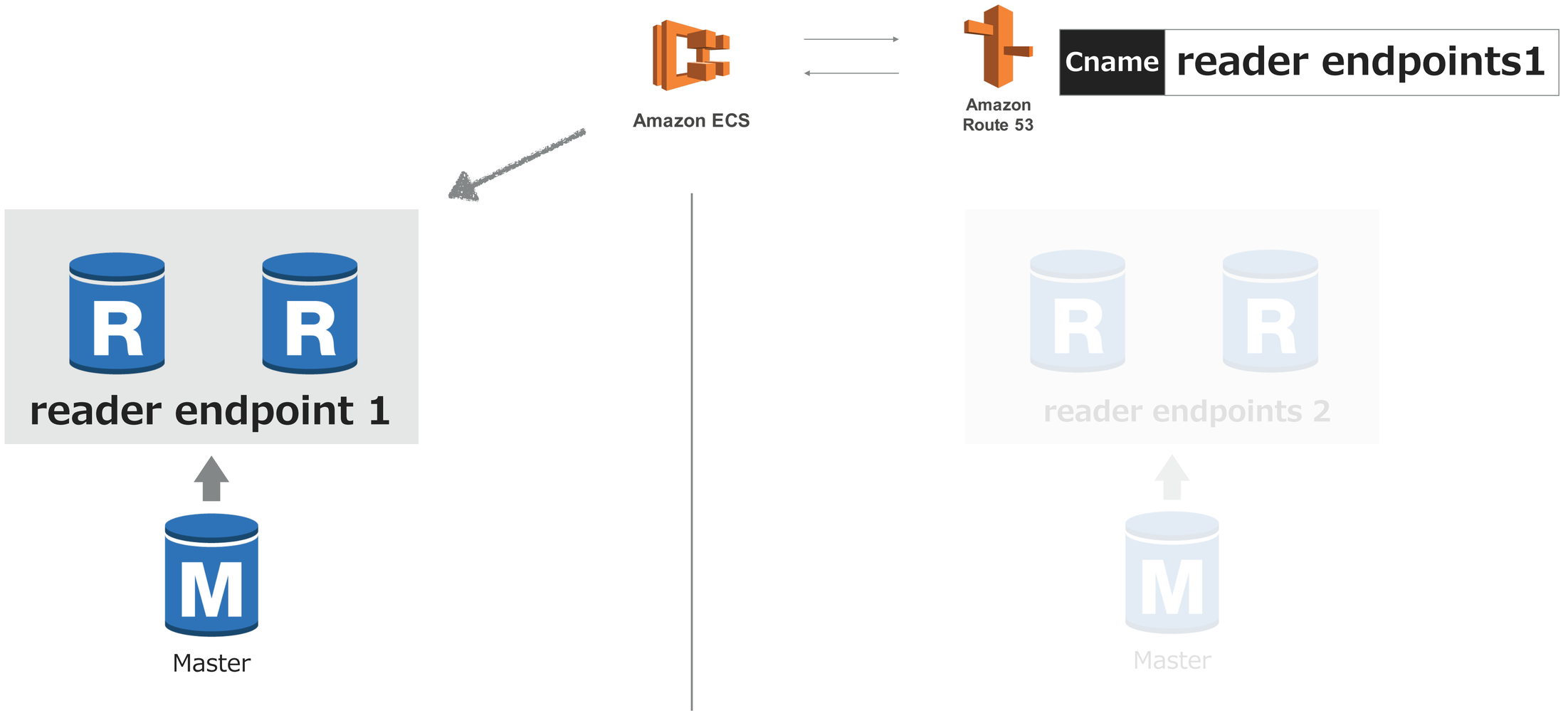
- reader endpointとRoute53を組み合わせたBlue/Green切り替えが容易
- リードレプリカの機能が便利
  - ロードバランシング機能
  - 自動修復機能

Amazon Auroraの活用事例

Auroraを採用した理由

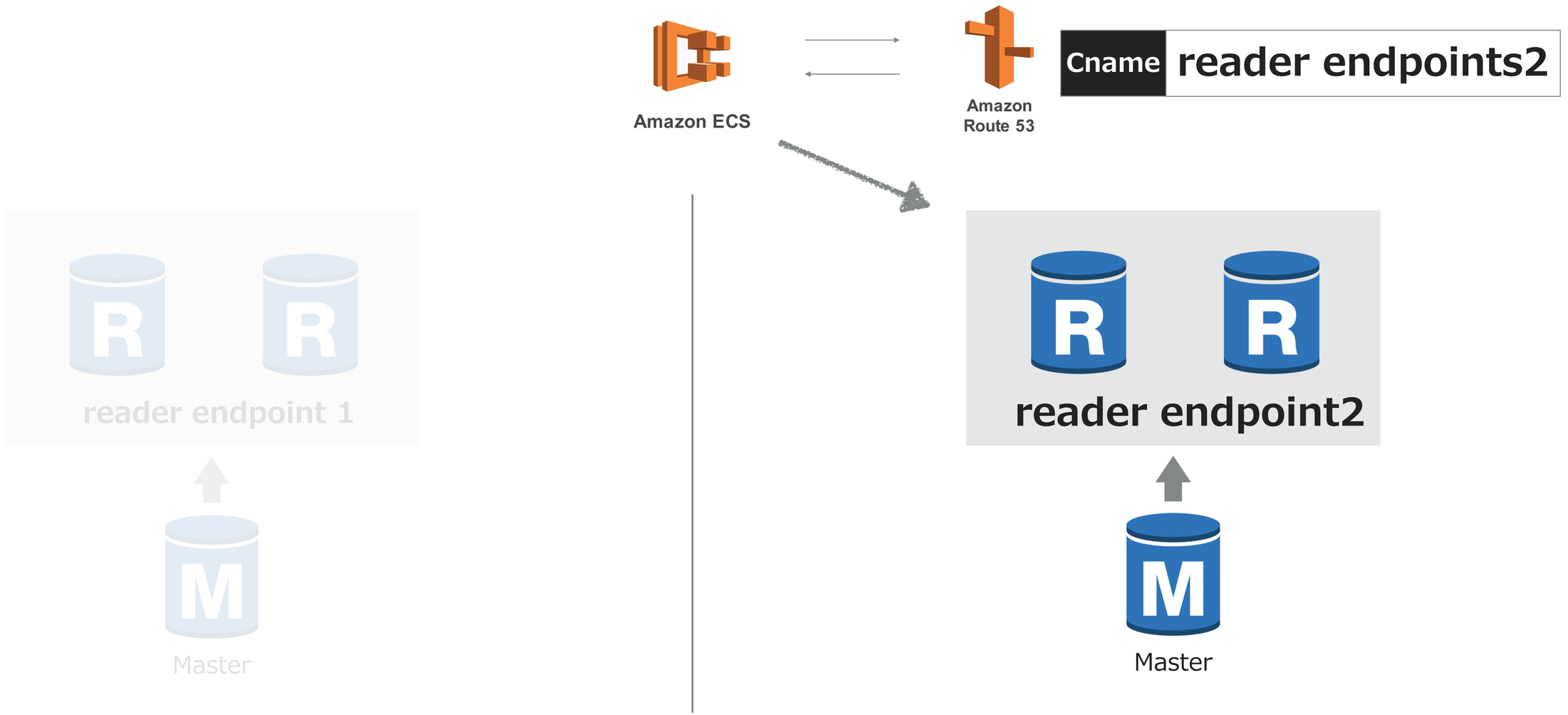
- **reader endpointとRoute53を組み合わせたBlue/Green切り替えが容易**
- リードレプリカの機能が便利
  - ロードバランシング機能
  - 自動修復機能
- 運用が楽

Amazon Auroraの活用事例



Route 53のCnameレコードを更新することで、参照先のread endpointの切り替えを実施

Amazon Auroraの活用事例



Route 53のCnameレコードを更新することで、参照先のread endpointの切り替えを実施

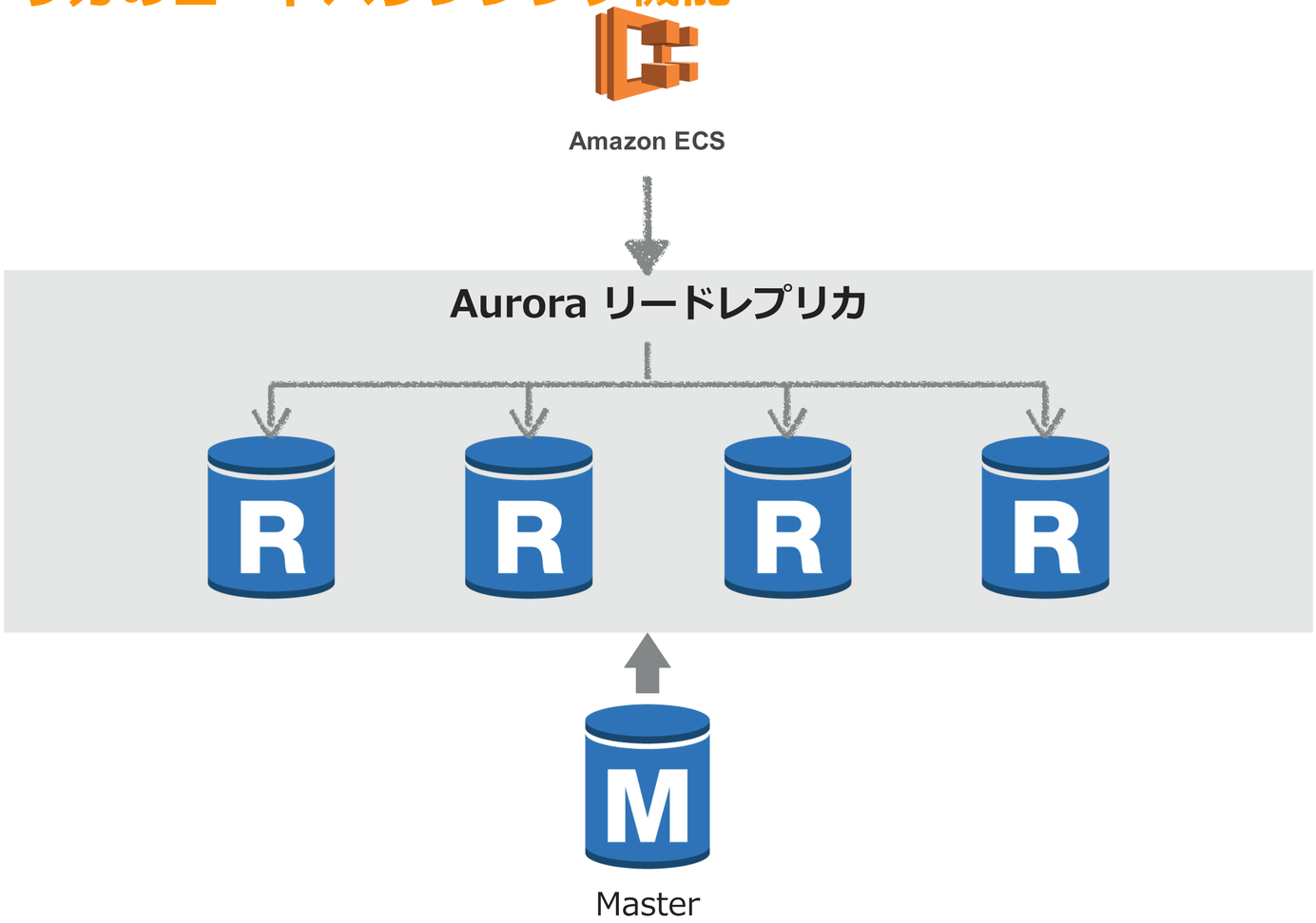
Amazon Auroraの活用事例

Auroraを採用した理由

- reader endpointとRoute53を組み合わせたBlue/Green切り替えが容易
- **リードレプリカの機能が便利**
  - **ロードバランシング機能**
  - **自動修復機能**

Amazon Auroraの活用事例

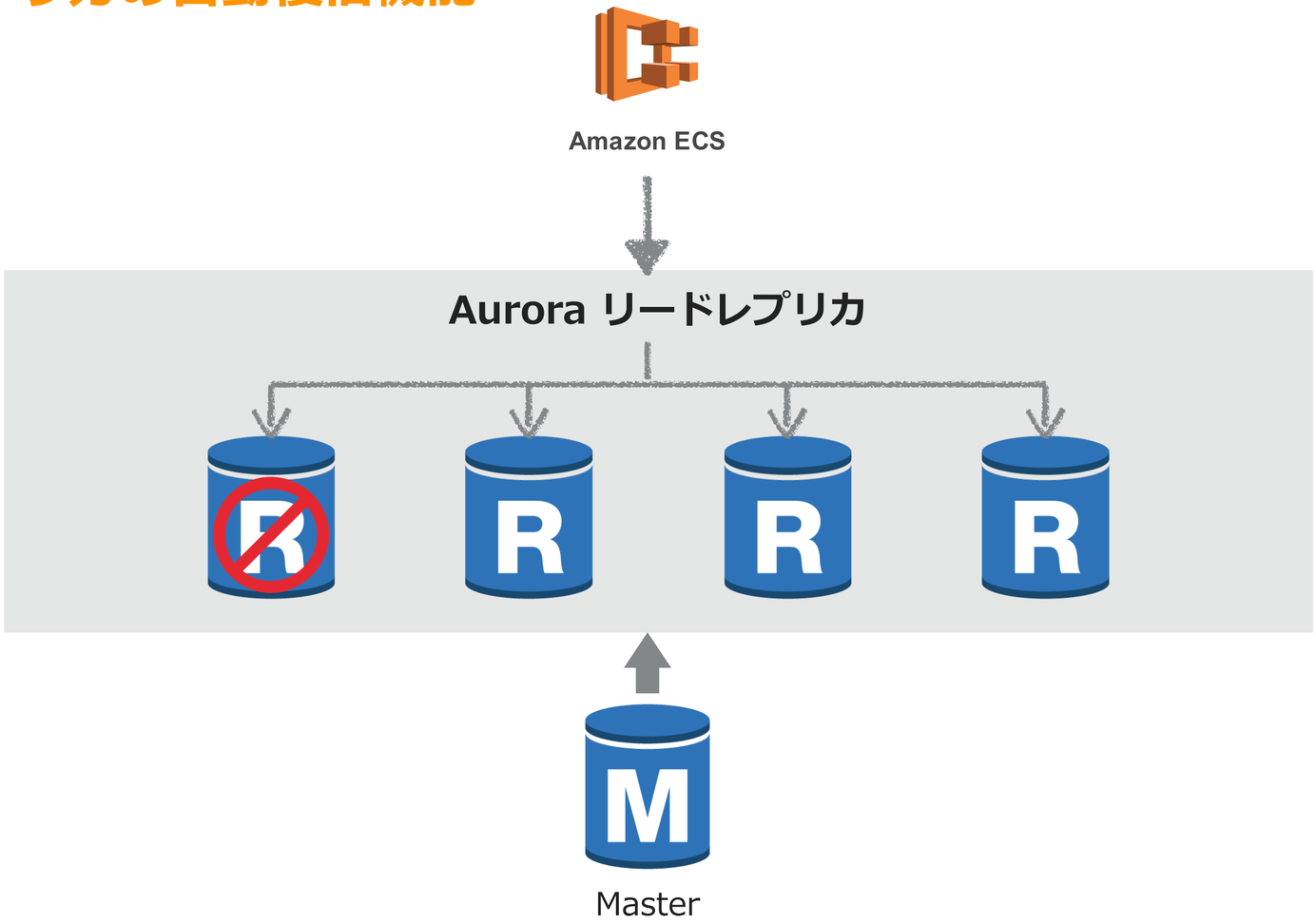
便利な点：リードレプリカのロードバランシング機能



リードレプリカに**ロードバランシング機能**がある為
Haproxy、MyDNS等のツールをDBの前段に構築する手間が省ける

Amazon Auroraの活用事例

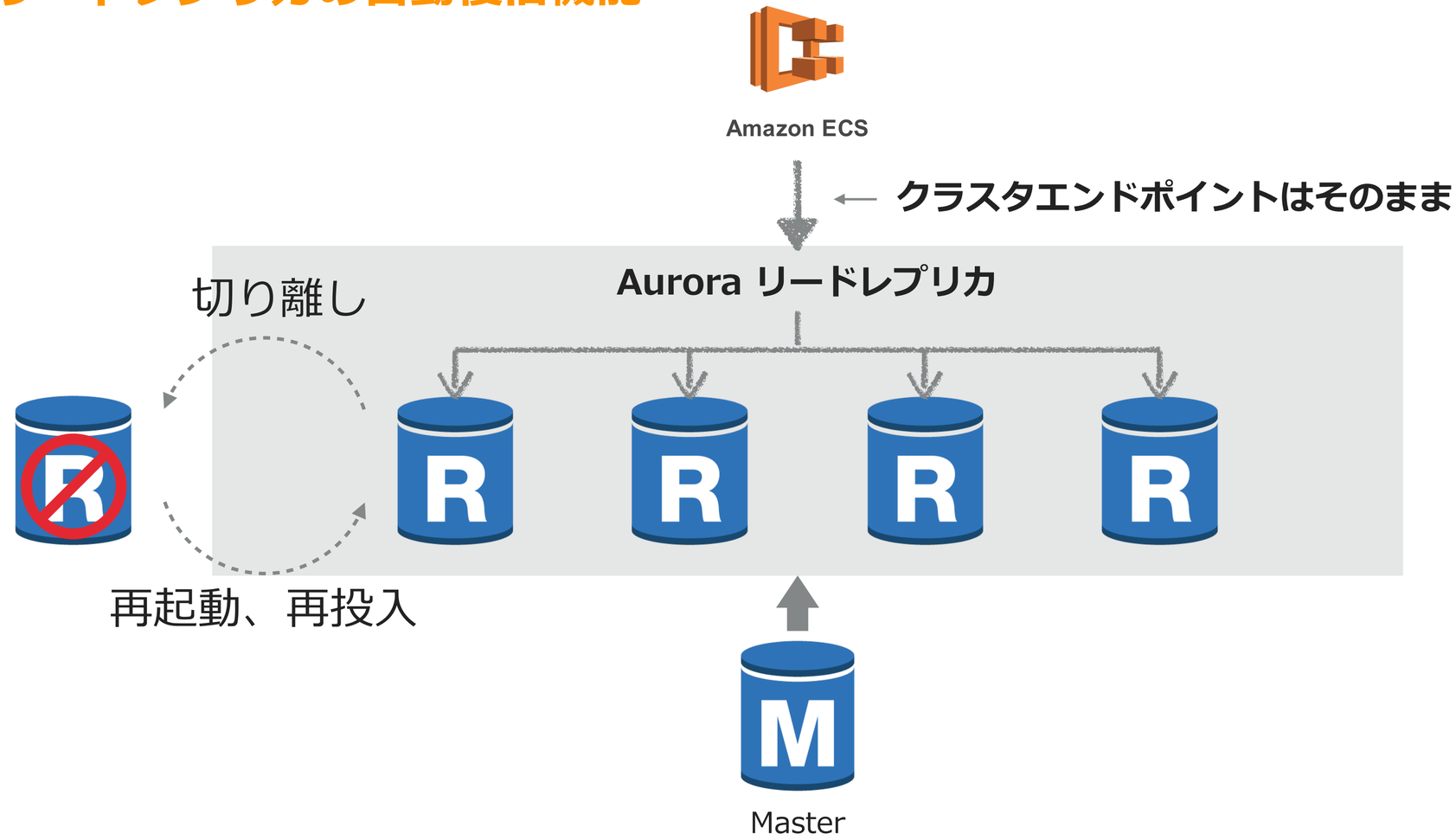
便利な点：リードレプリカの自動復旧機能



リードレプリカに**自動復旧機能**がある為、切り離し、再起動、再投入を自動でやってくれる

Amazon Auroraの活用事例

便利な点：リードレプリカの自動復旧機能



リードレプリカに**自動復旧機能**がある為、切り離し、再起動、再投入を自動でやってくれる

検証/切り替え

パフォーマンスに関する懸念

オートスケール

インフラエンジニア
の運用コスト削減

リードタイム短縮

Amazon ECS

AWS
CloudFormation

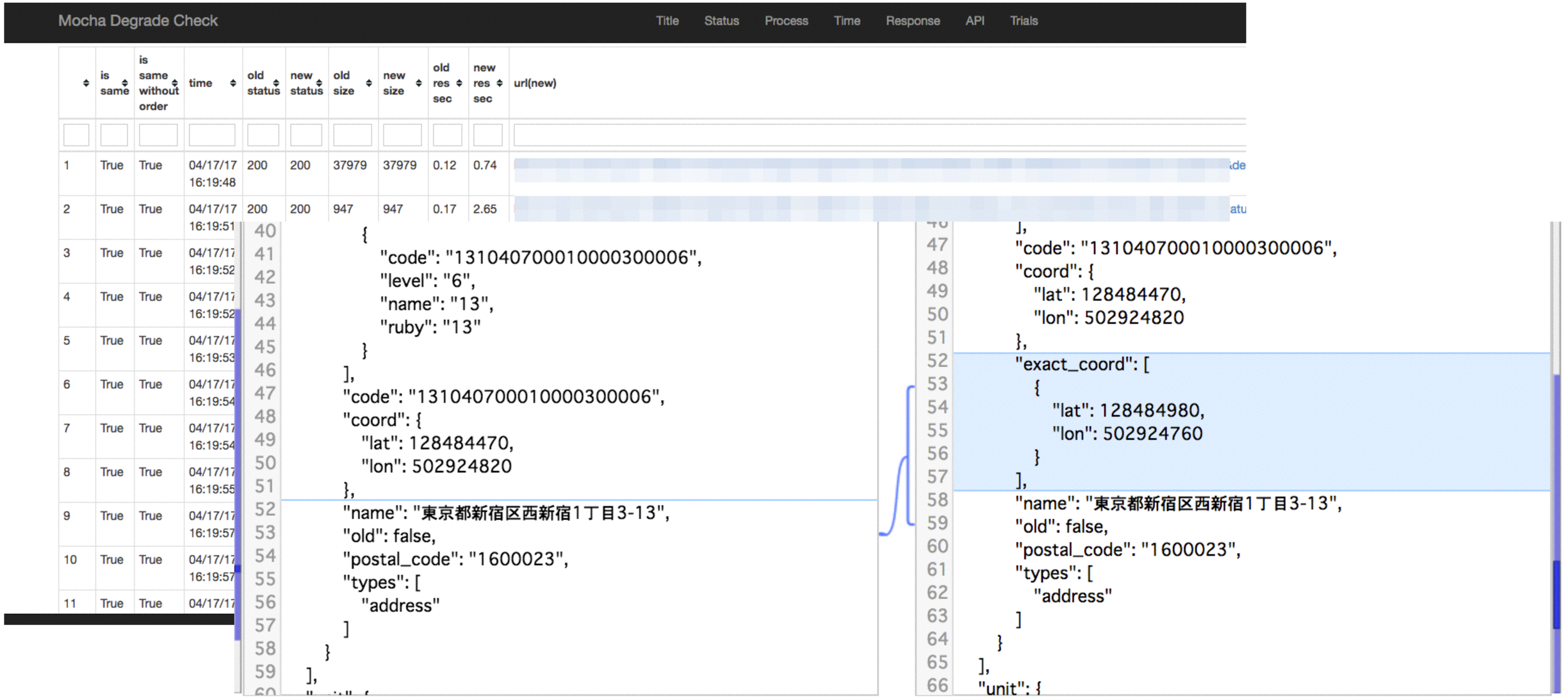
経路探索の
パフォーマンス/品質

日次検証でチェック

行った検証内容

- 各APIが返却するレスポンスBodyをオンプレとAWSで比較し、差分がない事を確認する運用を日次で実施
- JMeterクラスターを使ったロングラン負荷試験

レスポンス比較

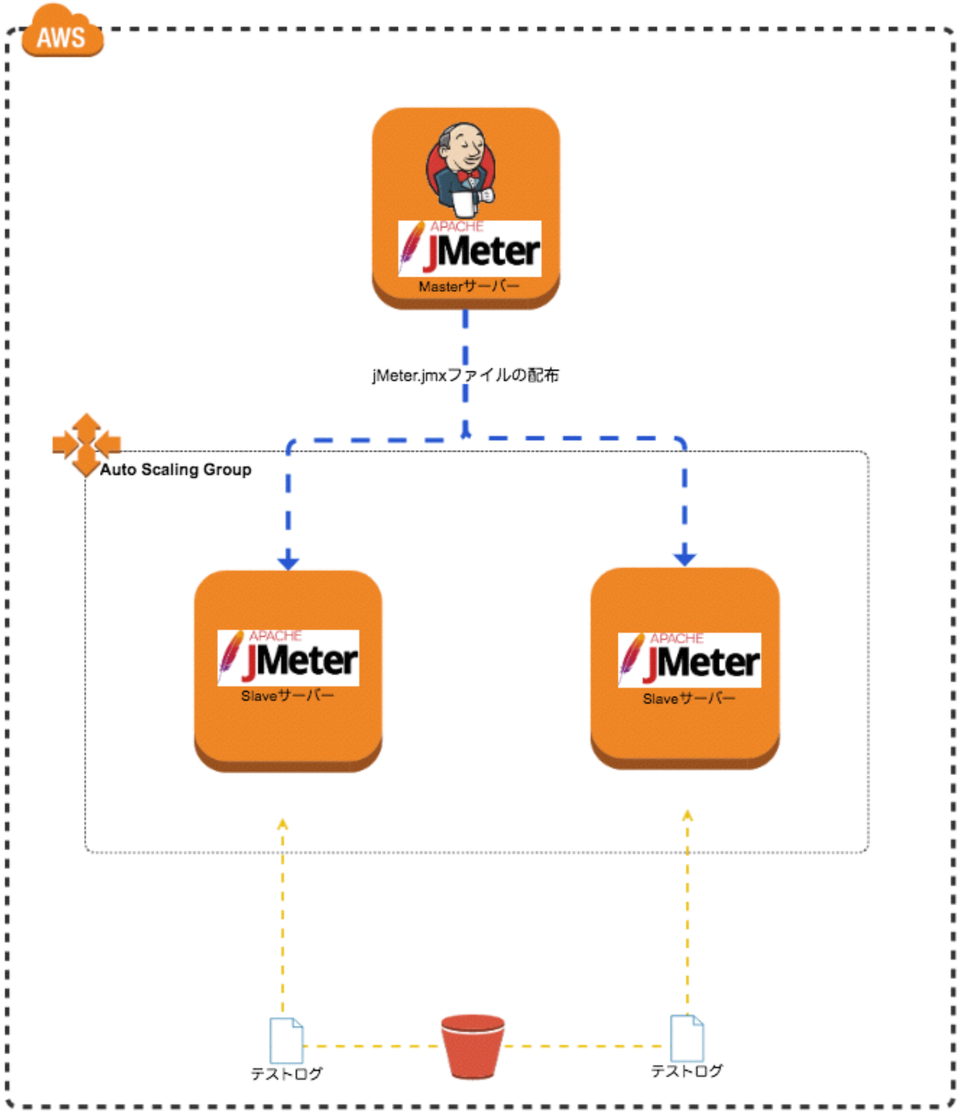


内製ツールを使い、オンプレとAWSでAPIのレスポンスに差分がでていない事を確認

JMeterクラスターを使ったロングランテスト

AutoscalingグループでJMeterクラスターを構築

- 数週間テストリクエストを送信
- 並列度はAutoscalingGroupで指定
- 予期しないエラーを検知

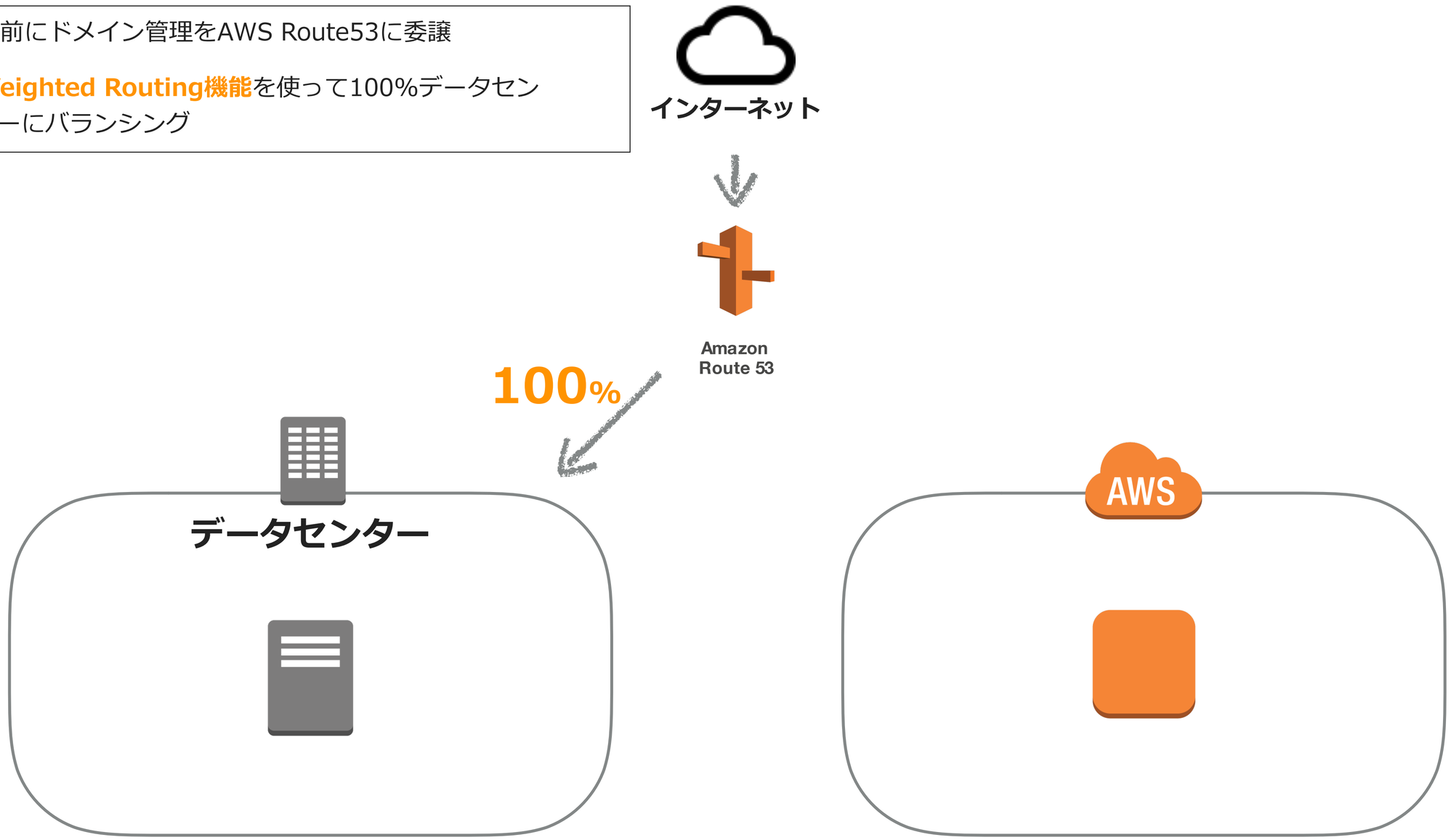


ECS上で稼働させた経路探索サーバーの品質・パフォーマンスに問題がない事を確認！

データセンターからAWSへの切り替え

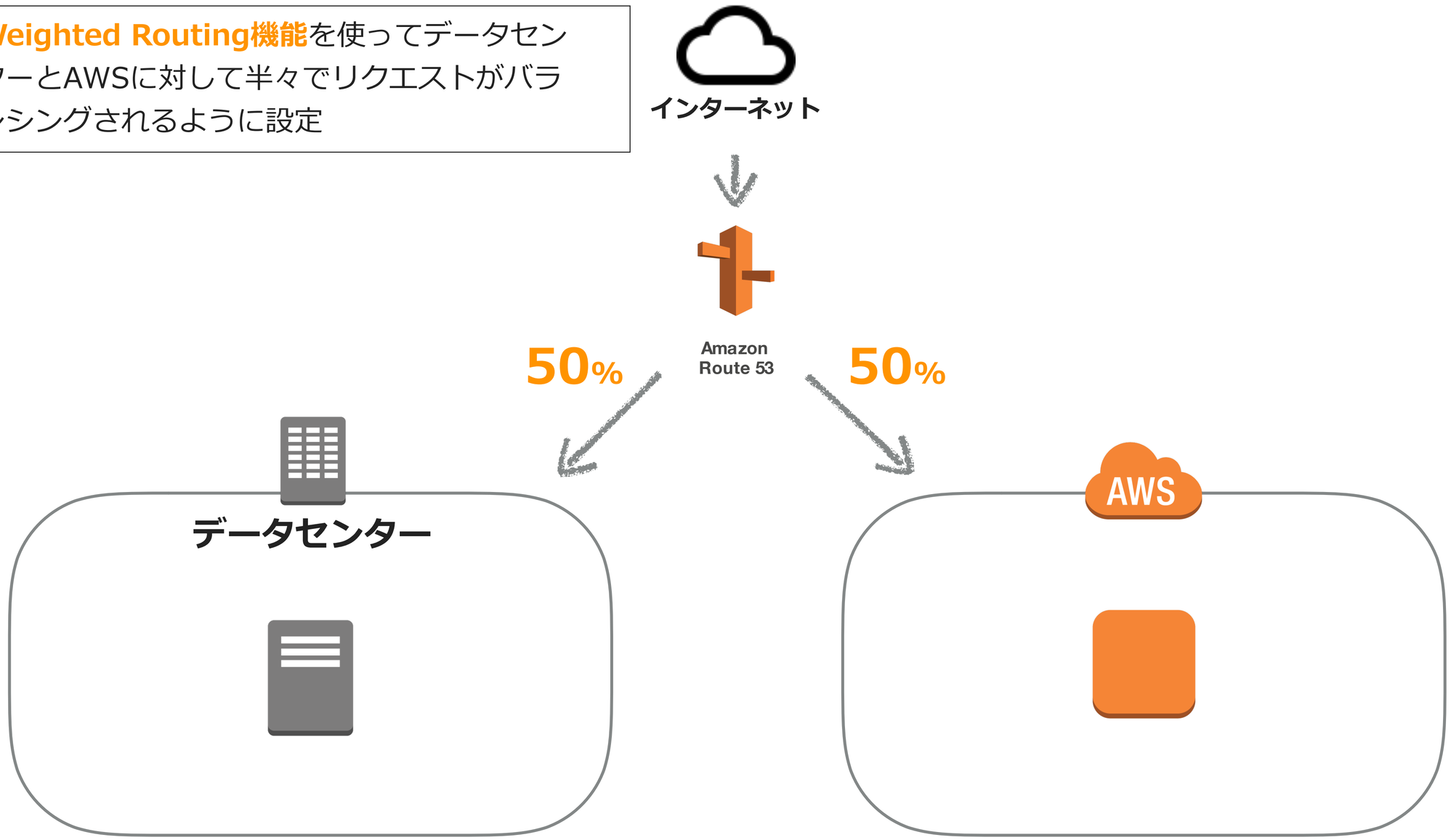
移行前

- 事前にドメイン管理をAWS Route53に委譲
- **Weighted Routing機能**を使って100%データセンターにバランシング



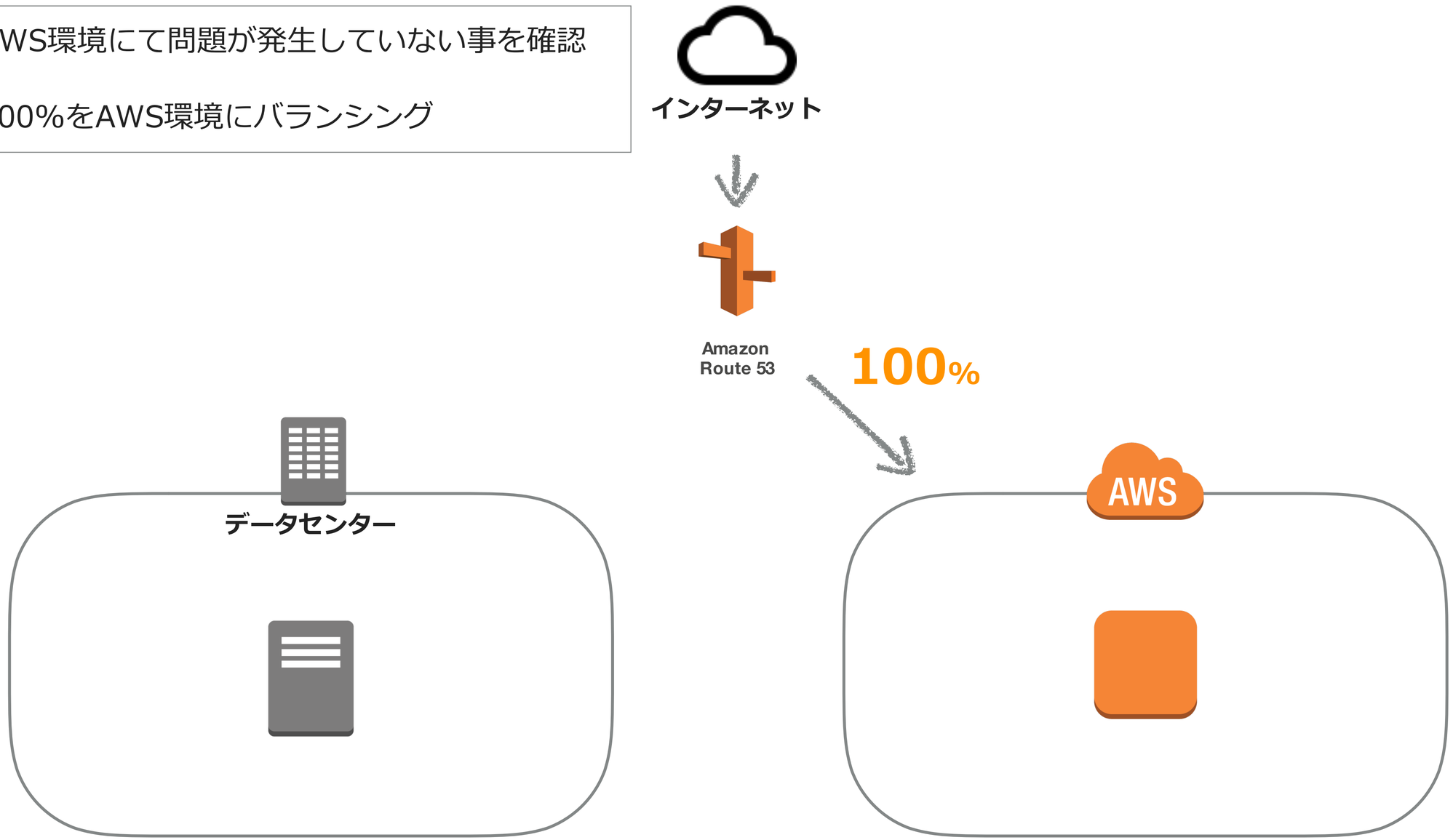
データセンターからAWSへの切り替え

移行日	•**Weighted Routing機能**を使ってデータセンターとAWSに対して半々でリクエストがバランシングされるように設定



データセンターからAWSへの切り替え

完全切替え	•AWS環境にて問題が発生していない事を確認 •100%をAWS環境にバランシング



クラウド移行とコンテナ化が
もたらした効果

インフラ構築手順をCodeした結果

- 環境構築にかかる時間が大幅に短縮
- Docker化によりアプリケーションのポータビリティが上がった

環境構築にかかる時間が短縮 - 新規サーバー構築依頼～サービスインまでのリードタイム

移行前

依頼受理～実施日調整
(タスクの待ち行列があるので着手までのリードタイム)

仮想/ベアリソースの割り当て

OSインストール、環境設定

ミドルウェアのインストール、設定

AP開発者がアプリケーションをDeployして動作を確認

サービスIN（バランサー、クラスターに追加）

リードタイムが最大**2週間**

移行後

ALB、ECSクラスターの生成

CloudFormation用の設定ファイルとECSタスク定義の作成

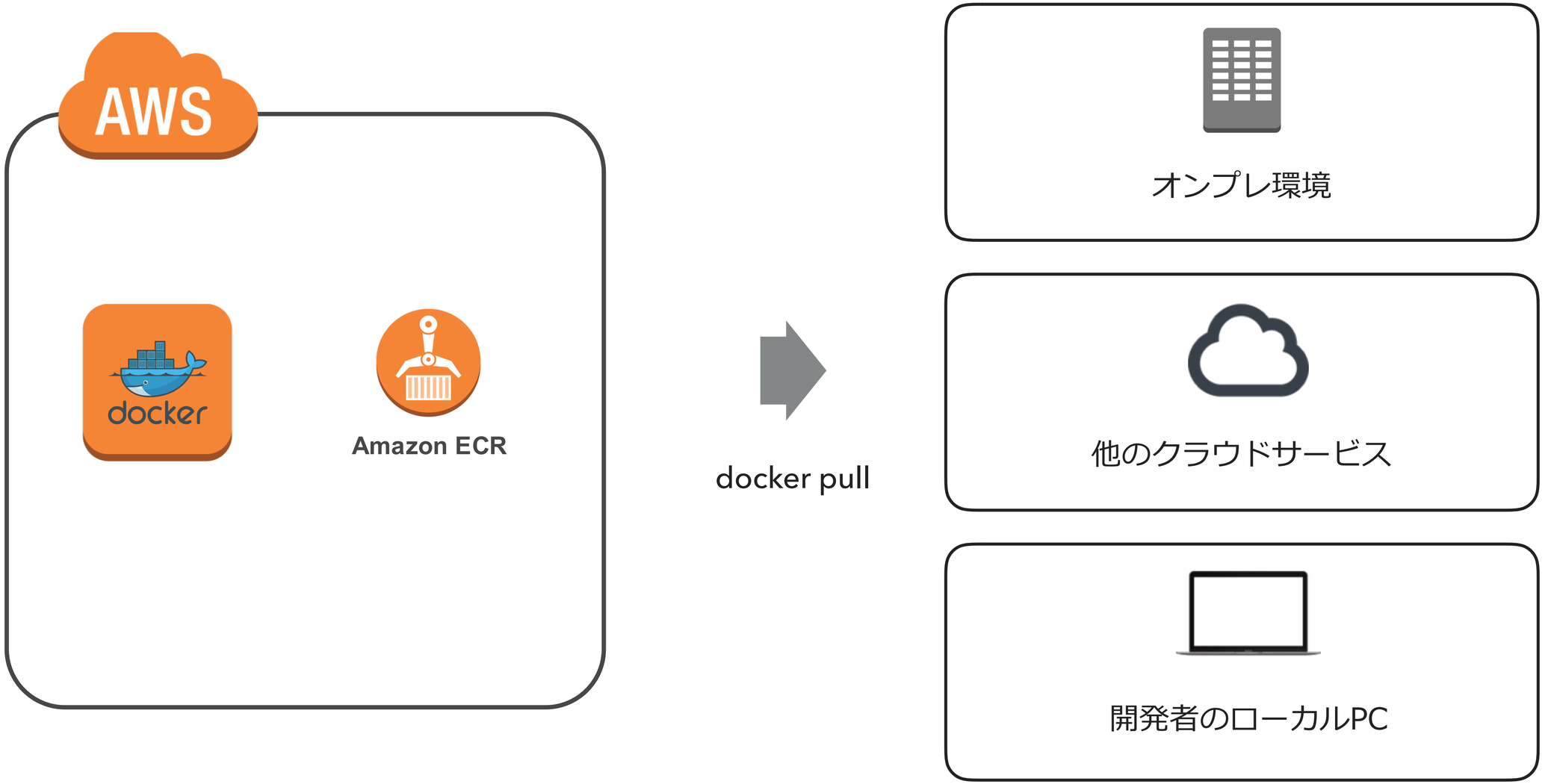
プルリク承認後にCloudformationの**create stack**を実行

内製ツールを使ったAPサーバーのデグレ試験

サービスIN（バランサー、クラスターに追加）

リードタイムが**数時間**に短縮

Docker化によりアプリケーションのポータビリティが上がった

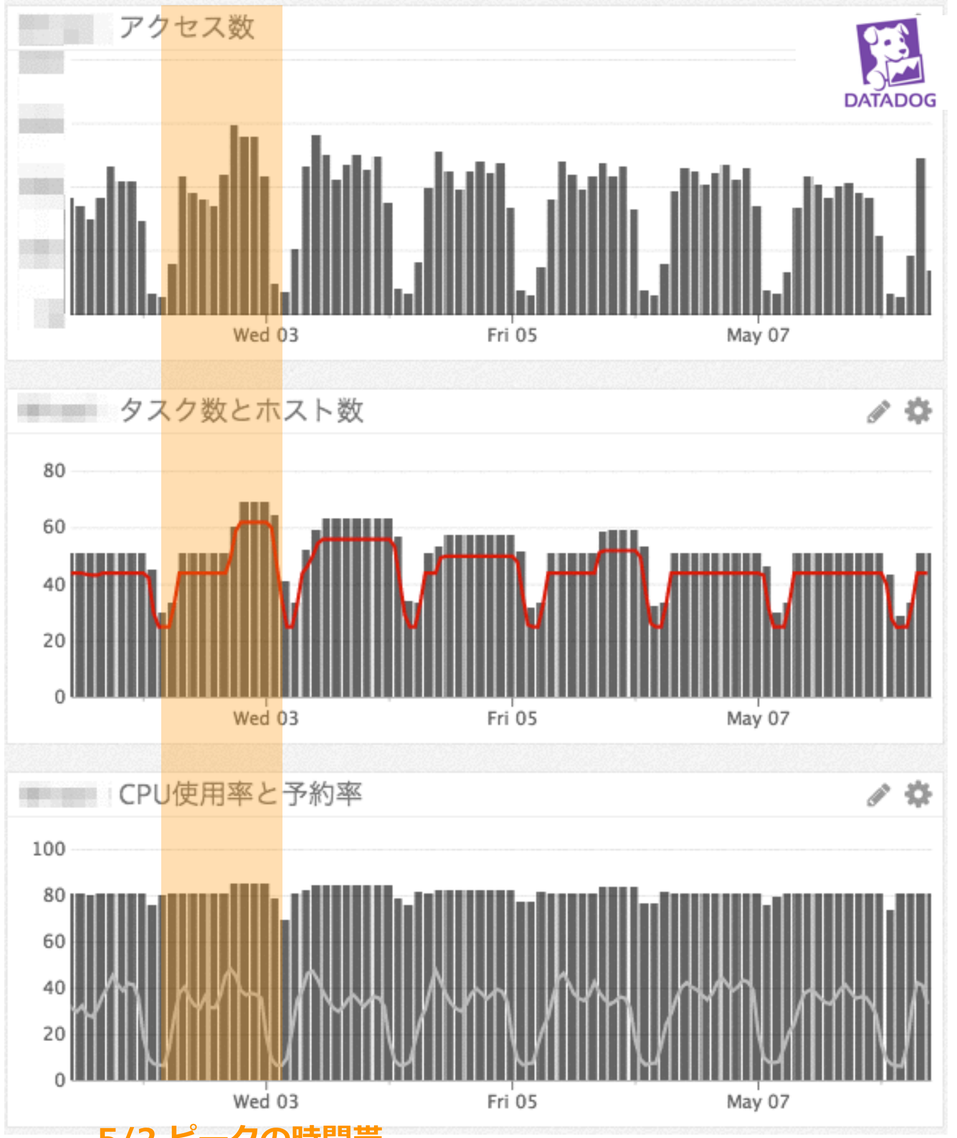


AWS移行で得られた効果

- アクセス数に応じた**オートスケール**
- インフラエンジニアの人的な**運用負担の減少**

アクセス数に応じたオートスケール

- CPU使用率によりコンテナがオートスケール
- GW期間のピーク時間帯(5/2の夕方)でも問題が発生していないので安心！



5/2 ピークの時間帯

インフラエンジニアの人的な運用負担が減った

- AWS移行～1ヶ月は障害調査/対応があった為、運用負担は減らなかった。
- 4月～は安定稼働しており、4月～5月のインフラエンジニアの運用負担はほぼゼロ
  - 繁忙期に備えたインフラ構築
  - 障害発生時のスケールアウト/再起動作業
  - 脆弱性対応の為のミドルウェアアップデート作業

今後やりたいこと

EC2インスタンスコストの最適化

- 現状
  - 移行実施後に数回障害が発生している為、少し多めにサーバーを起動させている。その為、インフラ費用が当初の見込みより高い状態に。
- 今後
  - **必要な台数だけ起動**する運用にシフト
  - リザーブドインスタンスだけでなく、**スポットフリートインスタンス**の並行利用を検討

さらなるコンテナ化

- コンテナ化できていない部分をコンテナ化

めざせ **100%!**

他サービスのコンテナ化とクラウド移行

今後、他のNAVITIMEサービスのクラウド移行を推進します。

第3回テーマ

生産性（プロダクティビティ）

Docker、**Vagrant**、**継続的なデプロイ**にご興味がある方は、是非connpassからご登録ください！

https://minami-aoyama-night.connpass.com/event/58102/

NAVITIME

ご清聴ありがとうございました