

DMMにおける 会員基盤プラットフォームへの AWS導入から活用事例の紹介



株式会社DMM.comラボ 岩崎 磨 / 飯田 涼太 / 西村 政輝

2017.6.2 AWS Summit Tokyo 2017

法務情報

- DMMは第三者の製品・サービスについて、特定の製造者やサービス提供者につき、製品やサービスを評価するものではありません。
- 当社事例は、あくまで当社事案であり各社のシステム・サービス要件等によって、機能、パフォーマンスその他の面で該当しない場合があります。
- 本プレゼンテーションは当社構築したAWSシステムに関する技術者の現時点での感想に基づいています。
- Copyright © 2017 DMM.com Co., Ltd. All Rights Reserved.
無断複製・転載を禁じます。

発表者紹介



IT戦略本部
本部長

I ♥ Infrastructure.



システム本部
チームリーダー

I ♥ Kanazawa.



システム本部
リードエンジニア

I ♥ Scala.

アジェンダ

- はじめに
 - DMMのクラウドへの取り組み状況
- 事例前半
 - 会員基盤フロントエンド改修に合わせてAWS移行した話
- 事例後半
 - 内製基盤アプリケーションのAWS移行におけるマネージドサービス活用例

DMMのサービス紹介



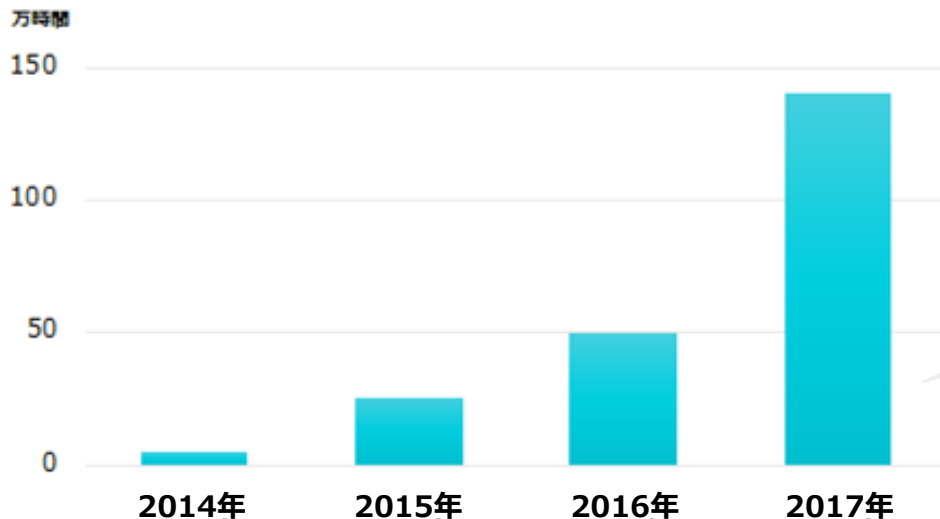
DMMのクラウド利用状況



DMMでのクラウド利用状況推移

- 毎年前年度比率2倍強で利用リソース量増加
- ゲームでの活用から基盤への展開フェーズへ

EC2利用時間



2017年データは
5月時点での予測値

DMMクラウド推進の取り組み



大切にしていること

・ スピード感

何かあっても作ってからごめんなさいするスタイル（DS方式）

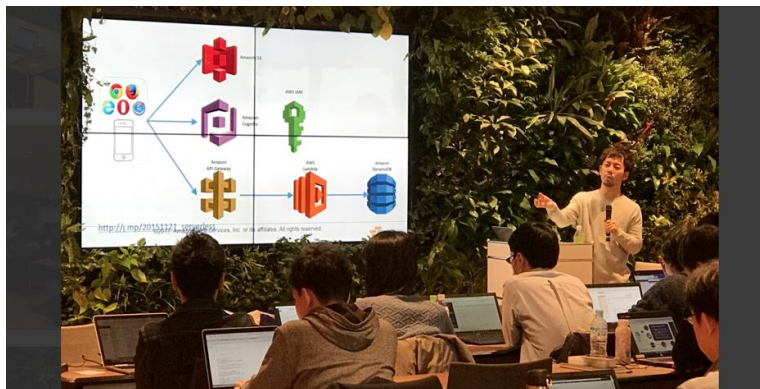
・ 標準化ではなくリファレンス化

ルール（標準化）の押し付けでは活用は広がらない
エンジニアが作った機能を横展開して皆の効率をあげていく軸

勉強会はたくさん開催

エンジニアはみんな勉強会大好き！

新社屋セミナースペースを活用してAWS勉強会を連日開催



開発者向けAWS自由利用環境の提供

DMMのエンジニア・デザイナーはAWS環境を私的利用でも自由に利用できるようにしました。

名付けて **“AWS実弾演習場”**

- DMMの開発者 1 人ひとりにアカウントを発行しAWSサービスが好きなように試せる環境を提供
- コストについては使いすぎ抑止のためにBillingAlertを設定
- 新機能のAWS Organizationsでアカウント作成、BillingAlert設定、権限剥奪については集中的に運用

会員基盤フロントエンド改修に 合わせてAWS移行した話

事例紹介前半
DMM会員機能の一部のAWS移行について



2017.6.2 AWS Summit Tokyo 2017

事例の概要

背景としてオンプレ上で稼働する
モノリシックなサービス基盤が存在。

急激な事業拡大による運用コスト肥大化が課題

もっと事業をリードできるスピード感を求めて
マイクロサービス化を進めていく中で
AWSを利用したら解決が捗りました！

アジェンダ

- 当時基盤の課題点
- 私たちの解決事例
- まとめ



当時の基盤の課題点



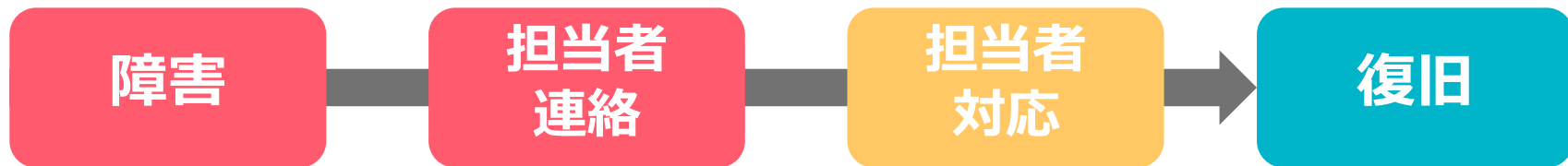
当時の基盤の課題点

基盤として、まだ改善の余地が見込める

1. もっと運用をツール化できる
2. もっと検証精度を上げられる
3. 検証時間も短縮できる

課題点の具体例1

インシデントやイベント対応など
運用がツール化できておらず属人化しがち
→ 対応が後手に回り 復旧時間が長期化



課題点の具体例2

サーバの維持管理を含めた費用対効果の観点から
検証環境と本番環境の構成が一部異なり

→ 検証環境では再現しない場合がある



課題点の具体例3

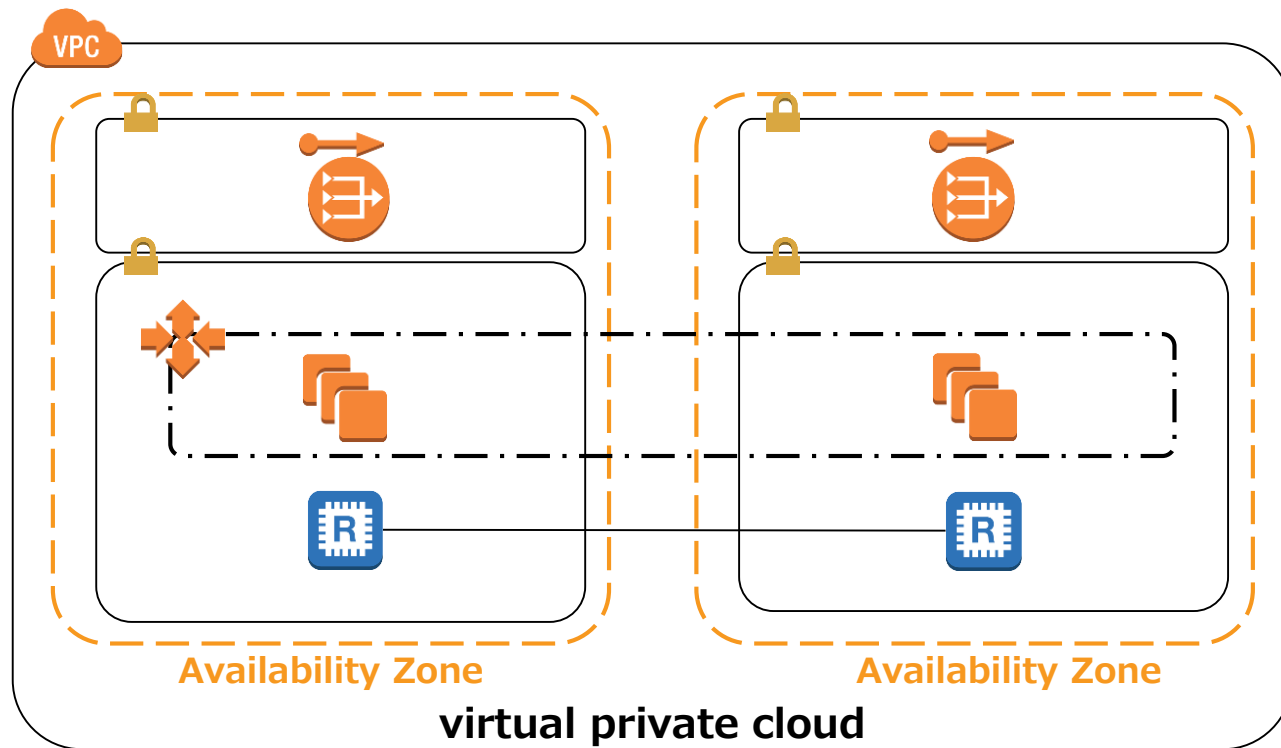
モノリシックかつ複雑な基盤となっており
変更は小さくても、影響が小さいとは限らない
→ 常にじっくり時間をかける保守的な対応



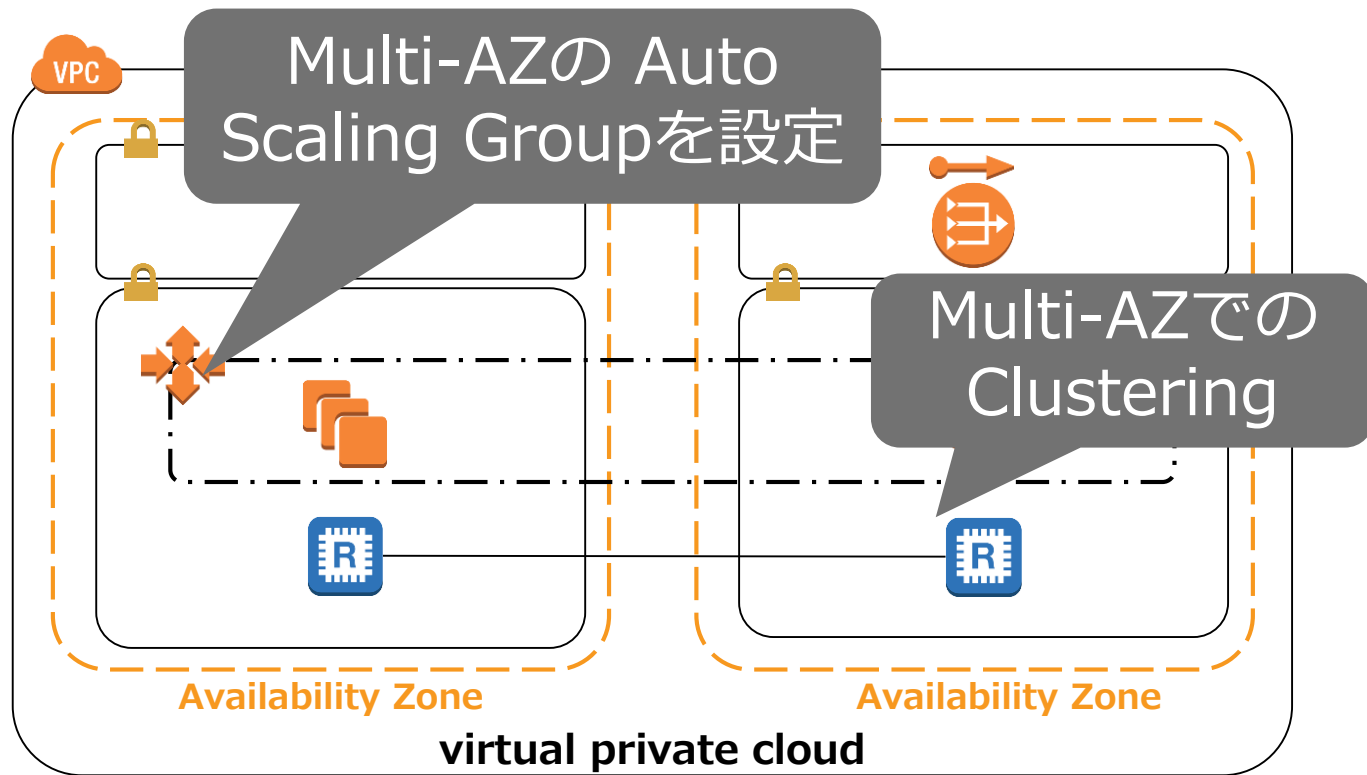
私たちの解決事例



課題点1: 復旧時間の短縮に向けて



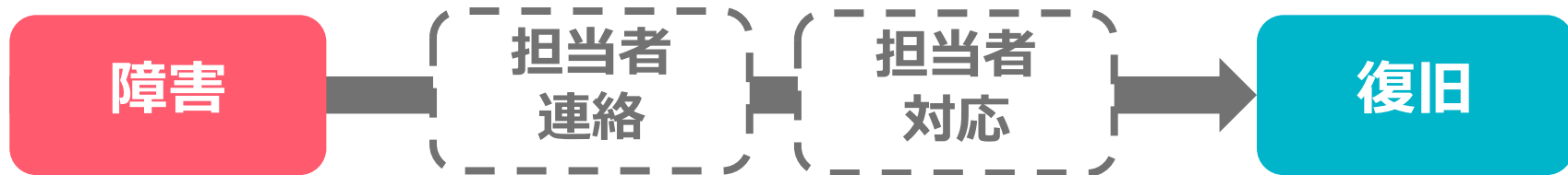
課題点1: 復旧時間の短縮に向けて



課題点1: 復旧時間の短縮に向けて

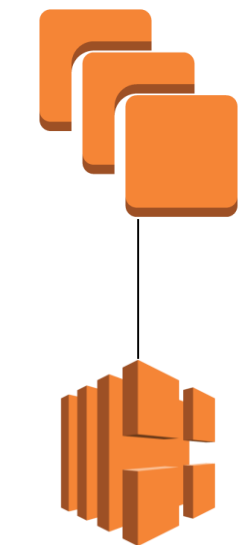
基本的に人の手を介さずに
設定に従った自動復旧を設定することで

→ Multi-AZ高可用性とAWSの
マネージドサービスにより復旧までを時短化

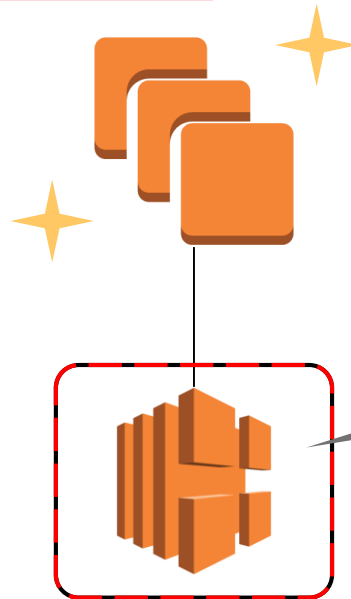


課題点2: リリース前の検証精度向上に向けて

必要な時だけ、全く同じ構成の環境を用意



ユーザー用

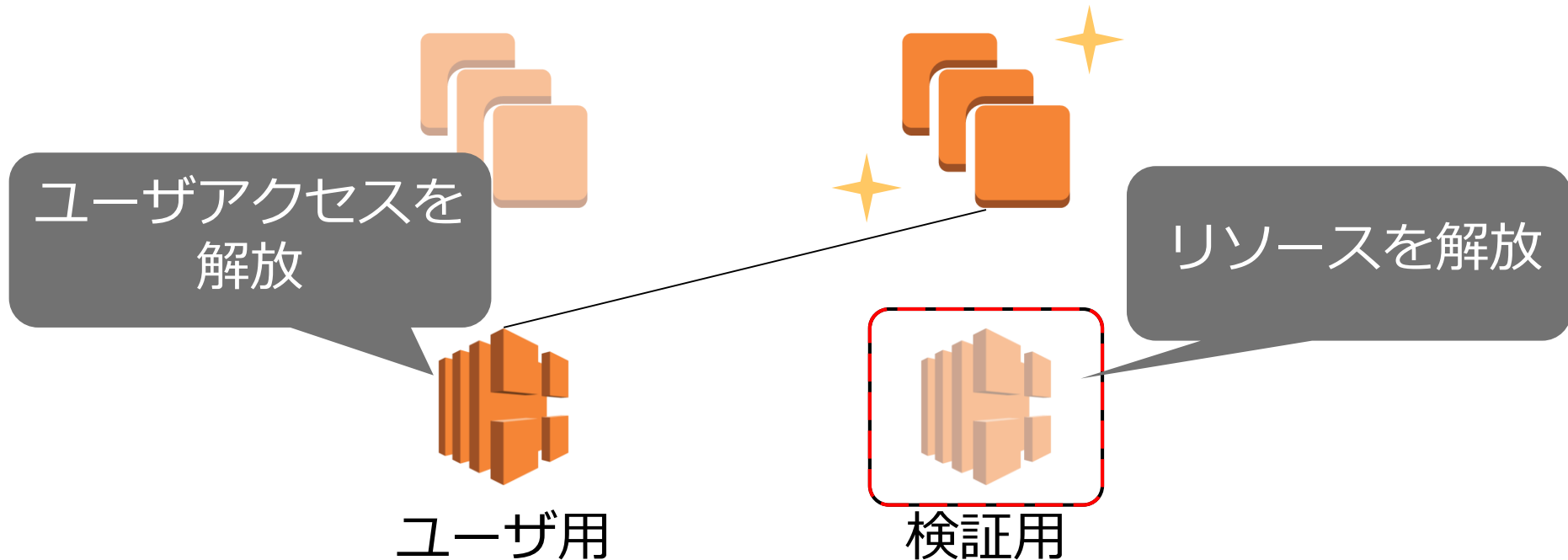


検証用

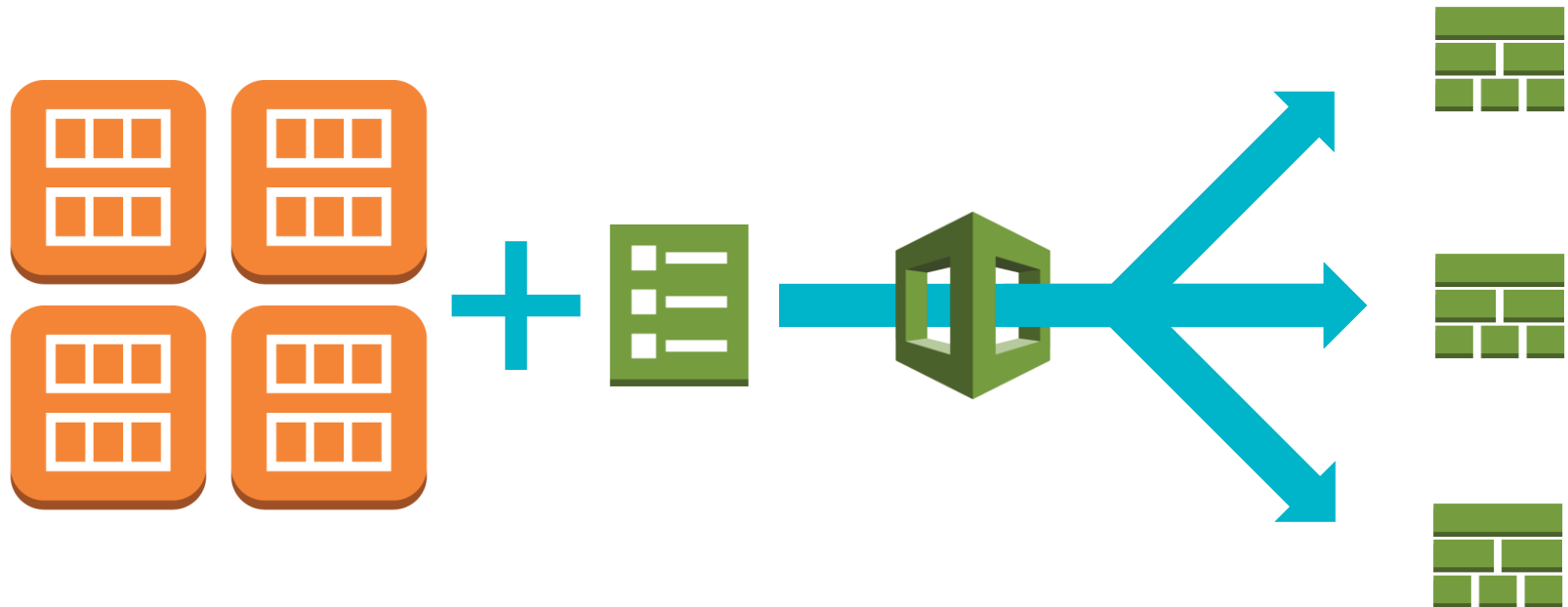
関係者ONLY

課題点2: リリース前の検証精度向上に向けて

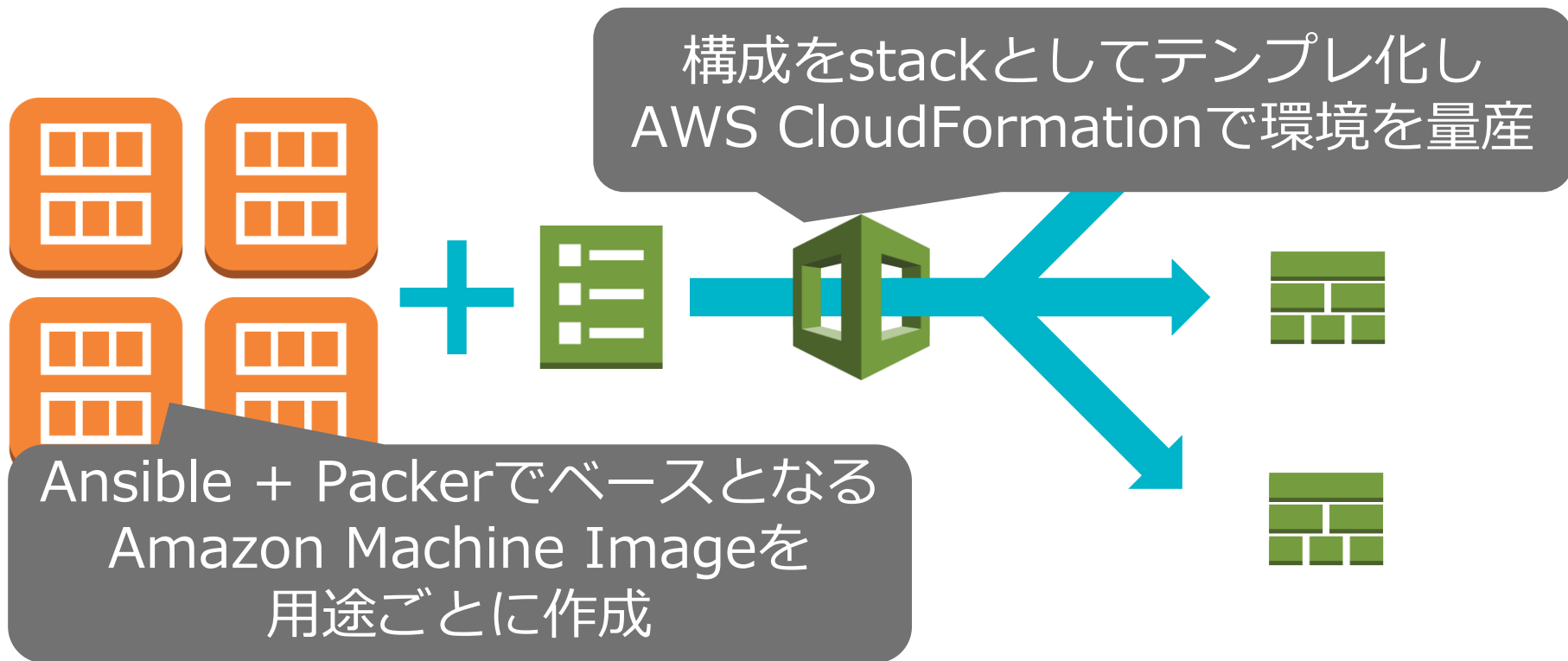
検証が済んだらアクセスもリソースも解放



課題点2: リリース前の検証精度向上に向けて



課題点2: リリース前の検証精度向上に向けて



課題点2: リリース前の検証精度向上に向けて

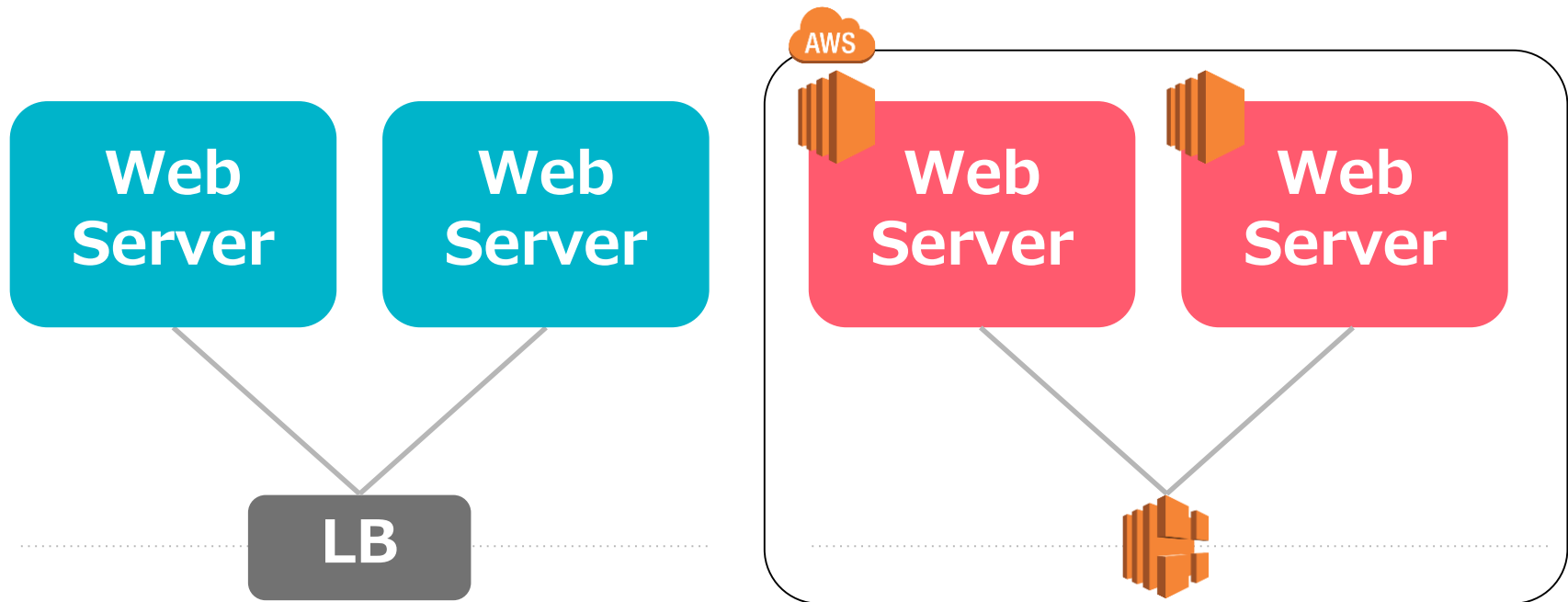
クラウドのオンデマンドなリソースに
AWS CloudFormationをかけ合わせることで
サーバを遊ばせずに、いつも同じ状態で検証可

→ 無駄もなく効率的に、事前検証精度を向上



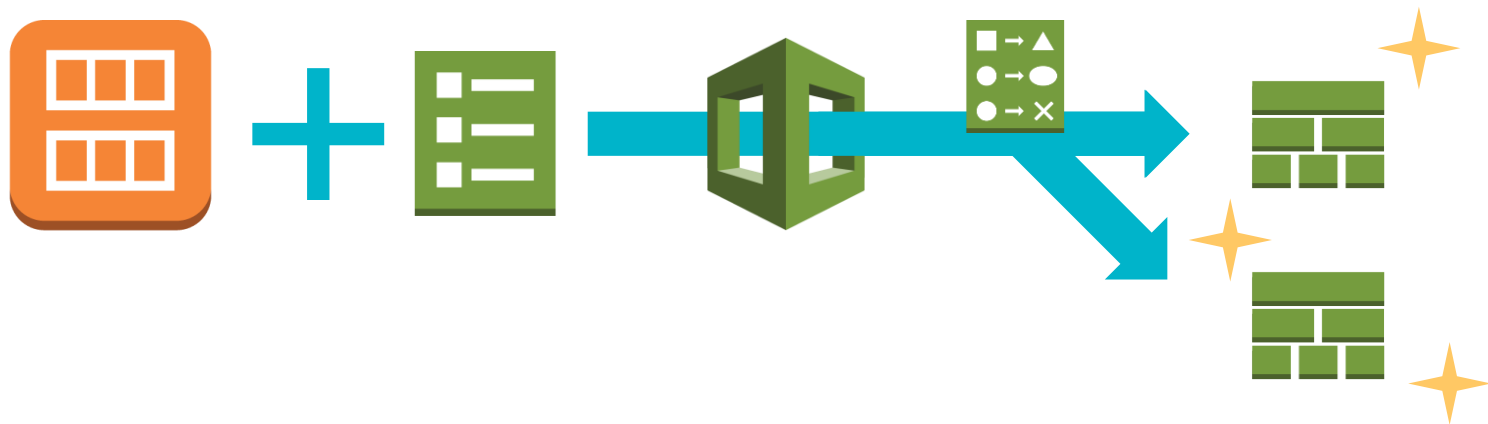
課題点3: 検証時間の短縮に向けて

マイクロサービスとして分離し、影響範囲を明確化



課題点3: 検証時間の短縮に向けて

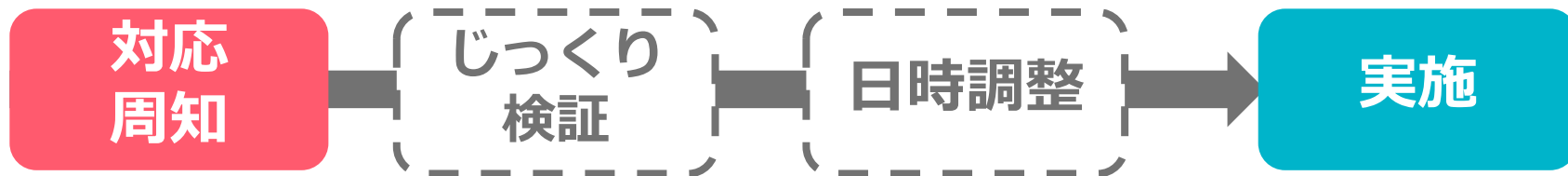
マネージドサービスを利用した
変更管理により漏れなく対応しつつ
新規に生成した環境でスピーディな検証が可能に



課題点3: 検証時間の短縮に向けて

AWSを利用することでマイクロサービスとして
影響範囲を明確にするだけでなく

その時々に応じた最適なリソースを
スピーディかつ正確に構成して検証可能に



まとめ



まとめ

- **マネージドサービスの活用**
 - Multi-AZの高可用性で障害を抑制
 - 障害時の自動復旧で時短化
- **潤沢なリソースとCloudFormationの利用**
 - 同一環境を無駄なく生成・破棄
 - 検証精度を向上しつつ、時間も短縮

内製基盤アプリケーションの AWS移植における マネージドサービス活用例

事例紹介後半

DMMの高ワークロードを支える API Gateway / OAuth2認可基盤



2017.6.2 AWS Summit Tokyo 2017

アジェンダ

- プロジェクト概要・構成図
- AWS移行で得られたもの
- AWS移行で注意したところ
- スケーリングと性能見積もり
- サービスディスカバリ

プロジェクト概要・構成図

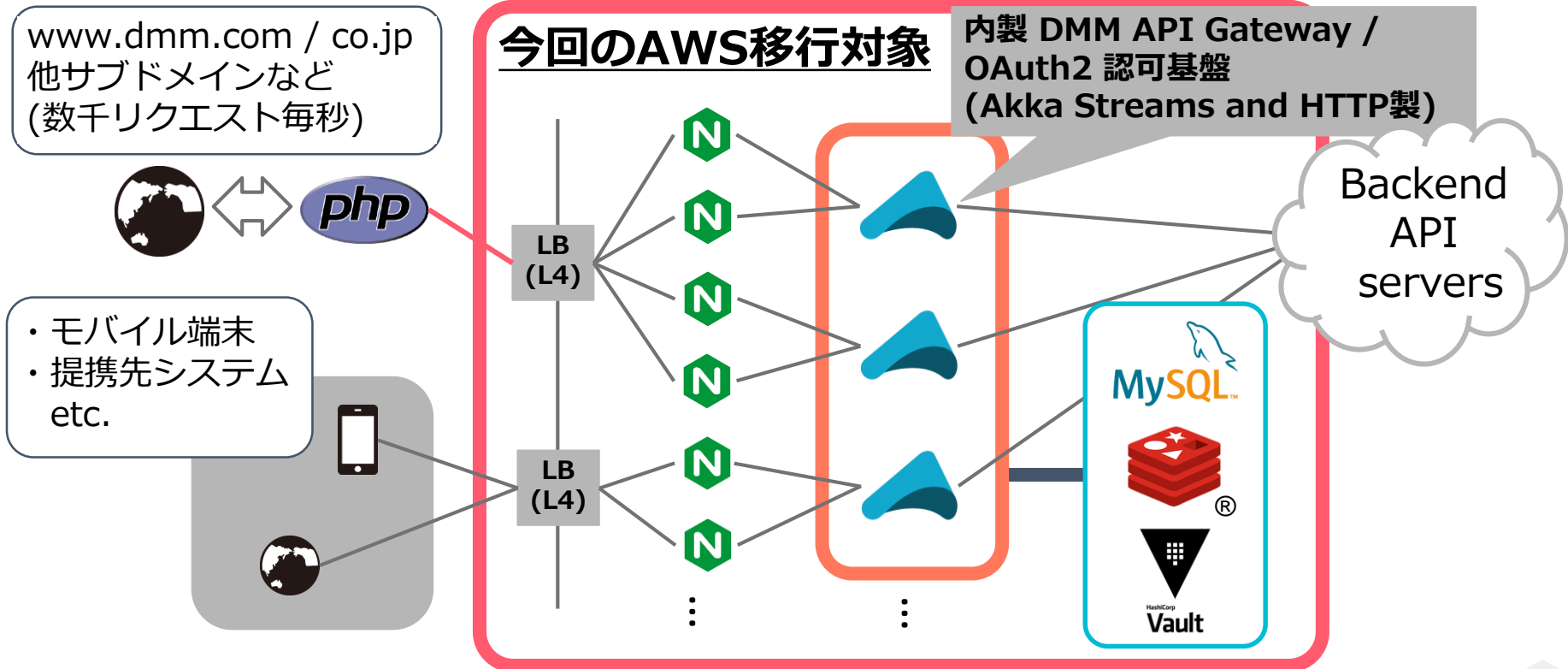


プロジェクト概要

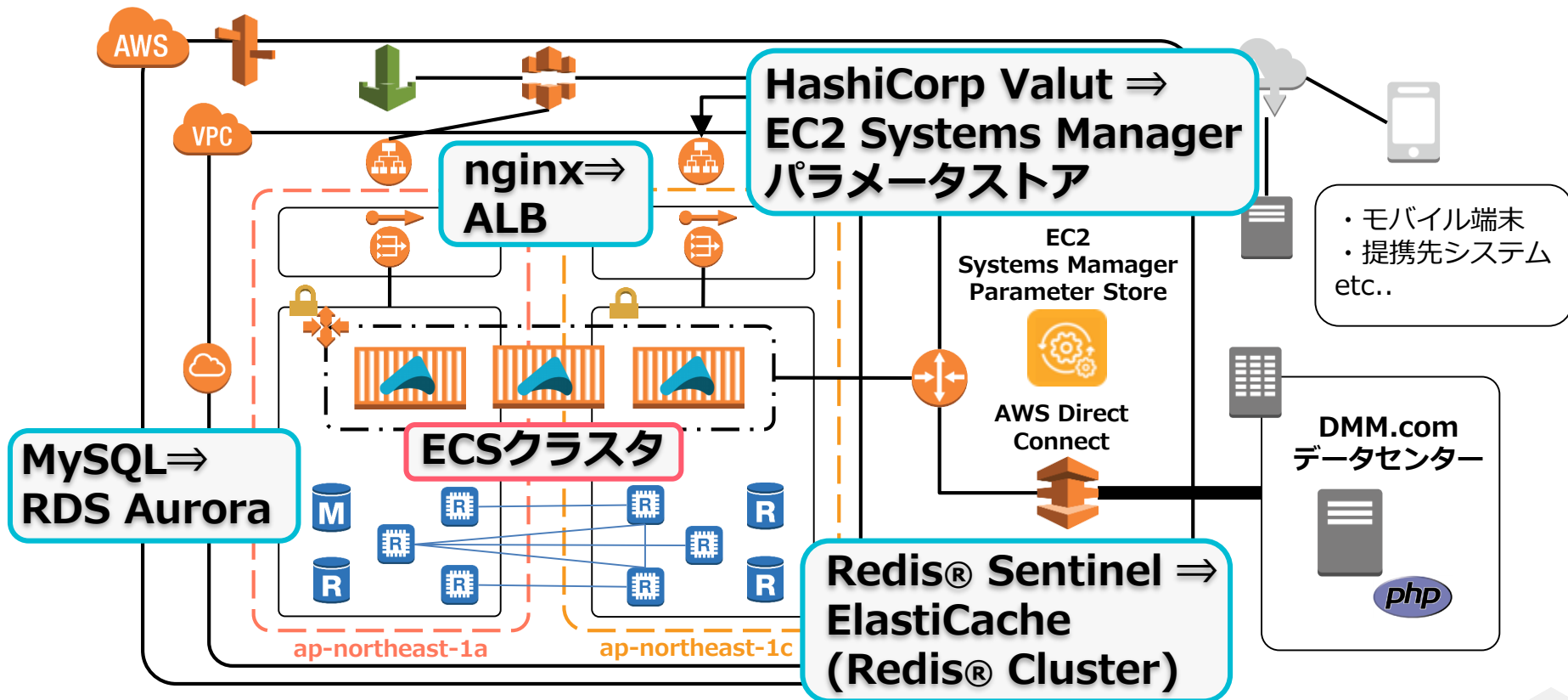
オンプレミスで製造・構築した内製 DMM API GatewayをAWSへ移行するプロジェクト

- 最重要なのは、**耐障害性**
 - DMM.comのプラットフォーム基盤の窓口となるシステム
 - 障害発生時の影響範囲が大きい
- APIの窓口としての**安定性**と**低いレイテンシ**も必要
 - オンプレミス側との通信にAWS Direct Connectも導入する

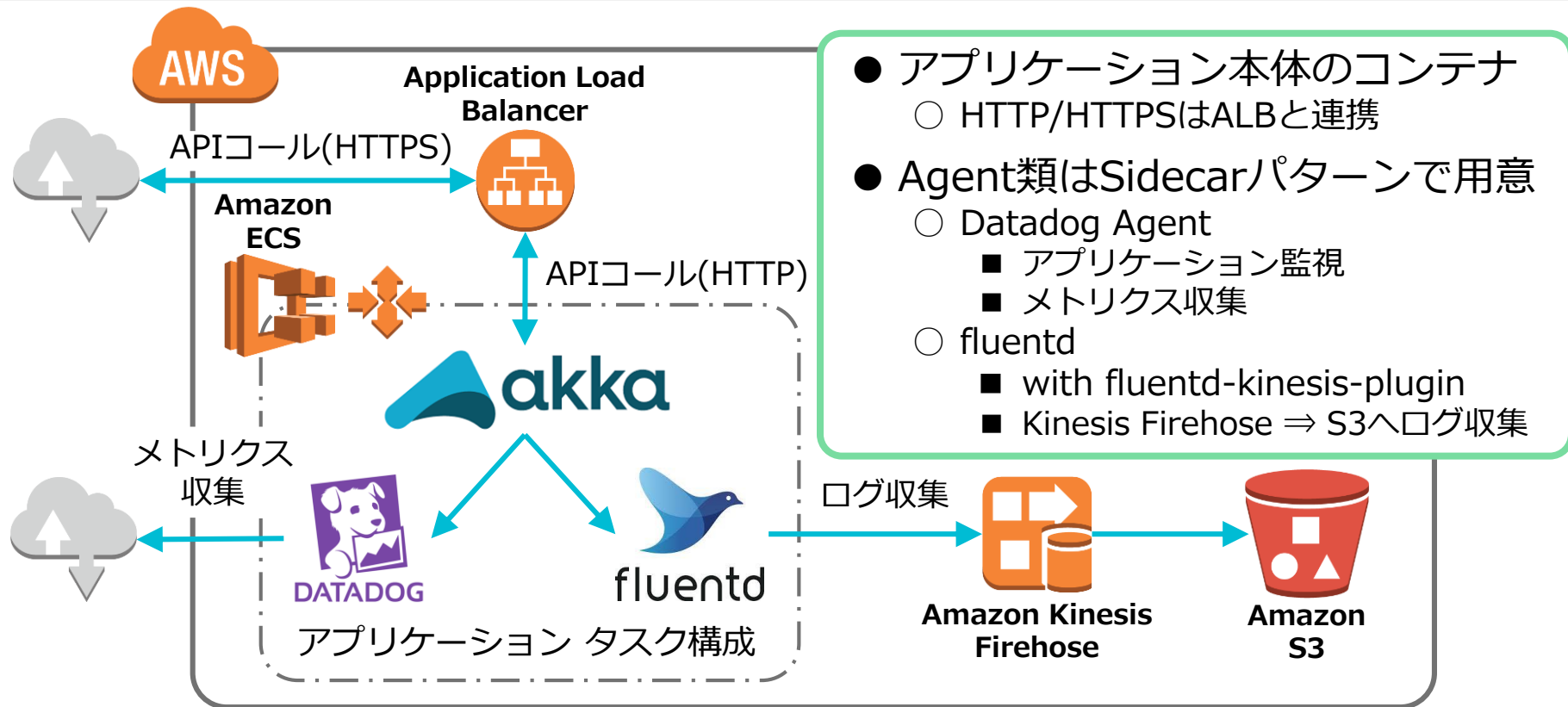
オンプレ時の構成図



AWS移行後の構成図



ECSタスクと関連するコンポーネント



AWS移行で得られたもの



AWS移行で得られたもの

1. 堅牢なインフラ環境の入手

- 複数データセンター級の冗長性(Multi-AZ)
- インスタンス・プロセス障害時のオートヒーリング

2. ミドルウェア運用からの解放

- 手段：フルマネージドサービスの活用
 - ALB+ACM、ECS、RDS、ElastiCache、Kinesis Firehose
- **EC2 Systems Manager パラメータストア**
↑ 採用例少ないと思うので次で紹介します!
(他はありふれてるサービスのため省略)

EC2パラメータストアでできたこと

- アプリケーション用のシークレット値の取得・格納
- 管理コンソールや、AWS CLI、AWS SDKを用いての操作

The screenshot displays the AWS Management Console interface for creating a parameter. On the left, the navigation pane shows 'パラメータストア' (Parameter Store) highlighted with a red box and a red arrow pointing to it. The main content area shows a table of existing parameters and a 'パラメーターの作成' (Create Parameter) form. The form includes fields for '名前*' (Name), '説明' (Description), and 'タイプ' (Type). The '値' (Value) field is highlighted with a red box. A red callout bubble points to the '値' field with the text: 'AWS KMSと連携し、シークレット値を暗号化できた。' (Can be encrypted with AWS KMS).

ここ!

※ 設定のイメージ例

AWS KMSと連携し、シークレット値を暗号化できた。

EC2パラメータストアで嬉しかったこと

シークレット値管理のミドルウェア運用等が省力化できた！

省力化対象	before (当社オンプレミス)	after (AWS)
ミドルウェア運用	HashiCorp Vault (HA構成) HashiCorp Consul (クラスタ構成) nginx	不要 (フルマネージド)
SSL証明書管理	必要	
ACL設計	HashiCorp Vault固有のACL	AWS IAMに統合

AWS移行で注意したところ



クラウド環境利用における注意点

1. 動的なスケーリングを前提としたElasticな構成

- AWS移行後は、インスタンス利用料が従量課金
⇒ 節約のためのスケーリングを行うことにした

2. 動的なネットワーク割当への対応

- (例)アプリの設定変更をオートスケール対象全台にプッシュしたい
⇒ AWSのサービスディスカバリ機能の活用で対応
 - 内部DNS
 - AWS CLI/SDK

ElastiCacheのレプリケーション遅延対策

懸念

- Redis® (Master-Slave構成) のレプリケーション遅延
 - 局面 : DMM API Gatewayへの連続リクエスト
 - 例: (Write) 認可要求 → (Read) API利用時の認可判定

対策

- ElastiCache Redis® Clusterを採用した上で、Redis® Clusterへの参照・更新をプライマリノードのみに限定
 - 実質的なMulti-Master構成として活用
 - メリット : レプリケーション遅延の考慮が不要
 - 負荷対策 : Redis® Clusterのシャーディングで分散

スケーリングと性能見積もり

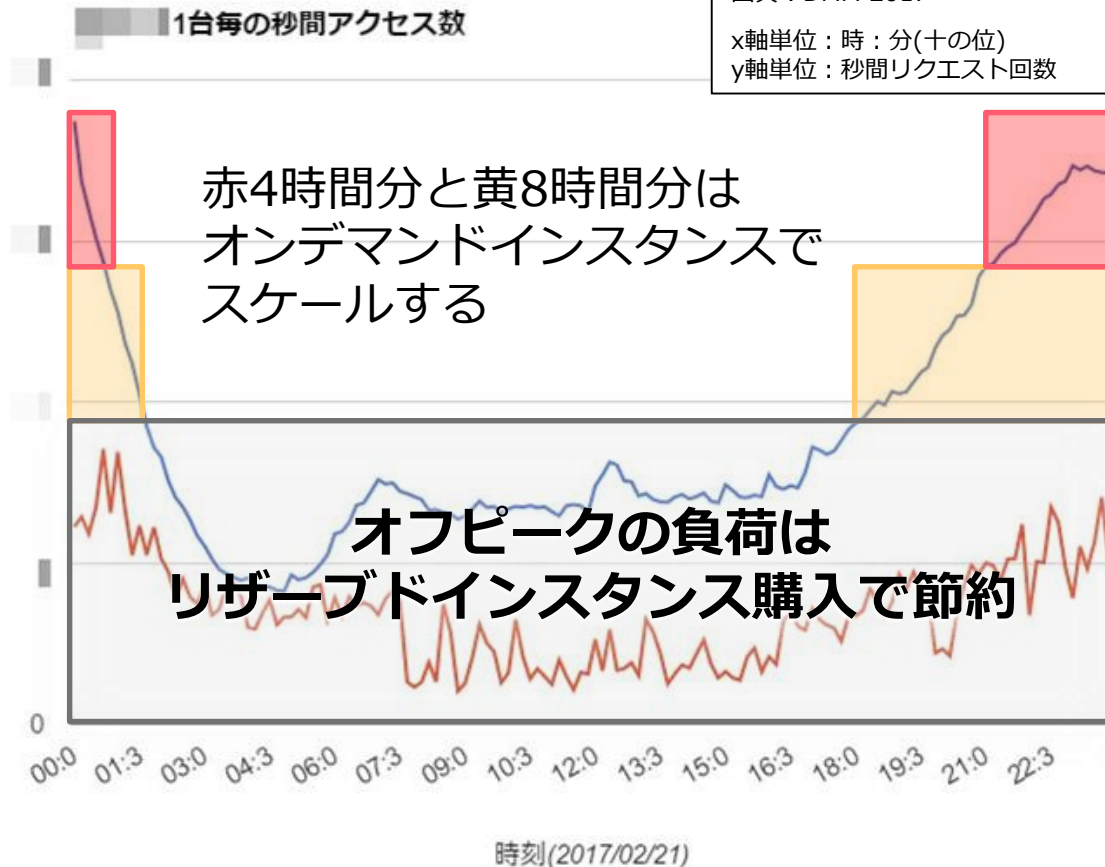


オートスケーリング 戦略



DMMのサービスの特性として、オートスケールはスケジュールベースを基本として問題ない。

主要商材がエンターテインメントということもあり、ピークタイムが夜間で不変であるため。



オートスケール設定で注意したこと

- スケールアウト時の注意点
 - 主としてスケジューラベース、副として負荷ベースを採用
 - 懸念：負荷ベースではインスタンス起動やウォームアップが後追い
 - イベント等のスケールリングだけ個別対応
- スケールイン時の注意点
 - ECSは**ドレイン後のEC2インスタンスを自動でターミネートしてくれない...**
 - 現状(2017.6.2現在)、Lambda等で仕組みの構築が必要！
 - 補足：オートスケールインはDMMの場合早朝時間帯
→ ※社員就寝中・・・.zzZ

ベンチマーク概要

観点

- リクエスト数ごとのレスポンスタイムはどの程度か
- インスタンスのスケールリングに応じてキャパシティもスケールするか

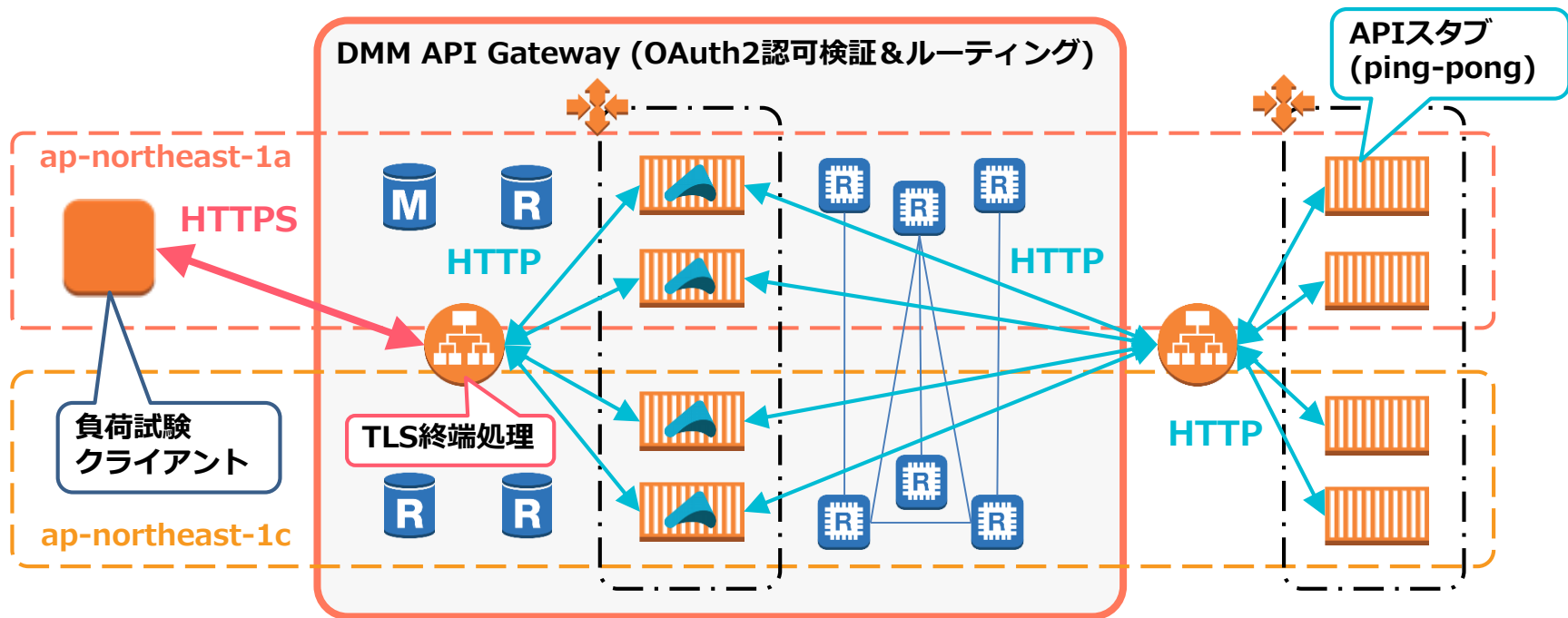
条件

- 下記インスタンスでの秒間1~10,000リクエスト時のレスポンスタイム

コンポーネント	インスタンスサイズ	台数
ECS	c4.xlarge (4vCPU/8GB)	4,8,12台で それぞれ確認
RDS (Aurora MySQL)	db.r3.xlarge (4vCPU/30.5GB)	4台 (M1・RR3)
ElastiCache (Redis® Cluster 3シャード)	cache.r3.large (2vCPU/13.5GB)	6台 (M3・RR3)

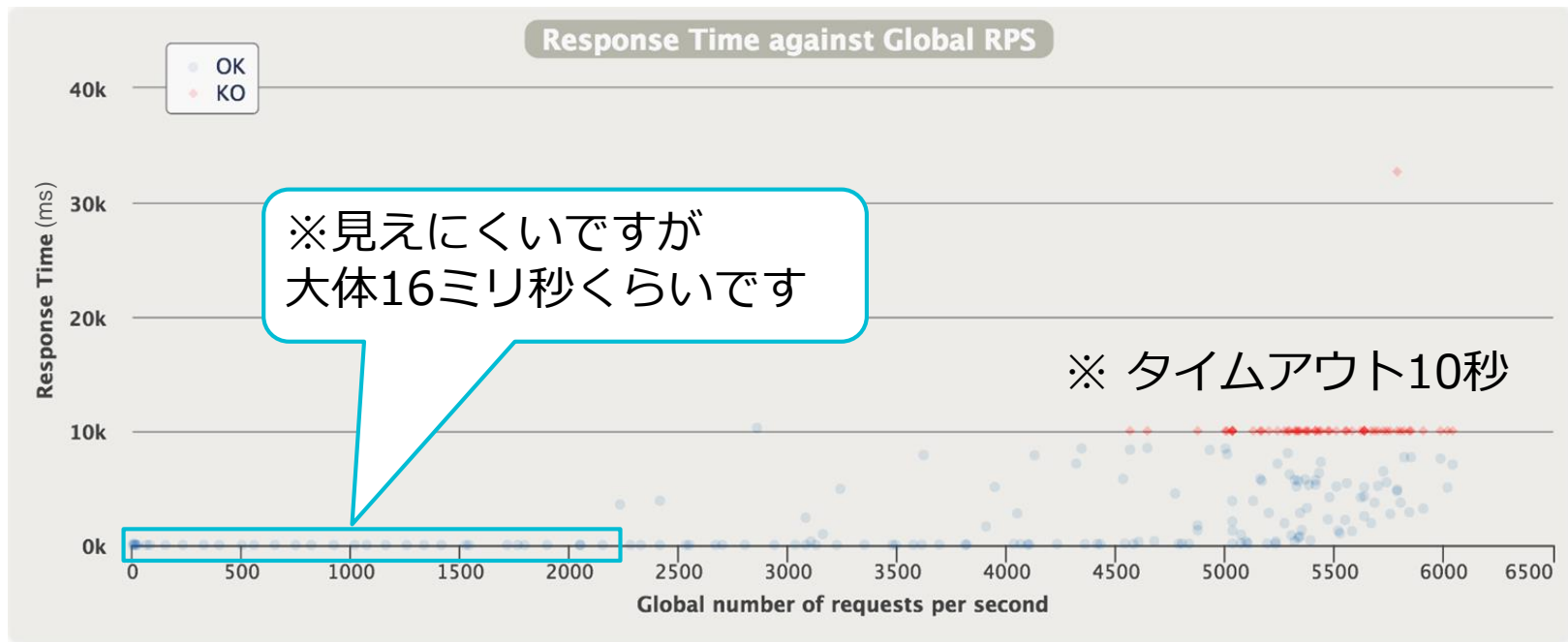
ベンチマーク構成図

DMM API Gatewayを中心に負荷試験クライアントとAPIスタブを用意



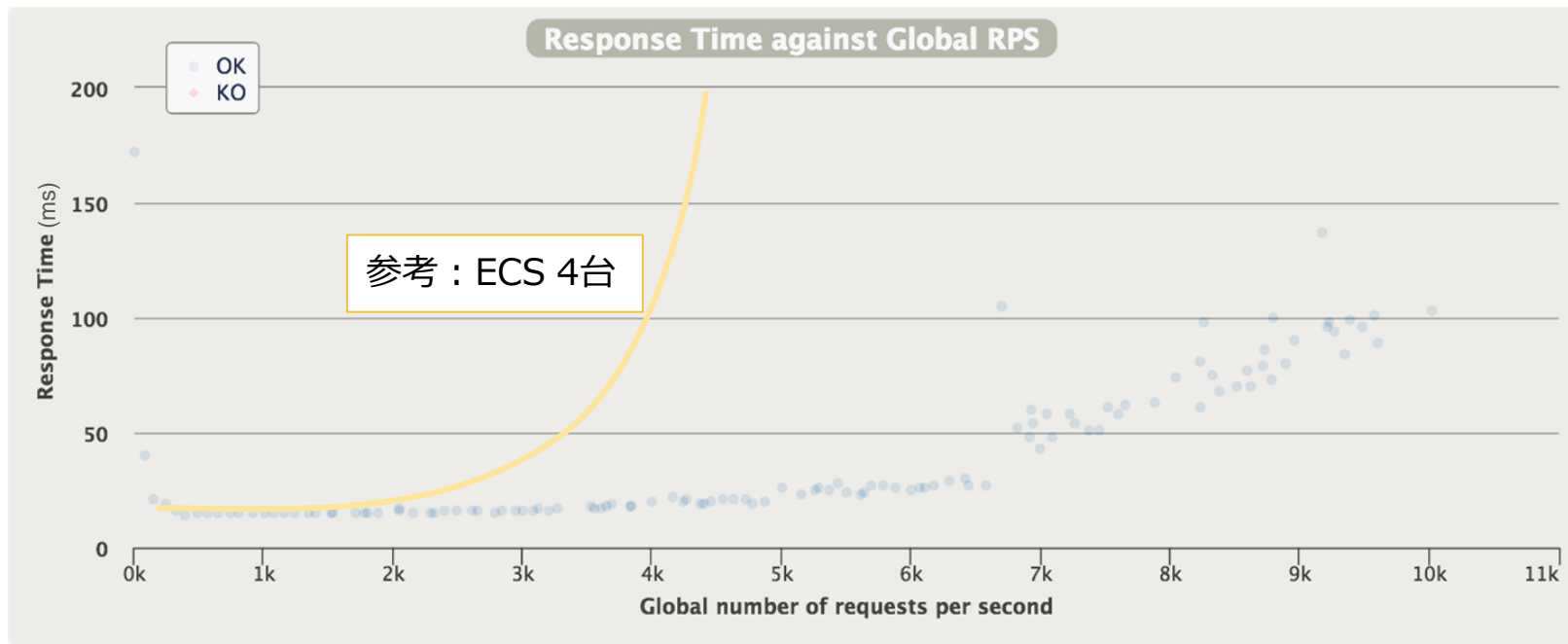
ベンチマーク結果(ECS 4台)

毎秒4,500リクエストが安定限界だったがオフピークはこれで十分そう



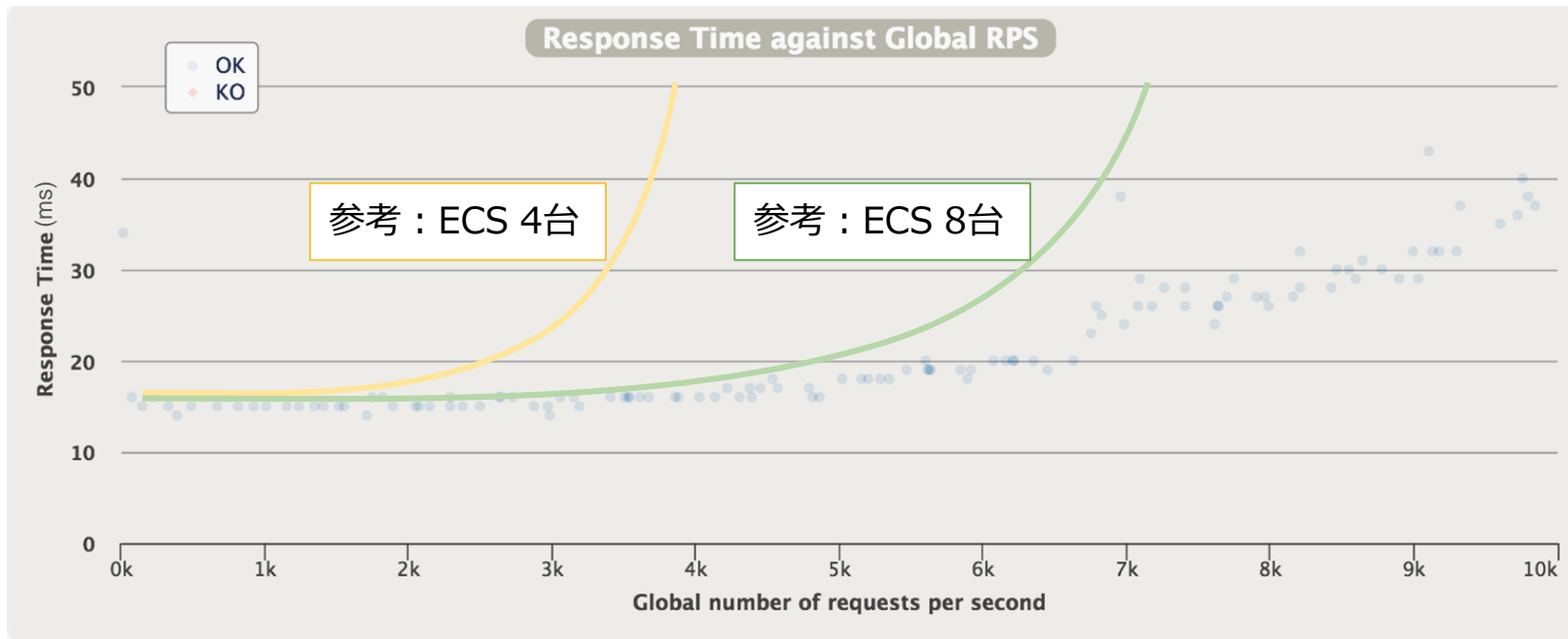
ベンチマーク結果(ECS 8台)

毎秒6,500リクエストまで良好な成績 & 毎秒1万リクエストにも対応



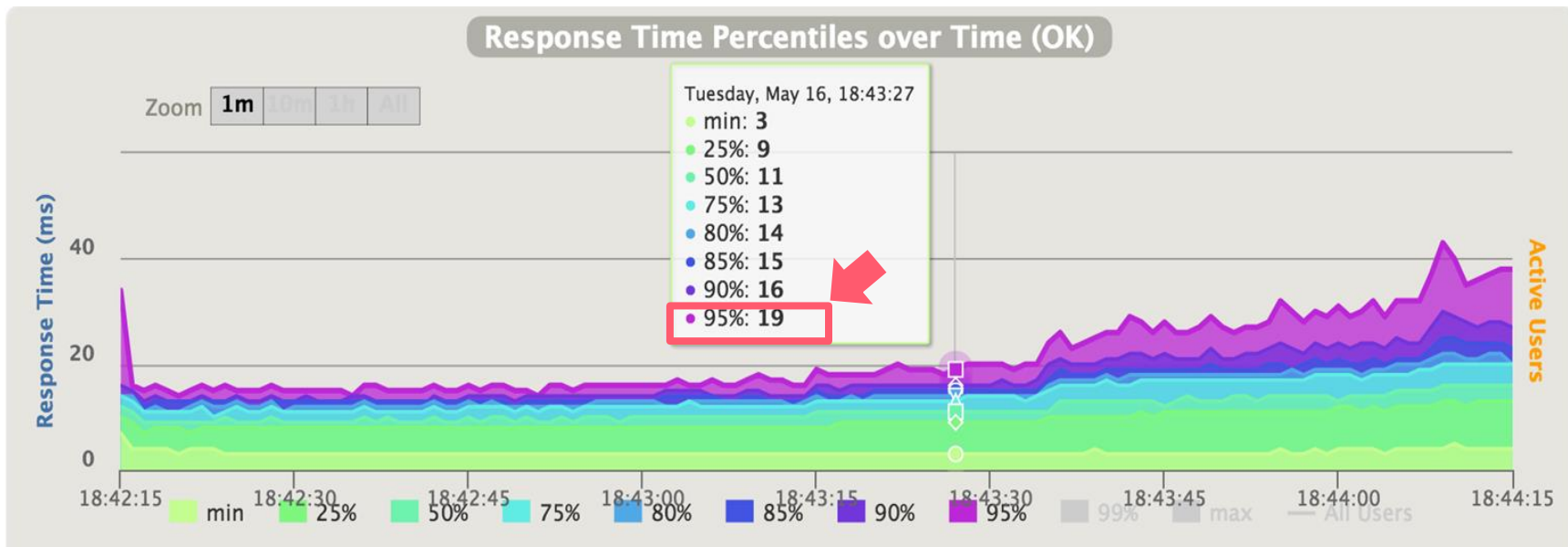
ベンチマーク結果(ECS 12台)

毎秒1万リクエストにも余裕の対応 明らかにオーバースペック



ベンチマーク結果(ECS 12台 percentile値)

AZ間の通信が頻繁に発生した場合でも19ミリ秒程度のレスポンスタイム



※ 外れ値を除外し 95パーセンタイル値まで掲載

サービスディスカバリ



LambdaベースでECSのディスクバリを実現

① ユースケースとして、APIルーティング追加やOAuthスコープ追加など、管理サイトからECS上のDMM API Gateway全台に更新をBroadcastしたい。

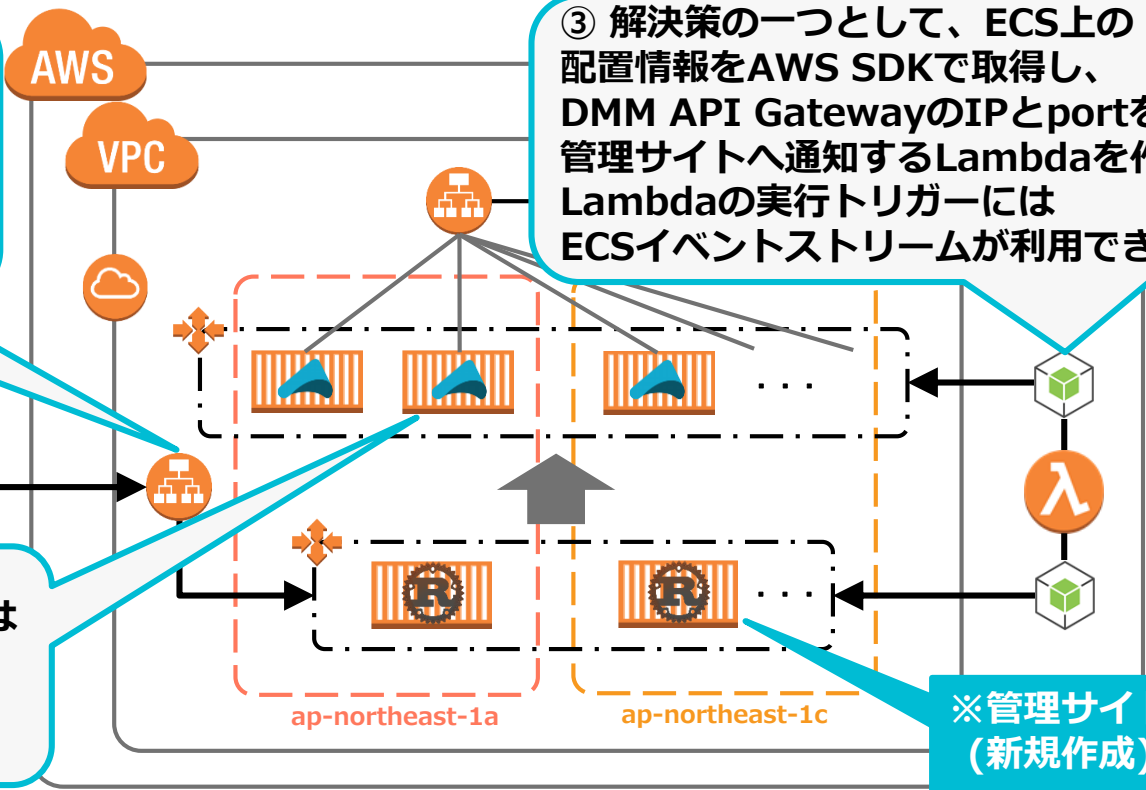
API Gateway
管理者



② しかし、ECS上のDMM API Gatewayはオートスケールする。IPアドレスとportが動的に変わっていく。

③ 解決策の一つとして、ECS上の配置情報をAWS SDKで取得し、DMM API GatewayのIPとportを管理サイトへ通知するLambdaを作成。Lambdaの実行トリガーにはECSイベントストリームが利用できた。

※管理サイト
(新規作成)



最後に



まとめ

AWS環境への移行で嬉しかったこと

- マネージドで高可用なシステムが従来より簡単に構築できた！
- 高ワークロードなシステムの構築も問題なかった！

AWS環境の使いこなしで気を付けたこと

- インスタンスの利用料はオートスケールの活用で節約
- サービスディスカバリはSDK等の活用で利用者側が作成

