サービス全断はダメ、ゼッタイ。途切れないテレビ会議システムを目指して

～AWS を最大限活用して可用性を高める秘策～

AWS Summit Tokyo 2017

梅原　直樹

31/5/2017

可用性

システムが継続して稼働できる能力

梅原　直樹

うめはら　なおき

- Twitter: @numeha
- 株式会社リコー 12年目
- オフィスのコミュニケーションサービスを開発する部署
- ソフトウェアエンジニア
- クラウド開発、モバイル(iOS, Android)開発、品質/テスト
- インフラリーダ 1年目

よろしくお願いします

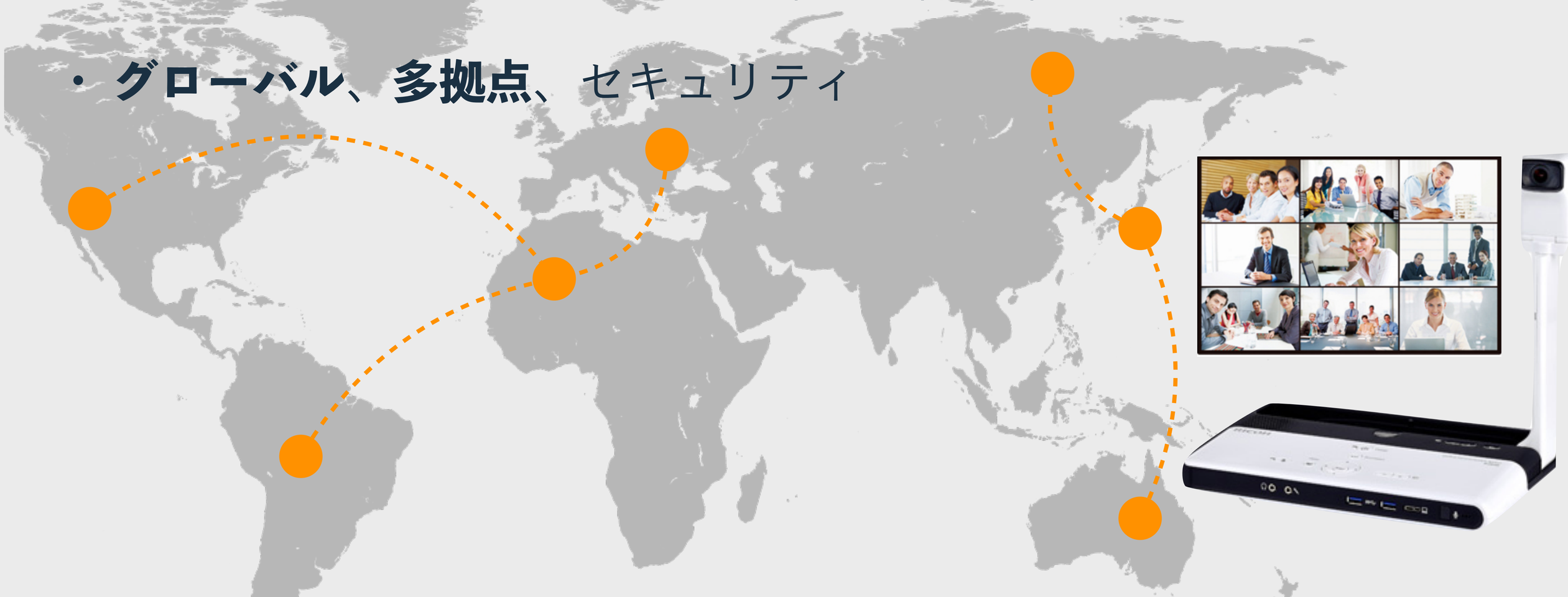
Agenda

- RICOH UCS（テレビ会議システム）が目指すもの
- AWSへの移行にこだわったこと
- AWSで運用してわかったこと

RICOH UCS
が目指すもの

サービスの特性

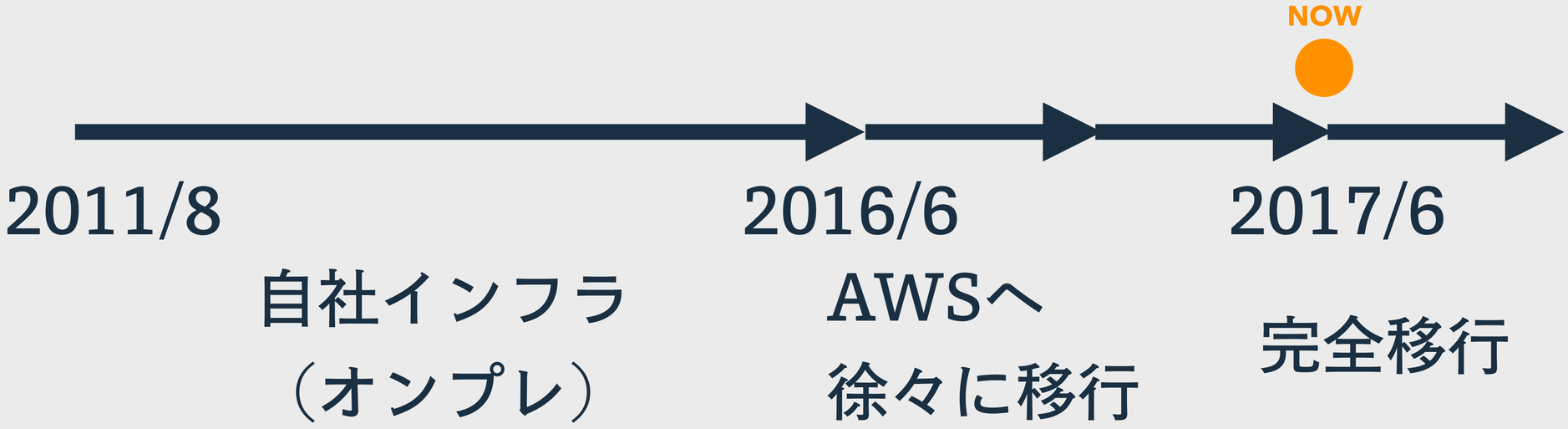
- **ビジネス向け**のインターネット経由で利用するテレビ会議システム
- **専用端末**、汎用端末（Windows, Mac, iOS, Android）
- **グローバル**、**多拠点**、セキュリティ



サービスの特性

- 世界中のビジネスアワーにほとんど使われる（9:00-17:00 JST）
  - 要するに24×365使われている
  - 年始、年度始め、月初は多い
- お客様の大事な遠隔会議で利用されている
  - 例えばある大事な遠隔商談の映像が途切れたら…😤
  - **とにかく切れちゃダメ**

RICOH UCSのクラウド



- 2016/6　ディザスタリカバリ用に利用
- 2016/12 メイン環境をAWSに移行
- 2017/6　完全移行完了

オンプレからパブリッククラウドへ

- 2016年からリコーグループ全社的な動きへ変化
- オンプレあるあるが主な理由
  - コスト高、リソース追加/削除、セキュリティ, グローバル
- 本プロジェクトが社内的には先進的な取り組み

インフラを含めた品質保証はできるのか

- 😱インフラは壊れてない前提で考えていると、障害が起きた時に大変なことになる。そして障害は必ずある
  - 2015年にインフラの大規模障害
  - 影響範囲が広く、いくらその上で仮想サーバを冗長化しても、全てが無駄…
  - エンドユーザは大きな損失、信頼低下

どんな理由でも

サービス全断はダメ

😫反省😫

そして目指すものは何か？

とにかく

可用性の向上👍

可用性の向上

- アプリケーション障害
- サーバ障害
- データセンター障害
- リージョン障害
- 全リージョン障害（IaaS事業者障害）

米国東部（バージニア北部、US-EAST-1）リージョンで発生した Amazon S3 サービス障害について

https://aws.amazon.com/jp/message/41926/

日本時間3月1日未明に米国東部（バージニア北部、US-EAST-1）リージョンにおいて発生いたしましたサービス障害に関する追加情報についてお伝えいたします。

この度、Amazon Simple Storage Service (S3) チームが S3 の請求システムの処理に通常よりも時間がかかるという問題のデバッグを進めておりました。
その過程におきまして、9:37AM PST(日本時間 2:37AM)に、適切に権限を与えられたS3チームメンバーが確立された手順に従い、S3 の請求システムが利用するS3サブシステムを構成する少数のサーバを削除するコマンドを実行いたしましたが、その際、コマンドへの入力が不適切であったため、想定よりも多くのサーバが削除される結果となりました。

今回誤って削除されたサーバは２つのS3サブシステムに関わるもので、１つは、Index（インデックス） サブシステムであり、当該リージョンにおいて全てのS3オブジェクトのメタデータやロケーション情報を管理し、全てのGET, LIST, PUT および DELETE リクエストを提供するものでした。もう１つは、Placement(配置) サブシステムであり、新規ストレージのアロケーションを管理し、Indexサブシステムに処理を要求するものでした。こちらは PUTリクエストの際、新規オブジェクトのためにストレージを割り当てるために利用されるサブシステムとなります。
キャパシティのかなりの部分が削除されるとこれらのシステムは再起動が必要となり、[illegible]起動[illegible]は サービスリクエストを処理できない状態となっておりました。

リージョン障害も起きる

また、S3 API が利用できなくなったことから、S3コンソール、Amazon Elastic Compute Cloud (EC2)の新規インスタンスの起動、Amazon Elastic Block Store (EBS) ボリューム(S3スナップショットからデータが必要であった場合)、AWS Lambdaを含め、US-EAST-1リージョンにおける、ストレージとしてS3を利用するその他のAWSサービスにも影響がございました。

S3 サブシステム群は、削除や顕著なキャパシティ障害の際にも、お客様への影響が軽微または影響のないよう設計されております。また、私どものシステムは障害が発生した場合のリカバリ対策も含めて構築しており、コアオペレーショナルプロセスの１つとしてキャパシティの削除や置き換えを行える能力に重きを置いてきました。本オペレーションは、S3のサービス開始以来システムを維持するために頼ってきたものではありますが、一方、長年にわたり広域なリージョンにあるIndexサブシステムやPlacementサブシステムを完全に再起動する機会はありませんでした。S3 は過去数年にわたる大規模な成長を経て、サービス再起動のプロセスおよびメタデータの正常性確認に求められる安全性のチェックに想定以上の時間を要する結果となりました。

今回影響を受けた2つのサブシステムのうち最初にIndex サブシステムの再起動を行いまして、12:26PM PST（日本時間 5:26AM）までに S3 GET, LIST および DELETE リクエストを提供するための十分なキャパシティを確保いたしました。 その後、1:18PM PST（日本時間 6:18AM）までに Indexサブシステムの復旧が完了し、GET, LIST および DELETE API が通常稼働する状態となりました。

Google Compute Engine、全世界のリージョンが同時に外部とのネットワーク接続を失うという深刻な障害が発生。ネットワーク管理ソフトウェアにバグ

http://www.publickey1.jp/blog/16/google_compute_engine_2.html

2016年4月19日

全リージョン障害だって起きる

クラウドのどこかで障害や災害が発生したとしても、その影響はアベイラビリティゾーンを超えることはなく、そのために複数のアベイラビリティゾーン（Google Compute Engineでは「ゾーン」）にシステムを分散して配置することで、クラウドの障害の影響を受けない高い可用性を備えたシステム構築ができる。これはクラウド（IaaS）に対応したシステム構築におけるもっとも基本的な考え方です。

しかし先週、2016年4月11日にGoogle Compute Engineで発生した通信障害は、アベイラビリティゾーンどころかリ

可用性の向上

- アプリケーション障害
- サーバ障害
- **データセンター障害**（だが、まずはここまで）
- リージョン障害
- 全リージョン障害（IaaS事業者障害）

可用性向上のために目指すこと

- **IaaSの障害を想定した設計（Design for failure）**
  - データセンターが丸ごと飛んでもどこかで動き続ける
- 障害検知を速くする
  - 自動的に復旧させる
- デプロイのダウンタイムゼロ

AWSへの移行
にこだわったこと

移行はどのようなシステムでも可能だが、
一度運用が始まってしまうとドラスティック
な変更は難しい。けど早く移行したい。

移行ポリシー

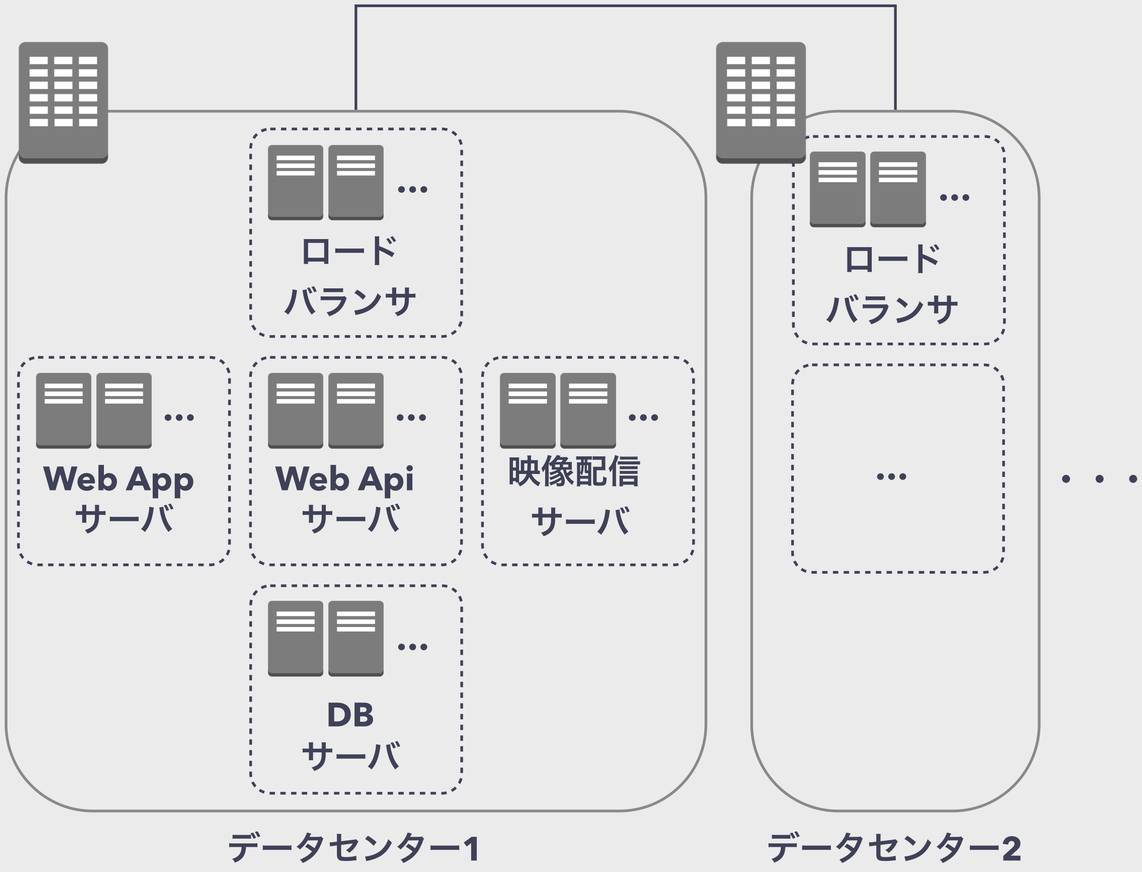
- **基本的に同じ構成で移行する**
  - ただし、コンテナ技術を採用する
  - ただし、マネージドサービスは積極採用する
  - 移行した後に最適化する
- **徹底的に自動化する**
  - オンプレで動いていたものを全てコード化して自動構築する
  - 作り直しても元と同じ状態になる安心感

移行ポリシー

- **基本的に同じ構成で移行する**
  - ただし、コンテナ技術を採用する
  - ただし、マネージドサービスは積極採用する
  - 移行した後に最適化する
- **徹底的に自動化する**
  - オンプレで動いていたものを全てコード化して自動構築する
  - 作り直しても元と同じ状態になる安心感

RICOH UCSの構成概要

- **会議する相手によって利用するデータセンターは 変わる**
  - データセンター間でDB間、API間等のやり取りを行う
- 映像配信サーバはグローバルに散らばっている
  - 遅延を最小化する
- 各種Web API, Appサーバ
  - Ruby, Java等、役割ごとに仮想サーバへ

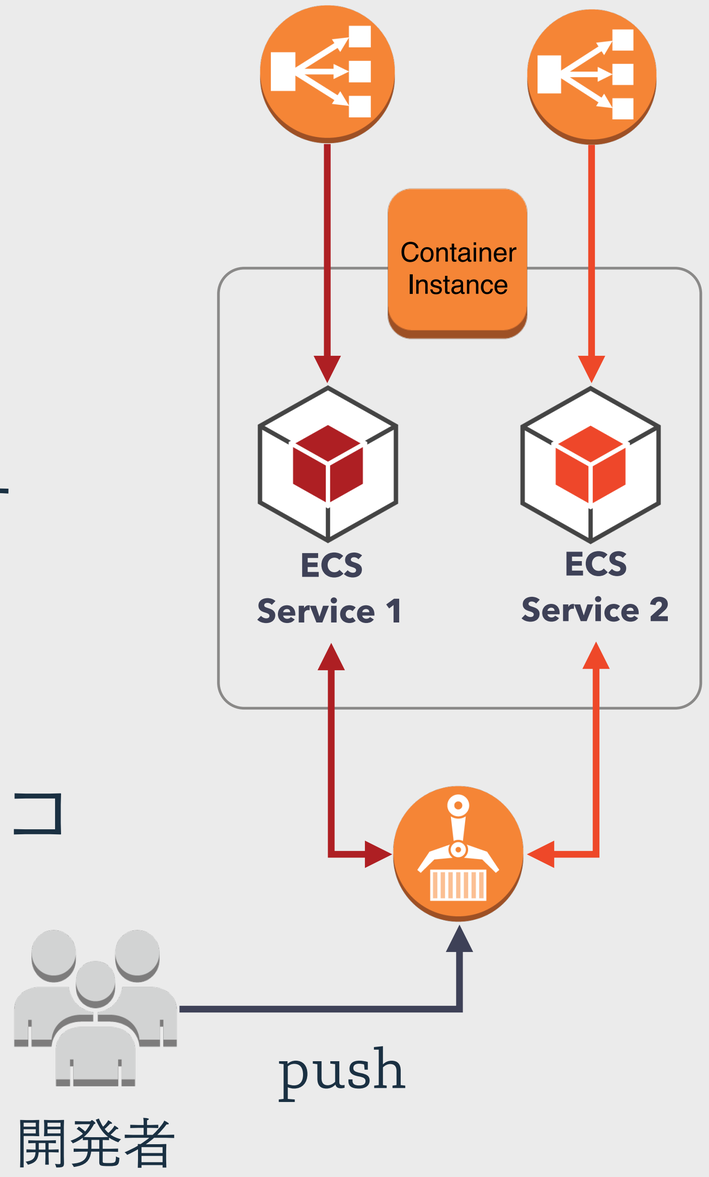


各種WebAPI, Appのコンテナ化

- 😄Dockerさえあれば動く
  - デプロイ/ロールバック容易、容易なスケールアウト、マイクロサービス
  - 言語、バージョンを気にしなくて良い（若者は新しい言語を使いたい）
  - ECSの安心感、2016/8/15 ECRがTokyo Regionに
- 😰運用実績 -> 作っていくしかない
  - 対応コスト -> 初期投資はかかるがその後の運用が楽

コンテナ化の基本構成

- 基本は、**ELB + EC2 + ECS + ECR**
- ECRがDevとOpsのI/Fになる
  - 開発者はDockerイメージをPUSHするだけでデプロイ可能
- アプリケーションの役割に応じて稼働コンテナ数(desiredcount)を変更



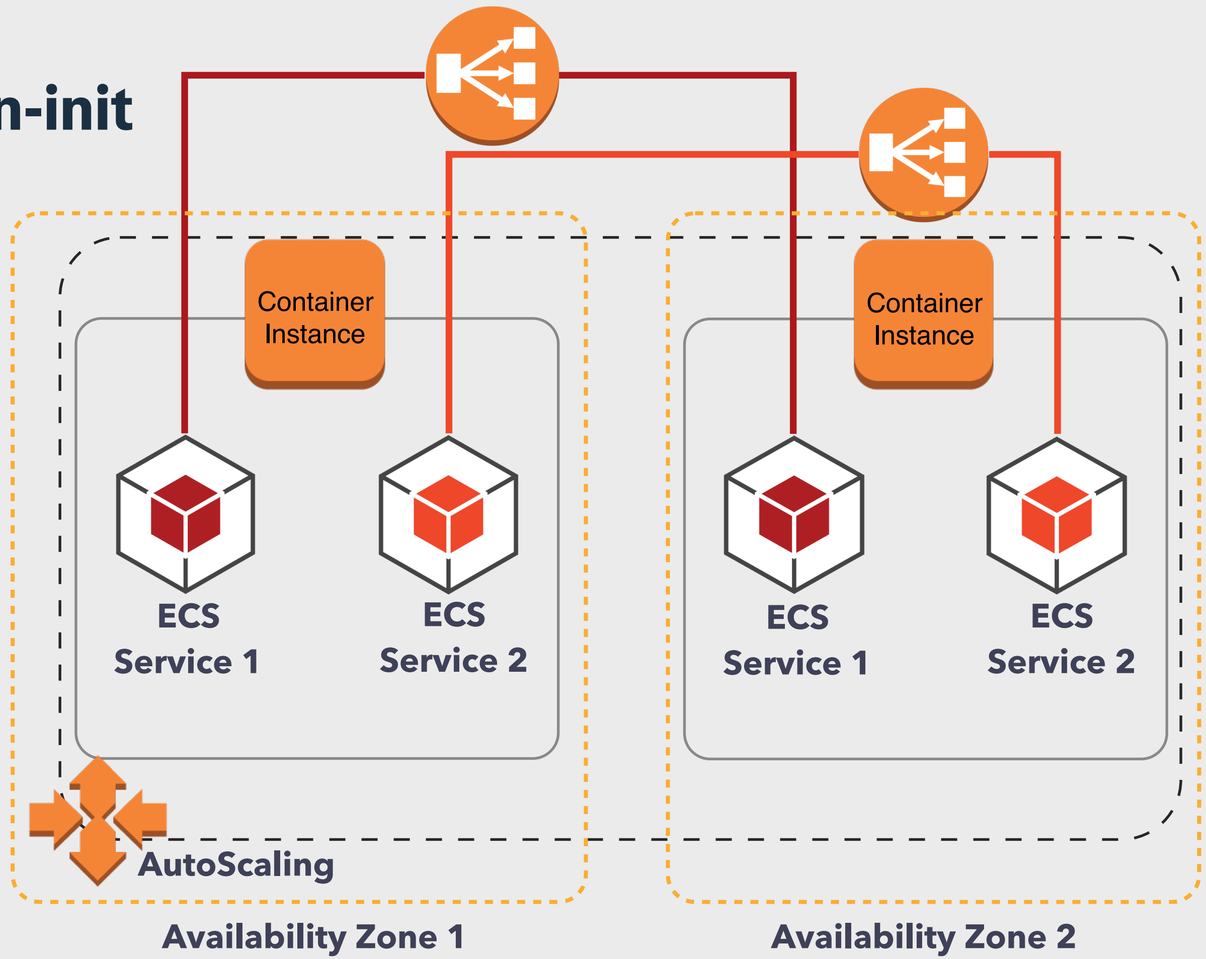
Elastic Load Balancing

Amazon ECS

Amazon ECR

コンテナ化による可用性向上

- インスタンスは壊れる前提で考える
  - ELB + EC2 + ECS + ECR + **AutoScaling** + **cfn-init**
- コンテナも壊れる前提で考える
  - ELBはAutoScalingではなく、ECSにつける
- Multi-AZにすれば**ダウンタイムゼロ**
  - **壊れても自動復旧**
- Single-AZでもダウンタイム数分
  - 役割によっては選択肢としてあり



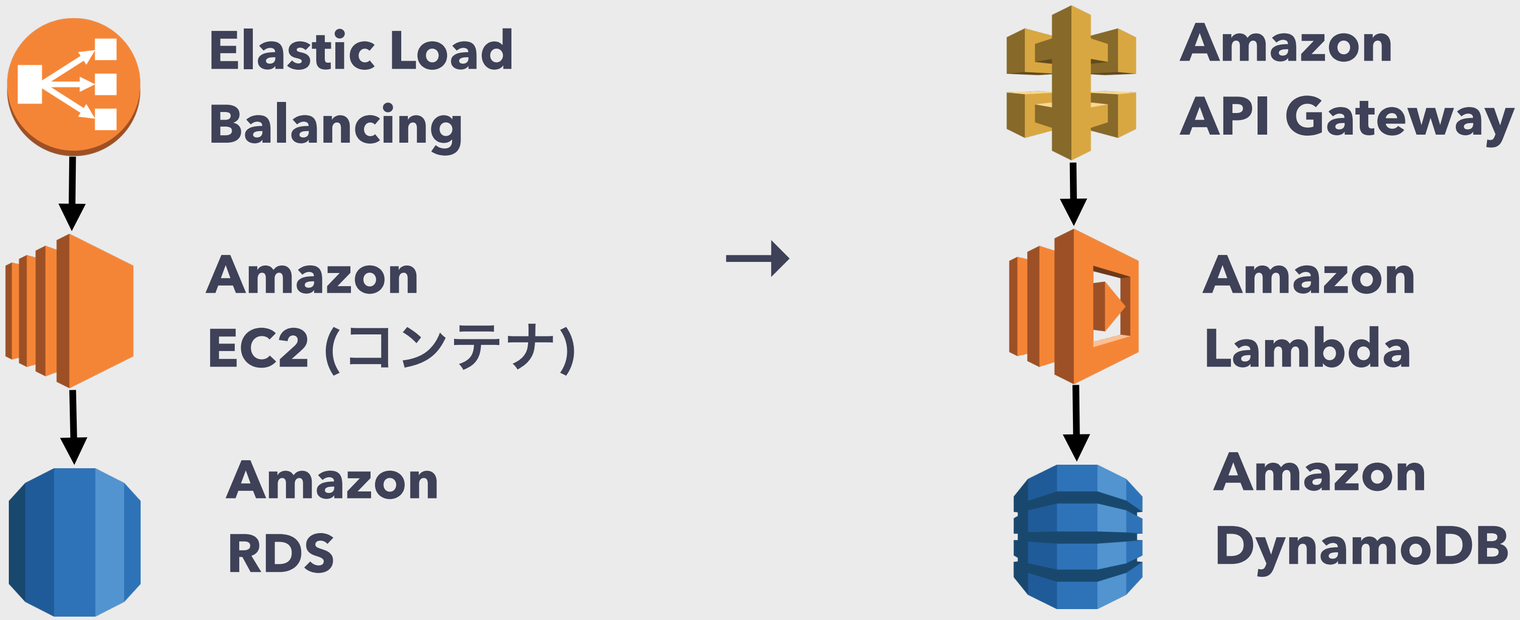
データストアはどうするか

- 😄マネージドサービスのRDS（MySQL）を利用
  - 元々MySQLを利用していたので問題は特になし
  - 自作の運用ツール（レプリケーターが等）が不要に
  - AZ障害を考慮するときは、基本はMulti-AZ設定にすればOK
- 😰以前の構成を踏襲した時に、クロスリージョンのレプリカのMulti-AZができない等の制約あり
  - このような制約が出た時点で構成変更検討が必要

Amazon RDS

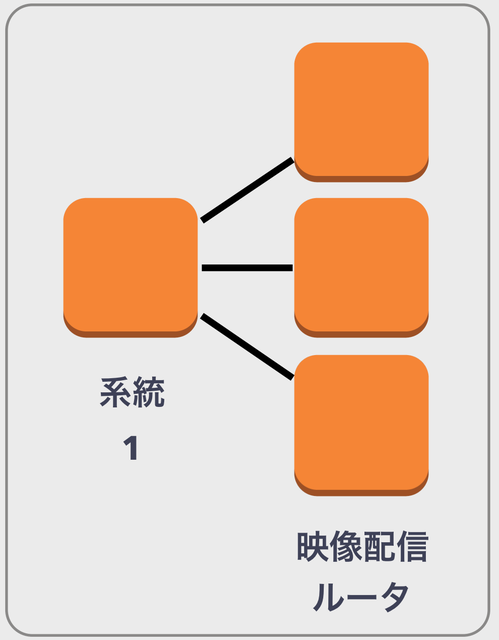
さらにマネージドサービスを利用する

- **新規APIのみ積極採用し、移行コストの関係で既存のAPIは移行しない**
- 😁サーバレス化によりAZを意識しない
  - ビルトインなのでメンテ不要
- 😰ブラックボックスなので本当に大丈夫なのかわからない



AWSの仕組みをうまく使えないケースもある

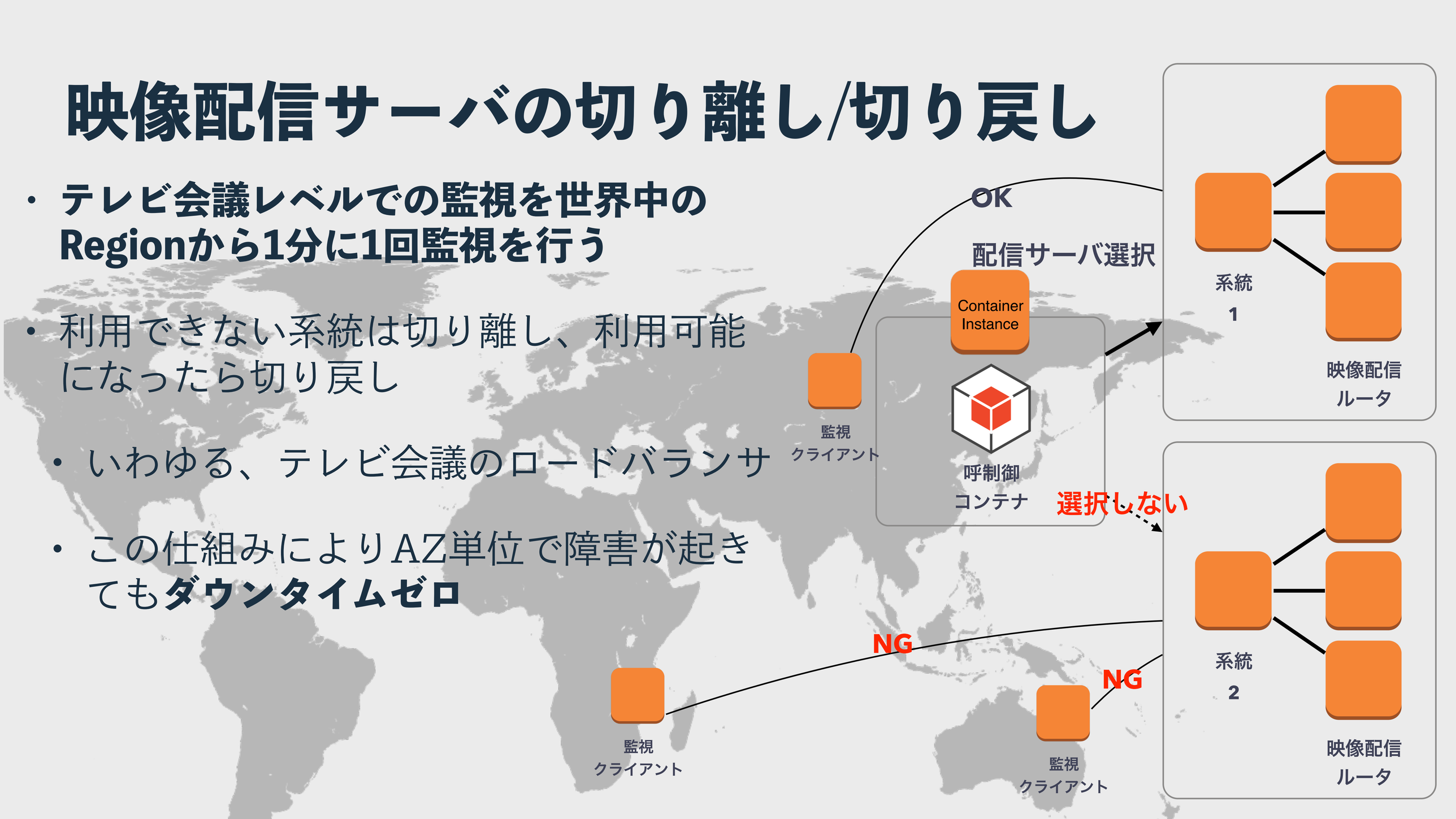
- **映像配信サーバは単純なヘルスチェックだけでは不十分**
  - 複雑なシーケンスで会議をしてみないことには正常かどうかわからない
- **AutoScalingとも相性が悪い**
  - つながっている端末がある限りは映像配信ルータを減らすことができない
  - 系統毎にしかアップデート出来ないのでAWSの仕組みにのることが難しい



. . .

映像配信サーバの切り離し/切り戻し

- **テレビ会議レベルでの監視を世界中のRegionから1分に1回監視を行う**
- 利用できない系統は切り離し、利用可能になったら切り戻し
  - いわゆる、テレビ会議のロードバランサ
  - この仕組みによりAZ単位で障害が起きても**ダウンタイムゼロ**



さらにさらに改善する

- RDS MySQL -> Auroraに変更
  - 高速フェイルオーバ（MariaDB Connctor）で最短数秒で復帰
- Health Check値の最適化
- 自動復帰しないパターンの観測・独自対応
  - 監視/復旧プログラムを投入
- **全ては検知を早く、復旧を1secでも早くするために**

利用サービス


Amazon
EC2


AWS
Lambda


Amazon
SES


Amazon
S3


Amazon
ECR


Elastic Load
Balancing


Amazon
SNS


Amazon
DynamoDB


Amazon
ECS


Amazon ES


Amazon
SQS


Amazon
RDS


Amazon
Route53


Amazon
CloudFormation


Amazon
API Gateway

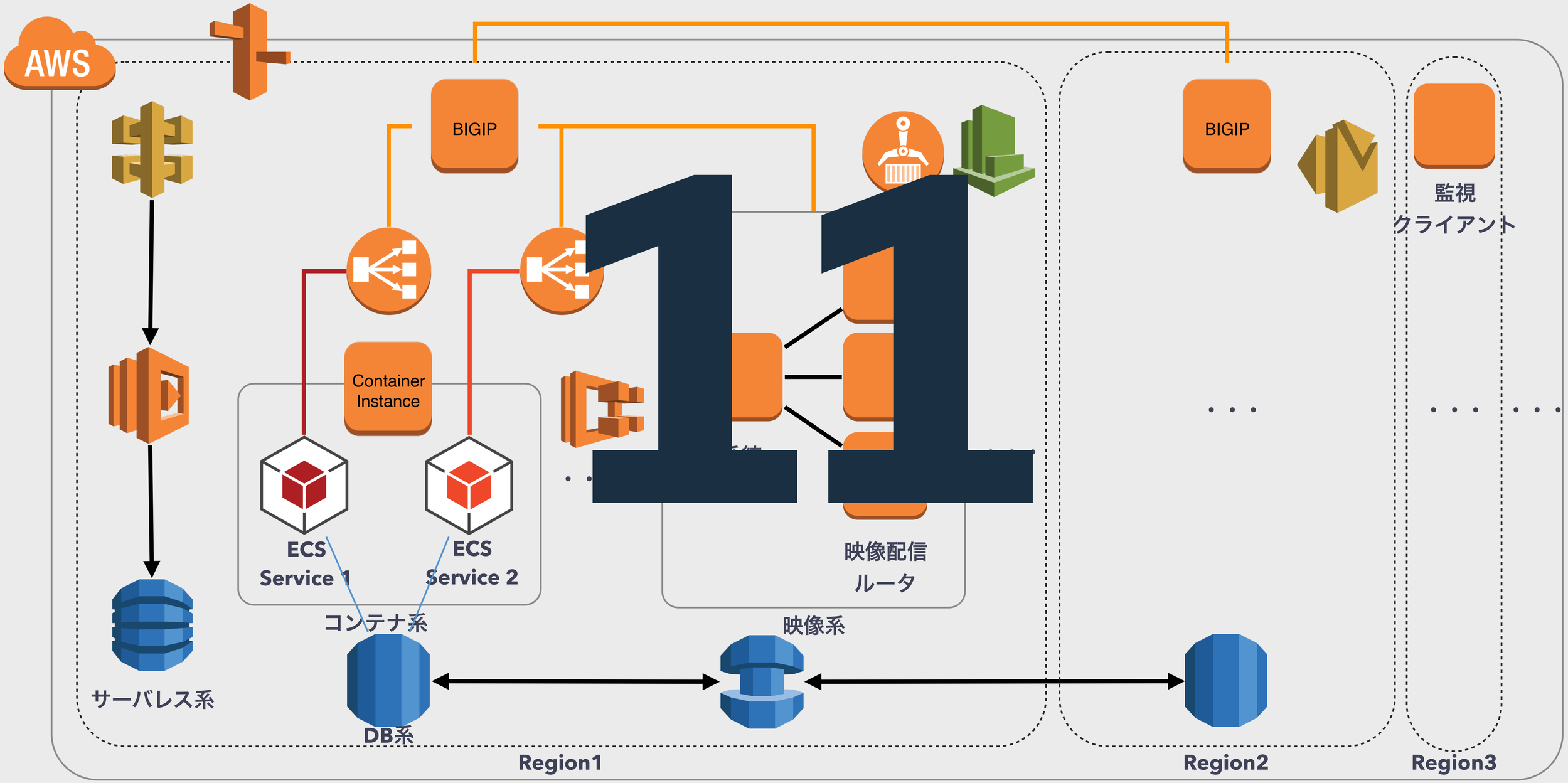

AWS
DMS


Amazon
VPC


Amazon
CloudWatch


Amazon
IAM

利用リージョン


AWS
BIGIP
BIGIP
監視
クライアント
Container
Instance
ECS
Service 1
ECS
Service 2
コンテナ系
映像配信
ルータ
映像系
サーバレス系
DB系
Region1
Region2
Region3

移行ポリシー

- **基本的に同じ構成で移行する**
  - ただし、コンテナ技術を採用する
  - ただし、マネージドサービスは積極採用する
  - 移行した後に最適化する
- **徹底的に自動化する**
  - オンプレで動いていたものを全てコード化して自動構築する
  - 作り直しても元と同じ状態になる安心感

愛情ある手作業でインフラを構築することは
　できる。しかしそれは一度だけだろうか。
二度以上あるなら自動化しようではないか。

環境は4つ

Dev環境
クラウドのメンバーが
日々開発している環境

Beta環境
クラウドを利用する端末向け
の評価環境

必要な時にあれば良いので自動化が特に役に立つ👍

常時稼働

Stage環境
本番投入前の練習場

Production環境
本番

インフラ構築を

徹底的に自動化する

徹底的に自動化する

- **インフラをコード化する**（Infrastructure As a Code）
- **Cloudformation**（Kumogata: Ruby製ツール）**を利用**
  - Cloudformation 60,000行 -> Kumogata 16,000行 に
  - 役割毎にテンプレートを分割したり、プログラミングと同様
- 環境名とバージョンだけを渡せば構築するようにバージョン管理
- **Dev環境は営業時間内のみ自動起動/削除**
  - 強制的に20:00にはシャットダウンされ、メンバーは帰るしかない
  - 毎日構築されるので、バグ混入のトラッキングがしやすい

イミュータブルインフラ

- **😁インフラのバージョン管理ができ、所定のバージョンに一から構築することができる**
  - デプロイの安全性向上
  - もしも壊れても作り直せば元通りになる安心感
- **😁コードベースで話すことができるようになった**
  - インフラ変更を加えたいときはコードに加える文化に
  - 職人芸がコードに

Blue-Green デプロイメント

- 方針
  - In placeデプロイのように一度作成したインスタンスに変更を積み重ねていくことはしない
  - Red/Blackデプロイのように全台を一斉にアップデートすることはしない
  - **役割に分けてBlue - Greenデプロイメントを可能にする**
- Cloudformationの構築/更新のみで実現
  - 基本は、Swap ECS services with ELB方式か、Swap Auto Scaling Group

Blue-Green デプロイメントをやってみて

- 😁各リソース単位で1から構築したインフラを使うので、事前にテスト済みのインフラを利用することができる
  - もしもミスが合ったとしても旧環境に切り替えるだけ
- 😰1から構築するのでデプロイに時間がかかる
  - Cloudformationのバージョン管理をしておかないと、旧テンプレートの更新にパラメータ不整合で失敗することがある

AWSで運用
してわかったこと

監視

- Pingdom
- CloudWatch
- Zabbix
- VCMon(独自会議監視)

ZABBIX

メール

Slack

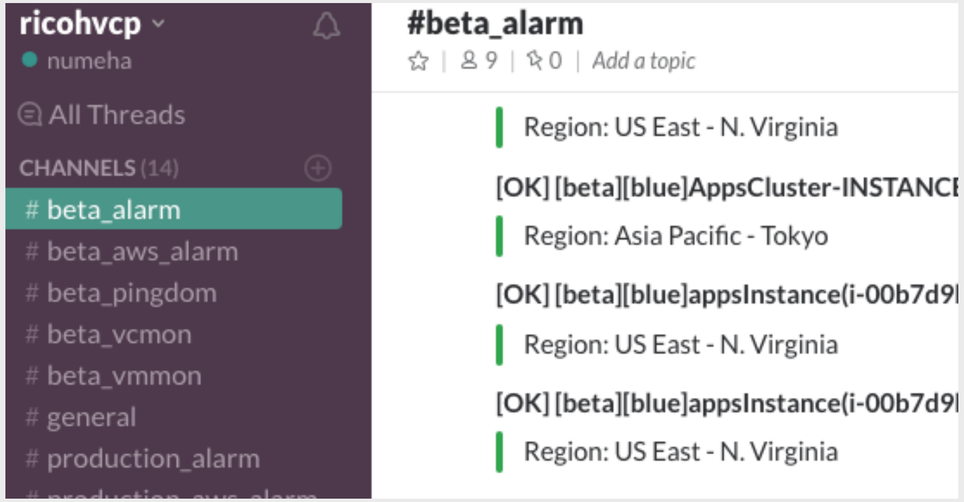
運用者

電話

本格運用開始して半年

- **安心😀**
  - 大規模障害からの解放
  - 稼働率の向上
  - インフラで何かがおきてもほとんど自動復旧
  - 手厚いサポート

本格運用開始して半年



- **不安😰**
  - アラームに怯える
    - AWSの癖を掴むまではビクビクする
    - このアラームのパターンならあれかな、これが続いたらやばい等
  - AWSのネットワーク品質低下問題
    - 外部/内部ネットワークで疎通不可

可用性は向上したのか

- 😁**事実向上した**
  - 以前までなら停止時間としてカウントされていた障害に遭遇
  - 自動復旧に幾度も助けられた
  - IaaS障害に強い設計になった
- 😁（初期投資は必要だけど）運用負荷は減った
- **”可用性のボトルネックが変わってきた”**と実感した

可用性のボトルネックはどこか

- IaaS障害=大規模障害 から **小規模障害** に変わってきた
- **アプリケーション側の工夫が余儀無くされるケースが出てきた**
  - 映像配信サービス（常時コネクション張りっぱなし）おいて、インフラの工夫だけで切断を伴わなくするのは困難
- **もはやインフラとアプリケーションの境目はない**
  - 自分達のアプリケーションの特性理解して、改善することが必要

まとめ

今までの
マインドをチェンジしよう

インフラを含めた品質保証はできない
インフラの障害は必ずある
インフラの障害を考慮した設計はできているか

「運用でカバー」は限界がある
インフラを理解してアプリケーションを改善する
アプリケーションを理解してインフラを改善する

サービス全断はダメ、ゼッタイ。途切れないテレビ会議システムを目指して

～AWS を最大限活用して可用性を高める秘策～

AWS Summit Tokyo 2017

おわり

梅原　直樹

31/5/2017