Architecting on AWS

モジュール1 はじめに

モノリシックアーキテクチャとマイクロサービスアーキテクチャ

- モノリシックアーキテクチャの場合
    - 機能追加の際に、既存サーバに新規機能を追加すれば良いだけ
- マイクロサービスアーキテクチャの場合
    - AWSを利用すれば、簡単に新たなリソースを追加でき、そこに機能を追加できる。

クラウドとは

- クラウドを作成するための技術
    - リソースの抽象化（仮想化技術を使っているだけ）
        - KVM, Xen, Hyper-V, VMWare ESXi
    - APIやセルフサービスポータルの提供（クラウド）
        - Clound Controller (OpenStack, CloudStack 等)

Well-Architected フレームワーク

下記、5つの観点をAWSのサービスを使う上で気を付けてほしい。
Well-Architectedフレームワークについては、kindle(英語)で無料公開されているからおすすめ。
日本語訳も以下で展開されている。(少し古いが。)
https://aws.amazon.com/jp/blogs/news/aws-well-architected-whitepaper/

1. セキュリティ
2. 信頼性
3. コスト最適化
4. パフォーマンス効率
5. 運用上の優秀性

AWS内のハードウェアについて

25GbEで構成されている。
https://d1.awsstatic.com/events/jp/2018/summit/tokyo/aws/21.pdf

AWS内のソフトウェアについて

Mapping Service
- VPC IDとEC2インスタンスのIPアドレスの組を「Server」と対応付ける機能

https://codezine.jp/article/detail/9790

日本国内のリージョンについて

大阪ローカルリージョンは、AWSに申請しないと利用できない。
東京に4つのAZがあるが、可用性向上のために、東京以外にも置きたいときに利用する。

エッジロケーション　現在187拠点

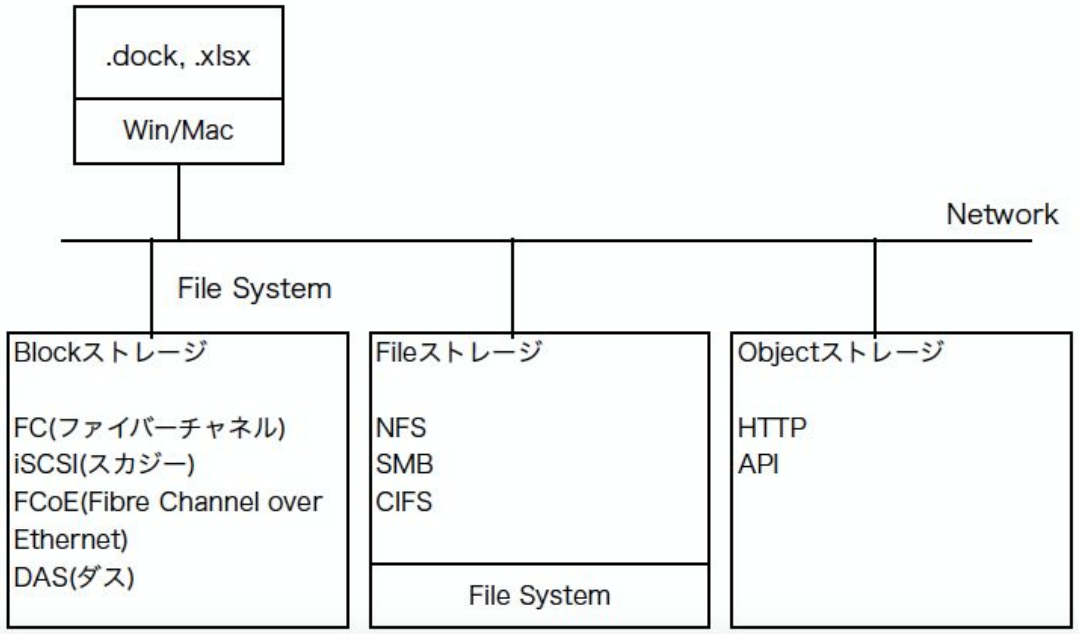
- Route 53　　: 権威DNS
  - DNSにおいて、あるゾーンの情報を保持し、他のサーバーに問い合わせることなく応答を返すことができるサーバーのことです。
- CloudFront　: CDN(Content Delivery Network)
- AWS WAF　　: Web App Firewall　(L7) : アプリケーション層
- AWS Shield　: DDoS緩和　　(L4) : トランスポート層

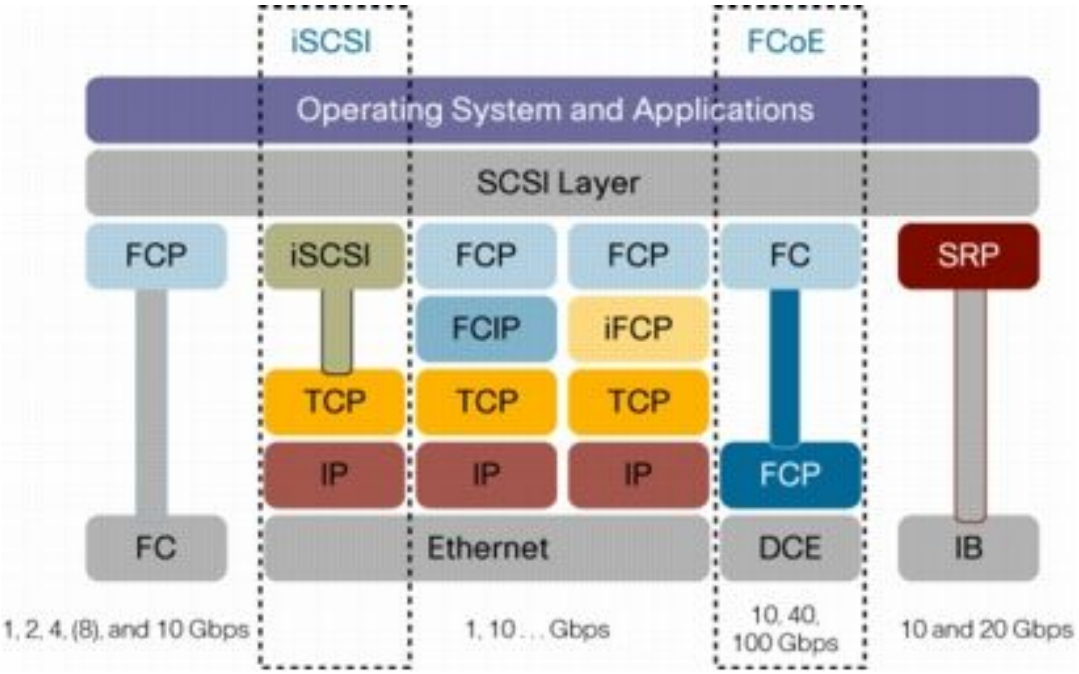
モジュール2 最もシンプルなアーキテクチャ

ストレージ製品について

FileSystemが、ストレージを管理している。
ストレージは、「メタデータ、データ、データ、、、」で構成されている。
USB等をフォーマットをすると、ストレージを上記のようにメタデータを付加して管理できるようにする。

- Blockストレージ
    - ストレージ内が細切れ(ブロックサイズごと)に分かれている。
    - Win/MacのFileSystem(NTFS, FAT, ext等)が、どこの細切れにあるデータに対してアクセスしてくれている。
    - アクセスの仕方については、C言語のポインタの考え方に似ている。
    - Win「`diskmgmt.msc`」で、ストレージ管理ができる。
    - アロケーションストレージサイズ = ブロックサイズ
- Fileストレージ
    - Win/Macからネットワーク越しにファイルにアクセスする。
    - Fileストレージの中に、FileSystemがあるから、利用できる。
- Objectストレージ
    - Win/Macからネットワーク越しにファイルにアクセスする。
    - namespaceで分離している。
    - オブジェクト = ファイル + メタデータ



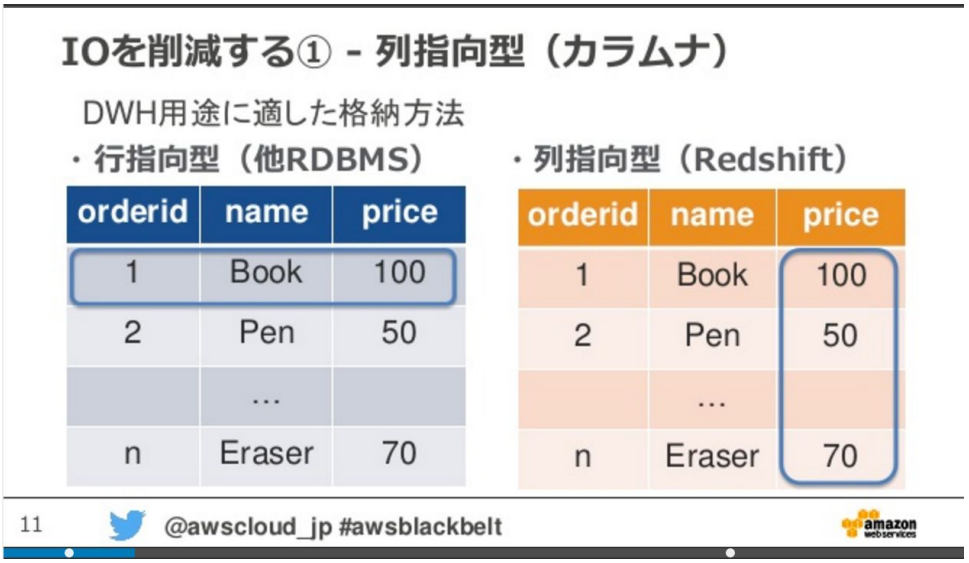


Amazon S3

- バケット形式
  - http://<bucket-name>.s3.amazonaws.com
  - 今後も利用可能
- パス形式
  - http://s3.amazonaws.com/<bucket-name>
  - 今後、廃止予定

Amazon S3のユースケース

- メディアや静的コンテンツの保存と配信
- 静的なウェブサイト全体をホスト
  - html, css, javascriptのみ (バックエンド処理(Java等)はできない。)
  - https://wiki.infra-workshop.tech/user/nacco_bron/静的サイトジェネレータHugoをさわろう
  - 静的ウェブサイトも、静的ウェブサイトジェネレータを利用すれば、簡単に作成できる。ECサイトなども裏でアクセスすれば構築可能。
- ビッグデータ
  - データレイク (池)　　: S3(分析に使えるかどうかを問わず格納)
  - データウェアハウス (貯水庫): RedShift(分析に使えそうなものだけ格納)
    - RedShiftはカラムナ型で列に対してアクセスできるから分析向き。

- 

Amazon S3へのデータ移行

- 大規模データの移行方法
  - マルチパートアップロード
  - Amazon CloudFront(エッジロケーション) 高速ネットワークの利用
  - AWS Snowball, AWS Snowmobile (80TB) 物理的にデータを移行する。

Amazon Glacier (Amazon S3の1機能)

- データのアーカイブ機能を提供してくれる。
- コストはS3の標準利用の1/3程度
- 監査ボールト(データの改竄防止に利用できる。)
- オブジェクト単位で、Amazon Glacierへ移行できる。
- ライフサイクルポリシーを利用すれば、一定期間利用されていないと、自動的に「標準利用」→「Amazon Glacier」→「データ廃棄」等を実施してくれる。

Amazon S3のコストについて

以下の軸がある。

- リージョンに応じて利用可能なサービスが異なるため、どのサービスを使いたいかでリージョンを決めるように。
- リージョンに応じて、コストが異なる。
  - 例：バージニアは安いから、あまり使用しないデータはバージニアに置いておいて、よく使用するデータだけ東京に置いておく等。
    (東京は近いからパフォーマンスが高い。)

モジュール3 コンピュートレイヤーを追加する

Amazon EC2でできることは何ですか？

AWS パートナー 事例大全集
https://pages.awscloud.com/AWS-Partner-Led-Customer-Reference-Book-DL.html

システム構成の事例に、EC2が頻繁に出てくる。
もし、すべてEC2にしておくと、
OSのパッチやバックアップ等、運用がすべてユーザー持ちになり、運用が死んでしまう。

良いシステムアーキテクチャとは、、、
- できるだけEC2の数は減らすようにする。
- EC2は使い捨て可能にする。
    - EC2にログデータやセッションを持たないようにする。

AMIからEC2を構築できる。

(ファミリー)(世代).(インスタンスサイズ)
- m5.large　　：CPUとメモリのバランスが良いもの。
- cファミリー　：cpu最適化
- rファミリー　：メモリ最適化

- 汎用タイプ
    - aファミリー
        - Armアーキテクチャ
    - tファミリー
        - x86アーキテクチャ
        - バースト可能（常に高負荷な処理には向いていない）
    - mファミリー(おすすめ)
        - x86アーキテクチャ
        - 常に高負荷な処理に向いている。
- 高速処理タイプ
    - pファミリー
        - 多用途(機械学習や深層学習)に向いている。
    - gファミリー
        - GPU(レンダリング処理に向いている。)
    - fファミリー
        - FPGA
- ストレージ最適化
    - hファミリー
        - 大容量のハードディスクを持っている。

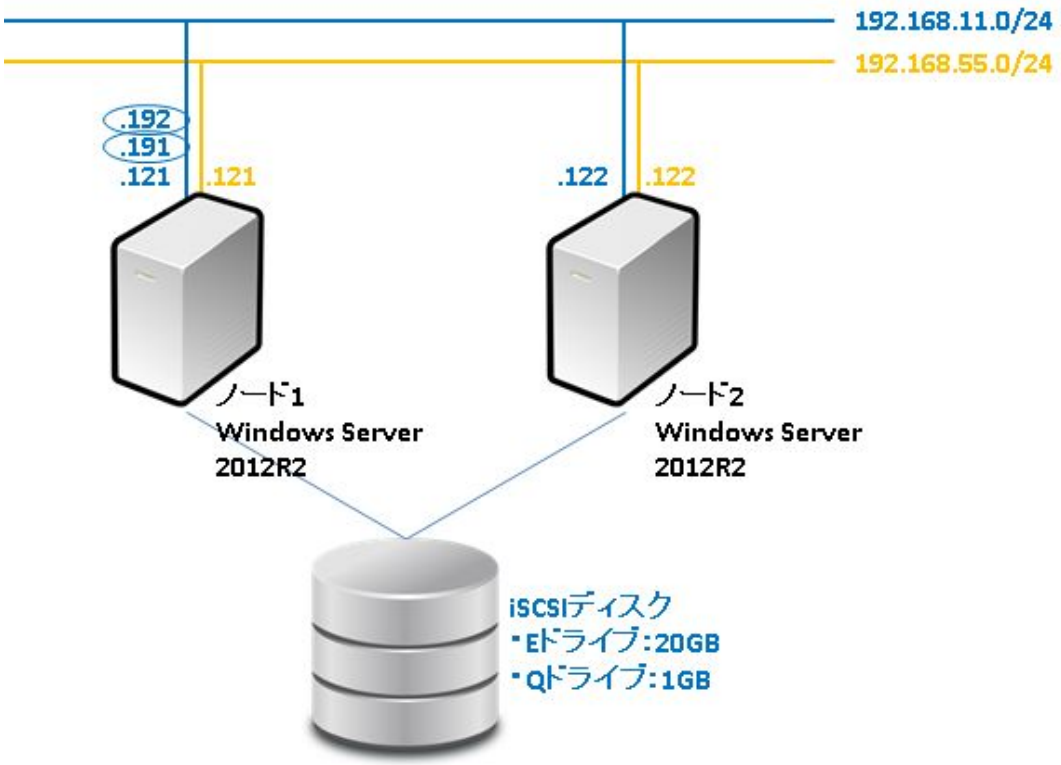
インスタンスストレージ

- EBS (Elastic Block Storage)
  アプリケーションからアクセスする頻度が高いデータはEBSに入れておいて、バックアップなどのデータをS3に置く構成が鉄板
  - EBSでは、WSFCができない。
  - 代わりに、DataKeeperを利用できる。
- SSD
  - IOPSが得意
    → IO/s
- HDD
  - スループットが得意
    MB/s, GB/s, TB/s

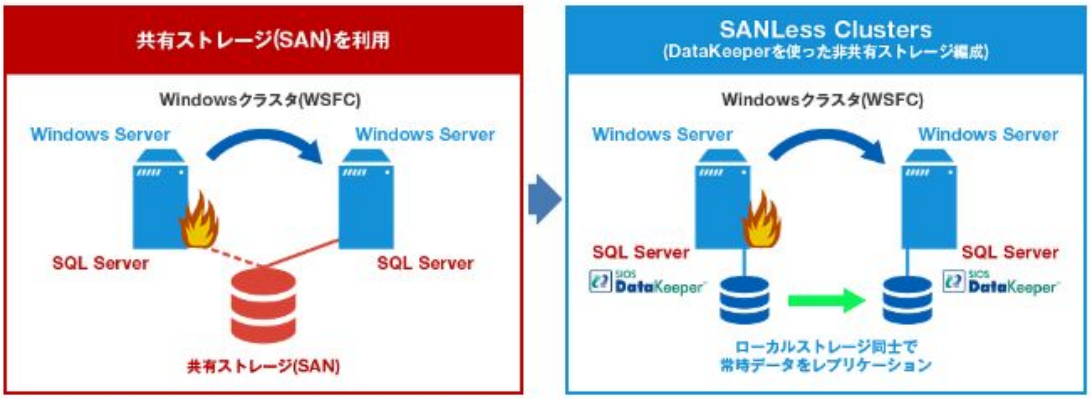
[補足]

==========

WSFCとは、split brain syndrome対策で、iSCSIを入れておいて、可用性担保するためのもの。(オンプレでよく使われてきたが、)ÅWSではこれができない。



DataKeeperでは、常時データレプリケーションが可能



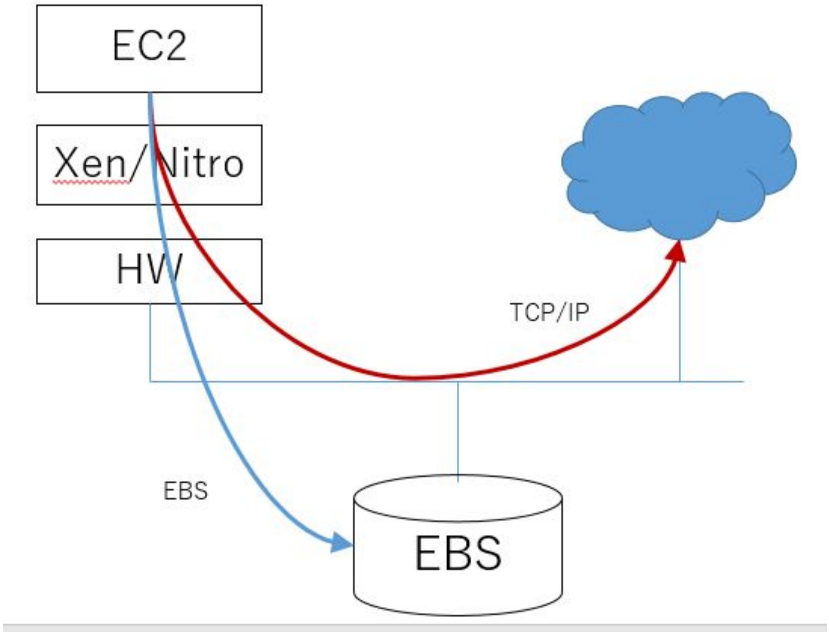
- クラウド環境など共有ストレージを構成できないケースがある
- 共有ストレージが単一障害点となる可能性がある
- 構築・運用には専門知識を持ったエンジニアが必要
- ベンダーロックインが課題
- 共有ストレージ自体が一般的に高価

- 物理・仮想・クラウド環境への対応が可能
- ミラーリングによる単一障害点の排除
- 構築・運用が比較的容易
- 構成は自由に組み合わせられる
- ローカルディスクで安価にHAクラスターを構成可

==========

EBS最適化

下記、EBSの通信を別の帯域を作成していて、最適化できる。
どのように最適化しているかは公開されていない。
（図はイメージ）



File System

- EFS
- FSx

料金オプションについて

- オンデマンドインスタンス
    - 利用した分だけ支払う
- リザーブドインスタンス
  これだけで50分くらいの講義できる。下がその資料。
  https://www.slideshare.net/AmazonWebServicesJapan/aws-black-belt-2017-cost-optimization-reserved-instance
  リザーブドインスタンスは権利であり、インスタンスは何度でも作り変え可能。
    - standard
        - 前払いによる割引率が高い
        - インスタンスタイプはその期間変更できない
    - convertible
        - 前払いによる割引率は普通くらい
        - 追加料金を払えば、途中でインスタンスタイプを変更できる

- スポットインスタンス
  使用されていないEC2インスタンスを購入する。
    - | RI (固定) | OI(変動) | SI(変動) |
        - OIが多く使われると、SIの量が少なくなり、SIは高くなる。
        - OIがあまり使われないと、SIの量が多くなり、SIが安くなる。
        - RI (リザーブドインスタンス)
        - OI (オンデマンドインスタンス)
        - SI (スポットインスタンス)
    - スポットブロックは、利用開始時間の料金で、最大6時間まで利用可能
    - スポットフリートは、分散処理で有効な方法。

モジュール4 データベースレイヤーを追加する

NoSQLについて

データと値の関係

- Key Value Store(KVS)
    - １：１
- ドキュメント
    - １：多
- グラフ
    - 多：多

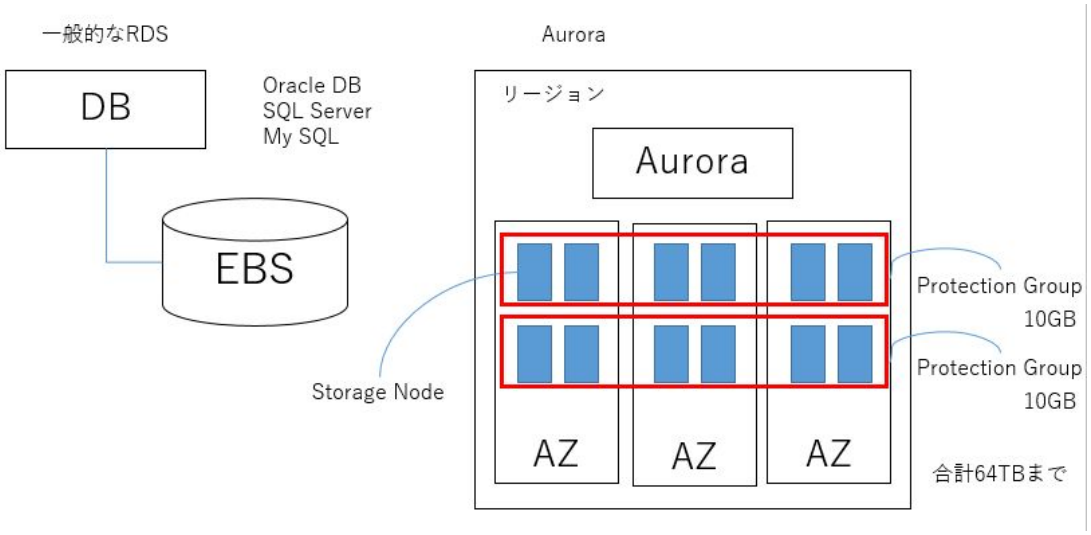
	リレーショナル/SQL	NoSQL
データストレージ	行と列	キー値、ドキュメント、グラフ
スキーマ	固定	動的
クエリ実行	SQL ベースのクエリ	ドキュメントコレクションにフォーカス
スケーラビリティ	垂直	水平

アンマネージド型データベース

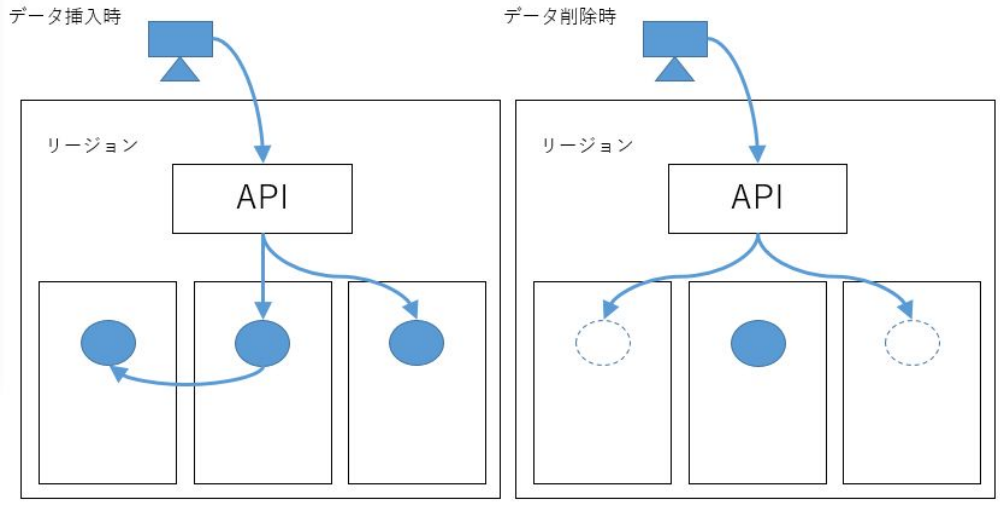
オンプレと変わらない

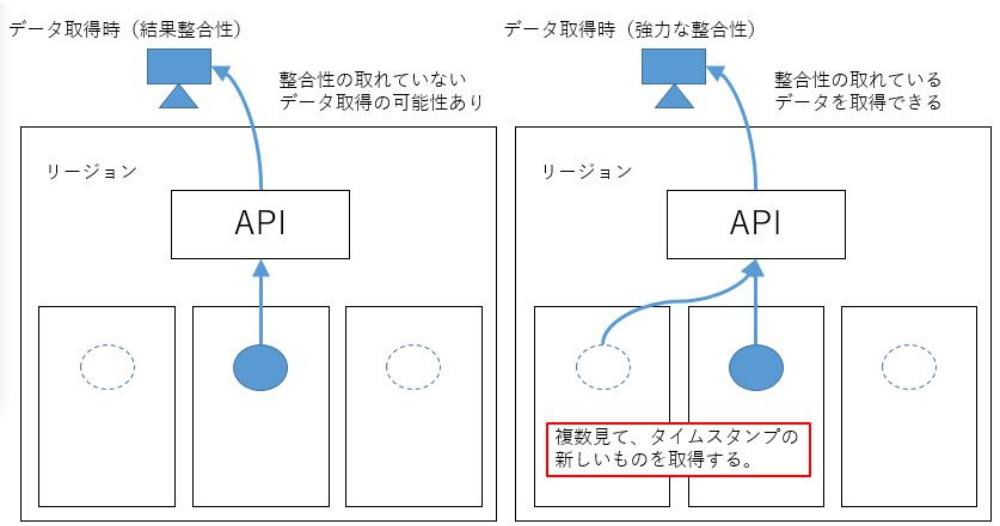
マネージド型データベース

- リレーショナルデータベース
  - Amazon RDS
    - SQLServer, MySQL, Maria DB, PostgreSQL, Oracle, Aurora…
    - Auroraについて



      詳しくは以下のURLへ
      https://d1.awsstatic.com/events/jp/2017/summit/slide/D3T1-6.pdf
  - Amazon RedShift
- 非リレーショナルデータベース
  - Amazon DynamoDB
    - パーティションキーは、分散されるようにする。
      すると、ホットパーティション（同じパーティションにアクセスが集中し、パフォーマンスが出ない事象）が起きにくい。
    - 整合性のオプション
      - 結果整合性
      - 強力な整合性





- Amazon ElastiCache
- Amazon Neptune
- Amazon DocumentDB
  - MongoDB, Redis, Kafkaでの商用サービス制限問題
  - 「MongoDBサーバーから期待する応答をエミュレートする。」
    by https://aws.amazon.com/jp/documentdb/
    つまり、MongoDBのような挙動をする。

モジュール5 AWSにおけるネットワーキング パート1

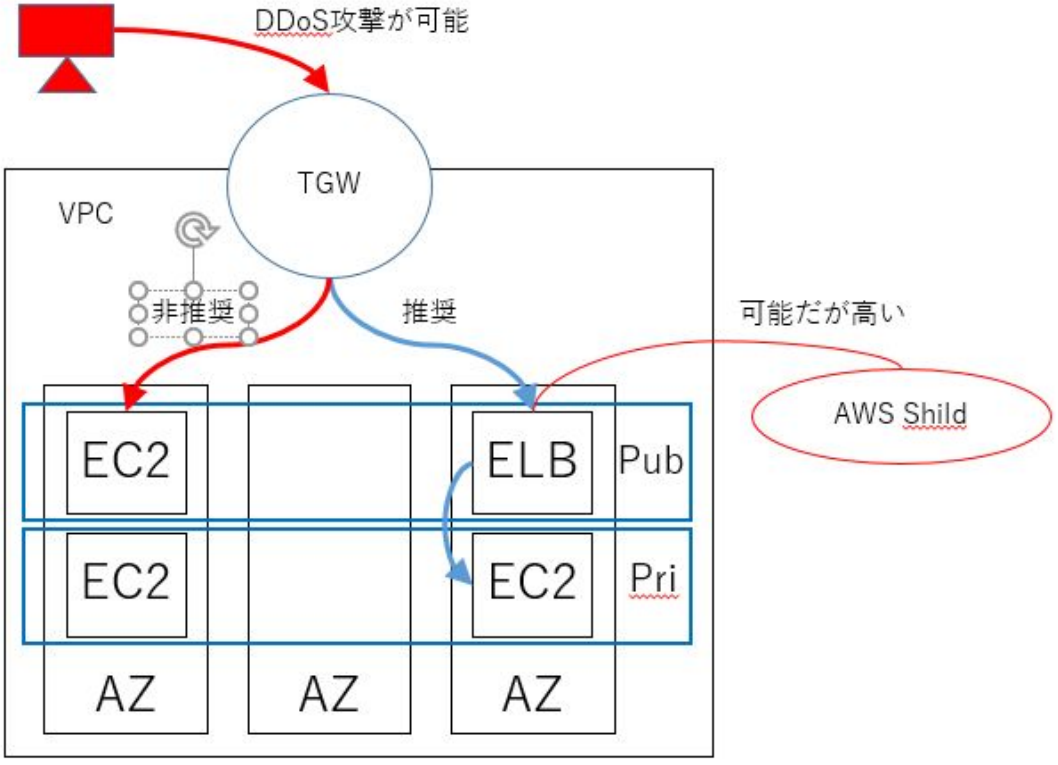
VPC (Virtual Private Cloud)

CIDR(Classless Inter-Domain Routing)（クラス間ドメインルーティング）：

ク ラスを使わないIPアドレスの割り当てと、経路情報の集成を行う技術です。

- リージョン毎に作成する。（複数のアベイラビリティゾーンは利用できる。）
- マルチVPC / マルチアカウント
    - マルチVPC
        - 単一のチーム、単一の組織で利用するのに適している。
        - 合計5つまでしかVPCが作成できない。（5VPC / リージョン）
          しかし、サポートに問い合わせをすることで「リージョン毎のVPC数」の上限緩和できる。
    - マルチアカウント
        - 大規模な組織や複数のITチームがある組織に適している。
        - 1アカウント 1事業部にしておくと、お金の管理も便利。
- IPアドレスの設定(CIDR)の例
    - CIDR 10.0.0.0/16
        - Network部　10.0.0.0
        - Prefix　16
          00000000 . 00000000 . 00000000 . 00000000
          ---------- . ---------- / ********** . **********
          256 * 256 → 65,536 IPsが利用可能
          00001010 . 00000000 . 00000000 . 00000000
          10 . 0 . 0 . 0
        - Prefix　17
          00000000 . 00000000 . 00000000 . 00000000
          ---------- . ---------- / ********** . **********

128 * 256 → 32,768 IPsが利用可能

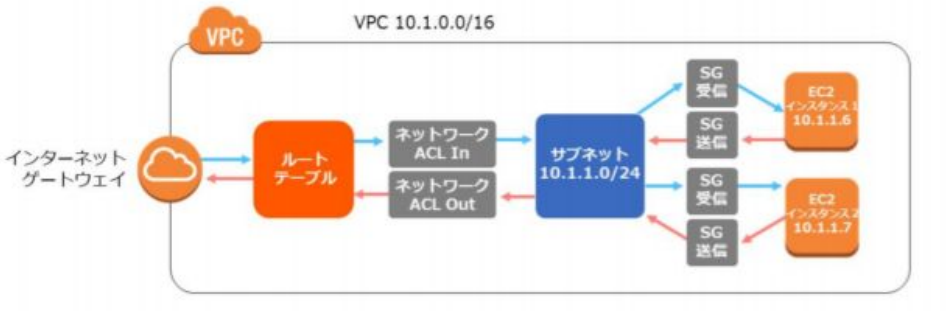


Elastic IP Address

VPCからIPアドレスを作成して、EC2等に紐づけられる。
再起動してもIPアドレスが変わらない。だからDNS等に登録できる。

2種類のファイアウォールの流れ

EC2をインターネットにつなげる流れ
ネットワークACLは、ステートレスだから、InとOutで違うところを通る。
SG(Security Gateway)は、ステートフルだから、InとOutで同じところを通る。



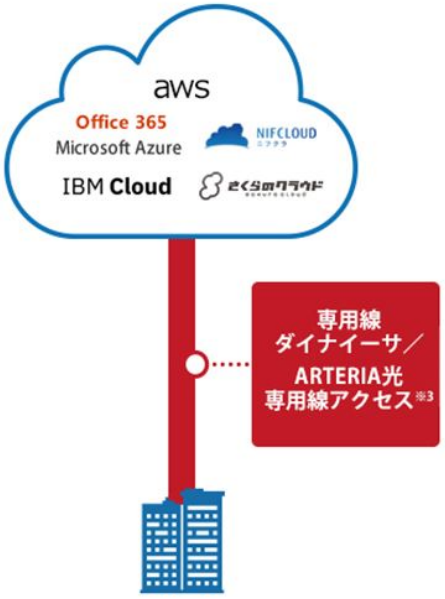
モジュール6 AWSにおけるネットワーク パート2

VGW (Virtual Private Gate Way)

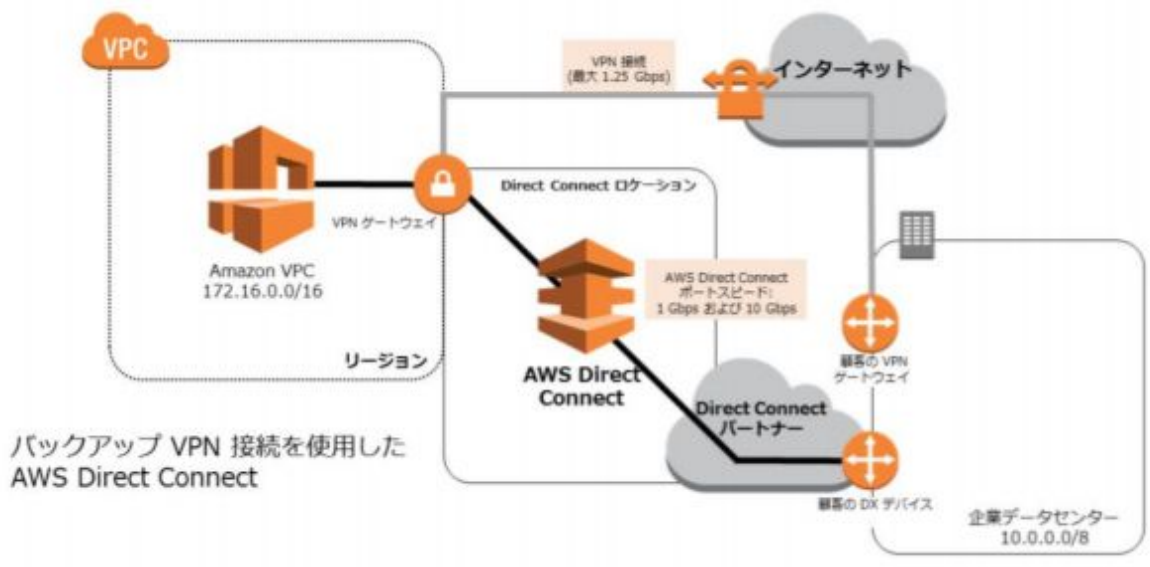
インターネット経由でオンプレとクラウドを接続できる。

DX(AWS Direct Connect)

オンプレとクラウドに専用プライベートネットワークで接続できる。
AWS Direct Pertner（会社）によって、値段が異なる。



ネットワークの構築方法（合わせ技：VPNとDX）

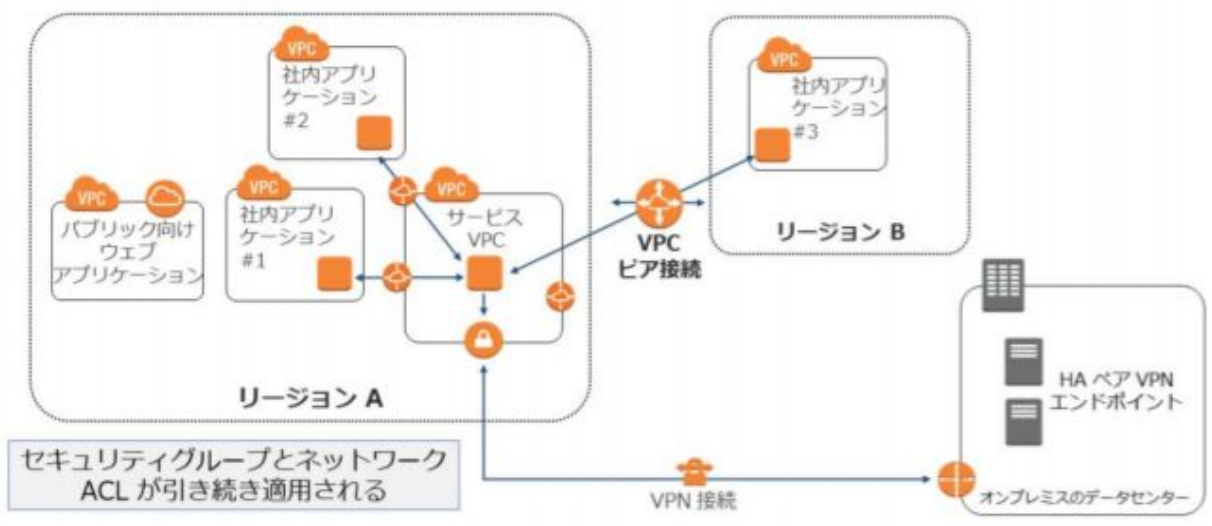


VPC間の接続

昔：VPC内でしか通信できなかった。
今：VPC peeringを利用することでVPC間/AWSアカウント間での通信ができるようになった。

接続するための条件

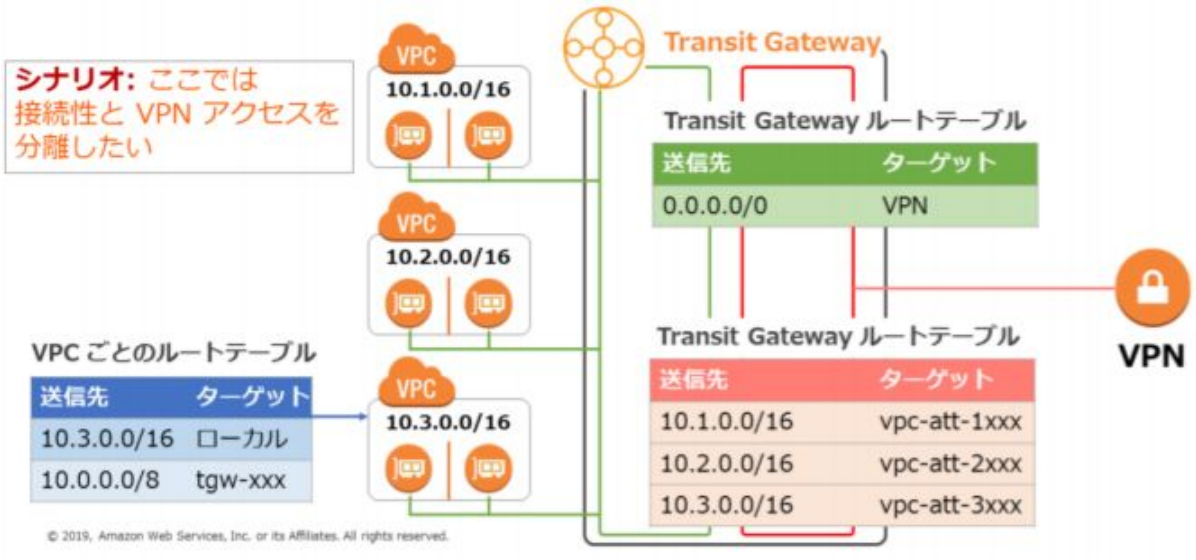
- VPCのCIDRは異なっている必要がある。



1. サービスVPCに認証機能などの共通処理を入れておく。
2. サービスが増えていくと、サービスVPCへのアクセスが増えていく。
3. サービスVPCはスケールアウトできるようにしておく必要がある。

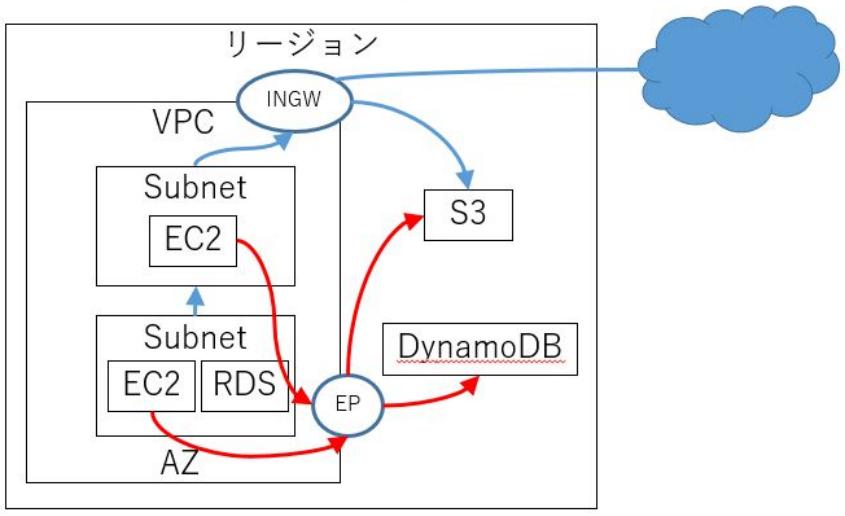
Transit Gateway

各VPCからオンプレにアクセスできるようにする例
（VPNからオンプレに接続されている。）

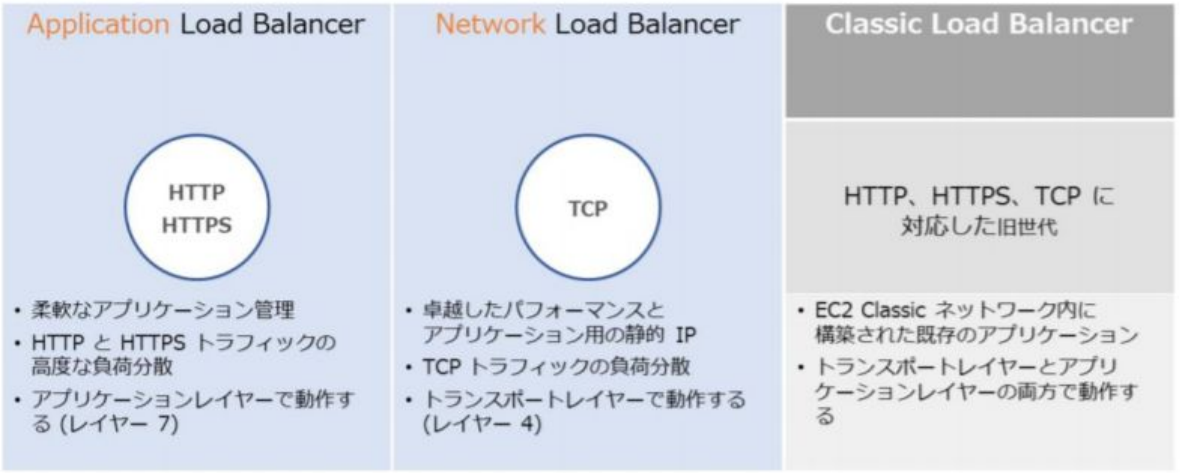


VPC EP(End Point)について

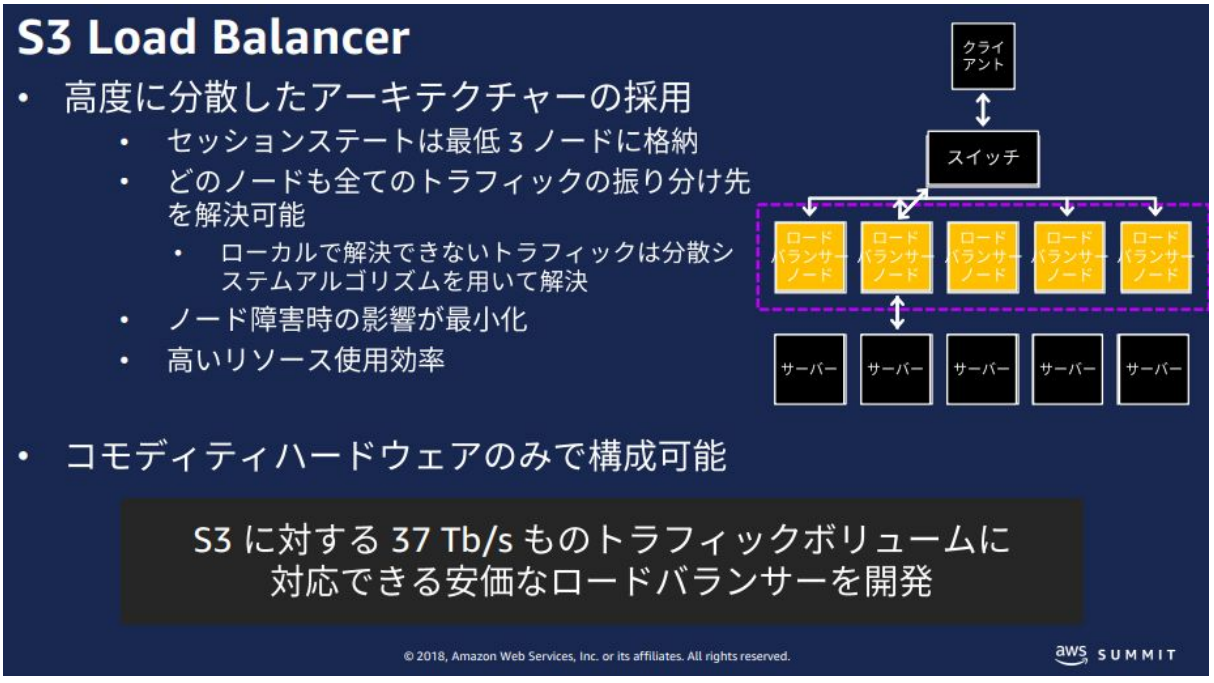
VPC EPを使うことで、インターネットに出なくても、ほかのサービスにアクセスできる。



ELB (Elastic Load Balancing)について

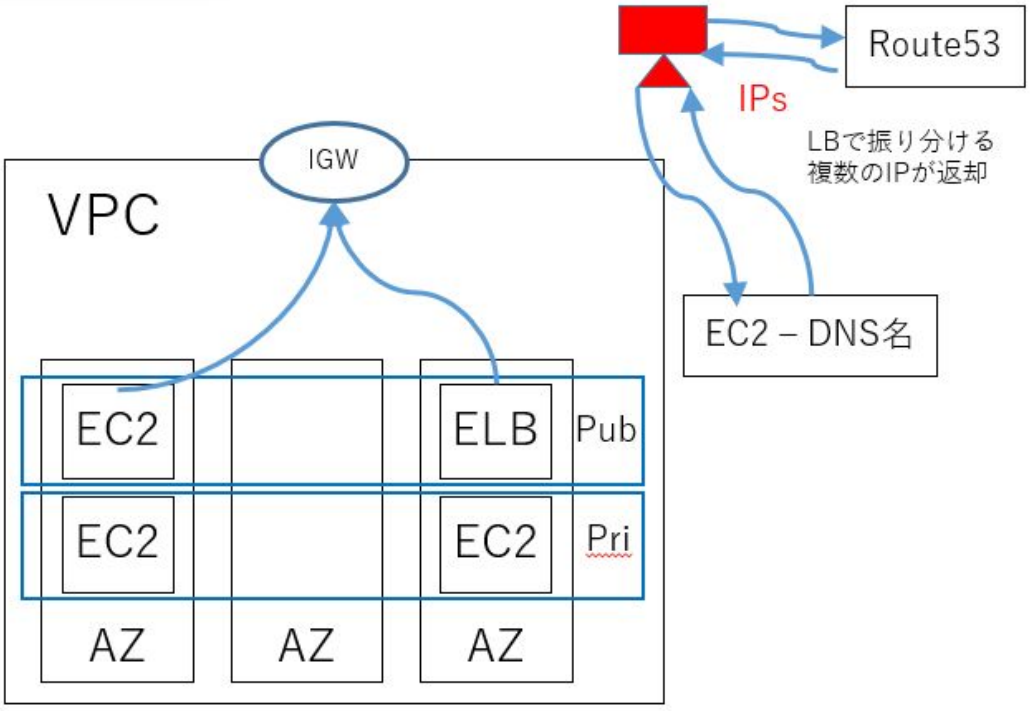


もともと、Classic LBだけだったけど、
Application LB(L7)とNetwork LB(L4)ができた。
Network LBは分散処理を実現するためにできた。
下記を応用した。
https://d1.awsstatic.com/events/jp/2018/summit/tokyo/aws/21.pdf



ELBの名前解決の動作について

Route53から複数のIPが返却される。



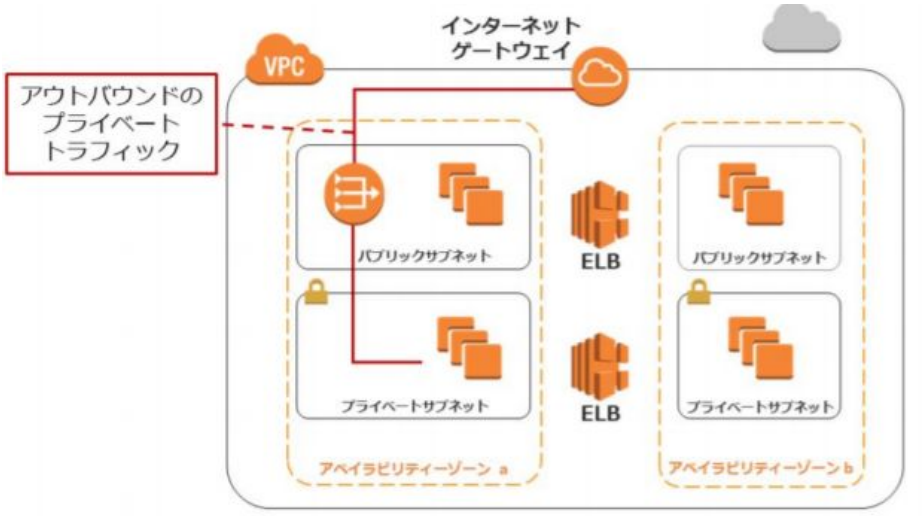
TLSターミネーション

内側は平文で、外側はSSLにできる。

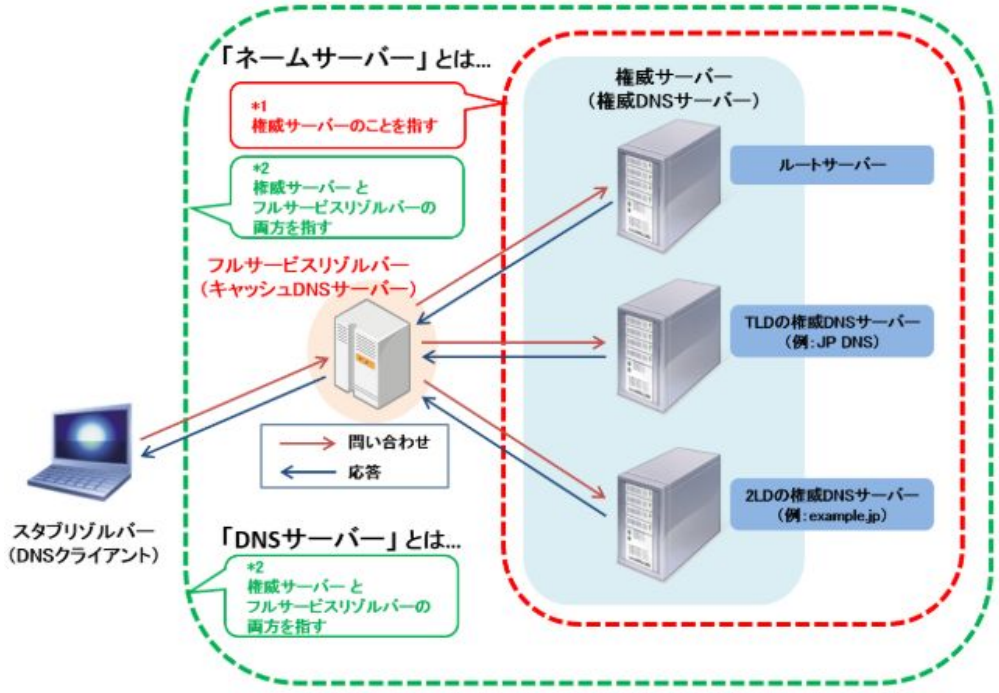
高可用性について

壊れたときは、手動メンテナンスではなくて、自動的メンテナンスにしましょう。

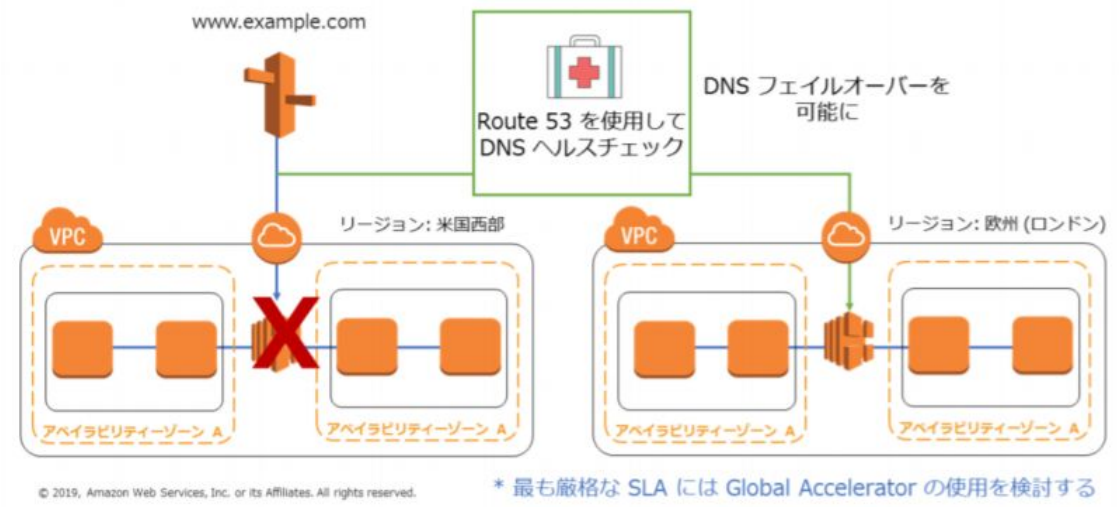
- AZ単位で壊れたとき
  AZごとに、NAT(Network Address Translation)を作成しましょう。
  NAT：プライベートIPアドレスをグローバルIPアドレスに変換するもの。



- リージョン単位で壊れたとき
  Route 53を利用しましょう。
  DNSサーバには2種類ある。
    - キャッシュサーバ（フルサービスリゾルバ）
    - コンテンツサーバ（権威DNS）⇒ Route 53



    - Route 53のオプションにヘルスチェックがある。



モジュール7 Identity and Access Management (IAM)

IAMとは

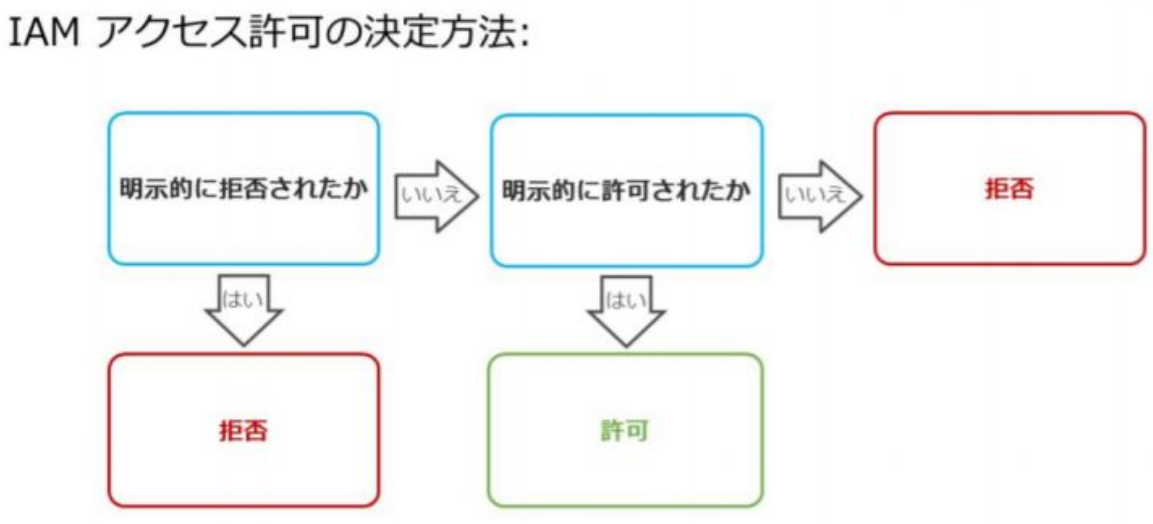
AWSのAPIに対する認証と認可の機能を提供するサービス

- 認証
  接続元が誰なのかを特定する処理
  ー　例：Twitterで認証
  ー　IAMユーザー
- 認可
  権限を割り当てる処理
  ー　例：Qiitaが機能を提供
  ー　①　ユーザに紐づくIAMポリシー
  ー　②　ロールに紐づくIAMポリシー

IAMユーザーを利用することがAWSの推奨。
(rootアカウントは利用しない。)

アクセス許可の付与

- リソースベース
  - AWSリソース（Amazon S3やAmazon Gracier等のポリシー）
- アイデンティティベース
  - IAMポリシーをユーザに対して割り当てること
  - IAMポリシーとは
    - 誰？（プリンシパル）
    - どこに？（ARN:Amazon Resource Name）
    - どんな処理を？（Action）



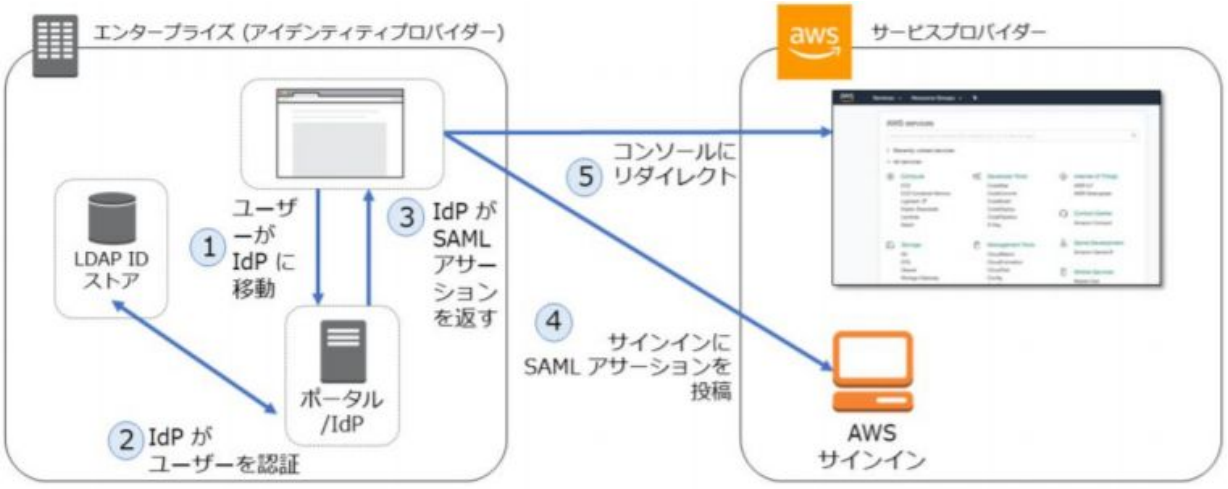
ユーザーフェデレーション

IAMロール

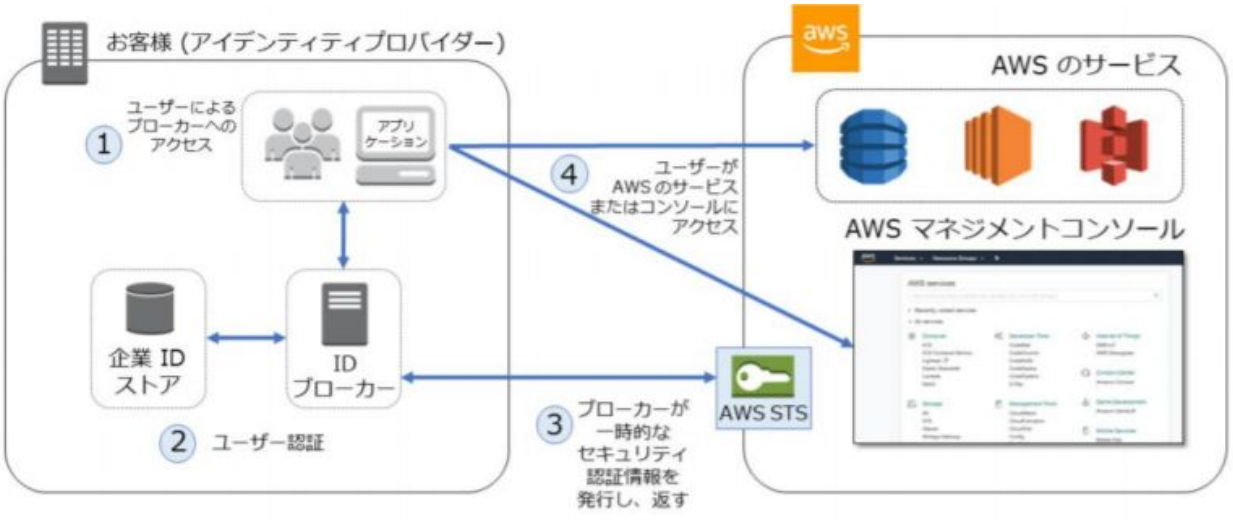
ユースケース:

- AWS リソースに AWS のサービスへのアクセスを提供
- 外部で認証されたユーザーにアクセスを提供
- 第三者にアクセスを提供
- ロールを切り替えて次のアカウントでリソースにアクセス:
  - 自分の AWS アカウント
  - その他の AWS アカウント (クロスアカウントアクセス)

SAMLを利用したユーザ認証の流れ（オンプレで認可権限を管理していた場合）

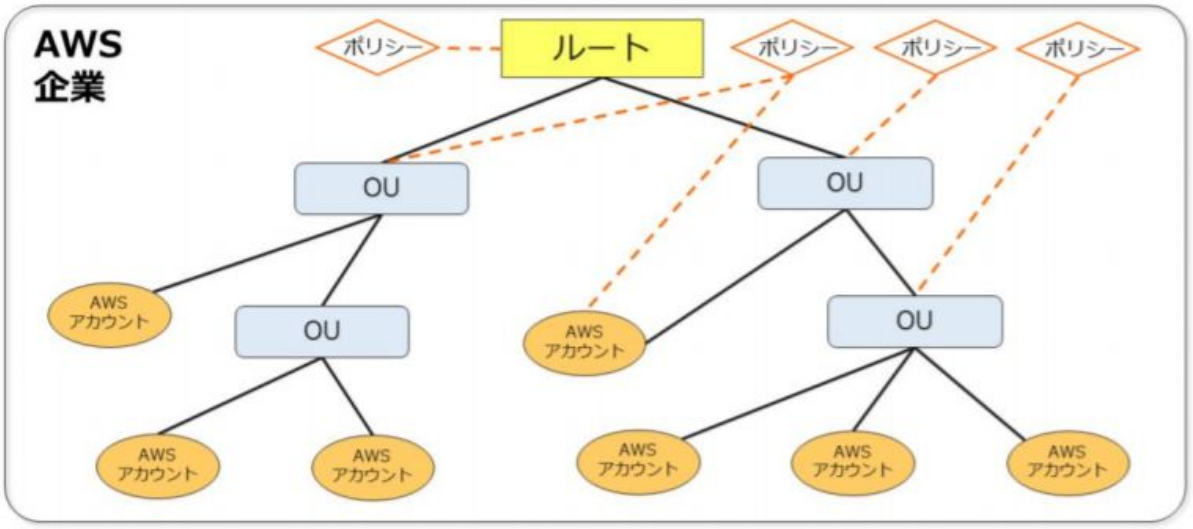


STS認証ブローカーを利用したユーザ認証の流れ



AWS Organizations

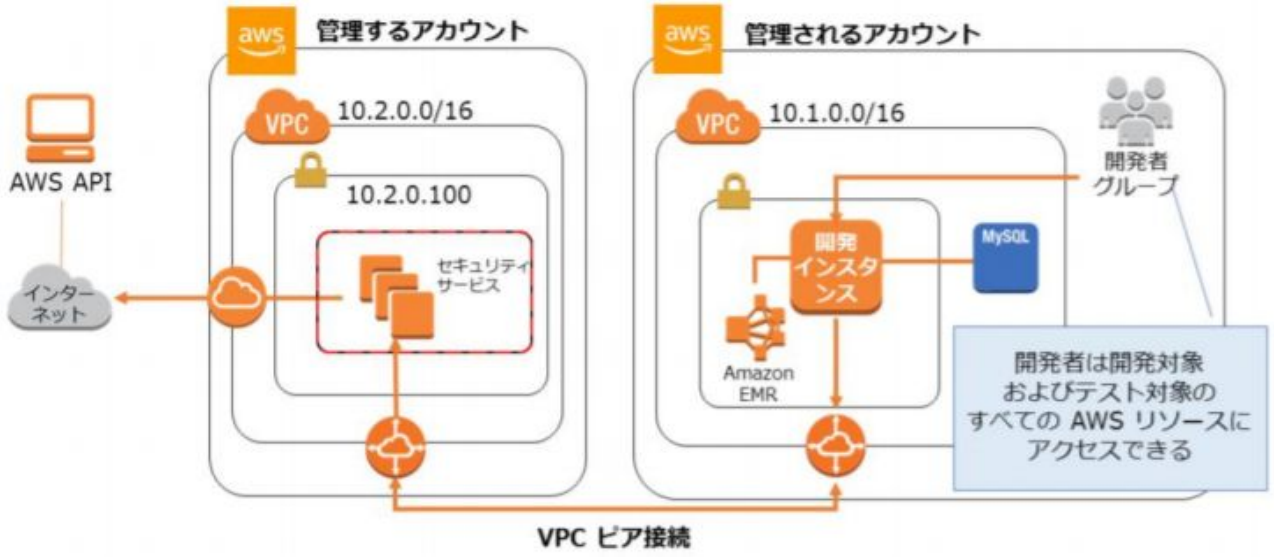
各AWSユーザの権限を管理できる。



ユースケース

<u>インターネットに出る場合は、管理するアカウントを通るようにする場合</u>

以下のような構成をする際に、AWS Organizationsを利用することで、各ユーザに
IGW(Internet Gate Way)を作成を制限できる。



モジュール8 伸縮性、高可用性、モニタリング

言葉についての復習

- 高可用性
    - HA(High Availability)
      ダウンタイム発生あり
      ソフトウェアで、障害があった際に切り替える。
      そのため、少しだけダウンタイムあり。
- 耐障害性
    - FT(Fault Tolerance)
      ダウンタイム発生なし
      24年間無停止で稼働し続けてきたサーバーが2017年4月にいよいよ引退
      FTサーバとは、ハードウェアですべてが冗長構成となっている。
      https://gigazine.net/news/20170201-ft-server-24years/

オートスケーリング

EC2, DynamoDBは使える。RDSは使えない。

伸縮自在性について

- 時間ベース
  予期できる場合に利用する。
  M-1グランプリでの時間ベースでの拡張例
  https://d1.awsstatic.com/events/jp/2017/summit/slide/D3T5-3.pdf
- ボリュームベース
  予期できない場合に利用する。

Amazon CloudWatch

何かのイベントをきっかけに、あるターゲットに対して、アクションを実行できる。

Amazon CloudTrail

プログラムやユーザが、プログラムコールを実行する。
実行結果のログがS3へ出力されるが、可視化されていない。
世の中の方が可視化するための方法を出してくれている。

参考URL:
「AWS CloudTrailのログをAmazon Elasticsearch Serviceへ転送して可視化してみた」
https://dev.classmethod.jp/cloud/aws/cloudtrail2elasticsearch/

Cost Explorer

どこで予算を消費しているのかが可視化されている。

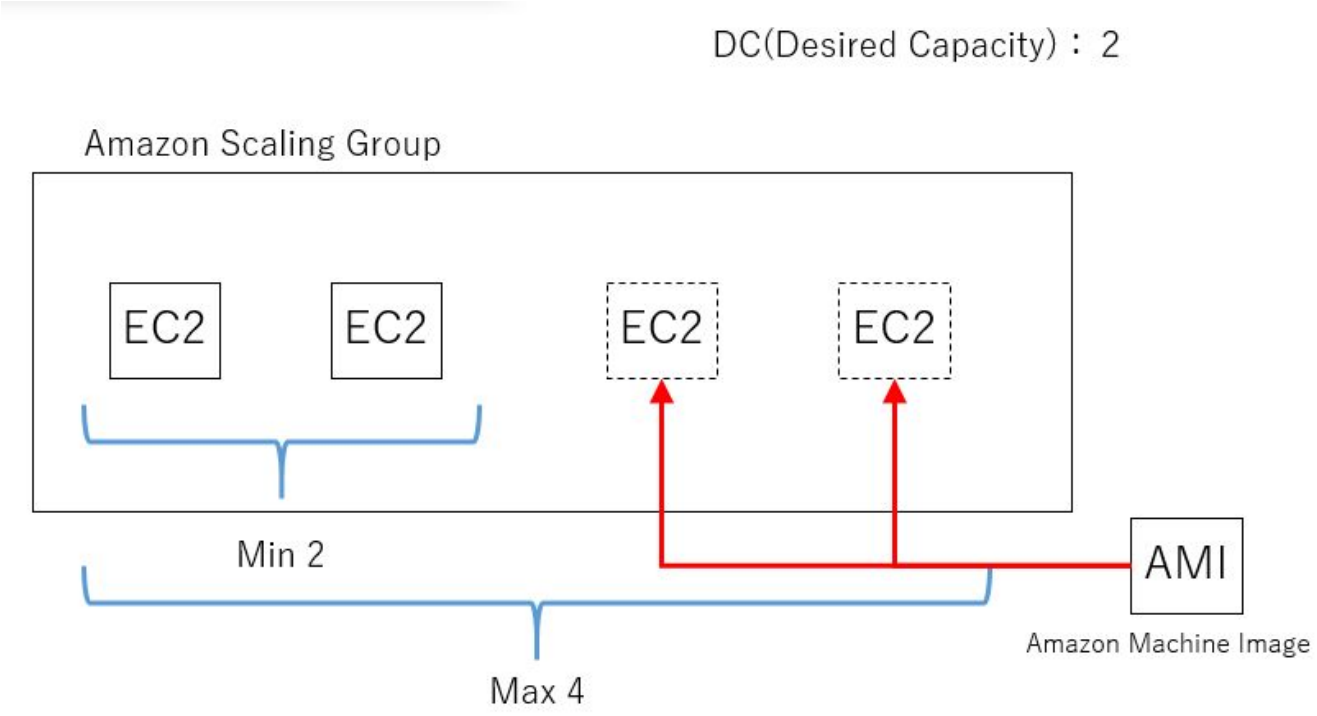
Amazon EC2 Auto Scaling

4つのAZで、4つのEC2があって、
2つのAZが壊れてしまった場合、残り2つのAZで2つずつのEC2が作成される。
結果、元と同じように、4つのEC2で稼働させることができる。

Auto Scaling Group

決めること

- 必要なキャパシティー
- 最小キャパシティー
- 最大キャパシティー



スポットインスタンス

あえて1つ前の世代を入れることで、スケーリング時のリソース枯渇を避けることができる。

参考URL
https://www.slideshare.net/GedowFather/gedow-style-aws-spot-instance

オートスケーリングの検討事項

Auto Scalingの良いルール：

- CPU 60%以上　EC2 + 1
    - スケールアウトは素早く
- CPU 20%以下　EC2 – 1
    - スケールインはゆっくり

Auto Scalingの悪いルール：

- CPU 60%以上　EC2 + 1
- CPU 40%以下　EC2 – 1

1〜4が繰り返し実行される。。

1. ASG内でのCPU利用率： (65 + 55) / 2 = 60
2. 1台増やす
3. ASG内でのCPU利用率： (65 + 55 + 0) / 3 = 40
4. 1台減らす

Amazon RDSのスケーリング

シャーディングを使用して、書き込み速度を早める方法。

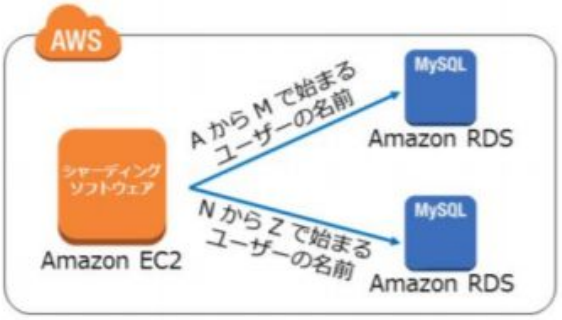
シャードがなければ、すべてのデータは
1 つのパーティションに入る

- 例: A から Z で始まるすべてのユーザーの名前が 1 つのデータベースに入っている

シャーディングを使用して、データを
大きなチャンク (シャード) に分割

- 例: 姓が A から M で始まるユーザーを 1 つのデータベースに、N から Z のユーザーをもう 1 つのデータベースに分ける

ほとんどの場合、シャーディングによって
パフォーマンスと運用効率が向上する



DynamoDBのスケーリング

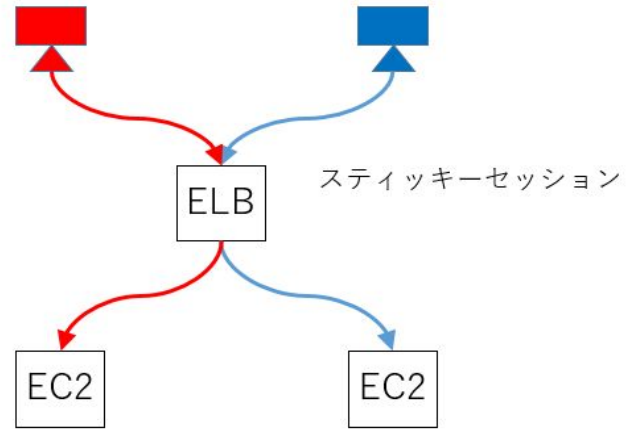
- プロビジョニングのスケーリング
  一気にアクセスが来た場合に、対応しきれない部分についてはエラーで返却される。
    - アダプティブキャパシティー
      プロビジョニングされたキャパシティーの合計 = 400WCU
      消費されたキャパシティの合計 = 250 WCU
      パーティション数は4つ。
      各パーティションでの消費量は、
      50/100, 50/100, 50/100, 100/100の場合、
      100のパーティションにアクセスが来るとエラーになる。

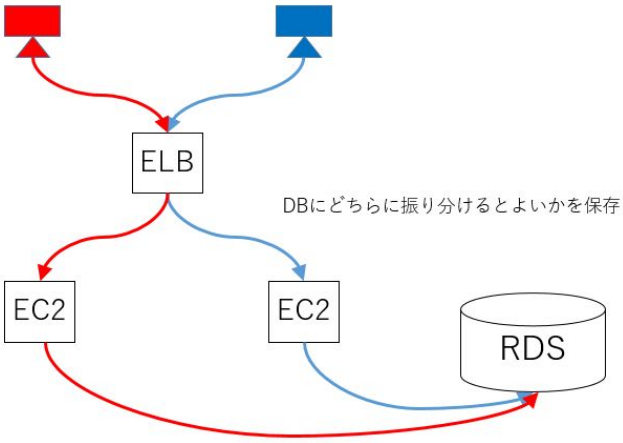
      アダプティブキャパシティの場合では、
      100のパーティションに対して追加でアクセスが来ても大丈夫なようにできる。
- オンデマンドのスケーリング
  一気にアクセスが来ても対応しきれる。

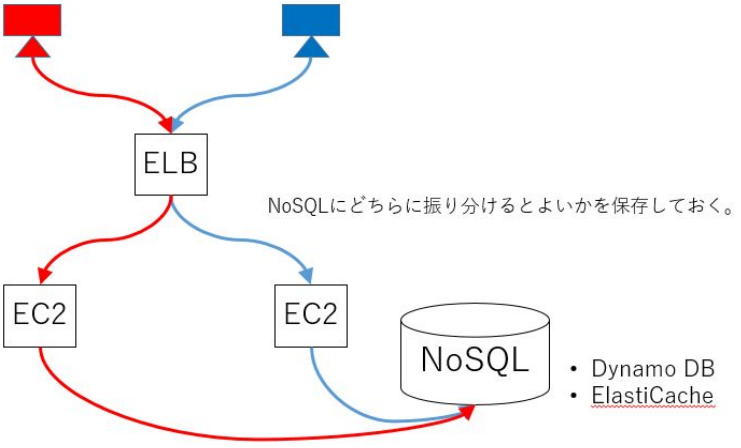
Auto Healing

- 単にEC2を作成した場合
    - マシンが壊れても自動復旧はしない。
- Auto Scalingを利用した場合
    - マシンが壊れた際に自動復旧できる。

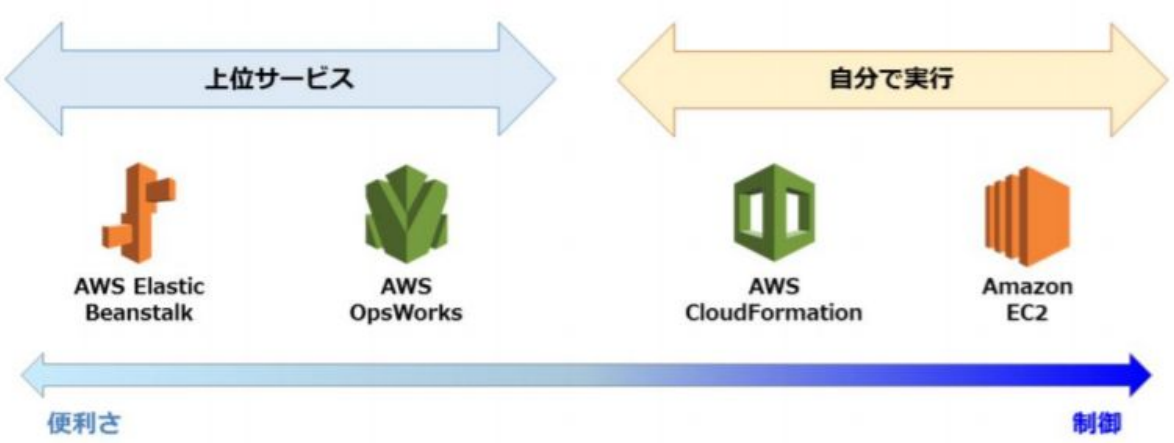
- DNSラウンドロビン
  一つのドメイン名に複数のIPアドレスを割り当てる負荷分散技術の一つ
- Connection Draining
  ソフトウェアのメンテナンスなどで使える機能。
  ELBから切り離してもリクエスト中のインスタンスへ指定秒数の間は通信は切れない。新規にリクエストがあっても既に切り離されたインスタンスへのアクセスはできない。
- スティッキーセッション
    - ELBを使っても、ユーザーごとに使うサーバーを固定できる。


ELB
スティッキーセッション
EC2
EC2


ELB
DBにどちらに振り分けるとよいかを保存しておく。
EC2
EC2
RDS


ELB
NoSQLにどちらに振り分けるとよいかを保存しておく。
EC2
EC2
NoSQL
• Dynamo DB
• ElastiCache

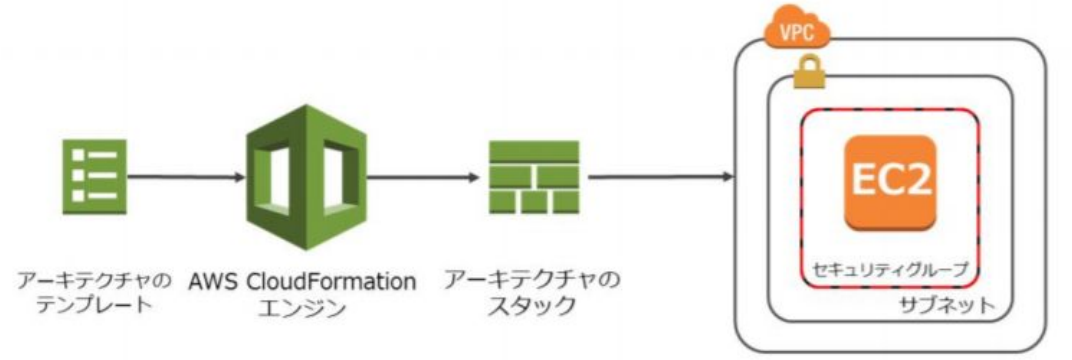
モジュール9 オートメーション



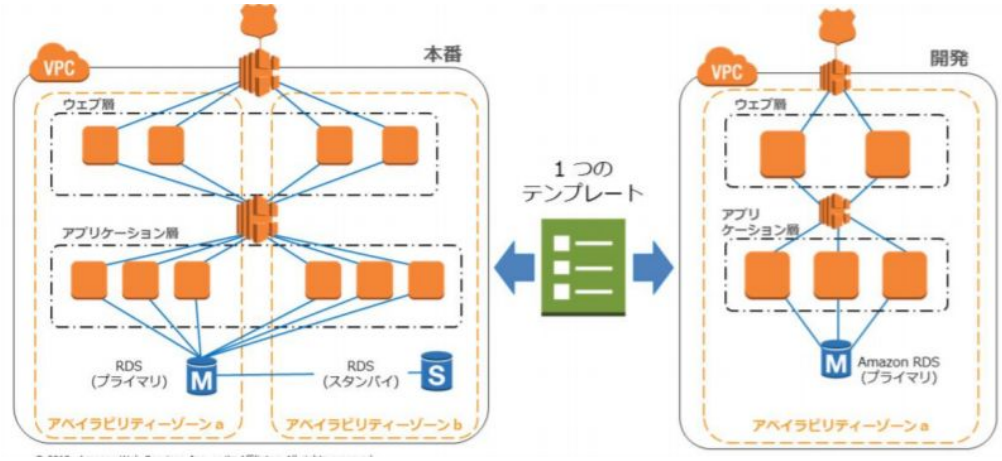
AWS CloudFormation (Infrastructure as Code)

JSONやYAMLから環境が生成される。

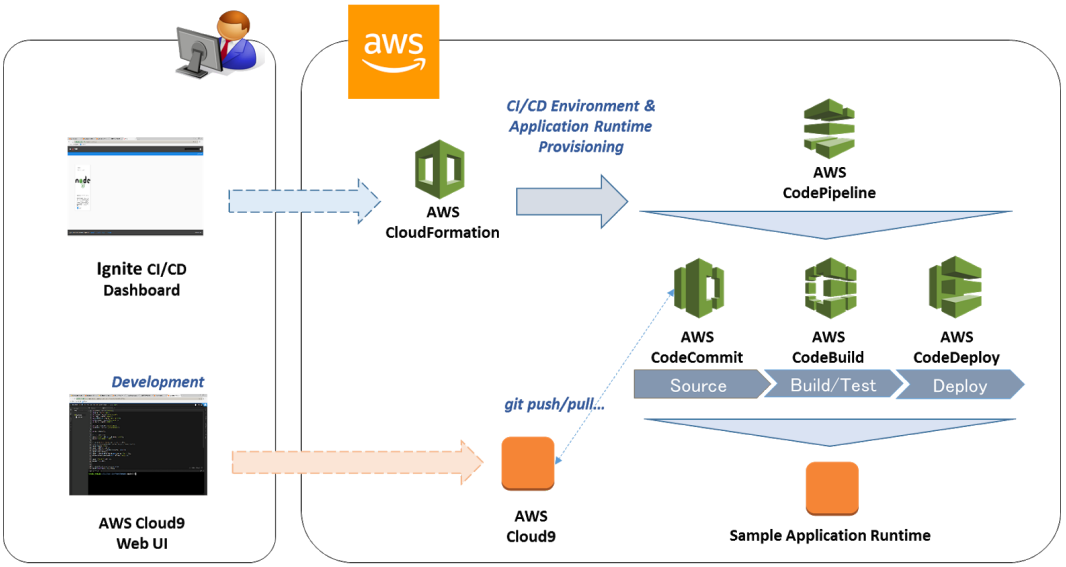
- JSONは、’,’の利用や、改行空白が無視される、コメント書けないので不便。
- YAMLは、タブで管理できて、コメント書けるので便利。



開発環境と本番環境が同じテンプレートから作成できる。



AWSでのDevOps関連サービスの全体像



Infrastructure as Codeとは

以下のような性質を持つもの

- 冪等性
    - 何度実行しても同じになる。
- ビルド-デプロイ-テストを繰り返すこと

AWS クイックスタート

すでに完成されている環境がある。
これをお客様の環境に合わせて、必要なものを足したり、不要なものを引いたりすると、楽に作成できる。
https://aws.amazon.com/jp/quickstart/?quickstart-all.sort-by=item.additionalFields.updateDate&quickstart-all.sort-order=desc

AWS System Manager

指定したサーバに対して、定期的にコマンドを実行させることができる。

- Windows Update
- ログ収集 など

AWS OpsWorks

- Ansible
  管理サーバ全てに対して、コマンドを一気に流すことができる。

AWS Elastic Beanstalk

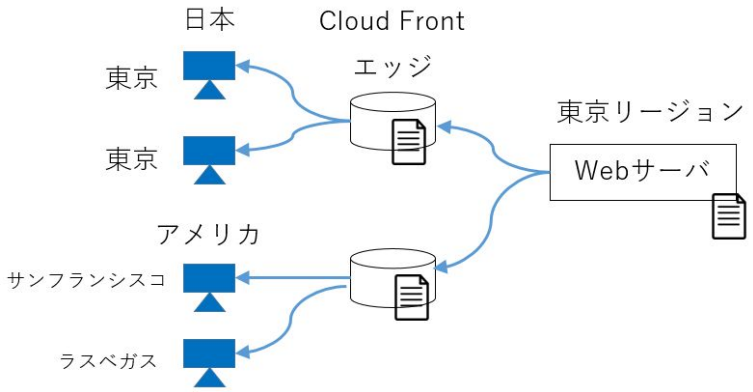
以下の環境を全自動で作成してくれる。
ただし、細かな設定などはできないというデメリットがある。

- ホスト
- オペレーティングシステム
- 言語インタープリタ
- アプリケーションサーバ
- HTTPサーバ

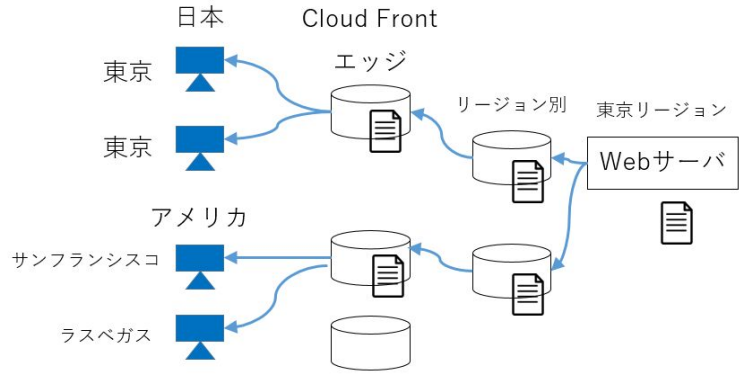
モジュール10 キャッシュ

しばらくの間、情報が古くなっても構わない場合にキャッシュを適用する。
（情報が古くなると、価値がなくなるものについてはキャッシュは適用すべきでない。）

- キャッシュの種類
  - サーバ側
    - エッジのキャッシュ
      CDN(Contents Delivery Network) ⇒ AWS Cloud Front
      Amazon Cloud Frontを利用すると、Amazon Shieldが動作して、
      DDoS攻撃を防ぐことができる。
      AZ別エッジ



      リージョン別エッジ



  - クライアント側
    - メモリ（ブラウザのキャッシュ等）
    - CPU

Amazon Cloud Frontを設定する方法

www.example.com
⇒ https://d～～～～～.cloudfront.net

- ２段階での名前解決
  ① http://www.example.com/
  ⇒ https://d～～～～.cloudfront.net
  (DNS CNAME　　：FQDN ⇒ FQDN) 任意のDNSでOK

  ② https://d～～～～.cloudfront.net　⇒ IP
  （DNS A　　：FQDN ⇒ IPアドレス) Route 53
  Route 53 Geoロケーションに基づくIPを返却する。
  意味：その場所に応じたDNSサーバを利用して、IPを返却する。
- 1段階での名前解決
  ① http://www.example.com/
  ⇒ https://d～～～～.cloudfront.net　のIPアドレス
  (DNS A Aliads　　：FQDN ⇒ IP) Route 53

キャッシュについて

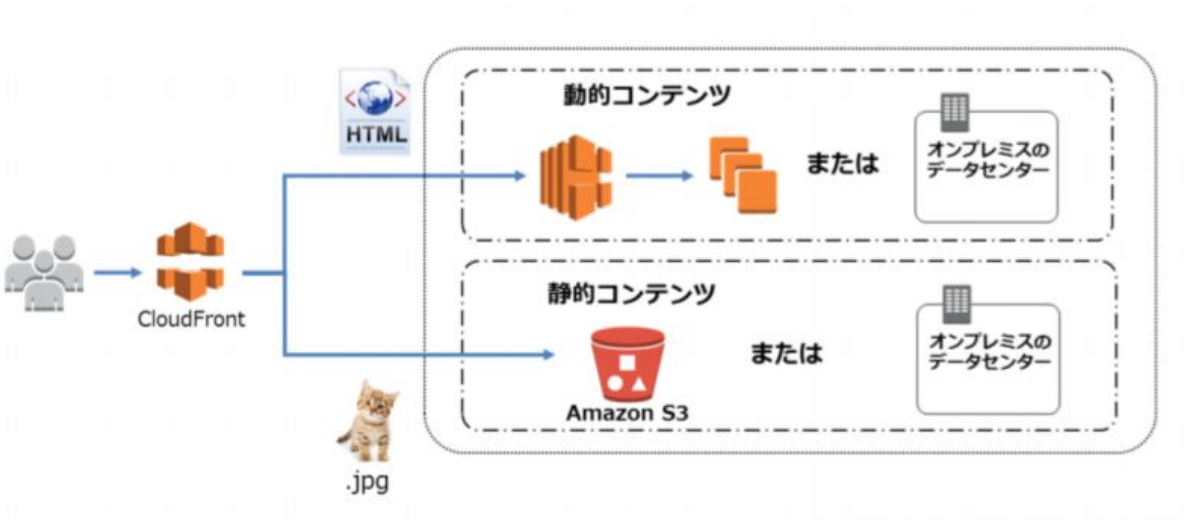
[**Amazonのクラウドエンジニアが選ぶ技術書**]
https://www.amazon.co.jp/技術書35選本/b?ie=UTF8&node=3517238051

HTTPについてのおすすめ本
https://www.amazon.co.jp/Real-World-HTTP-―歴史とコードに学ぶインターネットとウェブ技術-よしき/dp/4873118042/ref=sr_1_fkmr0_1?__mk_ja_JP=カタカナ&keywords=real+world+http+ミニ&qid=1565314234&s=books&sr=1-1-fkmr0
[ミニ版は無料 [Go言語のソースコードが載っていないだけ。]]
https://www.oreilly.co.jp/books/9784873118789/

メルカリでのキャッシュによる情報漏洩後の対応が模範解答

Cache-Control-Headerの設定が正しくなかった。
個人情報がキャッシュされた。
ほかの方がアクセスしたときに、ほかの方の情報を見ることができた。

詳細は下記の記事。
https://tech.mercari.com/entry/2017/06/22/204500

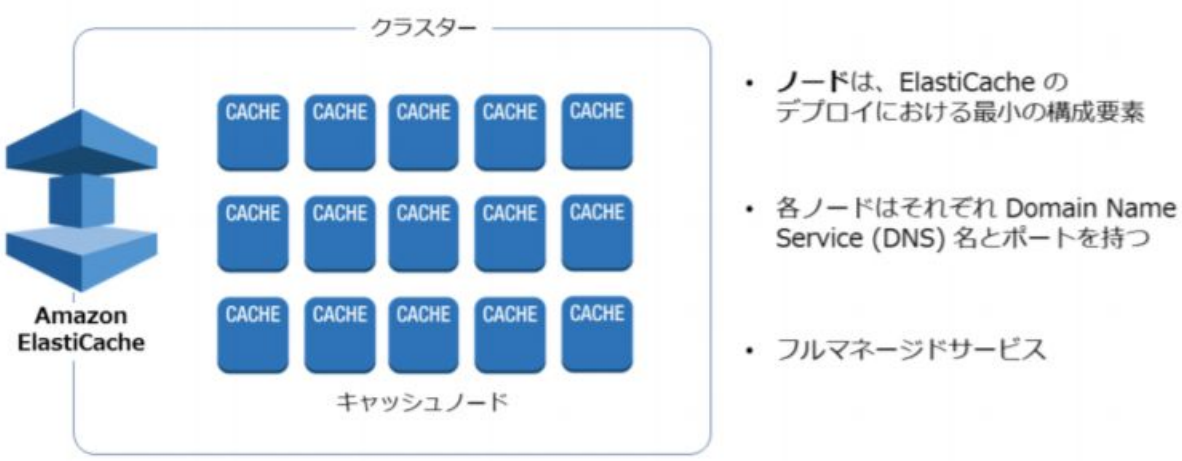


Amazon Cloud Frontのコンテンツの失効方法

Cache-Control-Headerで、失効期間を設定できる。
デフォルトでは失効期間は24時間になっている。

Amazon ElastiCache

DBへのアクセス内容をキャッシュさせることが可能。
応答はマイクロレベル単位。



- Redis
  永続性あり。（基本はメモリ、HDDへのデータの退避もしてくれる。）
  リモートディクショナリサーバーの略で、データベース、キャッシュ、メッセージブローカー、およびキューとして使用するための、高速でオープンソースのメモリ内 Key-Value データストアです。
- Memcached
  永続性なし。（メモリのみ）
  普通のメモリ使用のような使い方。

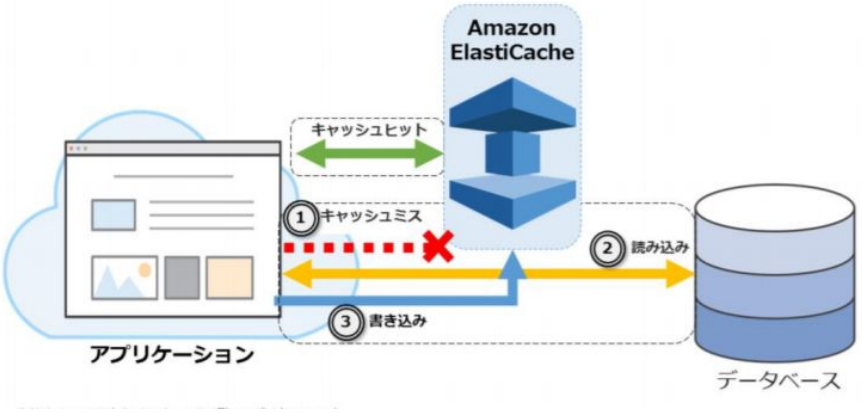
たいだいのフレームワークは、キャッシュサービスを提供している。
フレームワークが提供していない場合は、
フレームワークが対応している言語がサポートしているキャッシュの方法を調べる。

Redis と Memcachedの比較

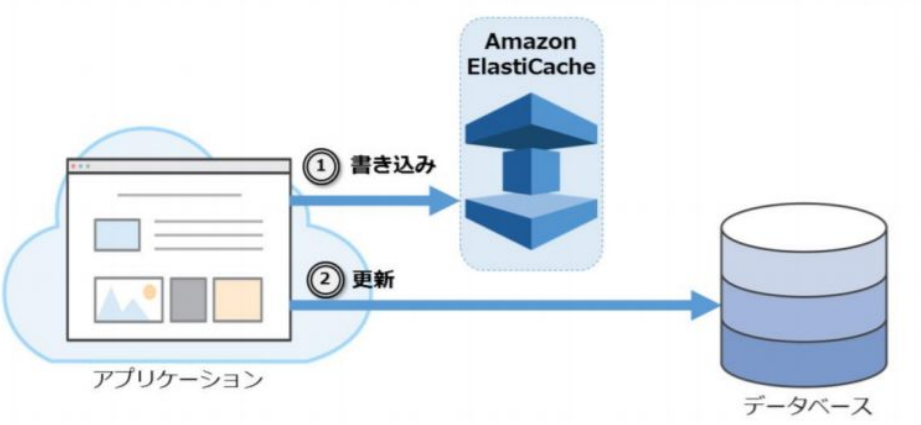
	Memcached	Redis
DB の負荷をオフロードするシンプルなキャッシュ	はい	はい
書き込み/ストレージの水平方向にスケールする機能	はい	いいえ
マルチスレッドパフォーマンス	はい	いいえ
高度なデータタイプ	いいえ	はい
データセットの並べ替え/ランキング	いいえ	はい
Pub/Sub キャパシティー	いいえ	はい
自動フェイルオーバー機能を備えた マルチアベイラビリティーゾーン	いいえ	はい
永続性	いいえ	はい

キャッシュへの書き込み方法について

- 遅延読み込み

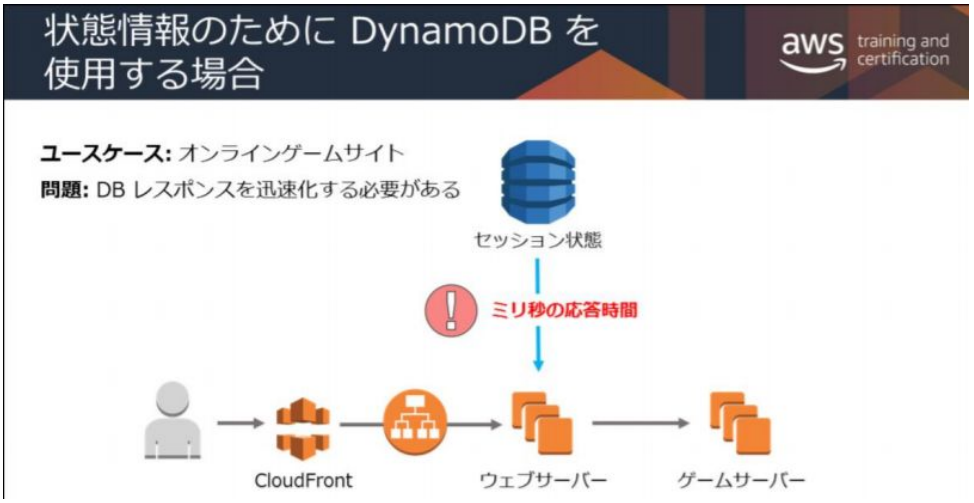


- ライトスルー

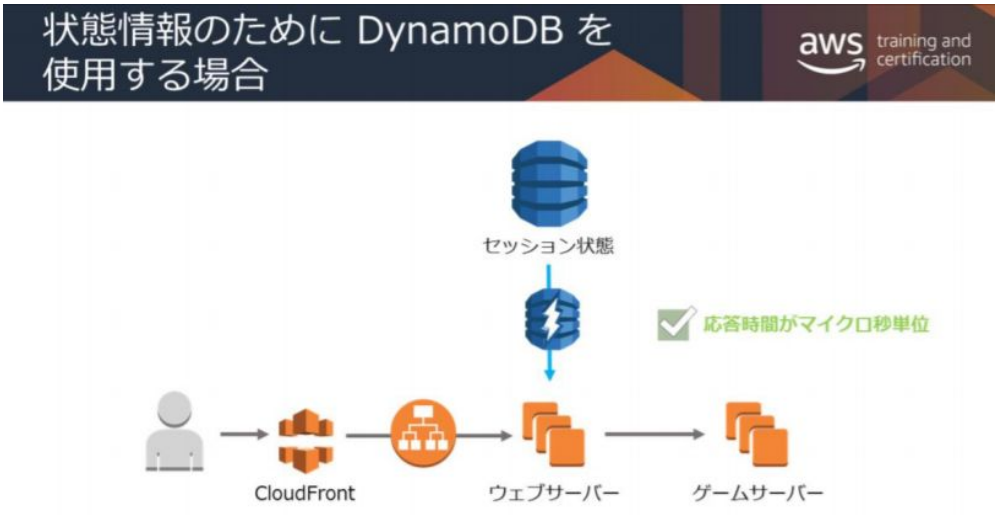


Amazon DynamoDB Accelerator

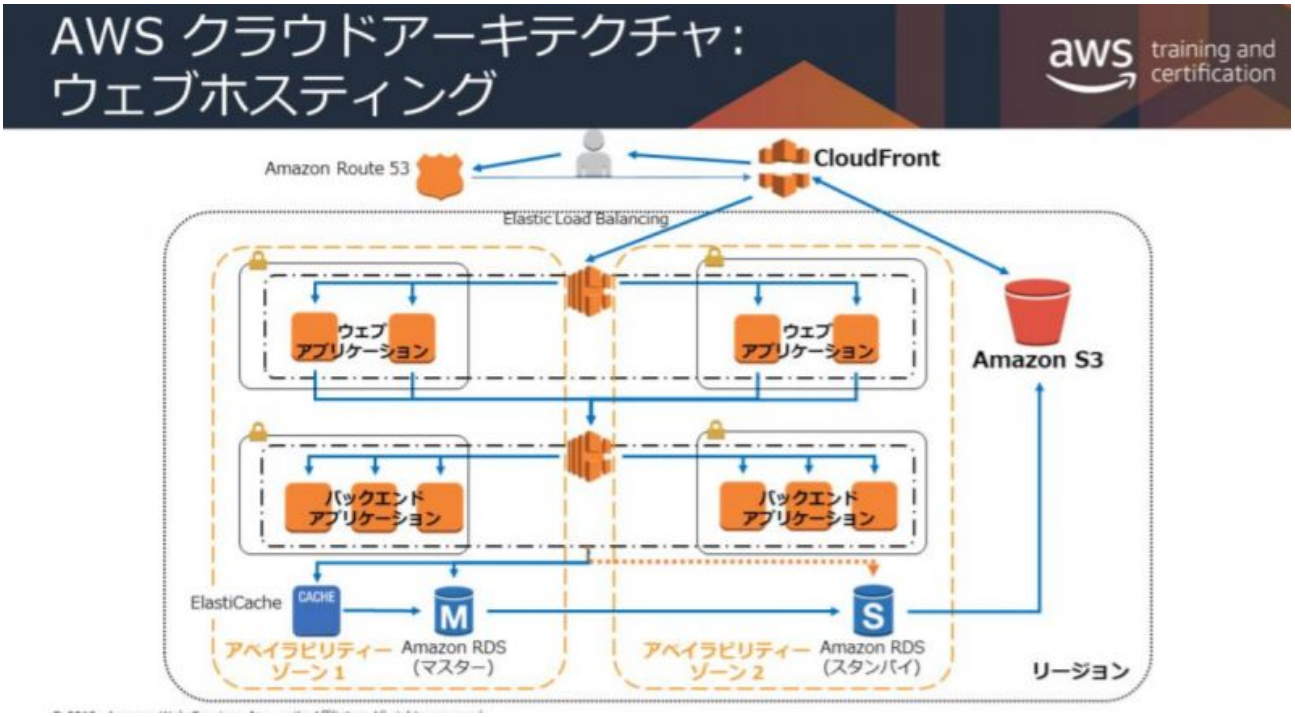
- DynamoDBを利用してミリ秒の遅延が発生する場合



- DynamoDB Acceleratorを間に入れることで解消できる。

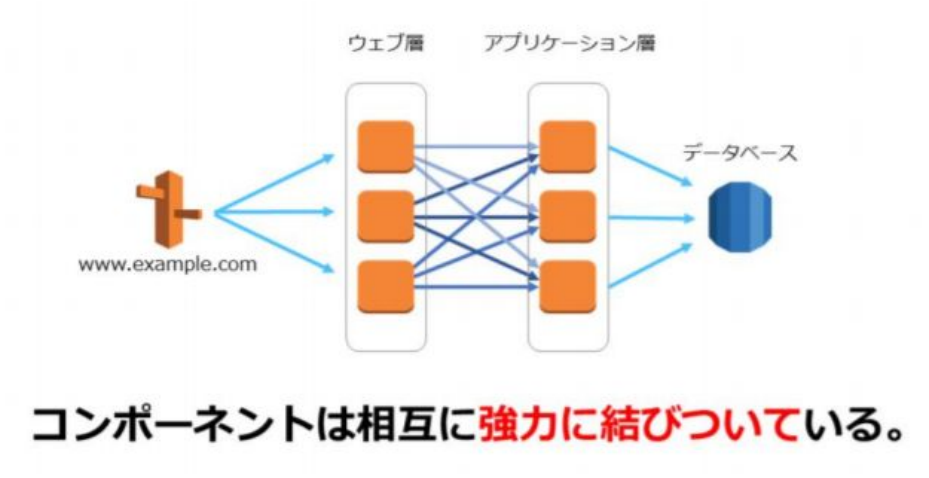


- Cloud FrontとElastiCacheを利用したクラウドアーキテクチャの例

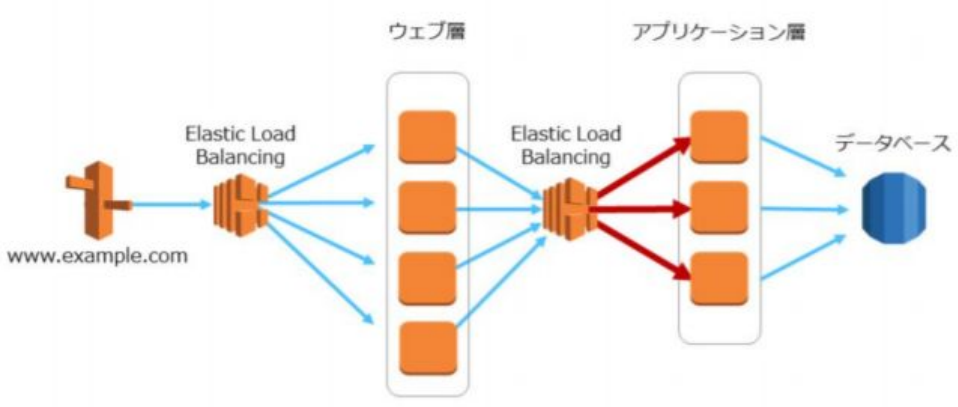


モジュール 11 分離アーキテクチャの構築

- 問題



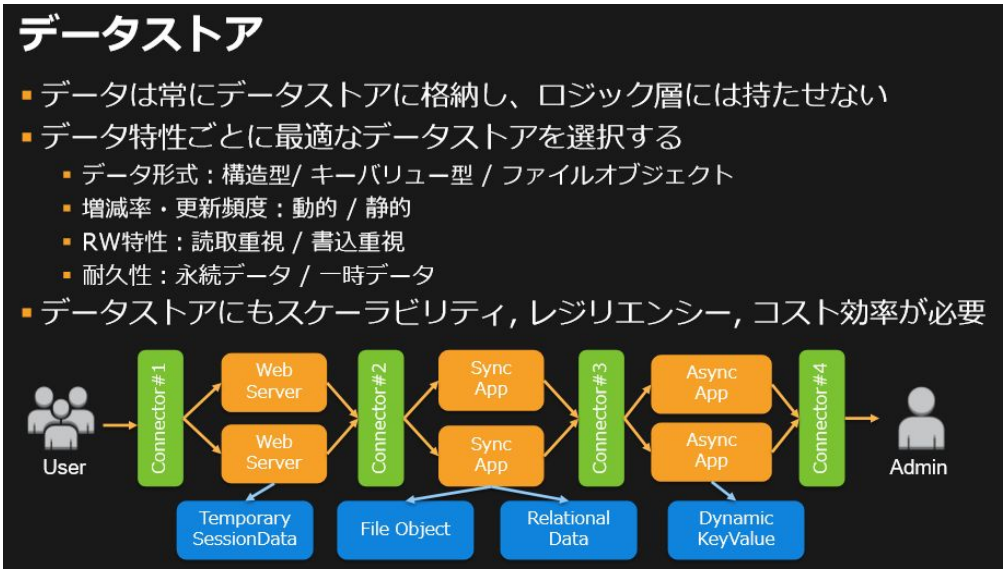
- 解決策



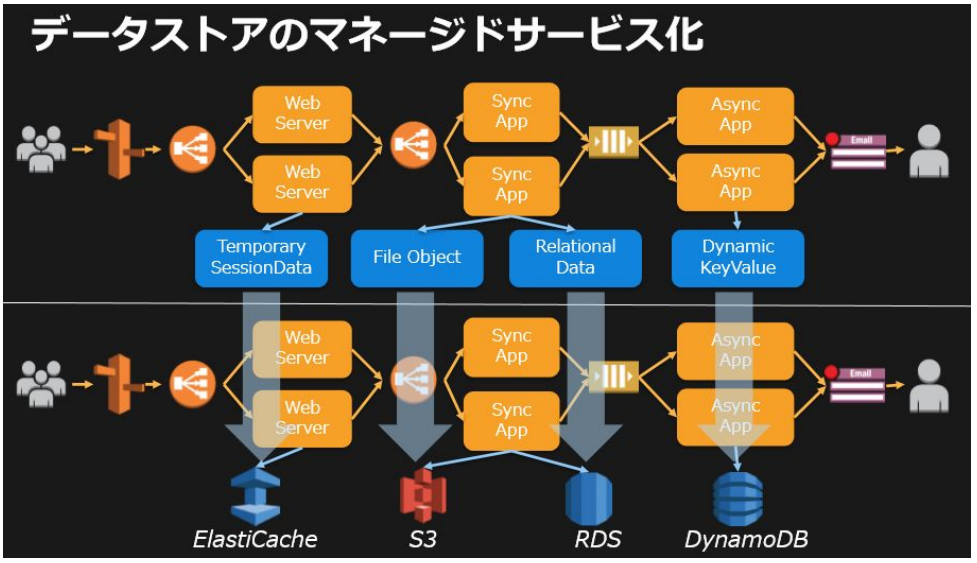
クラウド最適化アセスメント - Amazon AWS

https://media.amazonwebservices.com/jp/summit2016/3C-02.pdf

- 問題点（オンプレでTemporary SessionDataやFileObjectを管理すると、メンテ（運用）が大変になる。）



- 解決策（マネージドサービスを利用すれば運用が楽。）



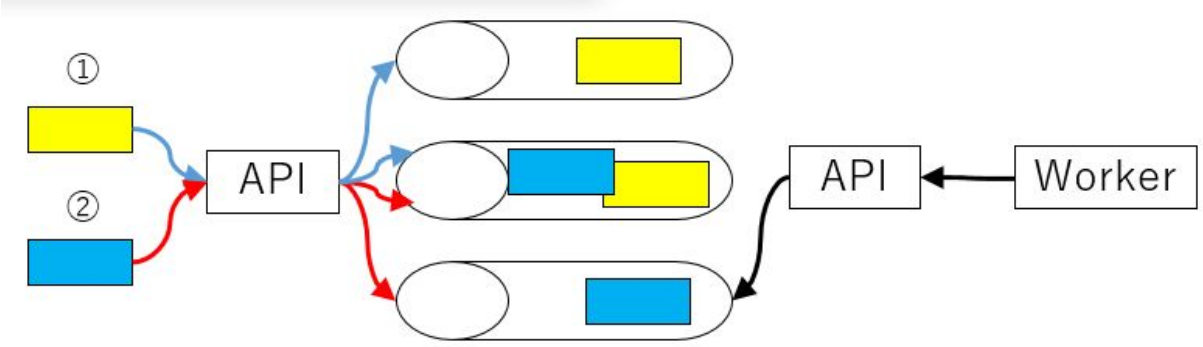
- サービスを疎結合につなげるための考え方。
  SOA サービス指向アーキテクチャ

Amazon SQS

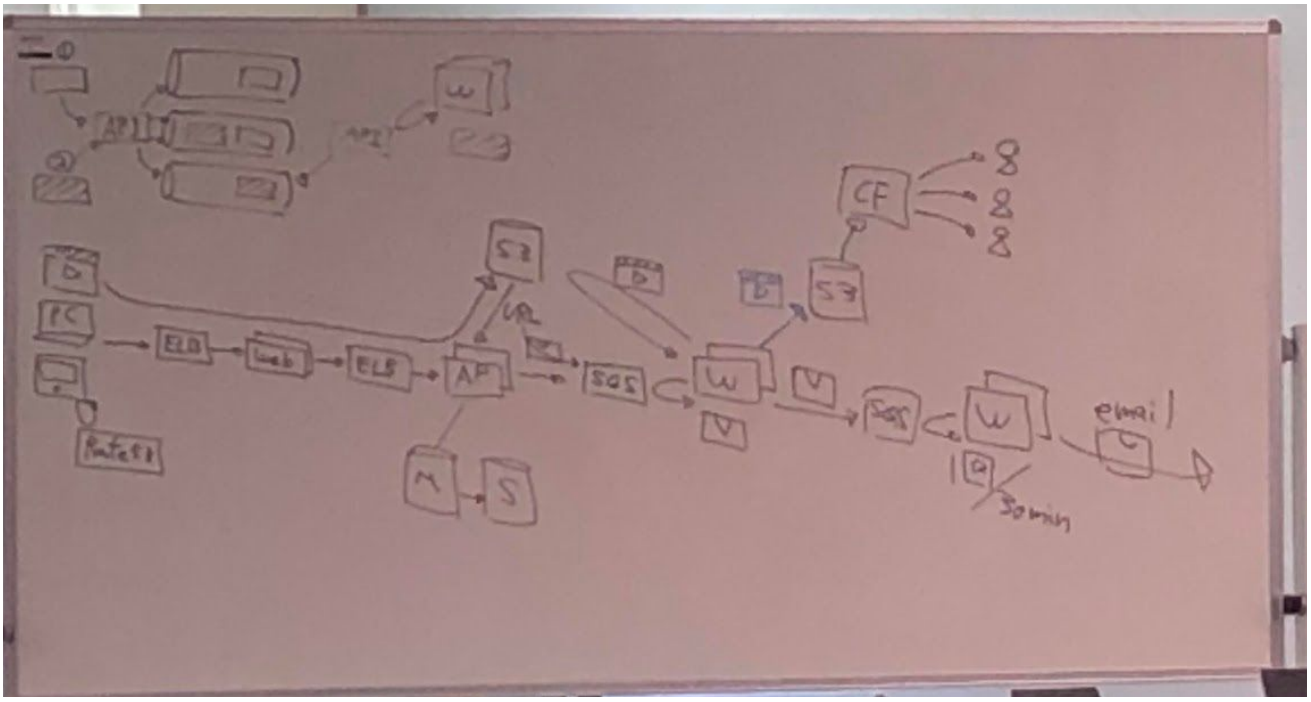
Amazon Elastic Load Balanceとの違いは、同期処理か非同期処理かが異なる。
SQSは、非同期処理。

キュータイプ

- 標準キュー
  - 少なくとも1回の配信
  - APIアクションあたり1秒あたりのTPS数はほぼ無制限
- FIFOキュー
  - メッセージが処理されるのは1回のみ
  - 1秒あたりの最大300件のメッセージをサポート

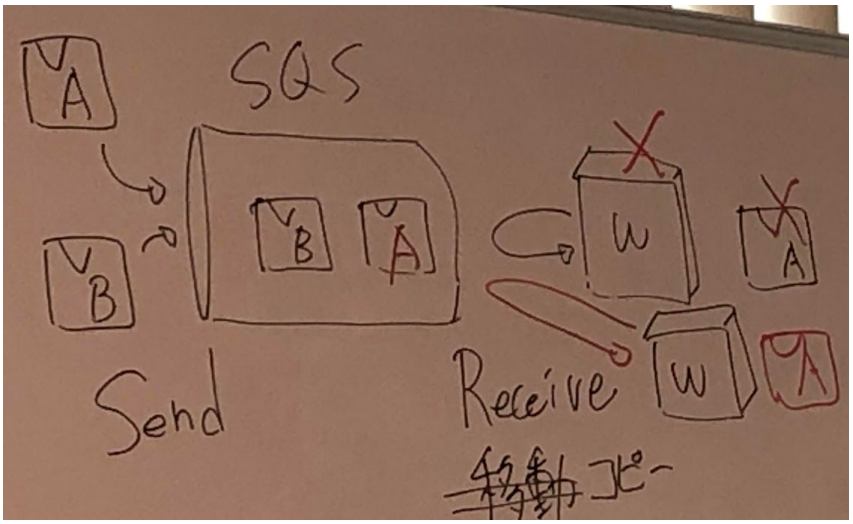


標準キューは速くて、無制限だけど、順番が保証されない



- ユースケース
    - 動画コンテンツなどは、SQSに入らないからS3に入れる。
    - SQSにはS3へ入れたときのURLを入れておく。
    - Workerで動画編集して、S3へ入れて、CloudFrontから配信する。
    - メール送信する内容はSQSに蓄えておく。（１件ごと送る訳にはいかないから。1ユーザー複数件の場合に迷惑。）
    - 1回/30minとかでSQSからデータを取って、ユーザにメールを送る。

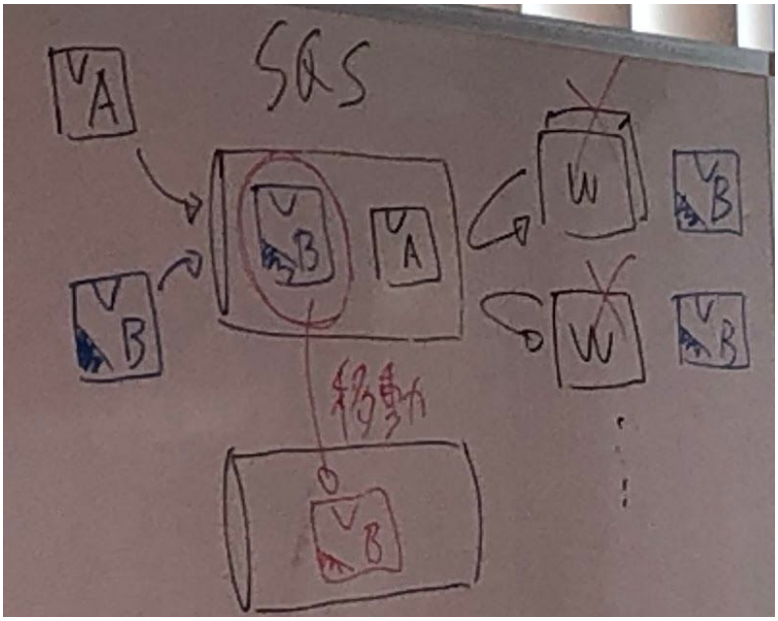
可視性タイムアウトについて



- 問題点
  SQSからデータを取得したWorkerが壊れた時

- 解決策
  レシーブした際に、30秒経てばSQSからデータが消える。
  データが処理できなかった場合、SQSにデータが復活して、処理できる。
    - 順番
        - send
        - recieve
        - delete

デッドレターキューサポート



VBは不正なメッセージのこと。

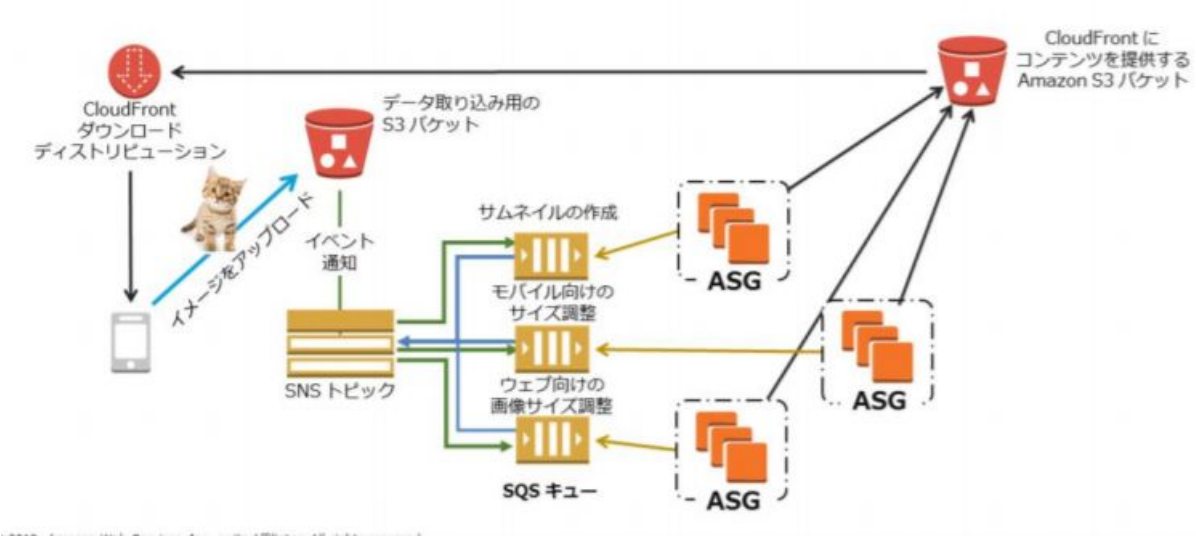
- 問題点
  不正なメッセージによりWorkerが壊れてしまい、
  同じメッセージを処理しようとして、何度もWorkerが壊れてしまう。
- 解決策
  何度か同じ処理をしようとしてもできなかった場合に、デッドレターキューに格納してくれるサービス

Amazon SImple Notification Service (Amazon SNS)

SNSは蓄積せずに、すぐにメッセージを送る。
（イメージは、メーリングリストにメールを送るような感じ）

- ユースケース

- システムの構築例



- Amazon SNSとAmazon SQSの比較

	Amazon SNS	Amazon SQS
メッセージの永続性	なし	あり
配信メカニズム	プッシュ (パッシブ)	ポーリング (アクティブ)
プロデューサー/コンシューマー	発行/サブスクライブ	送信/受信
分散モデル	一対多	一対一

Pub/Subモデルについてのおすすめ本
https://www.amazon.co.jp/入門-Python-3-Bill-Lubanovic/dp/4873117380

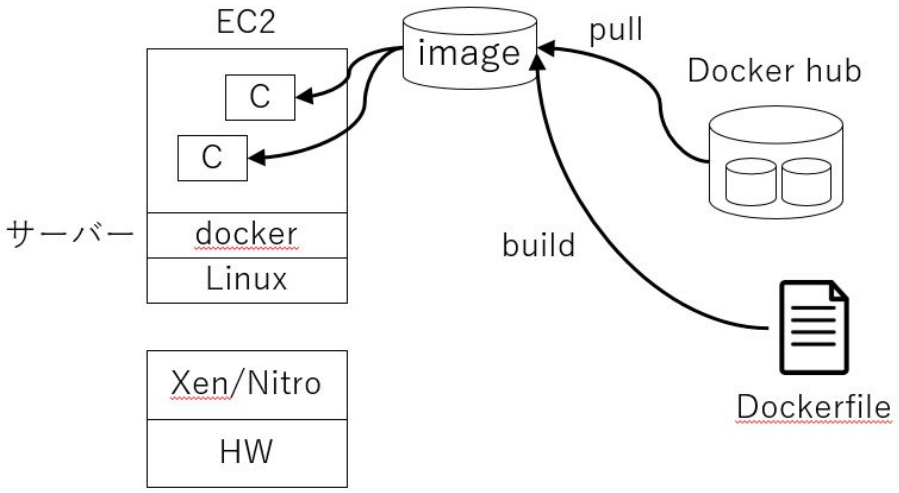
モジュール12 マイクロサービスとサーバレスアーキテクチャ

LINEのマイクロサービス
https://www.slideshare.net/linecorp/2017linespring

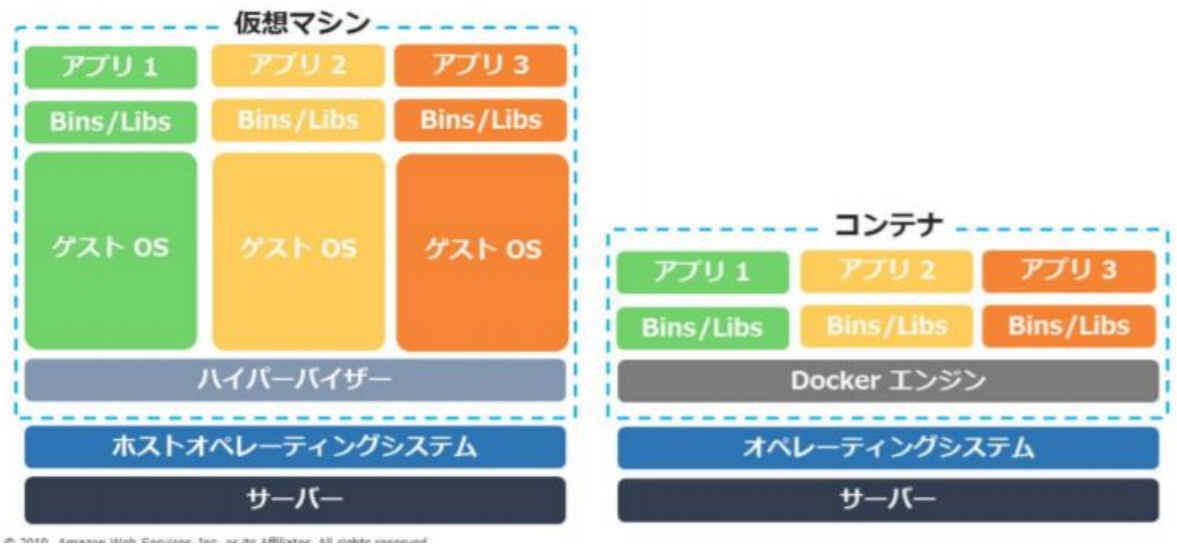
モノリシックからマイクロサービス化していく。（絞め殺しパターン）
https://codezine.jp/article/detail/11305

- 組織論
  - 開発チーム数は、6~8名くらいが良い。2pizza team
  - 人数は8名がちょうどよい。
    人数が多いとコミュニケーションが取りずらくなっていく。
    https://www.amazon.co.jp/HIGH-OUTPUT-MANAGEMENT-アンドリュー・S・グローブ-ebook/dp/B01MU055XH
  - コンウェイの法則
    組織の形から、ソフトウェアのアーキテクチャが決まっていく。
  - 逆コンウェイの法則
    ソフトウェアのアーキテクチャを決めてから、組織の形を決める。
  - プロダクションレディマイクロサービス
    本番対応力のあるマイクロサービスを構築する手法が紹介されている。
    https://www.oreilly.co.jp/books/9784873118154/

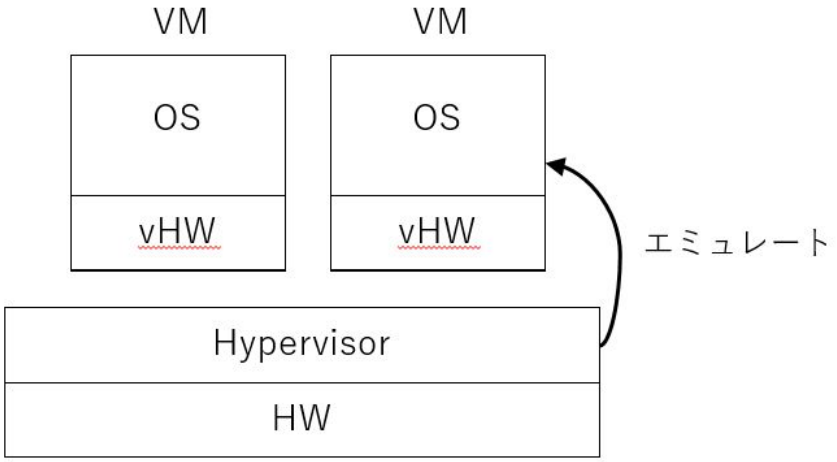
- コンテナについて
  solaris zoneやlxcなどもあるが、動作確認がしやすいという点からdockerが主流。



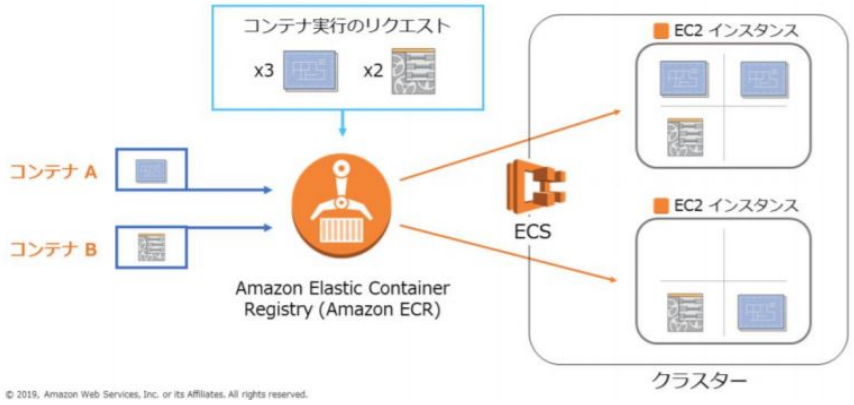
- コンテナと仮想マシンの比較
  コンテナはプロセス：プロセス間で干渉しないようにnamespaceやCapability、CGroup技術を使っている。
  https://www.slideshare.net/maruyama097/containername-space-isolation



仮想マシンは、vHWをエミュレートするから重たくなる。



- ユースケース
    - ECS(管理) + EC2(リソース)



    - ECS(管理) + Fargate(リソース)
      Fargateとは、、、
      コンテナを高速に利用するために使われる技術。
      https://aws.amazon.com/jp/blogs/news/firecracker-lightweight-virtualization-for-serverless-computing/

Amazon Lambda

Fargateの技術を利用して、イベントをトリガーとしてアプリケーションを動作させる。
Global IP（VPC内ではなく、非VPC）で動作させるのがおすすめ。
理由は、VPC内だと、他サービスと連携させるためのIPを作成する必要があり、1分くらいの時間がかかるため。

API Gateway

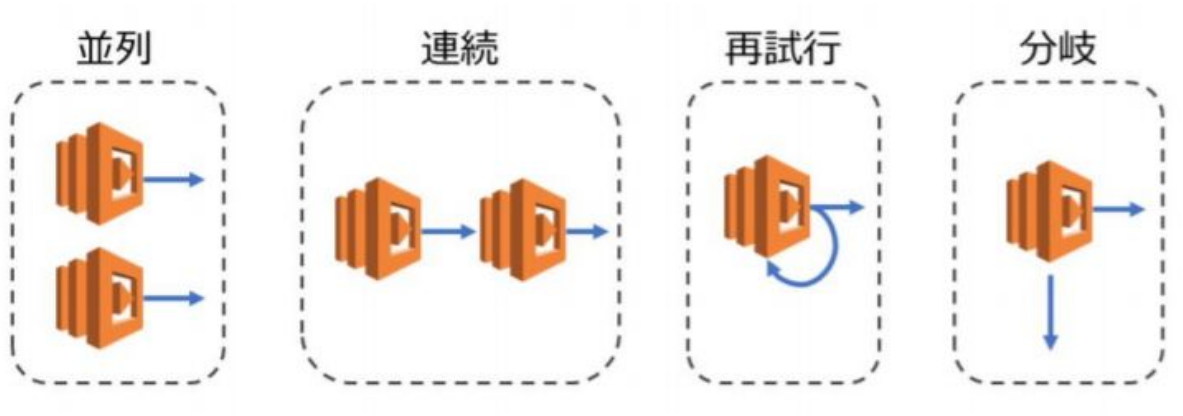
REST処理ができる。

- REST : HTTPのメソッド　→ CRUD処理
    - API Gateway
        - POST　　: C データ作成
        - GET　　: R データ取得
        - PATCH　　: U データ更新
        - DELETE　　: D データ削除
- GraphQL
    - AppSync

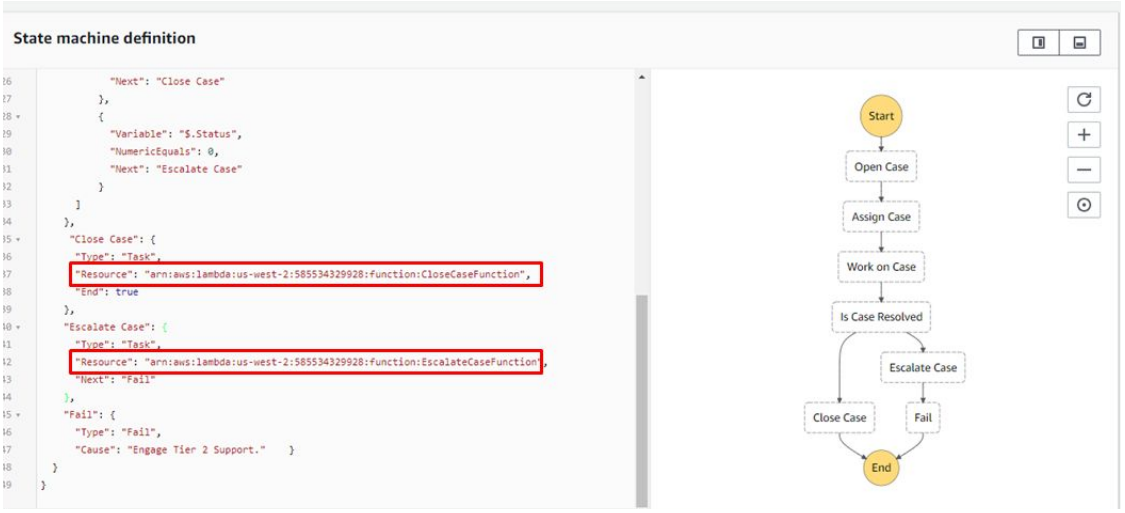
API Gatewayの実際の利用画面

AWS Step Functions

下記のように、AWS lambdaを使いたい場合に、AWS Step Functionsを利用する。

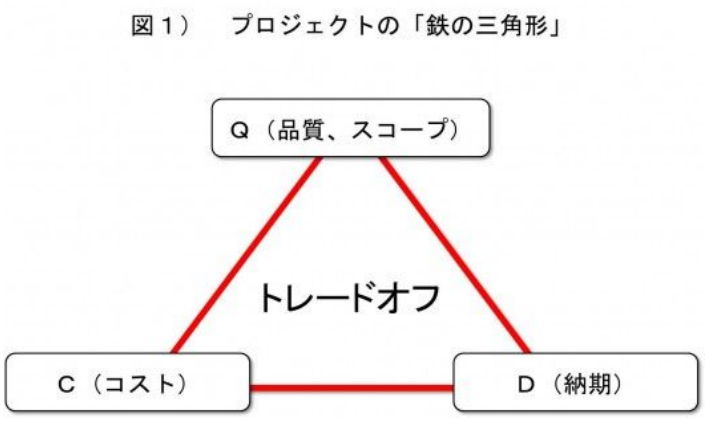


各ステップに対して、AWS lambdaのサービスを紐づける。



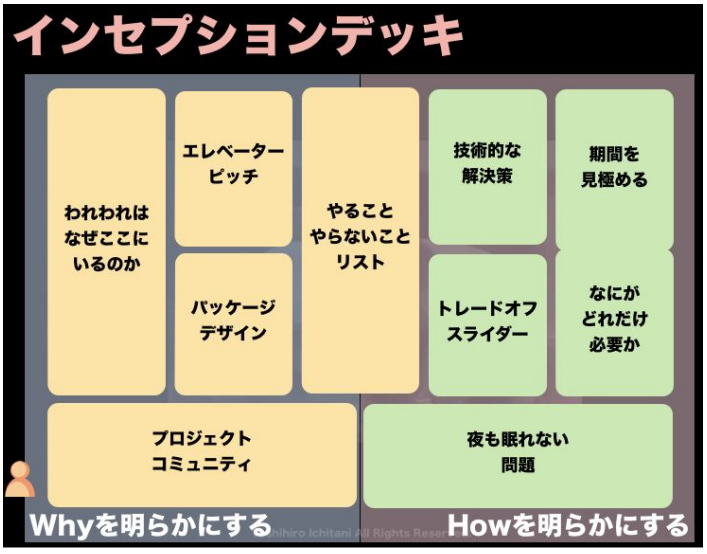
モジュール13 RTO/RPO および バックアップとリカバリの設定

鉄の三角形



インセプションデッキ

プロジェクトの全員が同じ方向を向き続けるために利用するためのもの。



インセプションデッキは、アジャイルサムライでも紹介されている。
https://www.amazon.co.jp/dp/B00J1XKB6K/ref=dp-kindle-redirect?_encoding=UTF8&btkr=1

アジャイル動物園 - 豚と鶏と他の動物たち
豚は肉(チームに貢献)を、鶏は卵を、カモメは、、、
話を聞かない人は、チームから追い出した方がよい。
https://www.ryuzee.com/contents/blog/4215

Googleのギークたちはいかにしてチームを作るのか
https://www.oreilly.co.jp/books/9784873116303/

災害対策

複数リージョンを選択しておきましょう。
災害時のことを考えたテストもしましょう。

カオスエンジニアリング

全世界に動画配信サービスを提供する Netflix が導入したことで注目されるようになった手法で、「本番稼働中のサービスにあえて擬似的な障害を起こすことで、実際の障害にもちゃんと耐えられるようにしよう」という取り組みのこと。

カオスモンキー

Amazonクラウド上のインスタンスをランダムに落としまくることで、サービスに対して仮想的な障害を引き起こしてくれます。 NetflixはこのChaos Monkeyを実環境で使うことで、本物の障害が起きたとしてもサービスが継続できることをテストし続けてきました。
SimianArmy

補足
AWS上に構築されたシステムのトラブルシュートコンテストの日のこと。

design for failure

design for failure：システムは壊れるものという前提で考える。
https://www.nttdata.com/jp/ja/data-insight/2012/090601/
FGOでのdesign for failureが欠けていたためにサービス停止してしまった例：
https://speakerdeck.com/isoparametric/grand-orderniokeru-deiraitowakusuliu-aws-dao-ru-and-huo-yong-shu

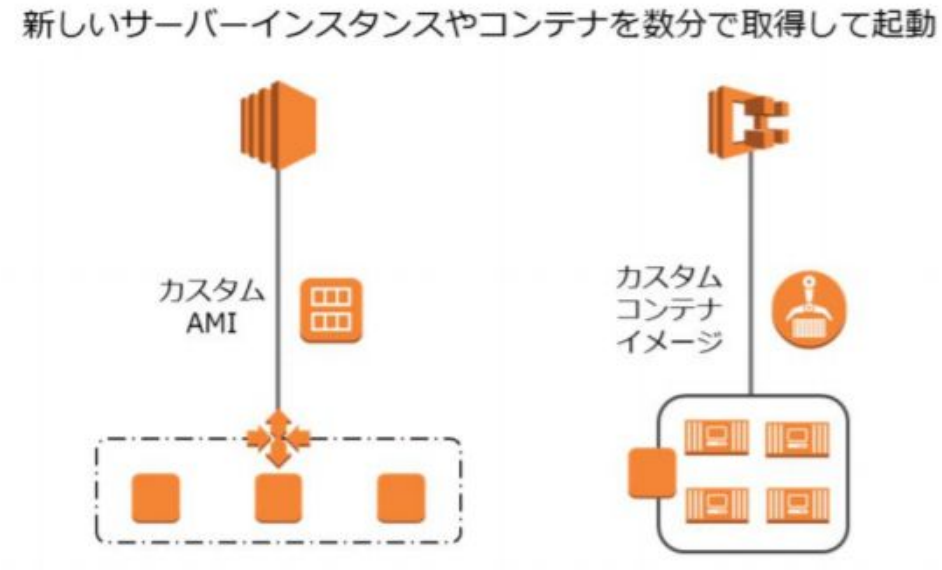
可用性の概念

- RPO(目標復旧時点)：Recovery Point Objective
  どれくらいの頻度でデータをバックアップすべきか？
- RTO(目標復旧時間)：Recovery Time Objective
  どれくらいの期間アプリケーションが利用できなくなるか？

ストレージの複製を作る

- Amazon S3：
  クロスリージョンレプリケーション
  非同期で複数リージョンでデータを取れる。
- Amazon Glacier：
  異なるリージョンのS3(耐久性：99.999999999%（**イレブンナイン**）)でデータのバックアップが取れる。
- Amazon EBS：
  S3にスナップショットのコピーが取れる。
- AWS Snowball
  異なるリージョンのS3にデータのバックアップが取れる。
- Amazon EFS
  異なるリージョンのファイルシステムにバックアップを取れる。

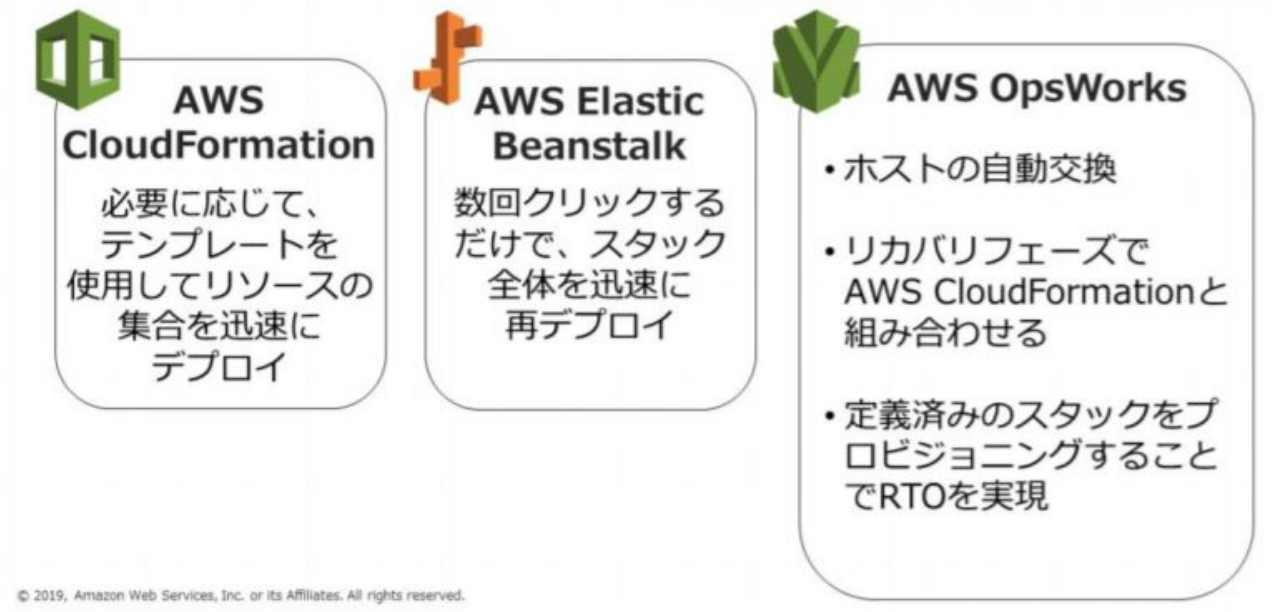
コンピューティングの可用性について



ネットワークの可用性について

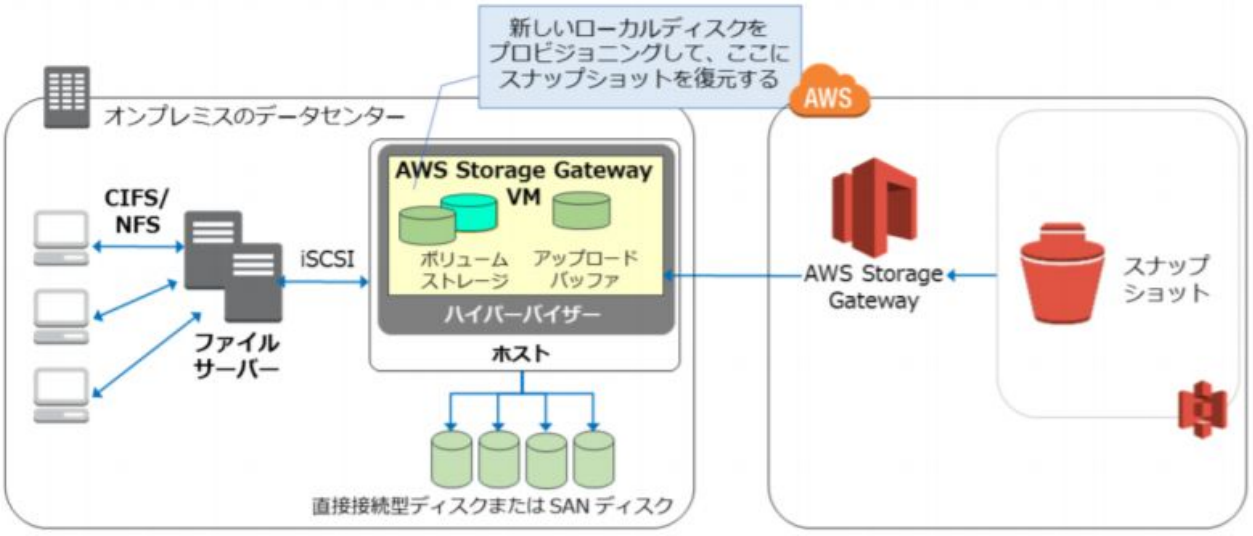
データベースの可用性について

オートメーションを使用してリカバリを迅速に

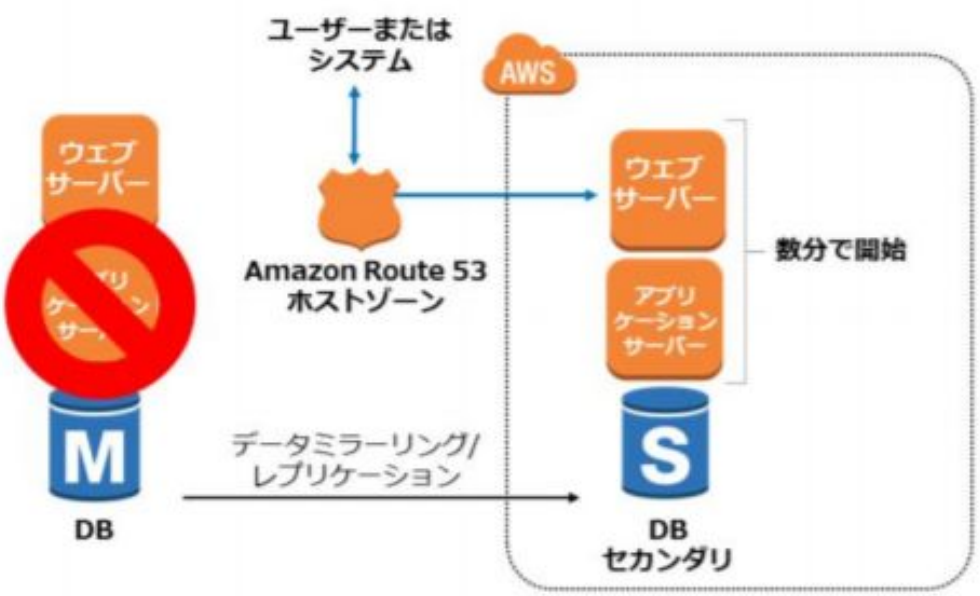


バックアップの復元例

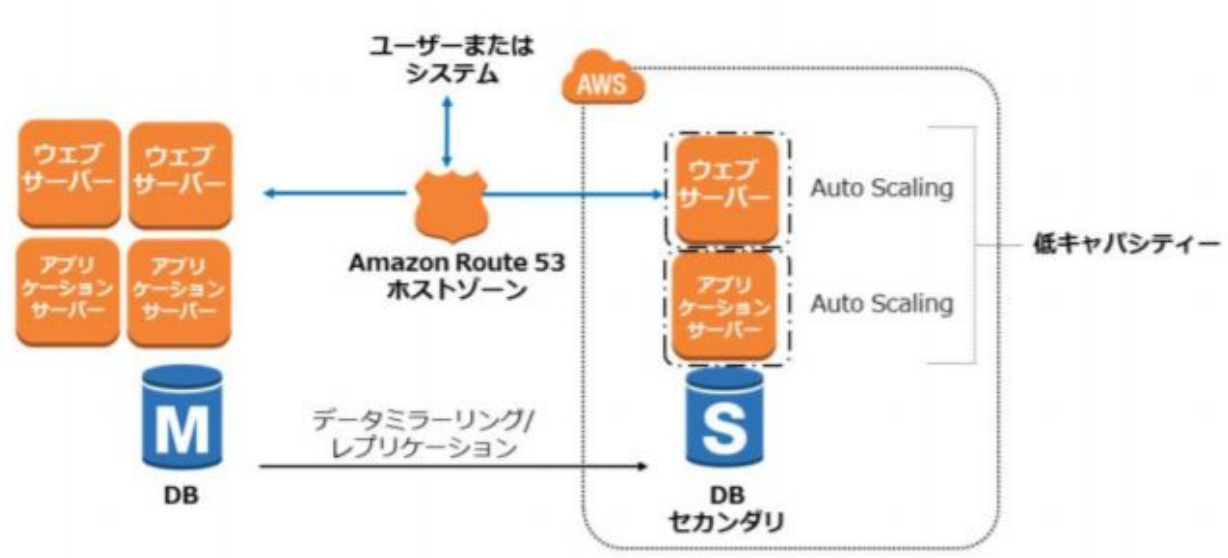
- オンプレにあるデータのリカバリ



- パイロットライトの例



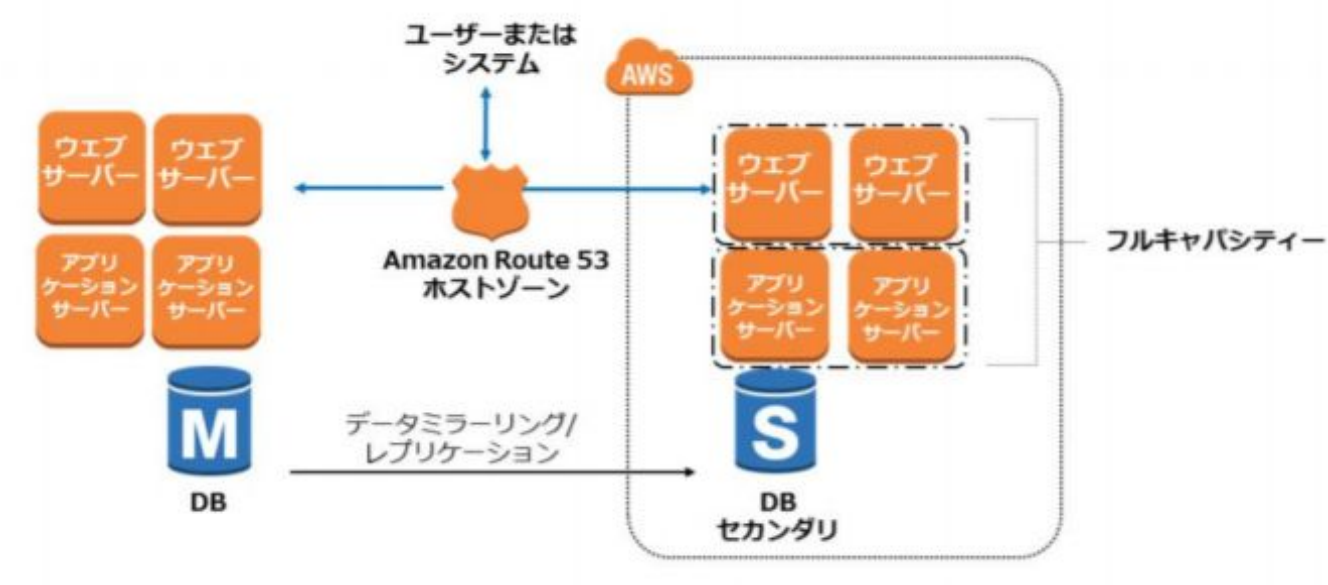
- 完全に機能する低キャパシティのスタンバイ



  - 書き込み処理がある場合
    低キャパシティは利用しない方がよい。
    ほかのリージョンにあるDB(Master)にいくため、処理が遅くなるため。
  - 書き込み処理がない（読み込み処理のみ）の場合
    低キャパシティは利用しても良い。

マルチサイト　アクティブーアクティブ

フルキャパシティ側からの書き込み処理は遅くなる。



まとめ
コストとRPO/RTOのトレードオフ

さいごに

オンデマンドインスタンスで試してから、リザーブドインスタンスにしましょう。

1. どれくらいのインスタンススペックが必要なのかを決める。
2. Trusted Advisorを利用して、どれくらいの容量が良いかを診断する。
   https://aws.amazon.com/jp/premiumsupport/trustedadvisor/