

Architecting on AWS 補足資料

モジュール:

クラスルームトレーニング コース

- 経験豊富な講師が提供する教室またはオンラインでのトレーニング

プロフェッショナル 上級コース

コース日数：3日

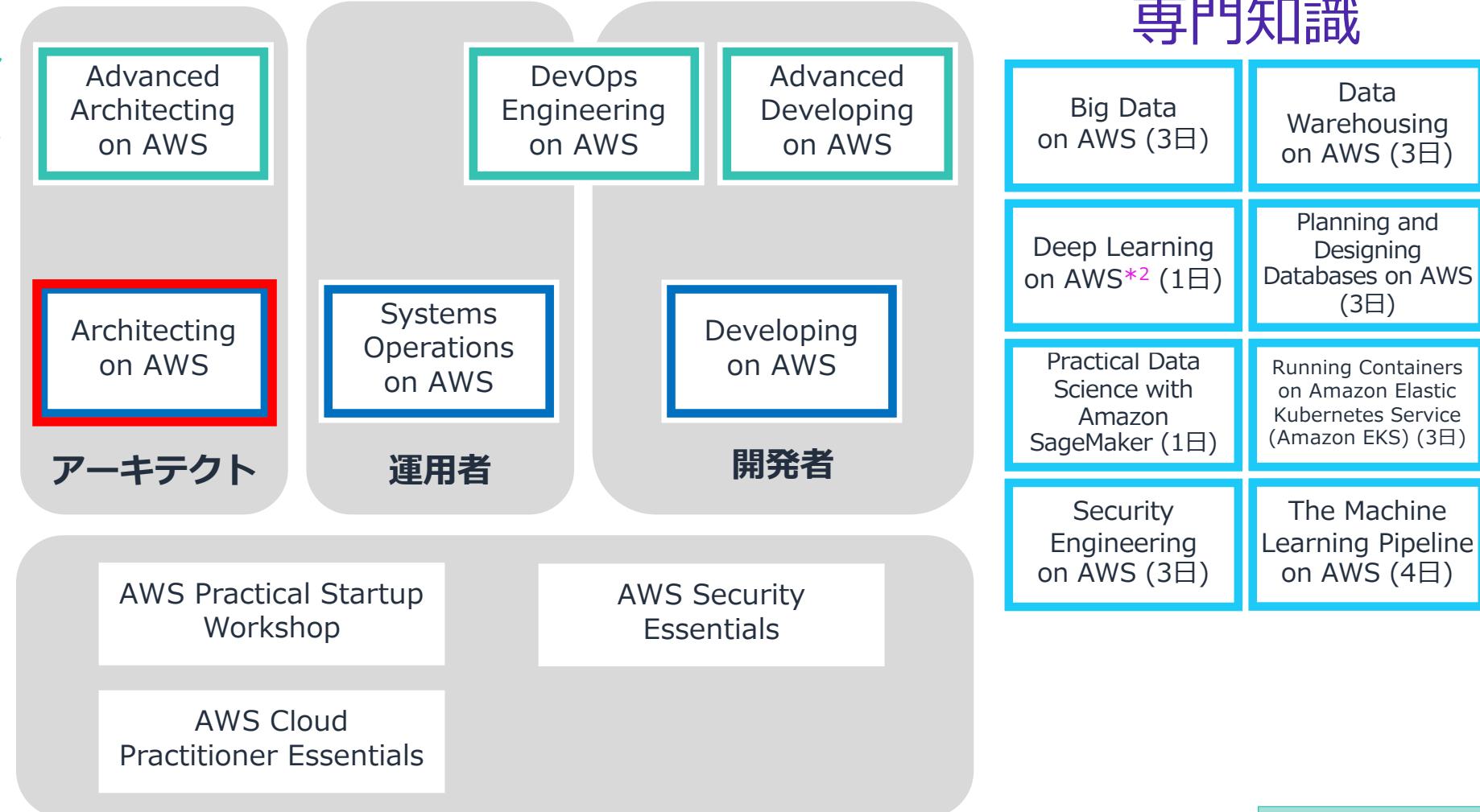
アソシエイト 中級コース

コース日数：3日

ベーシック 初級コース

コース日数：1日

専門知識



AWS 認定

- 技術スキルとクラウドの専門知識を検証し、キャリアとビジネスを成長

プロフェッショナル

2年間の AWS クラウドを使用したソリューションの設計、運用、およびトラブルシューティングに関する包括的な経験

アソシエイト

1年間の AWS クラウドを使用した問題解決と解決策の実施における経験

基礎

6ヶ月間の基礎的な AWS クラウドと業界知識



専門知識

専門知識分野において、AWS クラウドの技術的な経験



前提条件

- AWS クラウドプラクティショナーの基礎知識
- 分散システムの実務的知識
- 一般的なネットワーキングの概念の知識
- 多層アーキテクチャの実務的知識
- クラウドコンピューティングの概念の知識

お役立ち資料

- AWS クラウドサービス活用資料集 | サービス別資料
 - <https://aws.amazon.com/jp/aws-jp-introduction/aws-jp-webinar-service-cut/>
- AWS オンラインセミナースケジュール | AWS
 - <https://aws.amazon.com/jp/about-aws/events/webinars/>

1日目

- ・モジュール1: アーキテクチャ設計の基礎
 - ・ラボ1: AWS API を使用して EC2 インスタンスをデプロイする方法を確認する
- ・モジュール2: アカウントのセキュリティ
- ・モジュール4: コンピューティング
- ・モジュール3: ネットワーク 1

2日目

- ラボ2: Amazon VPC インフラストラクチャを構築する
- モジュール5: ストレージ
- モジュール6: データベースサービス
 - ラボ3: Amazon VPC インフラストラクチャでデータベースレイヤーを作成する
- モジュール7: モニタリングとスケーリング
 - ラボ4: Amazon VPC で高可用性を構成する

3日目

- モジュール8: オートメーション
- モジュール9: コンテナ
- モジュール10: ネットワーク 2
- モジュール11: サーバーレス
 - ラボ5: サーバーレスアーキテクチャを構築する
- モジュール12: エッジサービス
 - ラボ6: Amazon S3 オリジンで CloudFront ディストリビューションを設定する
- グループディスカッション
- モジュール13: バックアップと復旧

モジュール 1

アーキテクチャ設計の基礎

モジュールの目標

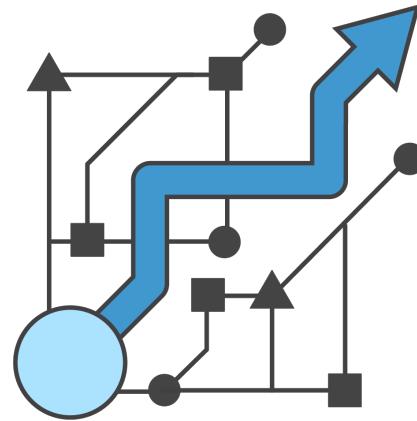
- AWS とはなにか？ビジネスにおいてどのようなメリットがあるのか？
- クラウドリソースへのアクセスを制御するにはどうすればよいか
 - → AWS の操作はどのように行うのか？
 - → 詳細は、モジュール2 で取り扱います
- AWS グローバルインフラストラクチャはどのような構造になっているか
- ベストプラクティスに従ってクラウドインフラストラクチャを構築していることをどのように確認できるか
 - → Well Architected Framework

AWS はどのように誕生したのか？

- ビジネス上の課題から始まった
 - 2000年頃にAmazon のEコマースプラットフォーム上で他社がショッピングサイトを構築できるような仕組みを構築しようとしていた
 - 1994年創業のAmazon は多くのスタートアップと同様、十分な計画無しでシステムを構築しており、ごちゃごちゃしたシステムなつていて他社に公開できるプラットフォームを作ることが困難であった
 - そこで社内すべてのチームがシステムを疎結合化しAPI でアクセスする方式に移行した(今日で言うところのマイクロサービス化)
- 次の課題は、インフラの構築
 - 各チームがシステムのインフラ構築に3ヶ月もかかっていた
 - Amazon の各チームが車輪の再発明をすること無く共通利用できるインフラストラクチャ・サービスを作成
- 2006 年販売開始
 - 2003 年頃にこのインフラの仕組みを外部に提供し新たなビジネスにできるのではないかというアイディアが生まれた
 - 2006 年にAWS リリース

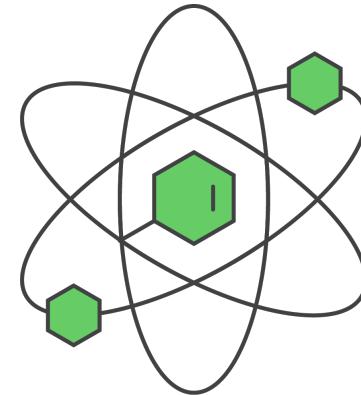
<https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/>

クラウドとは何か。AWS とは何か



オンデマンドで
ITリソースを
利用可能
(API による操作)

クラウドの特徴



ダイナミックに
拡張可能

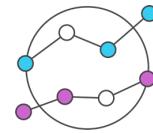


従量課金制

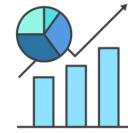
<https://aws.amazon.com/what-is-cloud-computing>

クラウドには他にどのような利点があるか？

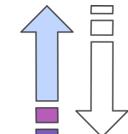
クラウドコンピューティングの 6 つの利点



固定の償却コストが変動コストに



スケールによる大きなコストメリット



キャパシティーの予測が不要に



速度と俊敏性の向上

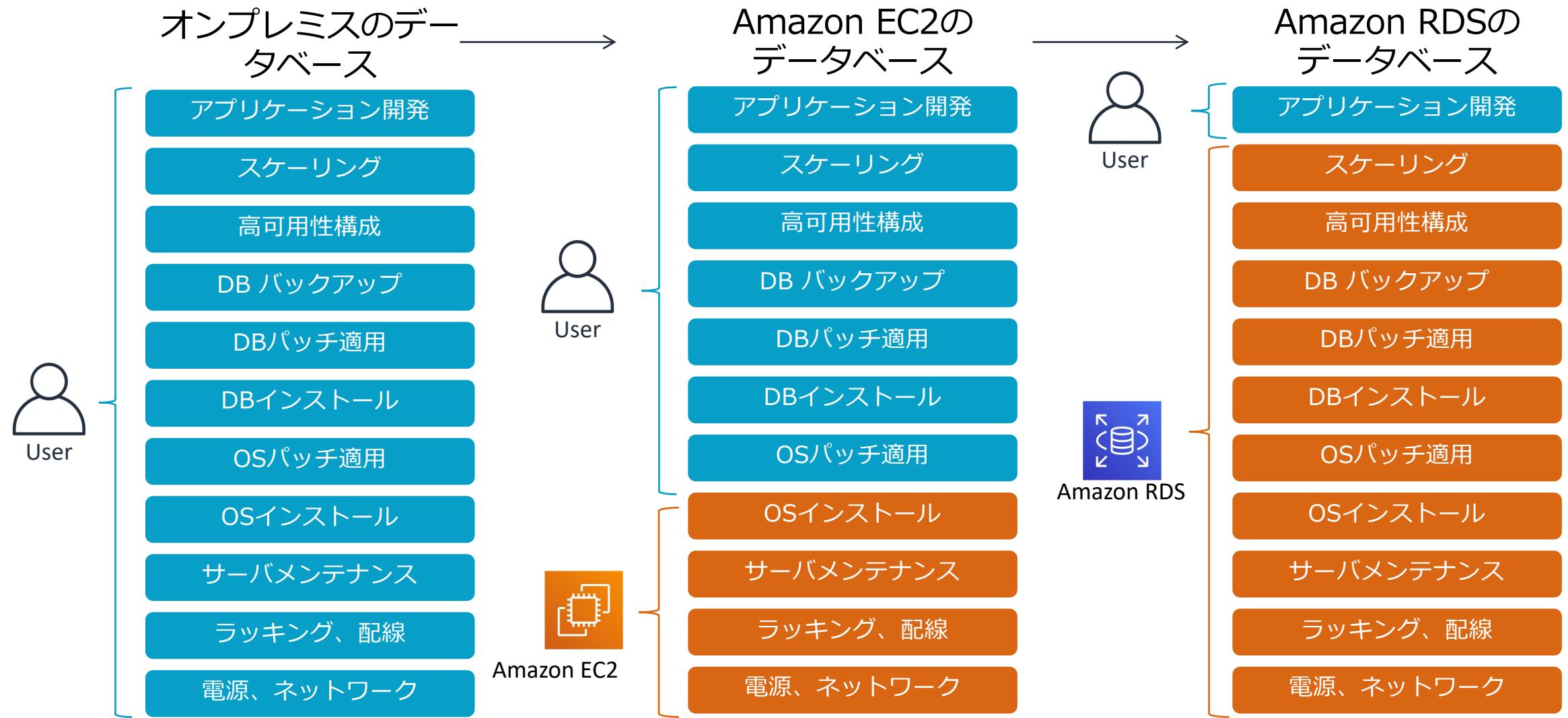


データセンターの運用保守投資が不要



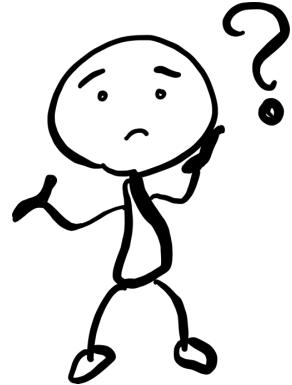
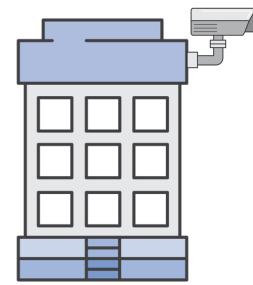
わずか数分で世界中にデプロイ

マネージドサービス



AWS データセンター

データセンター



データセンターが障害などで止まってしまったら、AWS は使えなくなってしまうのかな？

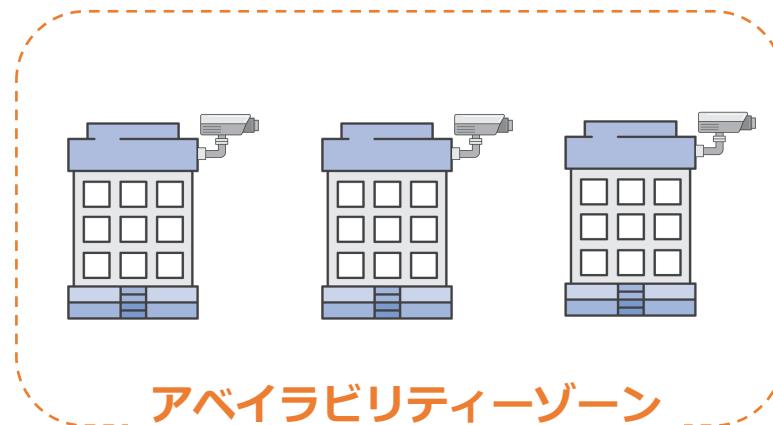
(参考)

2018 Data Center Industry Survey Results

- 過去1年でデータセンター障害の経験あり
 - 回答者の約3分の1 (**31%**)

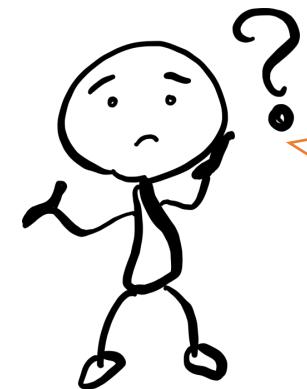
<https://uptimeinstitute.com/2018-data-center-industry-survey-results>

アベイラビリティゾーン(AZ)



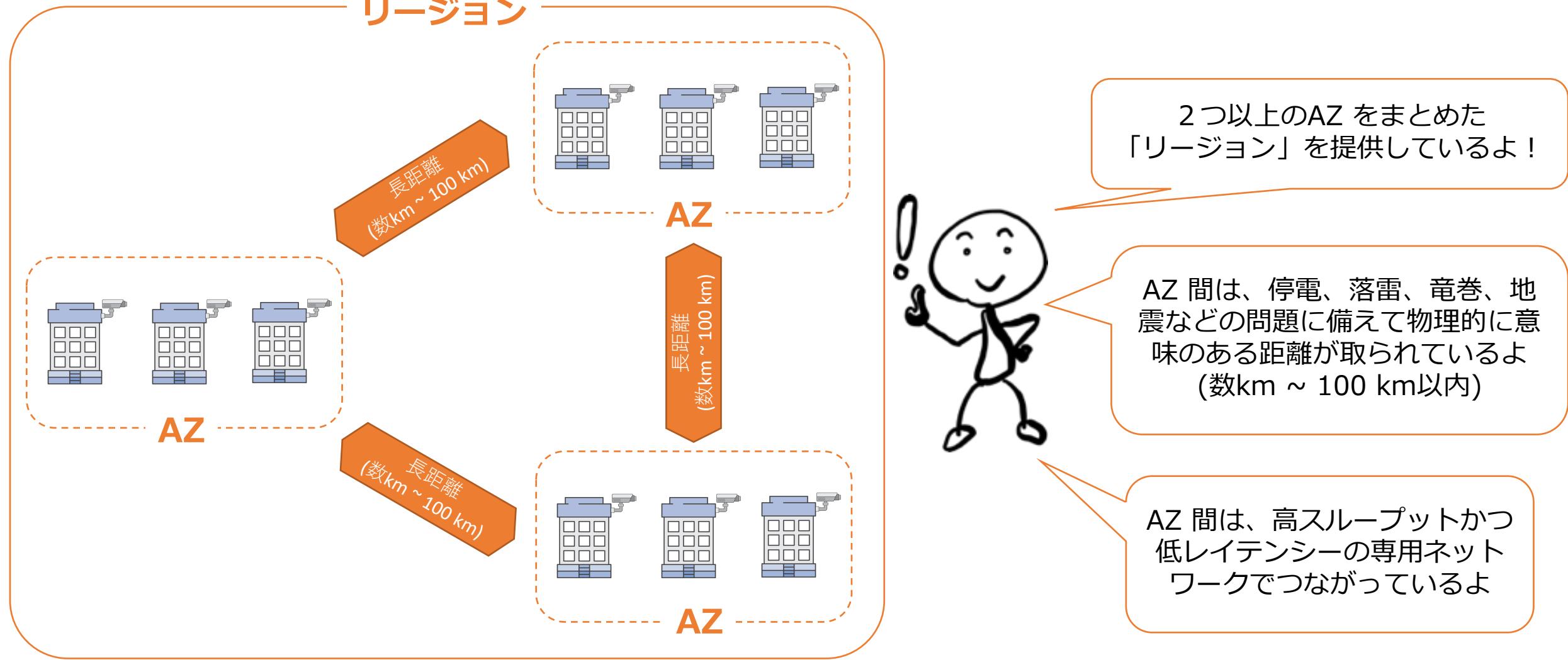
複数のデータセンターをまとめた
「アベイラビリティゾーン」が用
意されているよ

「アベイラビリティゾーン」は略
して「AZ」というよ

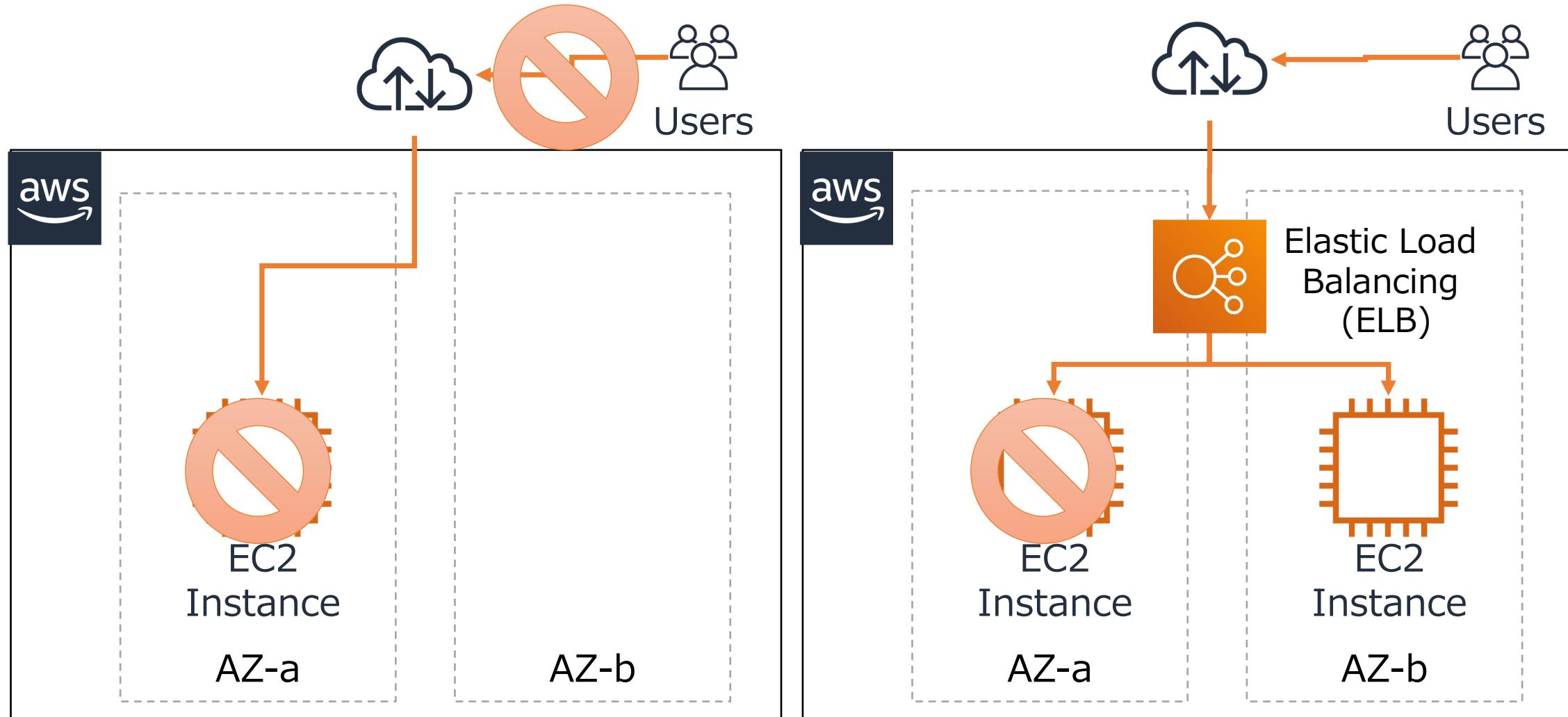


でも地震などの災害や大規模
な停電などで、AZ 内の複数の
データセンターが同時に止まっ
てしまったら、やっぱりAWS
は利用できなくなってしまう
の？

リージョン



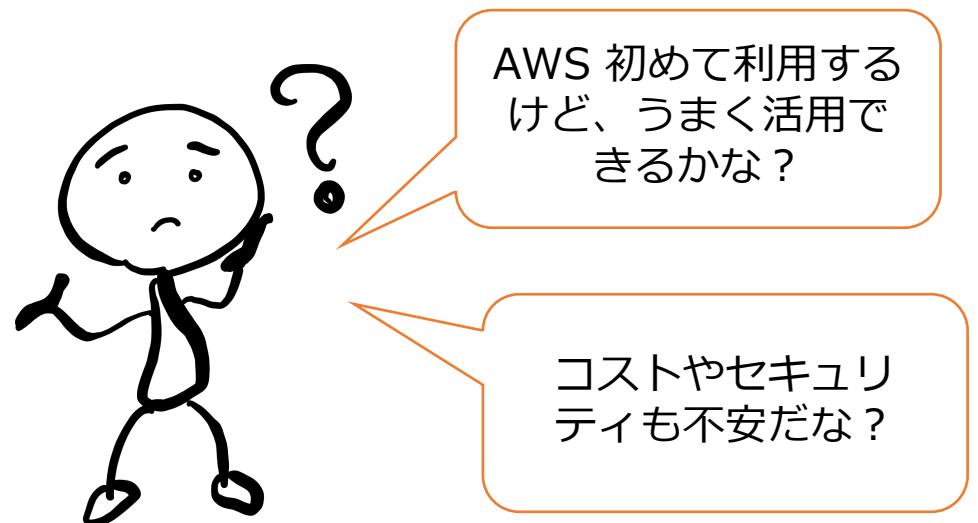
推奨設計: マルチAZ構成



AWS Well-Architected Framework(W-A) とは

Lean Horus

- AWS 上でのシステム設計・運用の原則とベストプラクティスをまとめたもの
- AWSのソリューションアーキテクト(SA)とお客様が 長年にわたり数多くの経験から作り上げたもの



AWS W-A

15 年以上 の経験から、数多くの
お客様 とAWSのSAが得た ベス
トプラクティスがすぐに入手出
来る

質問と回答形式でのベストプラクティス例

例：セキュリティの質問抜粋

SEC 3. 人とマシンのアクセス許可はどのように管理すればよいでしょうか？

- アクセス要件を定義する
- 最小限の権限をアクセスを付与する
- 緊急アクセスのプロセスを確立する
- アクセス許可を継続的に削減する
- 組織のアクセス許可ガードレールを定義する
- ライフサイクルに基づいてアクセスを管理する
- パブリックおよびクロスアカウントアクセスの分析

質問と回答形式でのベストプラクティス例

例：セキュリティの質問抜粋

SEC 3. 人とマシンのアクセス許可はどのように管理すればよいでしょうか？



全項目ベストプラクティスに
則っていないとダメなのか？

ベストプラクティスを理解した上で
「(ビジネス的な)判断をする」ことが重要
→ リスクや改善点の“顕在化”



モジュール 2

アカウントのセキュリティ

モジュールの目標

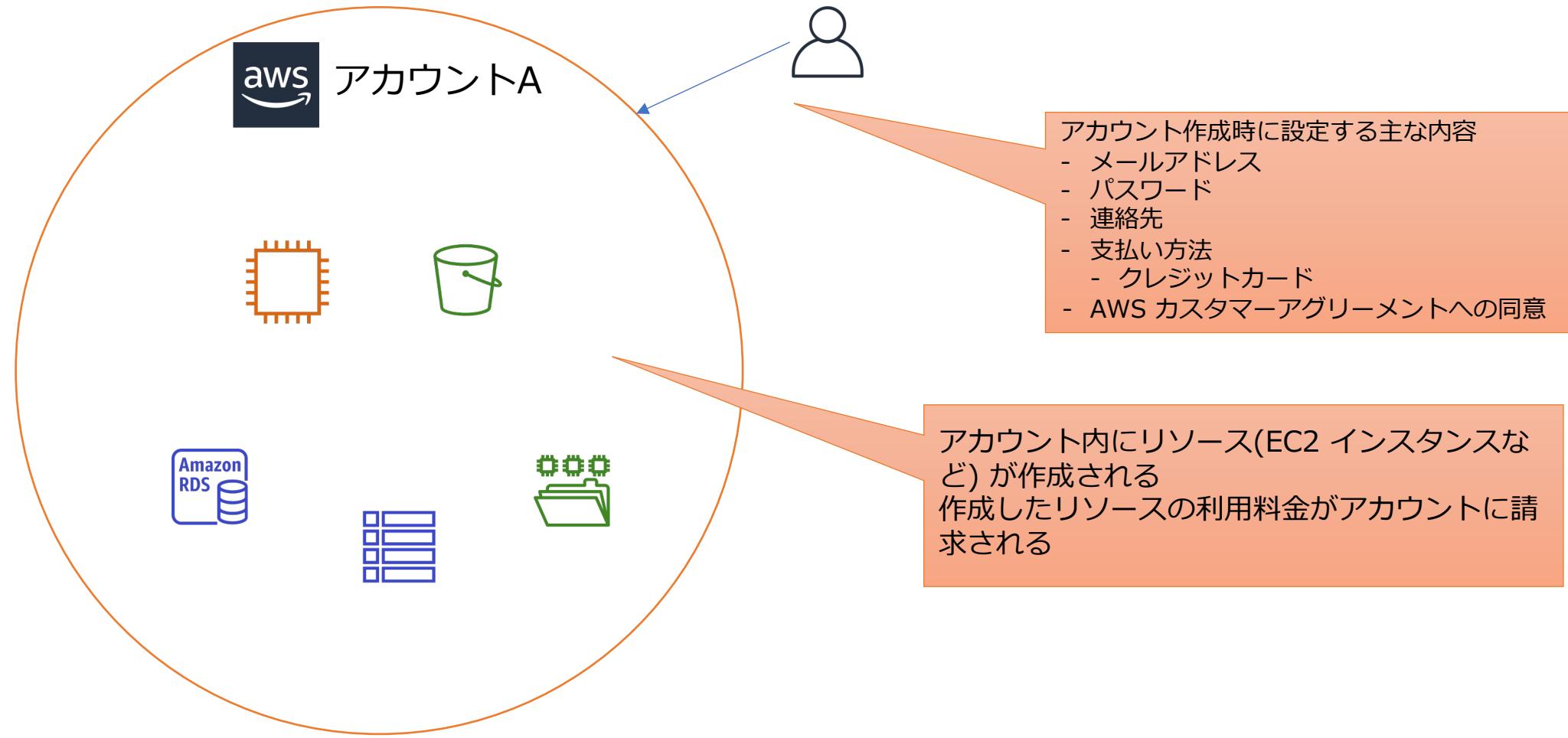
- AWS のアカウントとリソースへのアクセスを管理するベストプラクティスは何か
- 必要なリソースのみへのアクセス権をユーザーに付与できるか
- リソースへの一時的なアクセスを許可するにはどうすればよいか
- 複数のアカウントを管理する最適な方法は何か

責任共有モデル

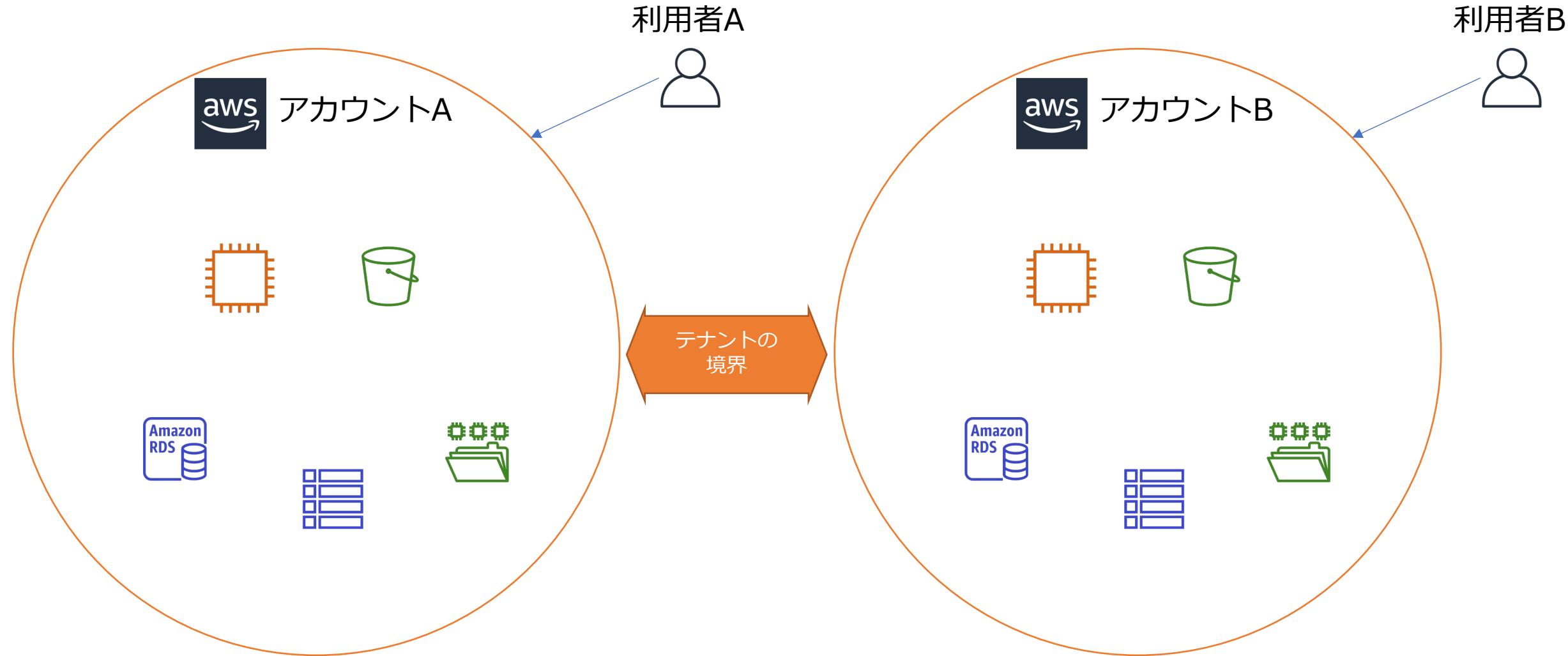
お客様	お客様のデータ		
	プラットフォーム、アプリケーション、Identity and Access Management		
	オペレーティングシステム、ネットワーク、ファイアウォールの設定		
	クライアント側のデータ暗号化	サーバー側の暗号化	ネットワークトラフィックの保護

AWS	ソフトウェア			
	コンピューティング	ストレージ	データベース	ネットワーク
	ハードウェア/AWS のグローバルインフラストラクチャ			
	リージョン	アベイラビリティーゾーン	エッジロケーション	

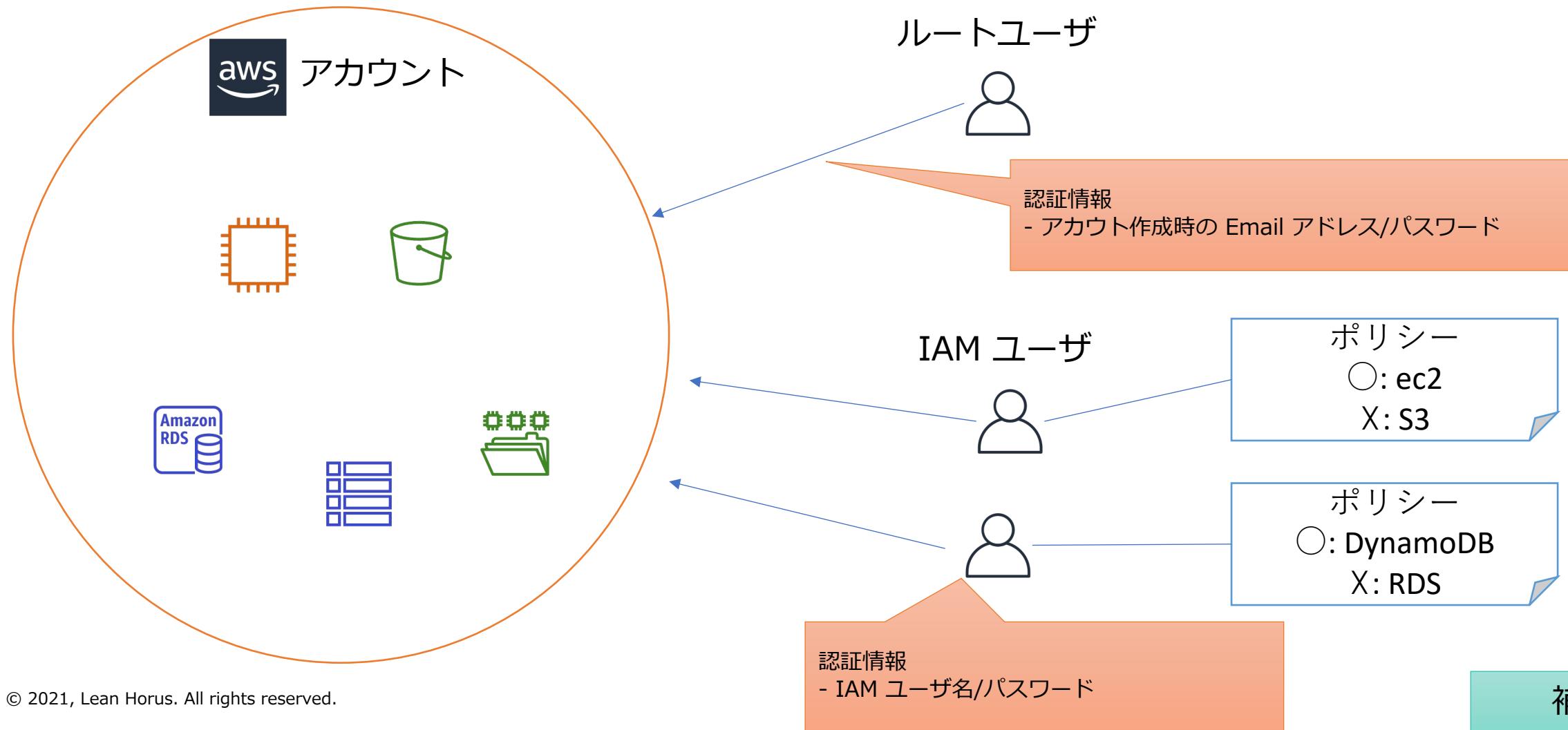
AWS アカウントとは？



AWS アカウントとは？



IAM ユーザ、ポリシー



ポリシー例

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Stmt1453690971587",  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject",  
                "s3:GetObject"  
            ],  
            "Resource": "arn:aws:s3:::examplebucket/*",  
            "Condition": {  
                "IpAddress": {  
                    "aws:SourceIp": "54.64.34.65/32"  
                }  
            }  
        }  
    ]  
}
```

IAM ポリシーにより以下を設定

- どのリソースに対して
- どのような操作を
- 許可する or 拒否する
- (Option) 追加条件



Effect

- Allow or Deny

Action (or NotAction)

- 権限の及ぶ操作

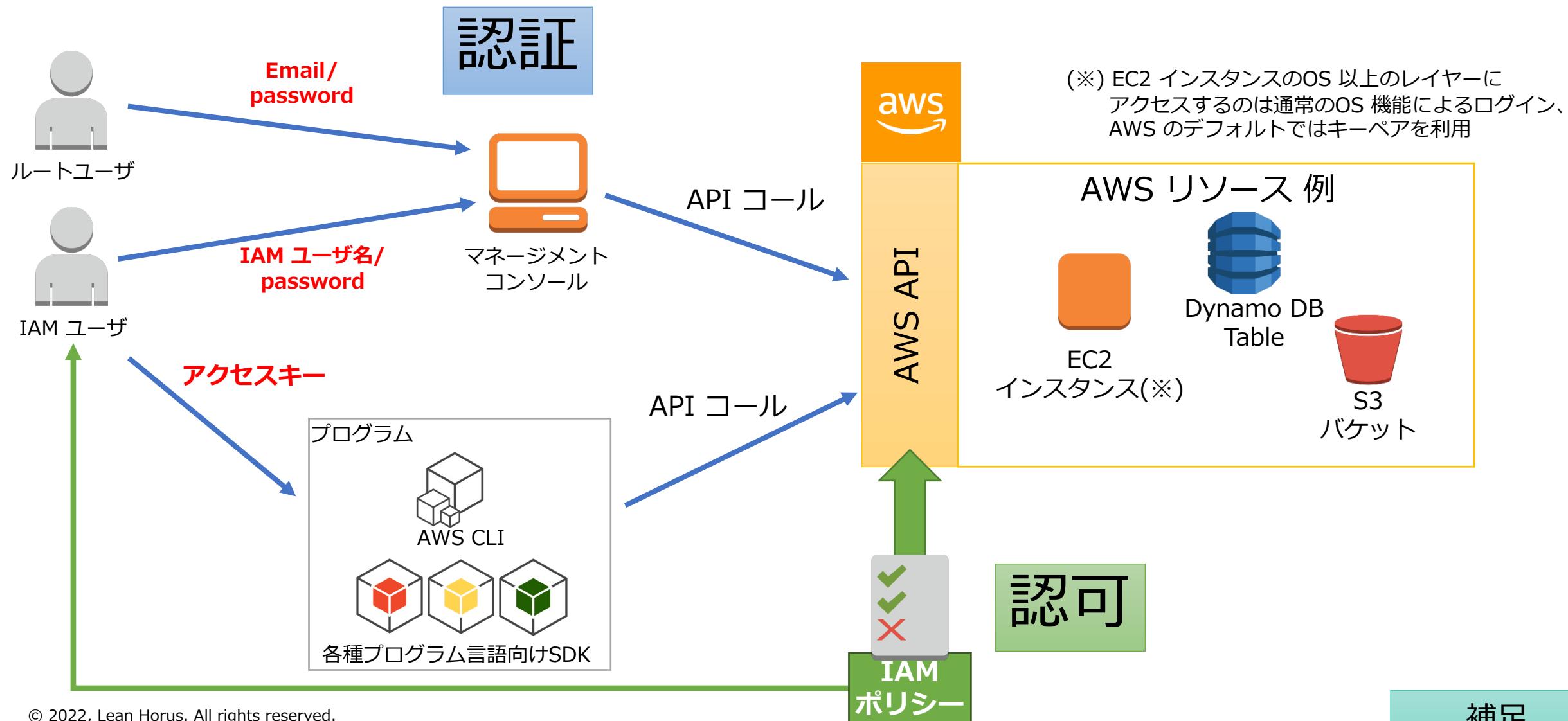
Resource (or NotResource)

- 権限の及ぶリソース範囲

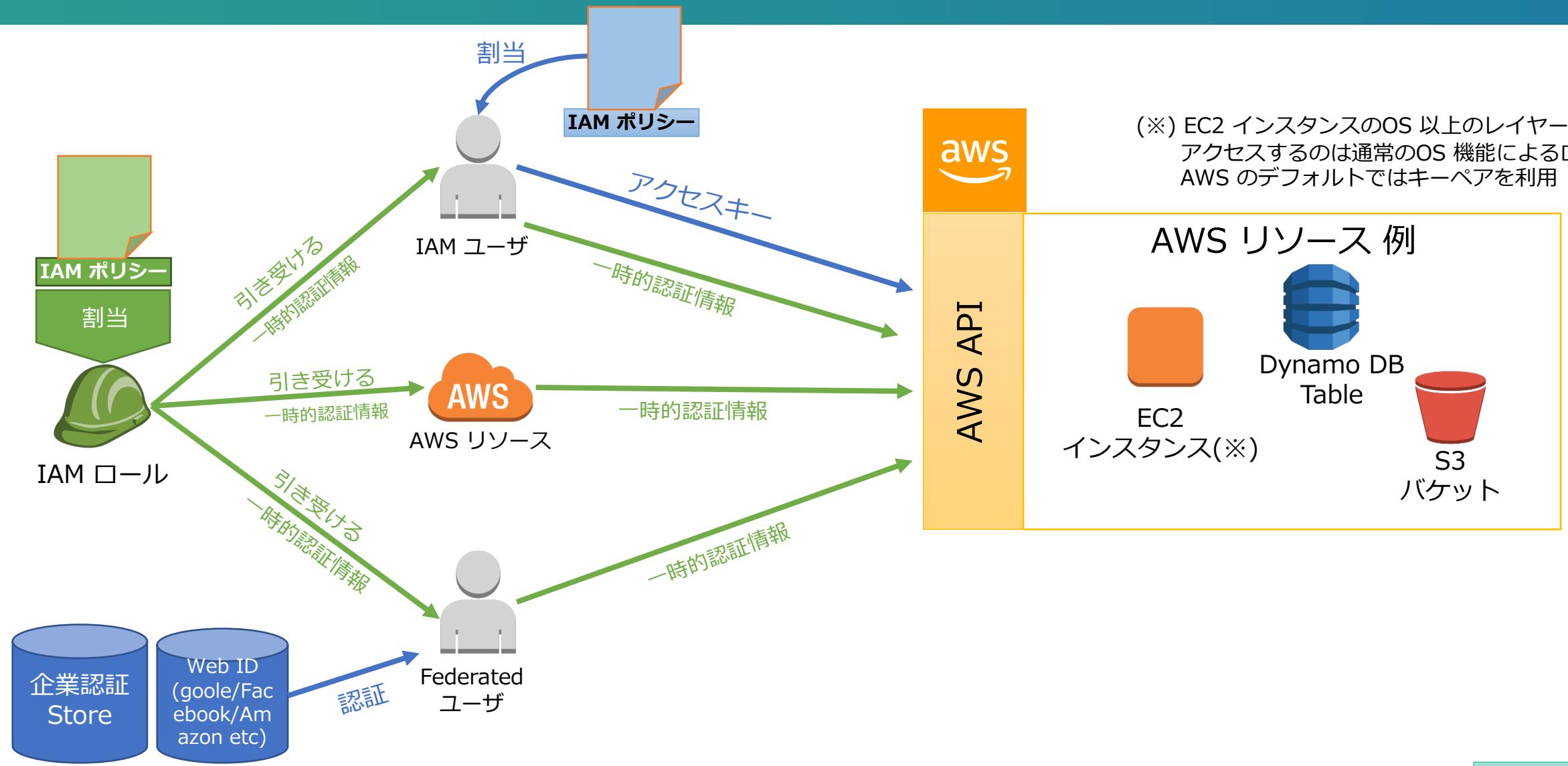
Condition (Option)

- 追加条件を記述可能
- 特定のIP アドレスからのリクエストのみを受け付けるなど

AWS IAM 認証 & 認可

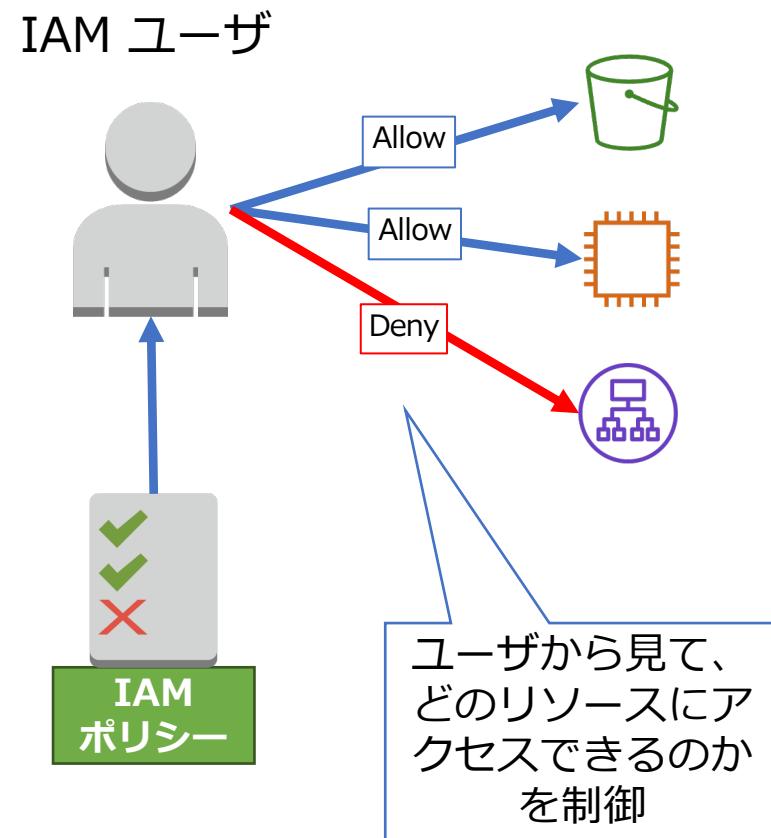


IAM ロールのユースケース

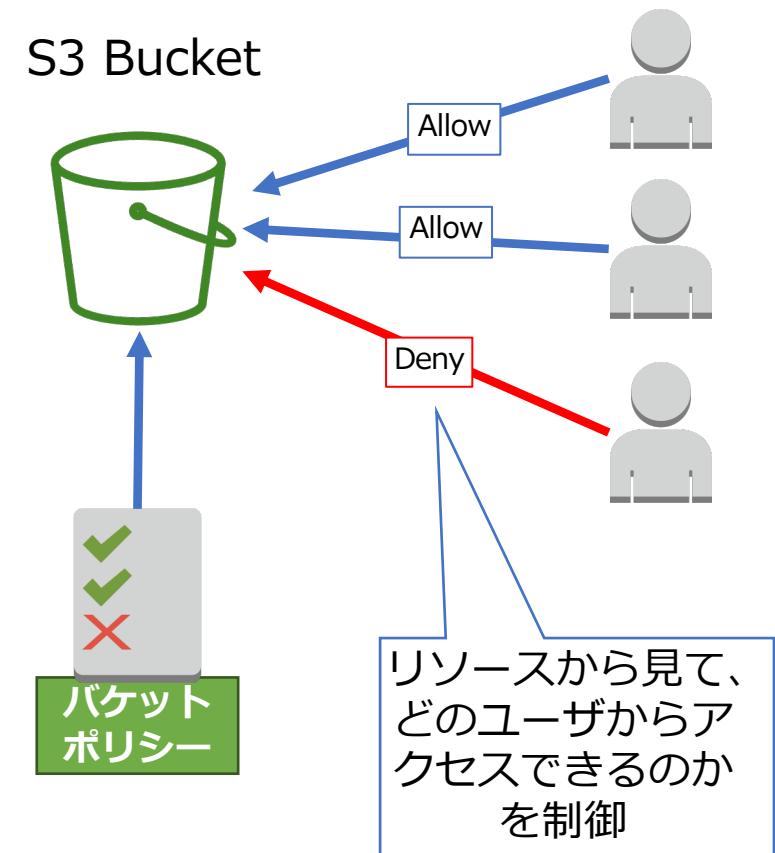


ポリシー：アイデンティティ vs リソース

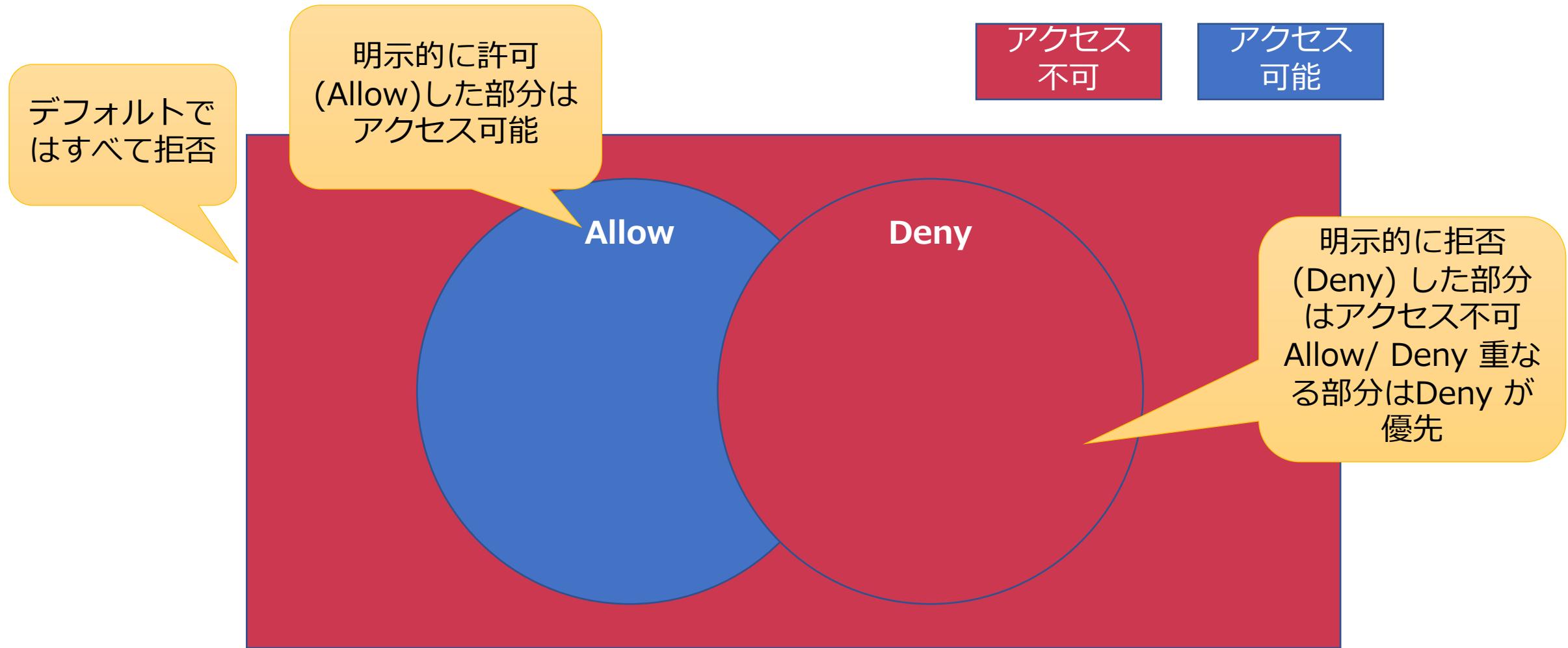
アイデンティティベース



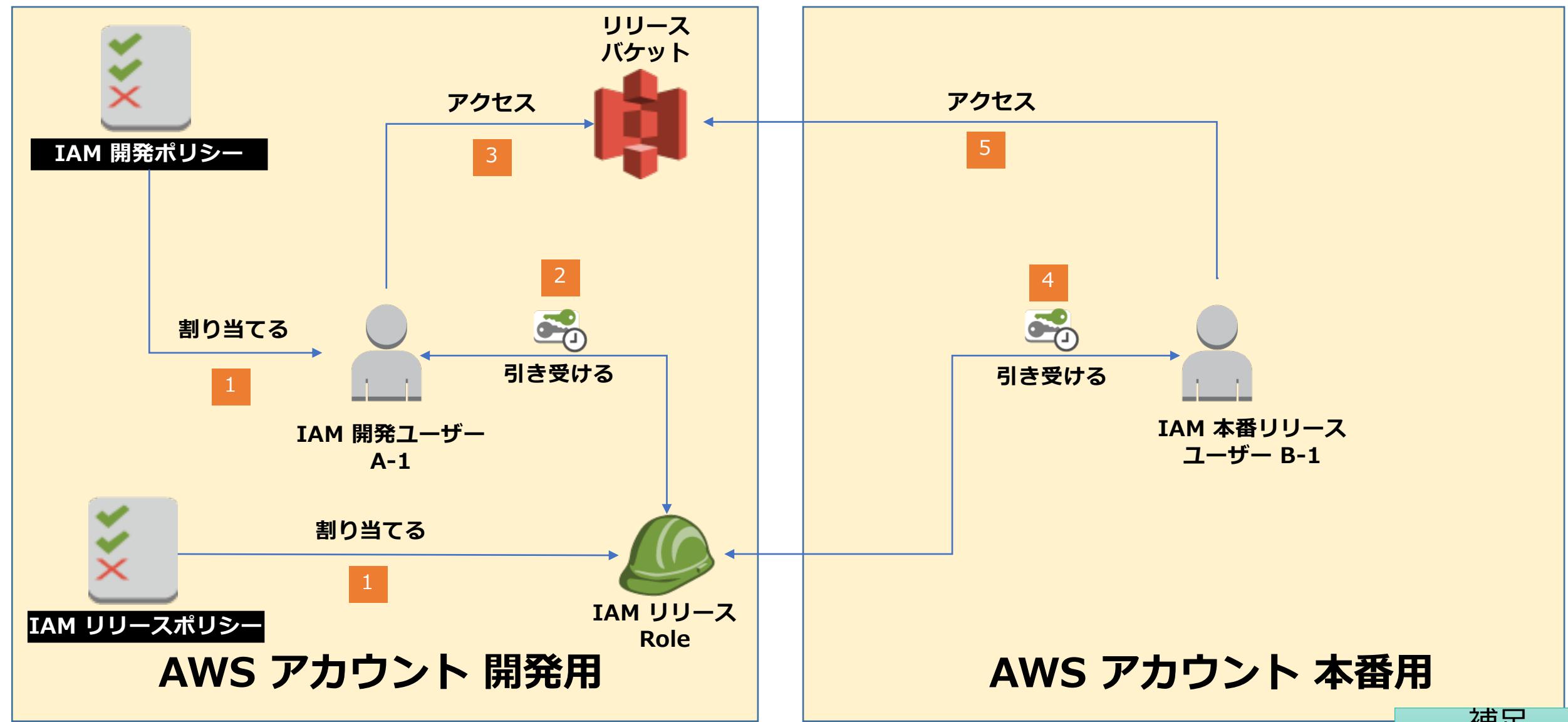
リソースベース



IAM ポリシーの適用ルール

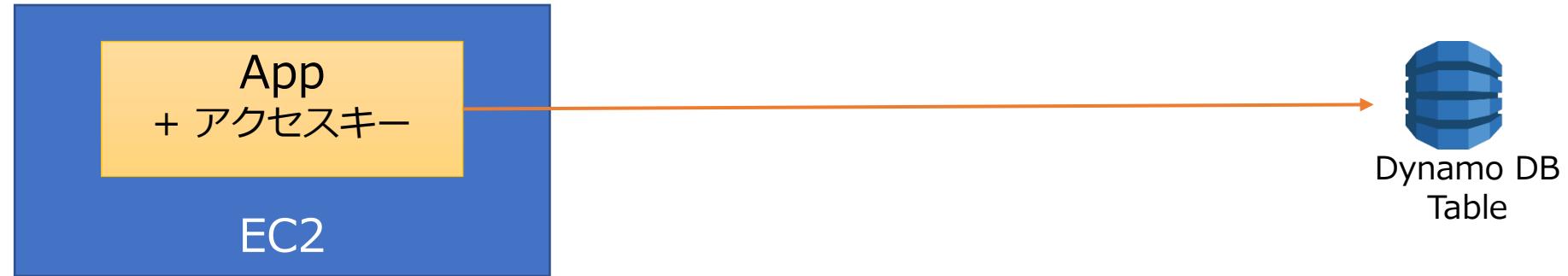


AWS IAM ロール - ロールを引き受ける

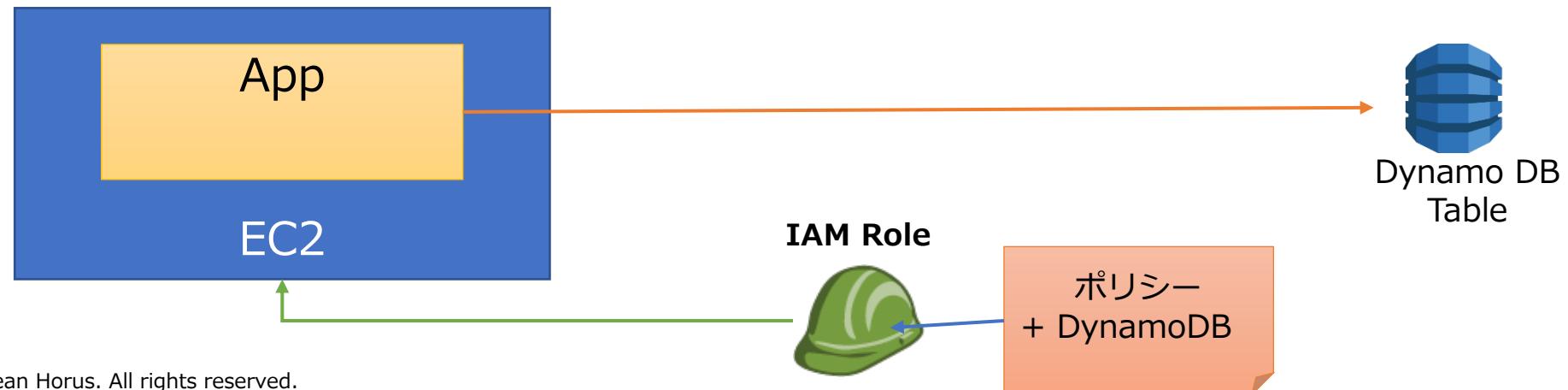


EC2 で IAM ロールを利用する

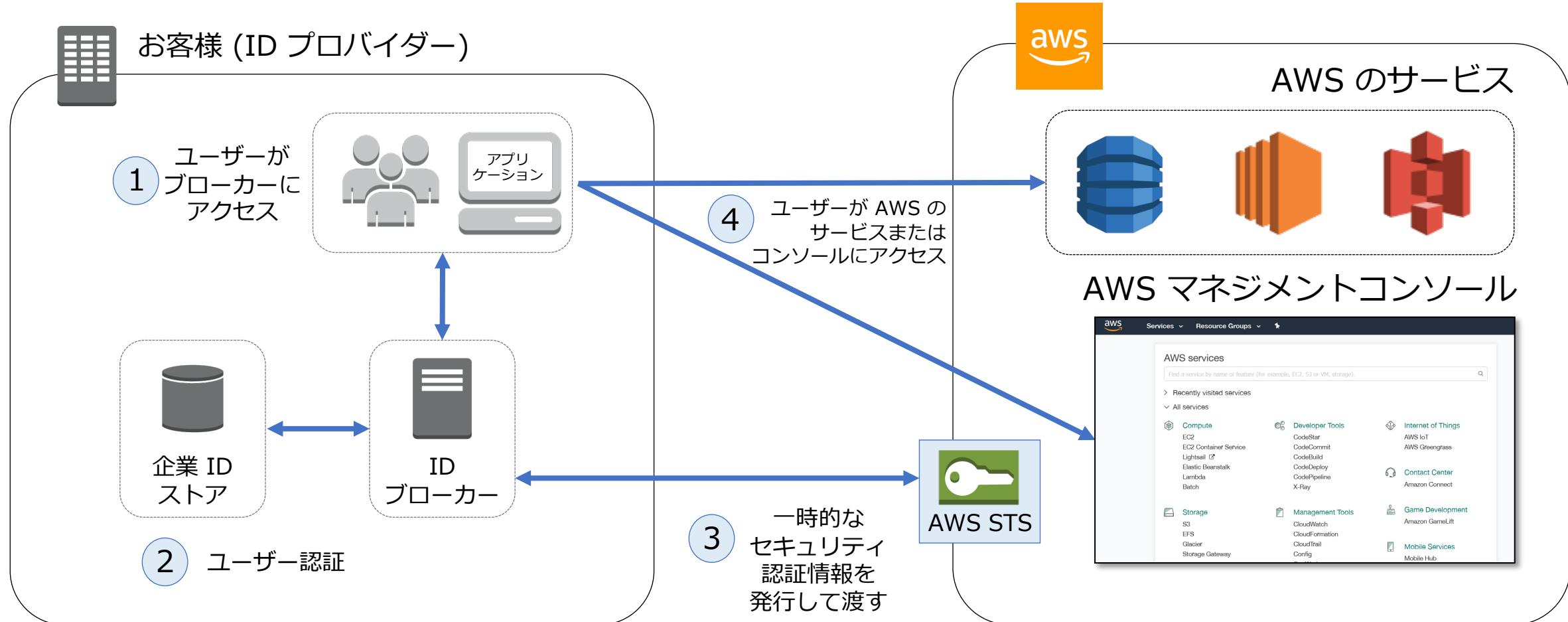
アクセスキーを利用するパターン (非推奨)



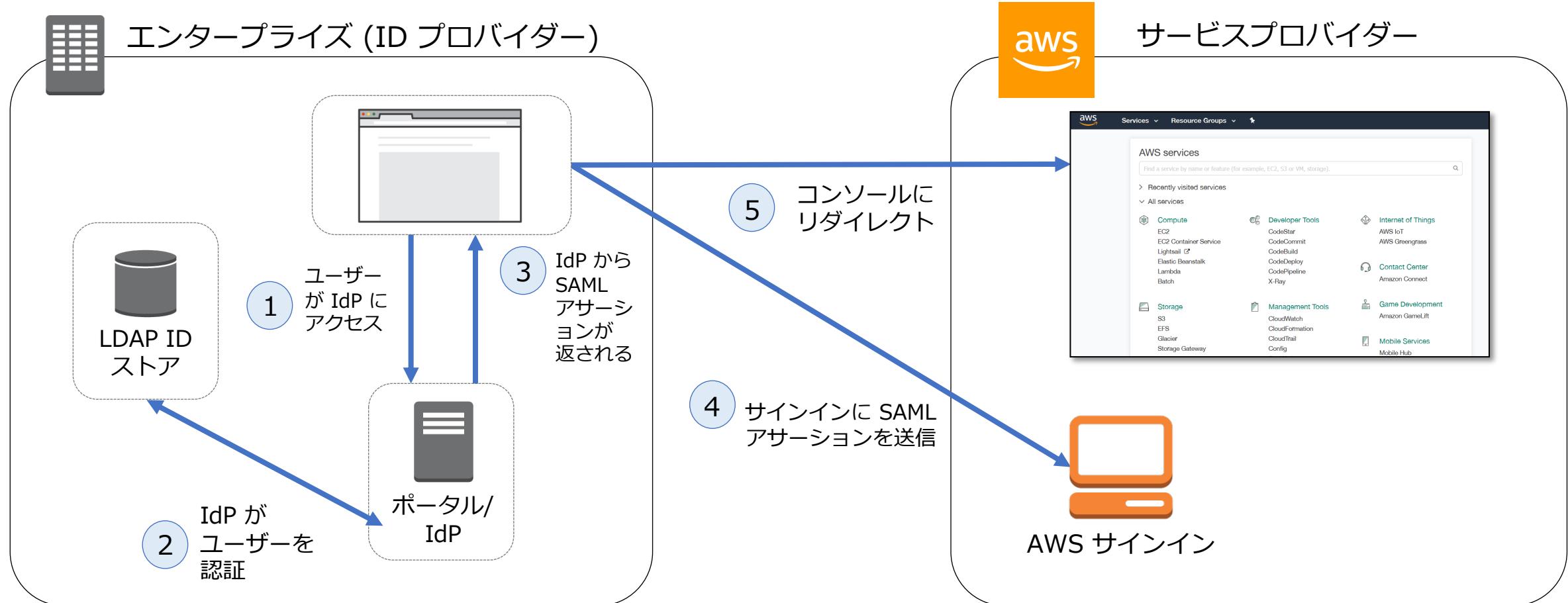
IAM ロールを利用するパターン (推奨)



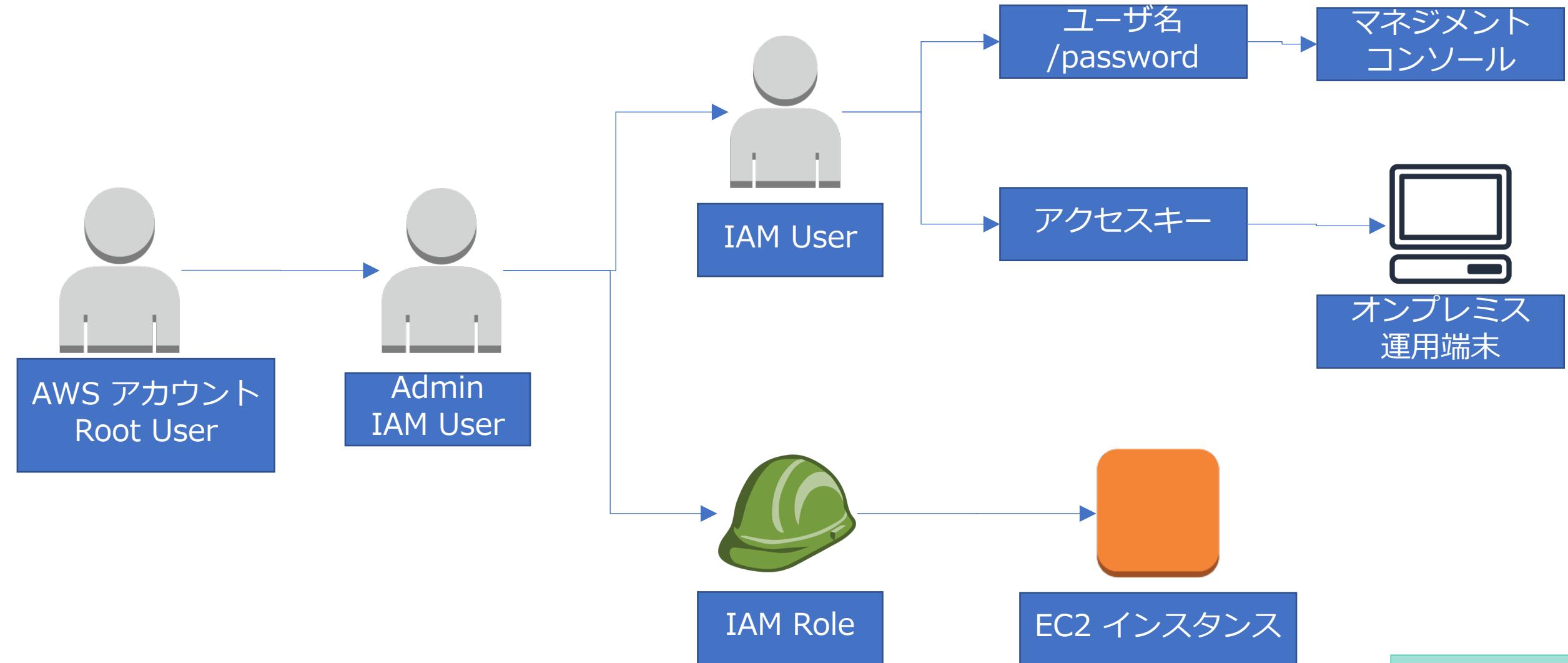
フェデレーション: STS ID ブローカー



フェデレーション: SAML



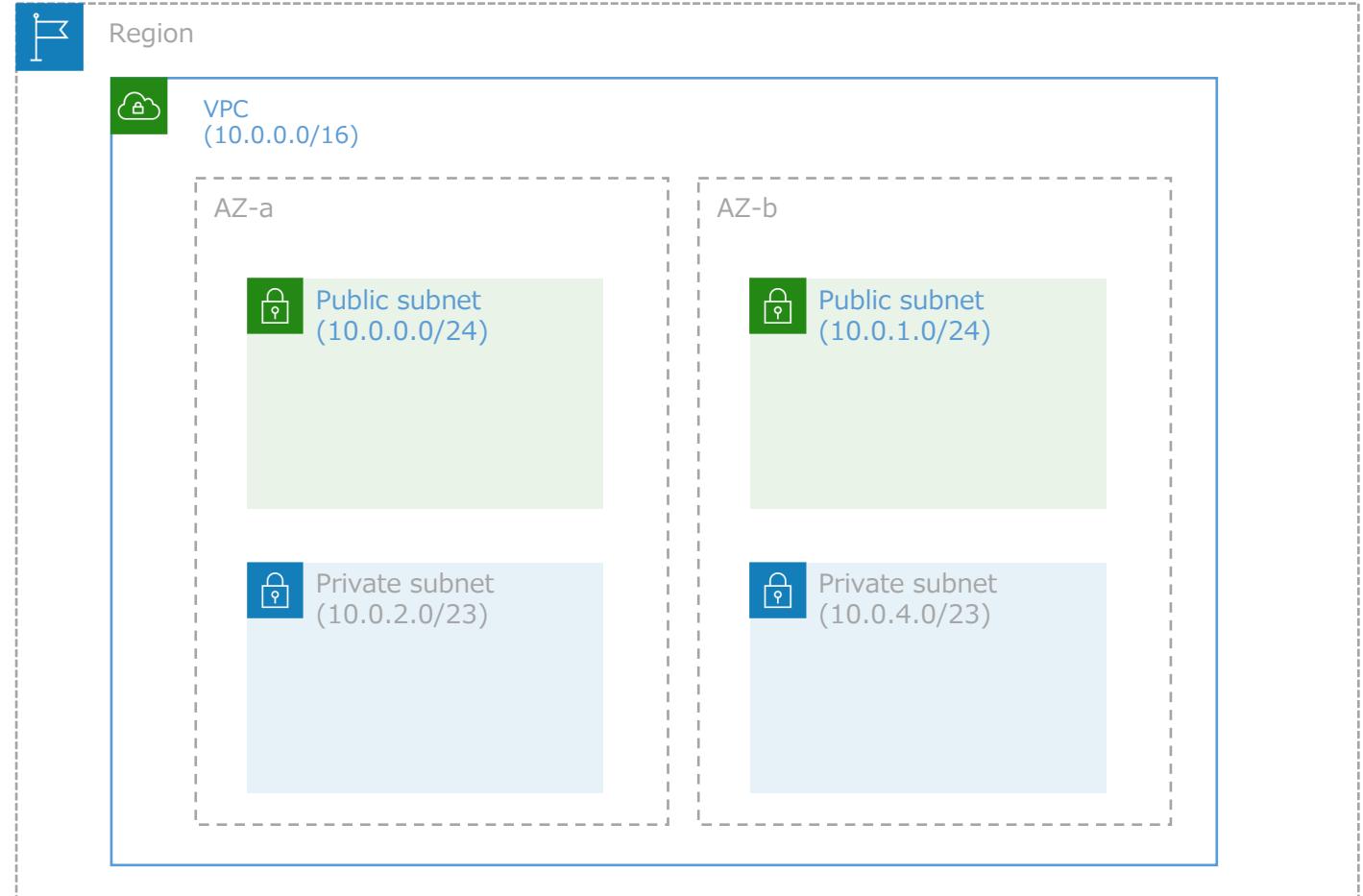
問3 解説補足



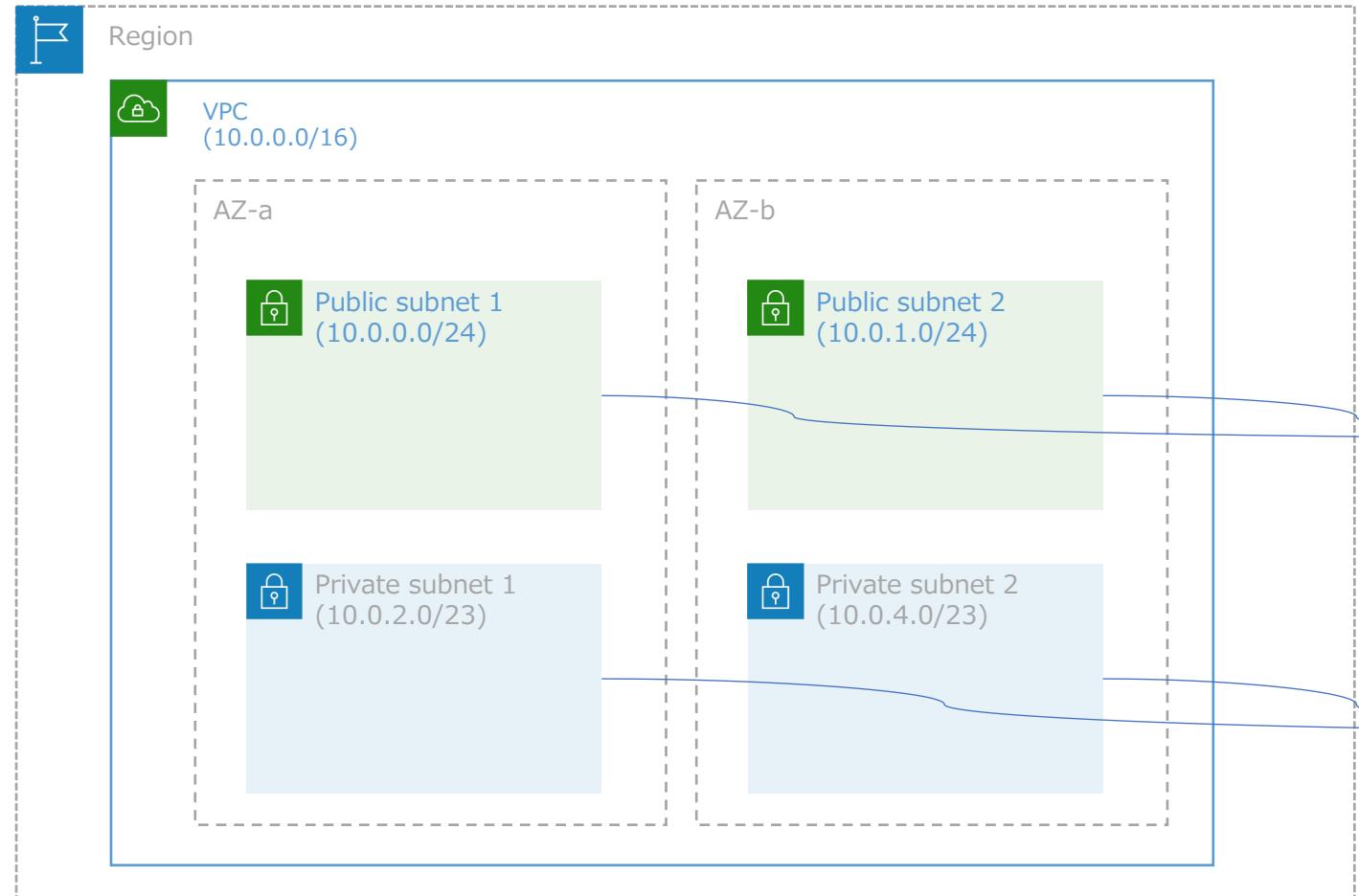
モジュール 3

ネットワーク 1

VPC 構成



VPC: ルートテーブル



メインルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

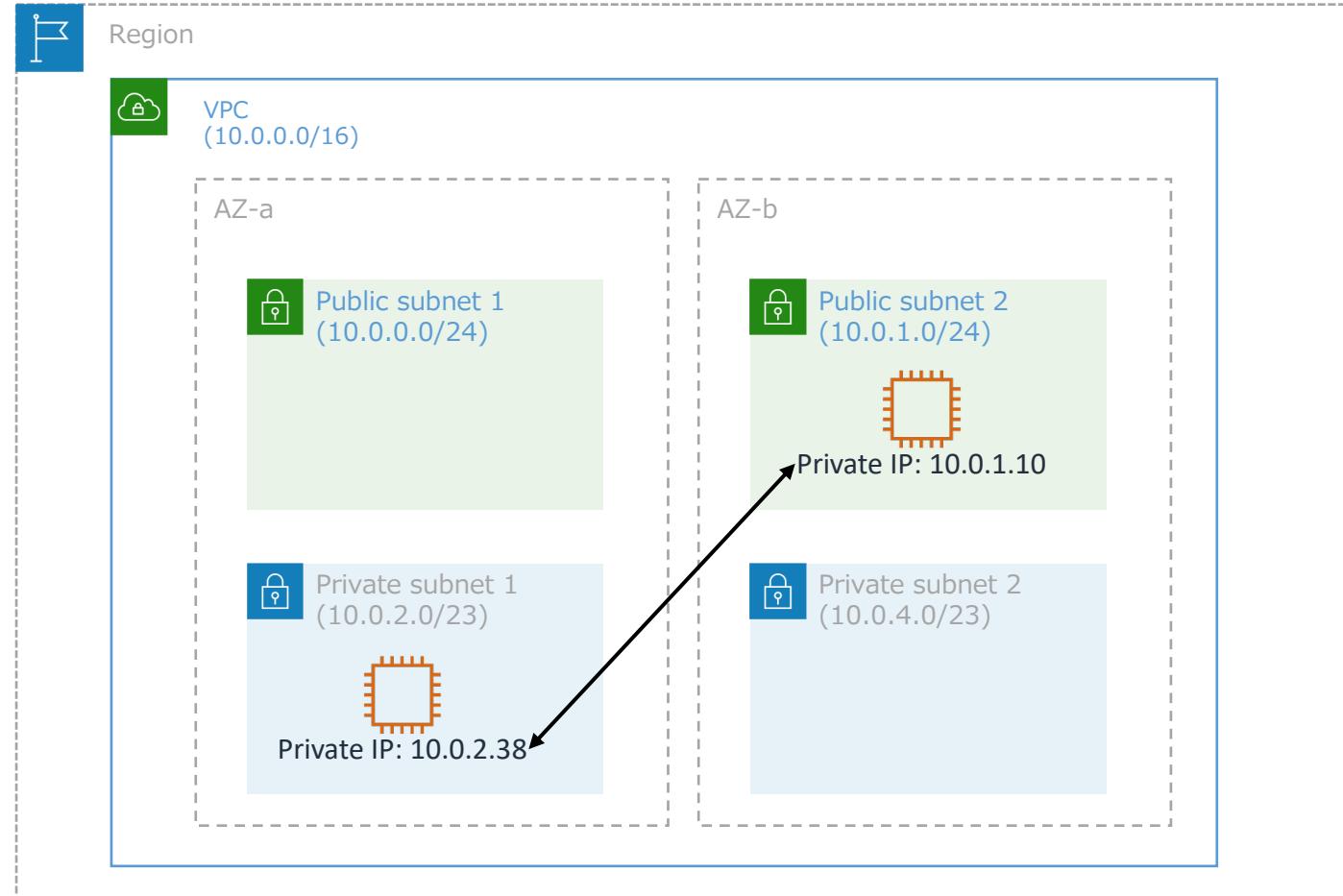
パブリックサブネット用ルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

プライベートサブネット用ルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

VPC: ローカル通信



メインルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

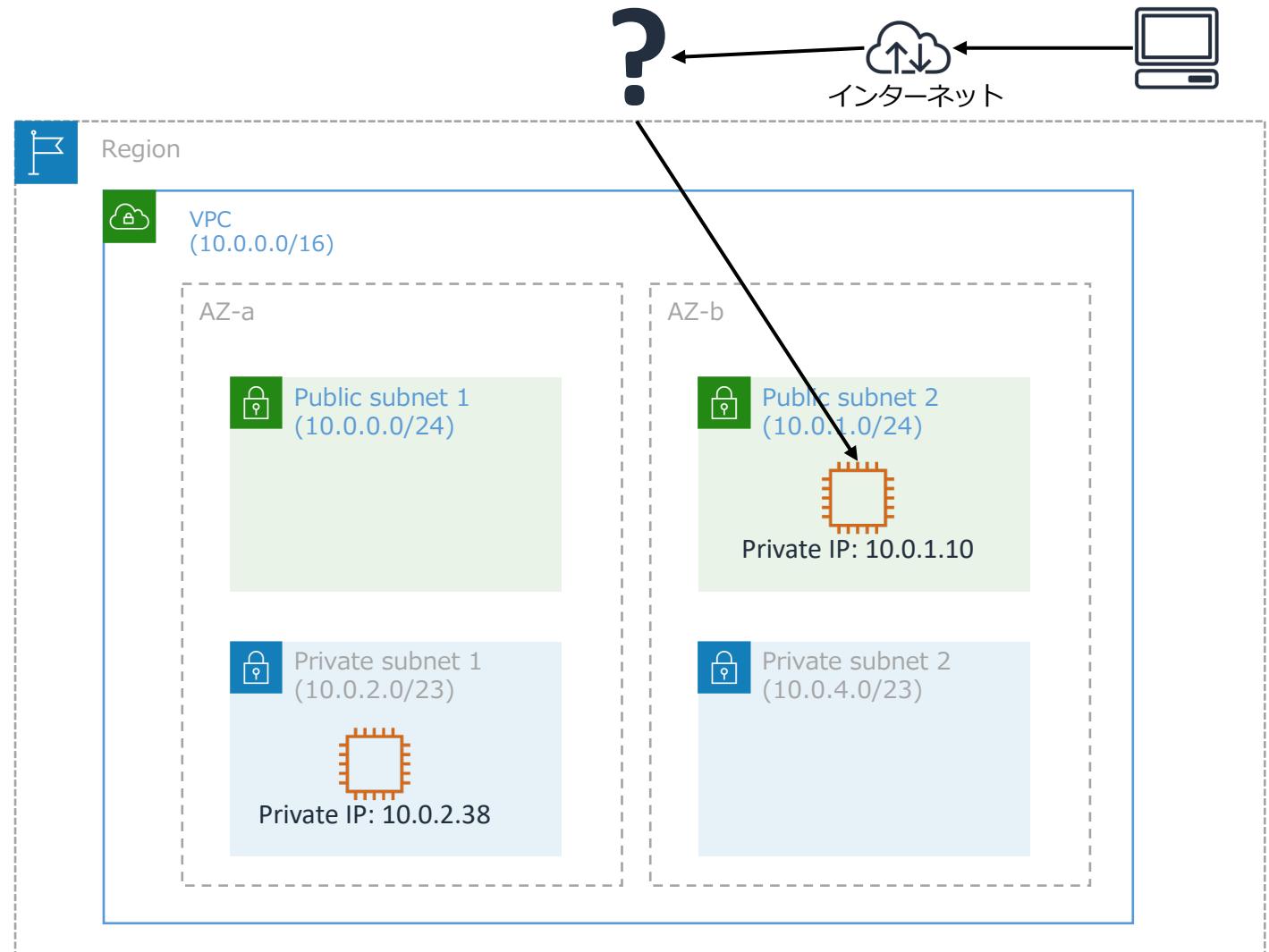
パブリックサブネット用ルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

プライベートサブネット用ルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

VPC: インターネットとの通信



メインルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

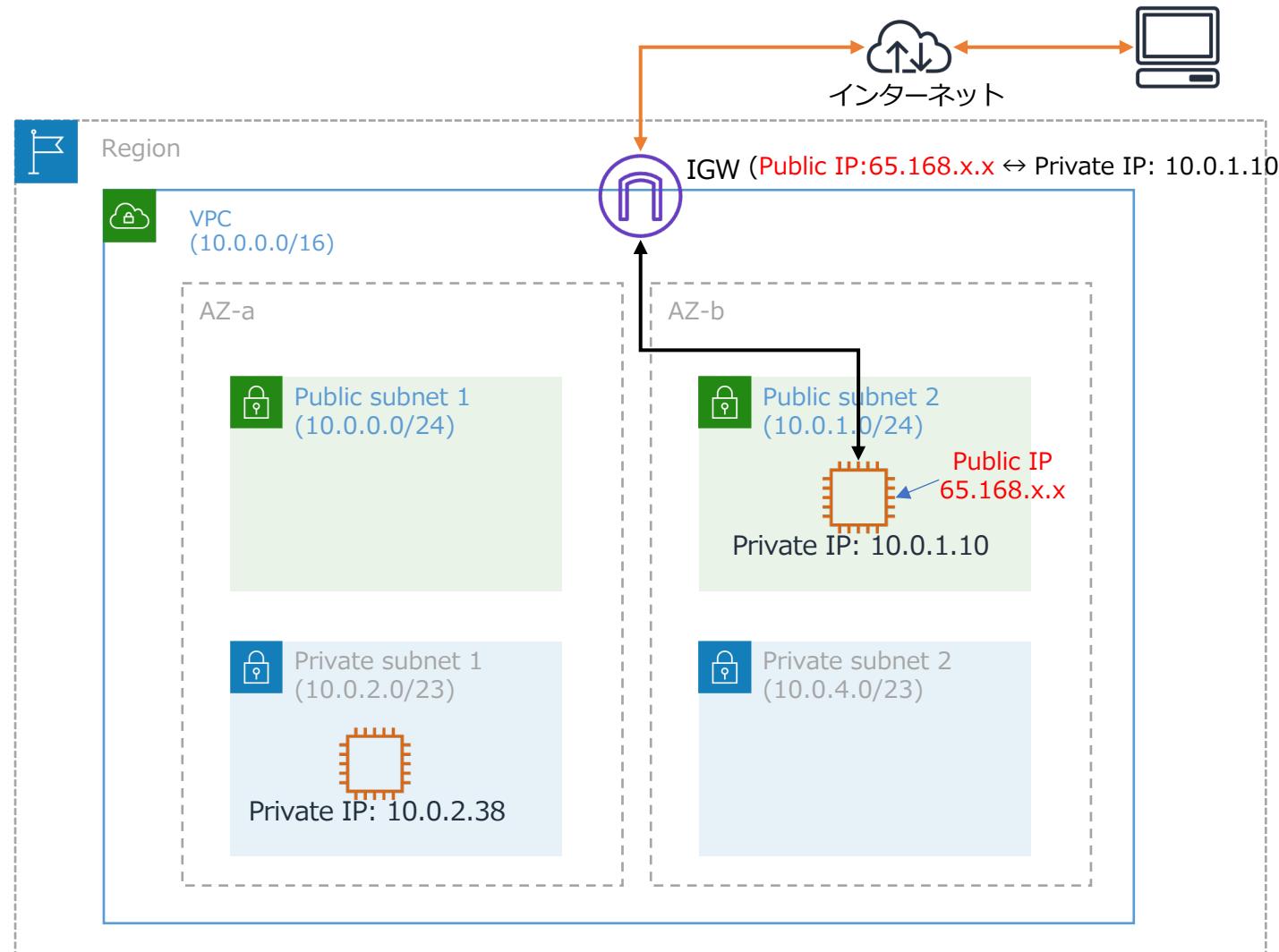
パブリックサブネット用ルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

プライベートサブネット用ルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

VPC: インターネットとの通信



メインルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

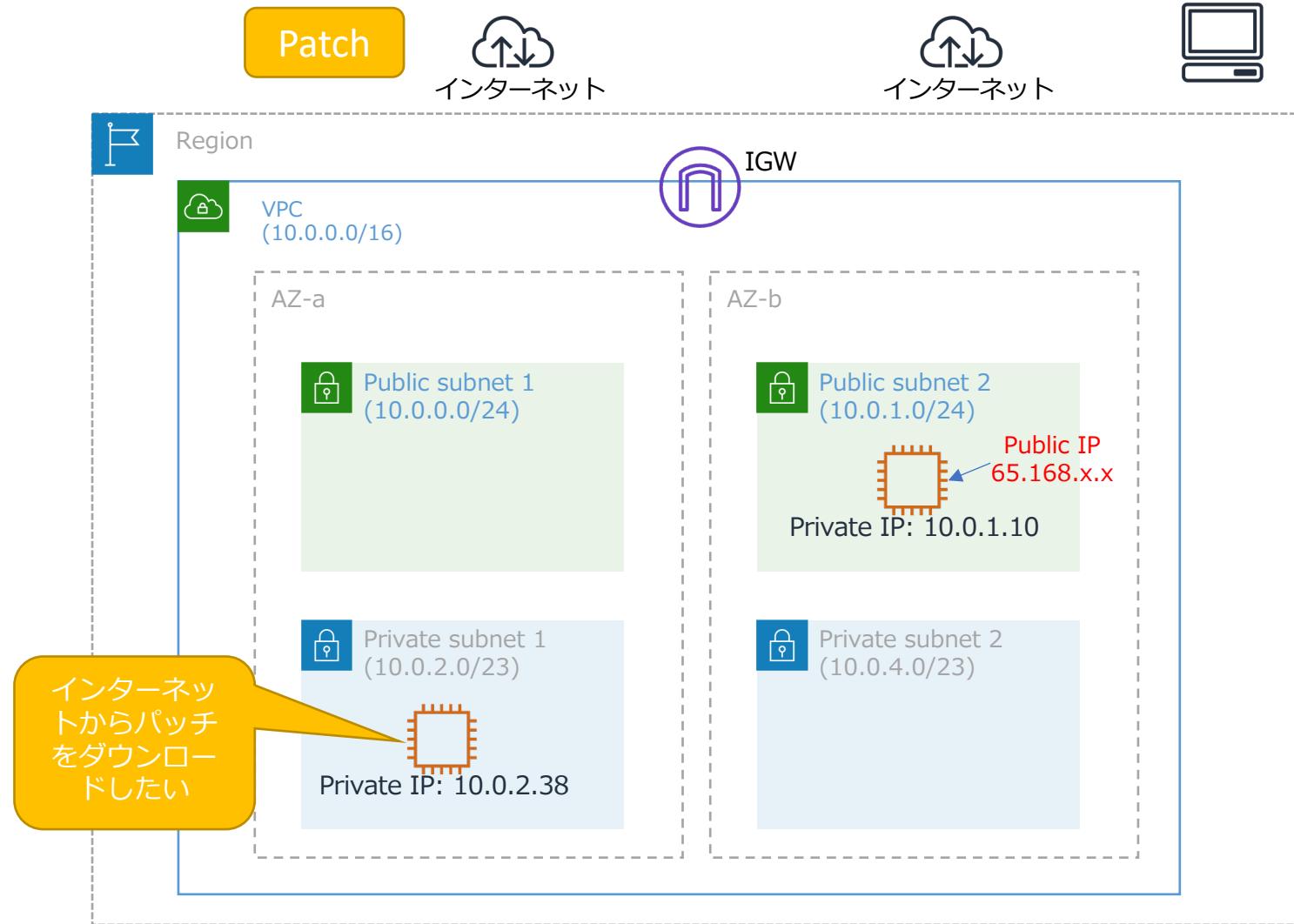
パブリックサブネット用ルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル
0.0.0.0/0	igw-xxxxxx

プライベートサブネット用ルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

VPC: インターネットとの通信



メインルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

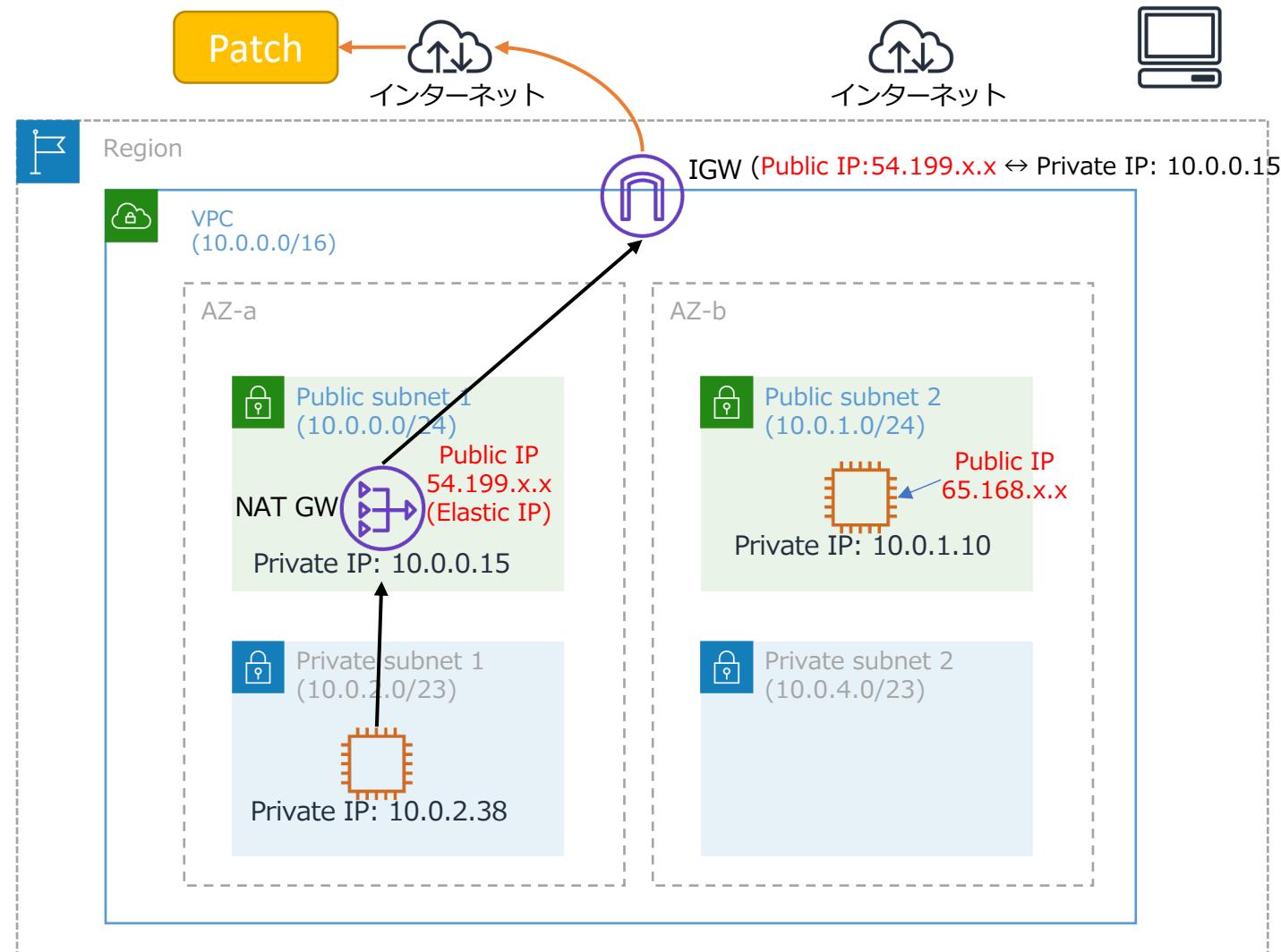
パブリックサブネット用ルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル
0.0.0.0/0	igw-xxxxxx

プライベートサブネット用ルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

VPC: インターネットとの通信



メインルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル

パブリックサブネット用ルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル
0.0.0.0/0	igw-xxxxxx

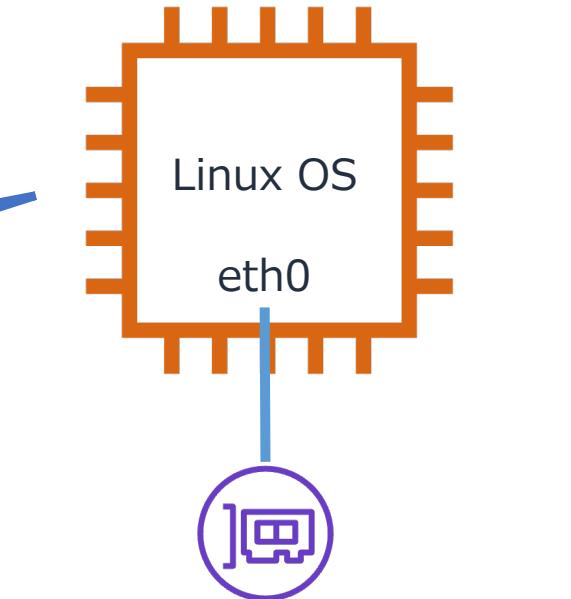
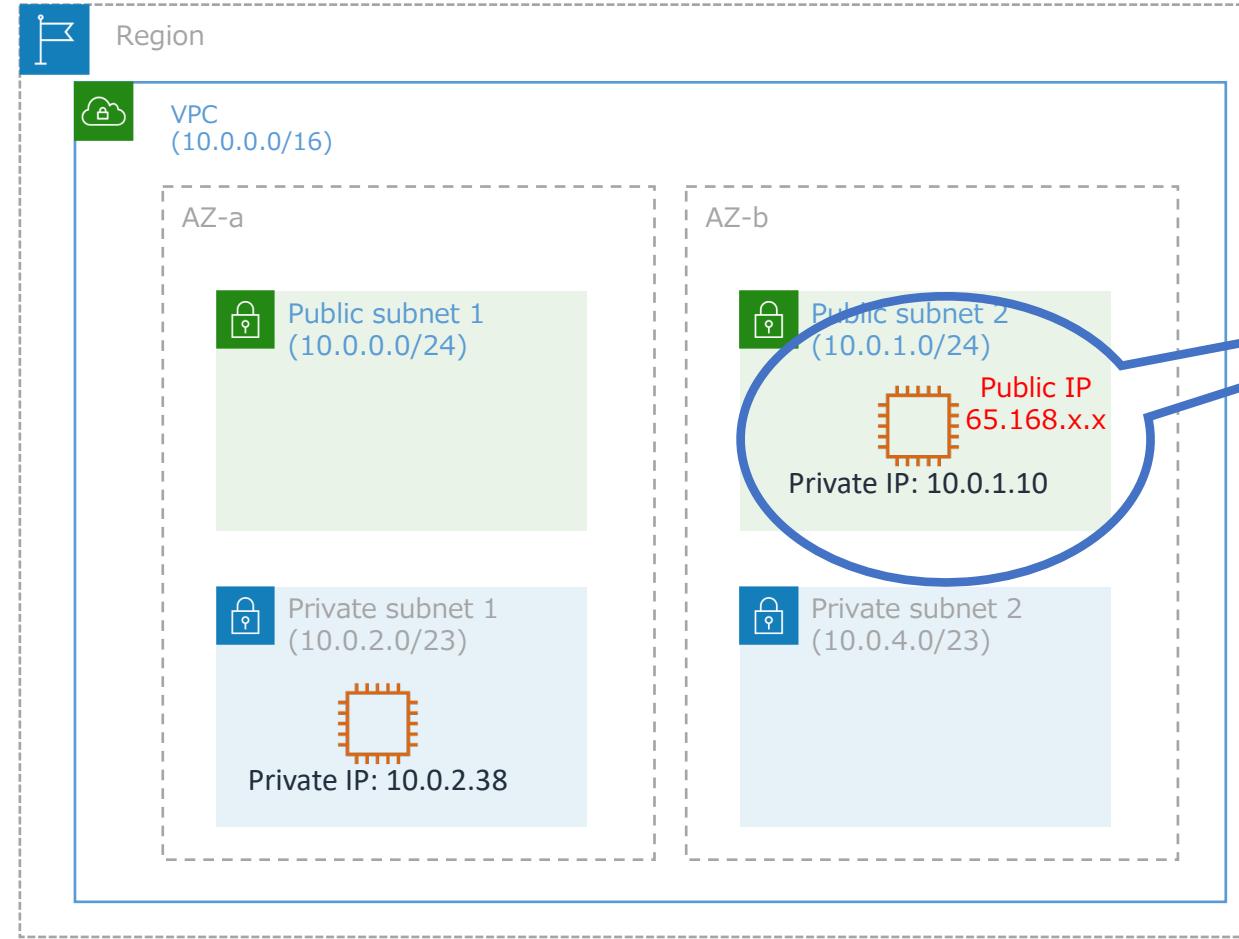
プライベートサブネット用ルートテーブル

送信先	ターゲット
10.0.0.0/16	ローカル
0.0.0.0/0	nat-xxxxxx

パブリックIP

- パブリックIP：インターネットから到達可能なIP アドレス
- インスタンスへのパブリックIP の割当
 - 自動割当パブリックIP
 - インスタンス起動時に指定
 - インスタンスを停止して起動すると異なるIP に変わる
 - Elastic IP
 - インスタンスのライフサイクルとは異なる管理
 - 任意のインスタンスに割り当てたり付け替えたりが可能
 - 固定的なパブリックIP を利用したい場合

Elastic Network Interface



Elastic Network Interface

Private IP: 10.0.1.10
Public IP: 65.168.x.x
Mac Address: xxxxxxxx
Security Group: sg-xxx

パブリックサブネット/プライベートサブネット

パブリック
サブネット

サブネットを使用してインターネットアクセスを定義する

パブリックサブネット

- ルートテーブルにインターネットゲートウェイへのエントリがあり、パブリックインターネットへのインバウンド/アウトバウンドのアクセスが可能



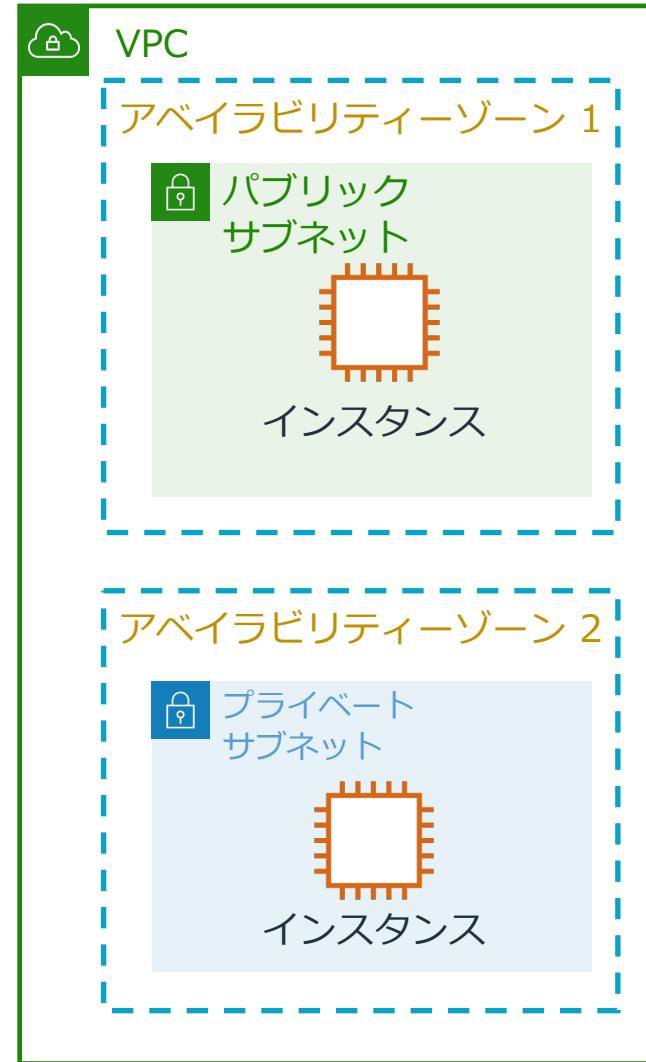
プライベート
サブネット

プライベートサブネット

- ルートテーブルにインターネットゲートウェイへのエントリがない
- パブリックインターネットから直接アクセスできない
- 通常は NAT ゲートウェイを使用して、パブリックインターネットへの制限付きアウトバウンドのアクセスをサポート

デフォルトのネットワークACL

- VPC を作成すると自動的にデフォルトネットワークACL も作成される
- 明示的にネットワークACL を指定しない場合は、デフォルトのネットワークACL が適用される



デフォルト NACL

インバウンド

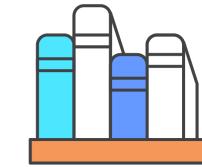
ルール番号	タイプ	プロトコル	ポート範囲	送信元	許可または拒否
100	すべてのトラフィック	すべて	すべて	0.0.0.0/0	許可
*	すべてのトラフィック	すべて	すべて	0.0.0.0/0	拒否

アウトバウンド

ルール番号	タイプ	プロトコル	ポート範囲	送信先	許可または拒否
100	すべてのトラフィック	すべて	すべて	0.0.0.0/0	許可
*	すべてのトラフィック	すべて	すべて	0.0.0.0/0	拒否

補足

サブネットのユースケースの例



データストアインスタンス



プライベートサブネット



バッチ処理インスタンス



プライベートサブネット



バックエンドインスタンス



プライベートサブネット

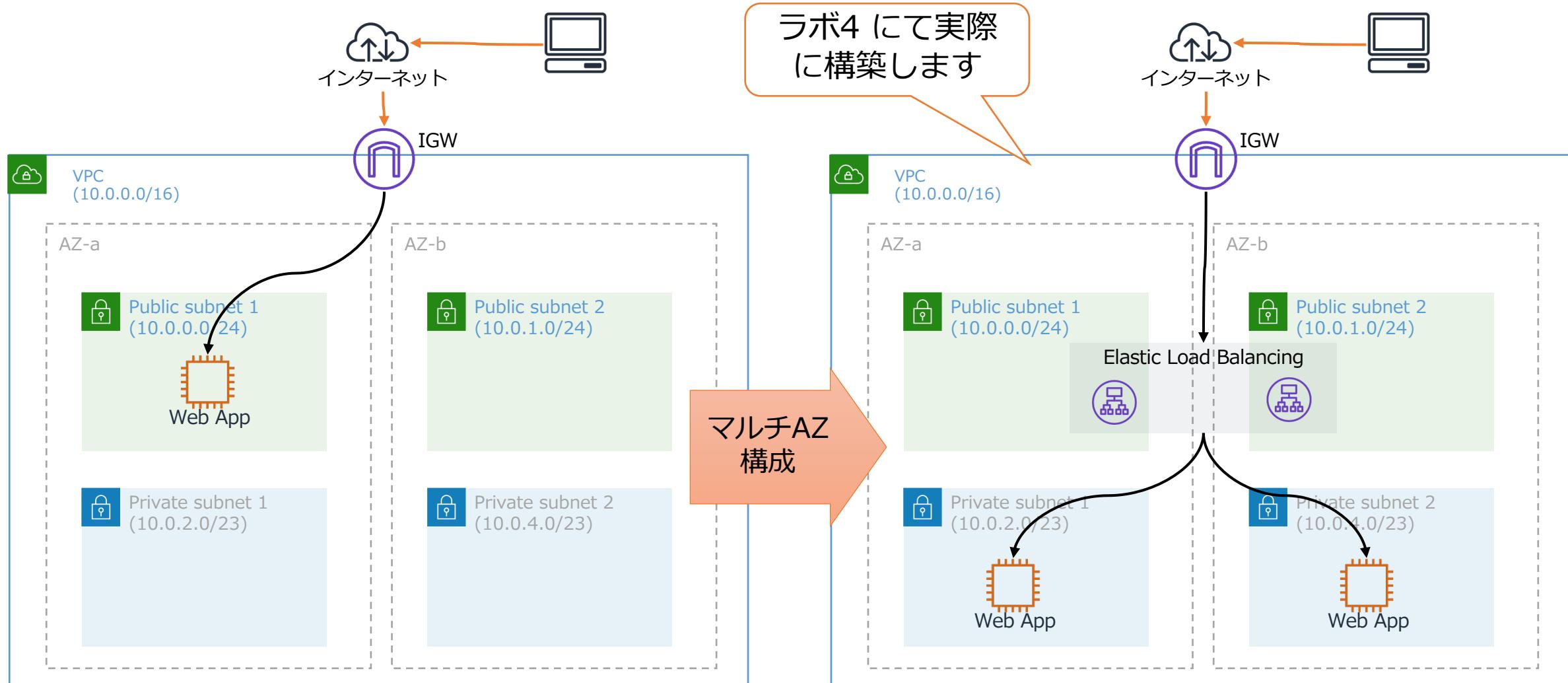


ウェブアプリケーション
インスタンス

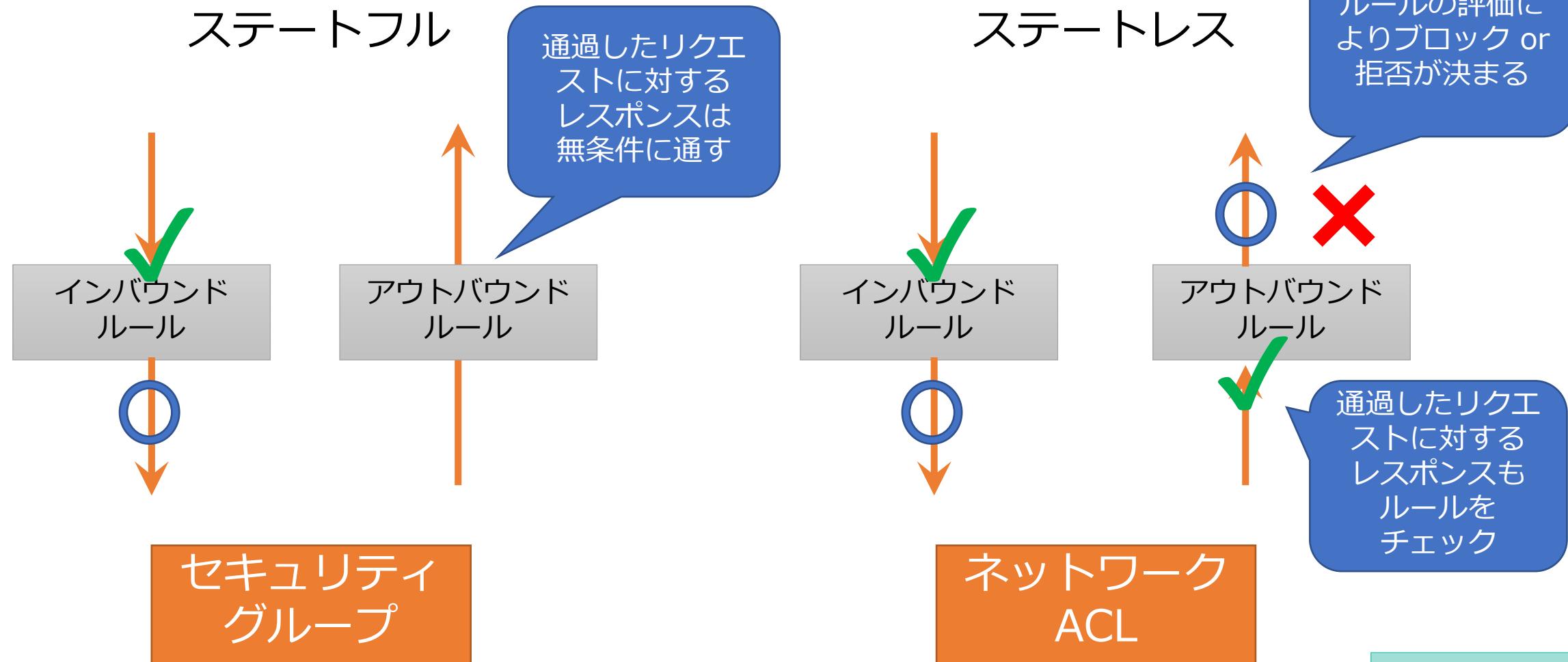


パブリックサブネットまたは
プライベートサブネット

推奨設計：マルチAZ構成



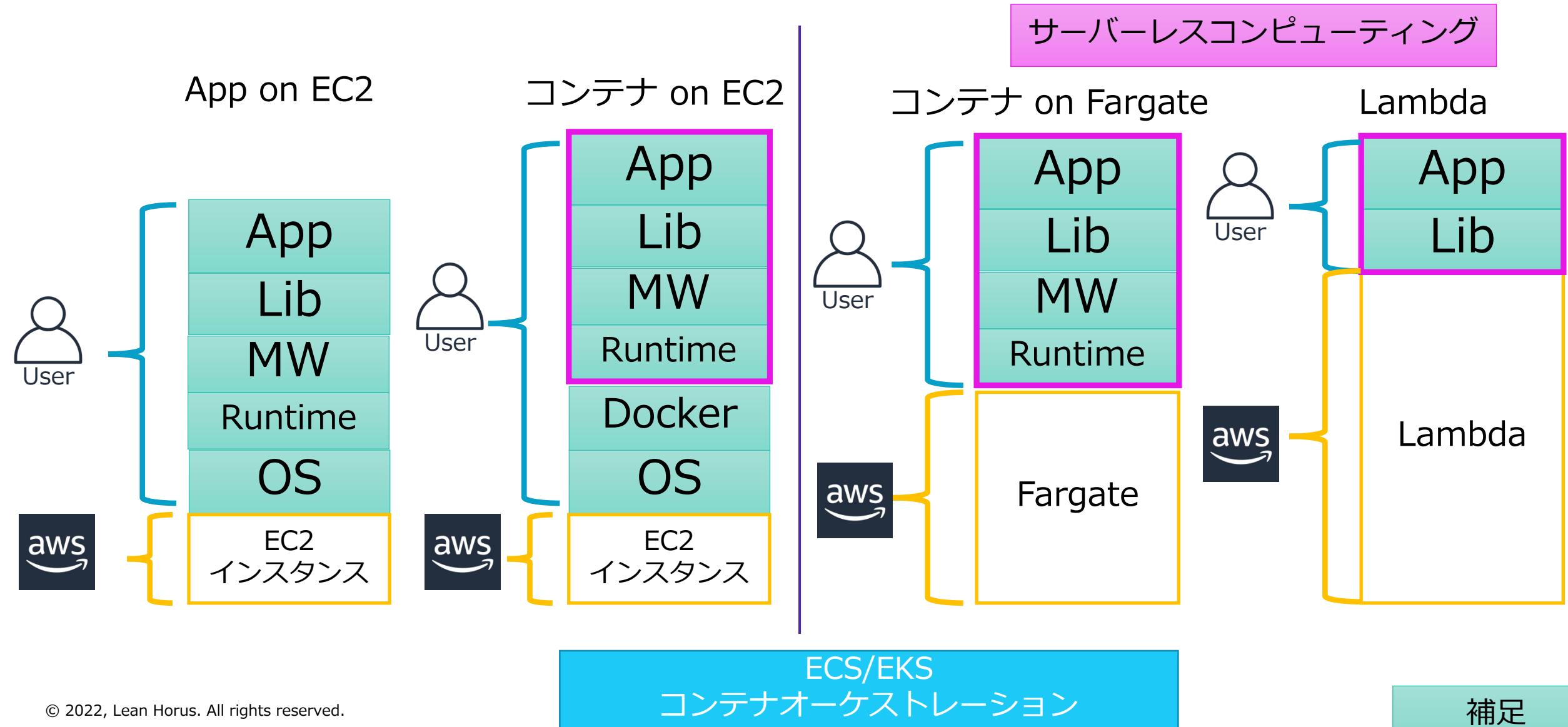
ファイアウォール: ステートフル vs ステートレス



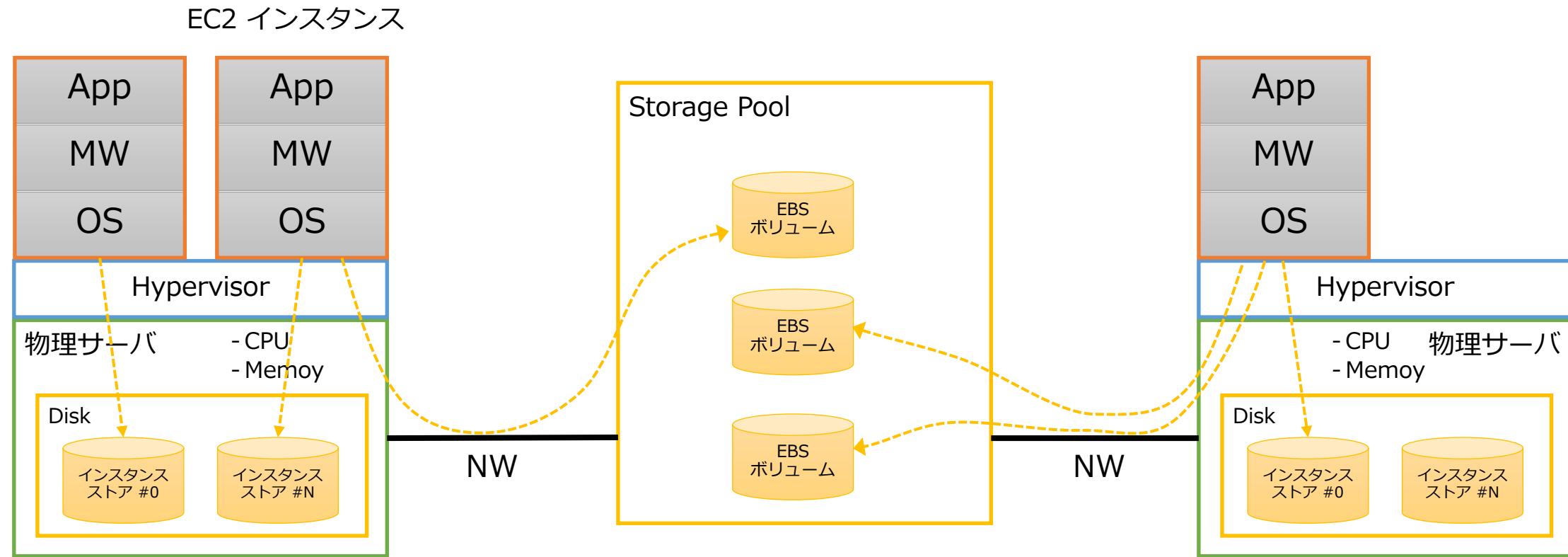
モジュール 4

コンピューティング

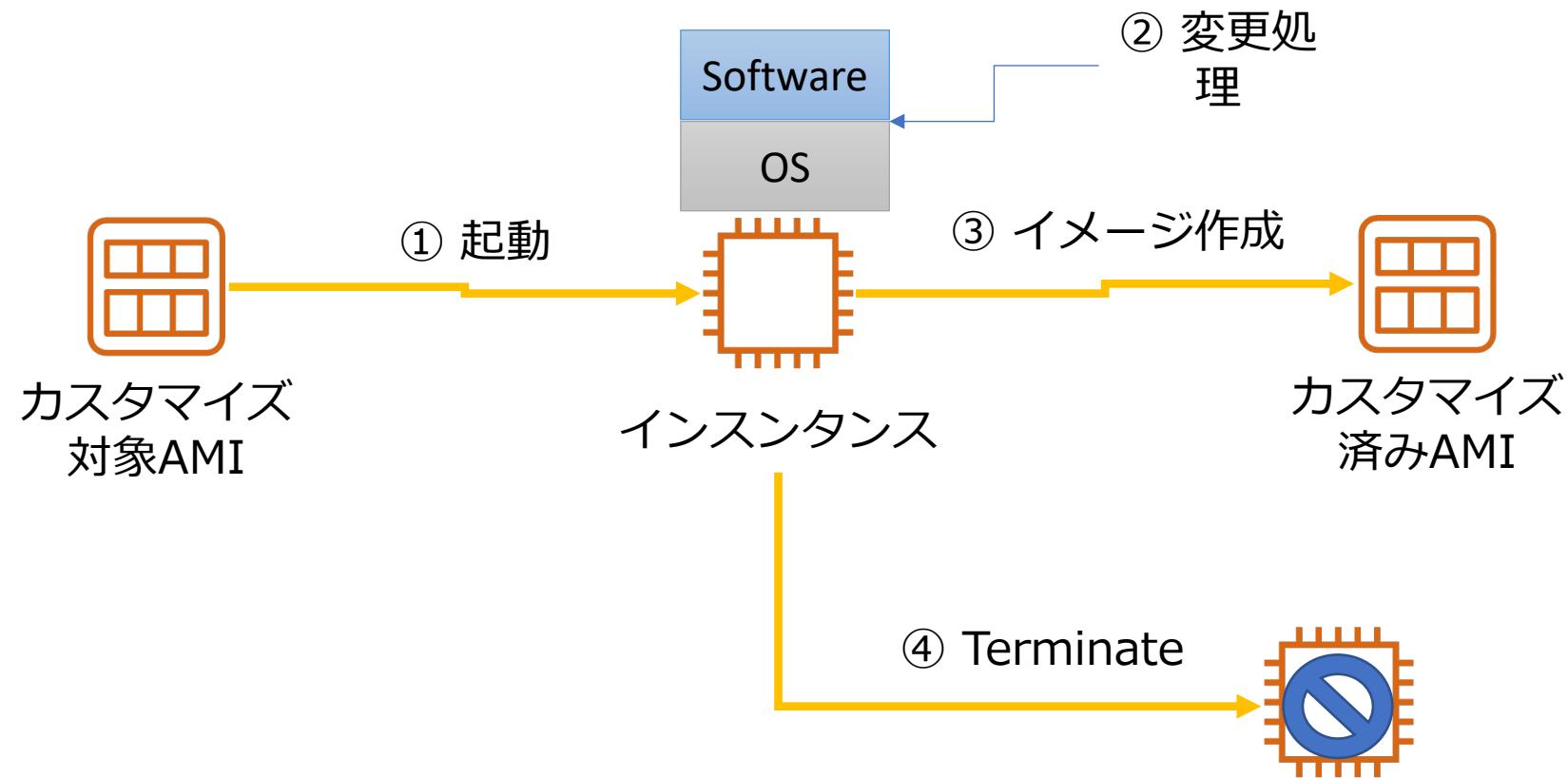
コンピューティング の進化



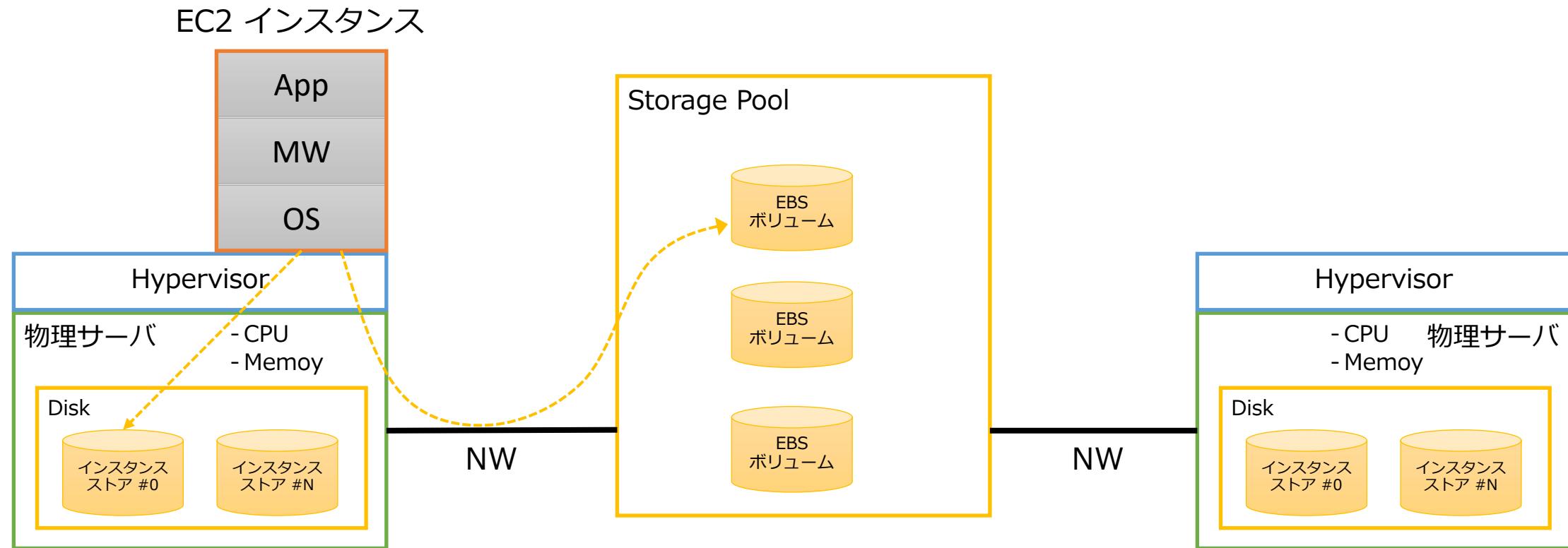
EC2 インスタンス



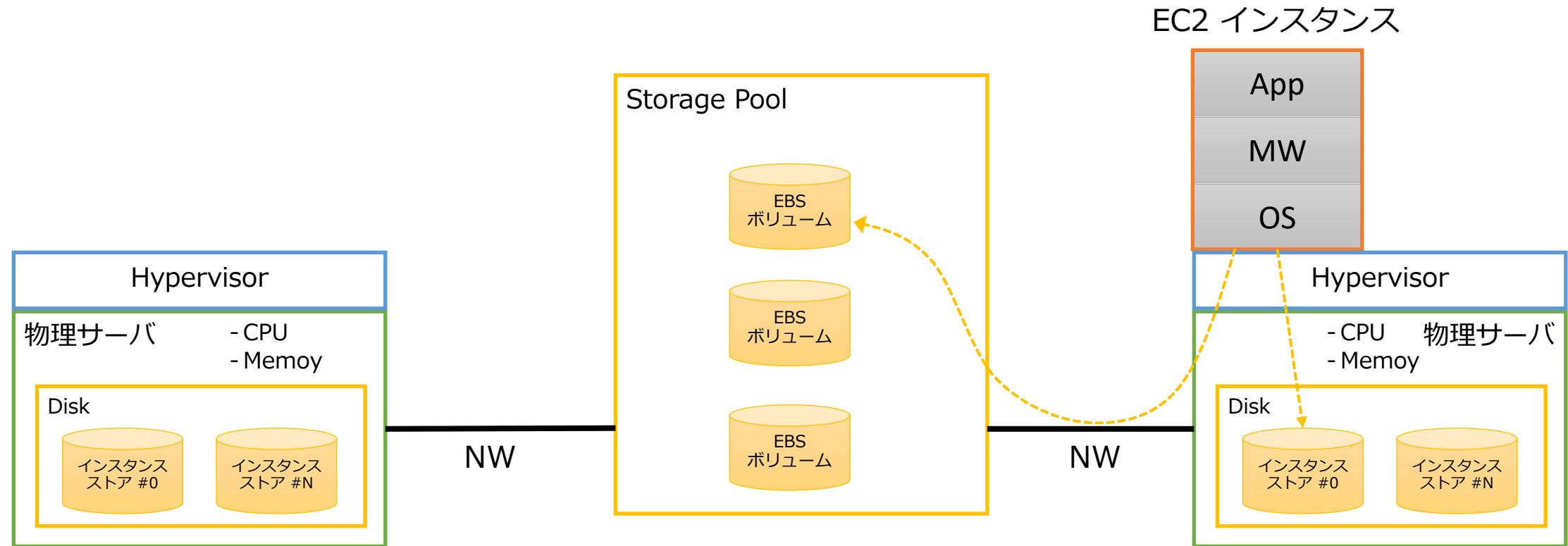
AMI のメンテナンス方法



EC2 インスタンスライフサイクルとストレージ

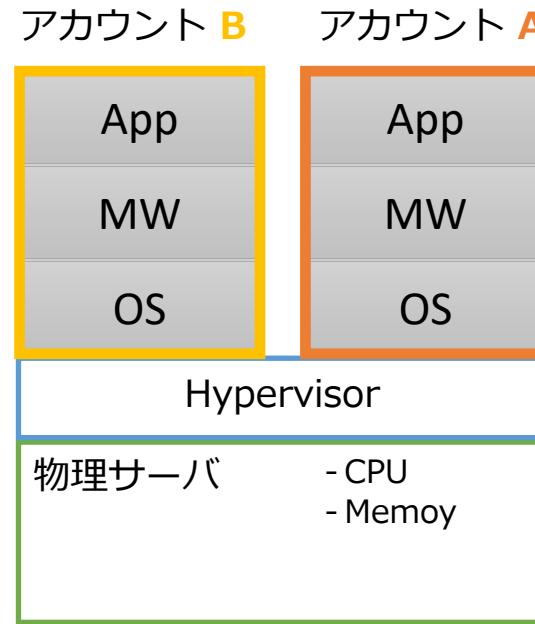


EC2 インスタンスライフサイクルとストレージ

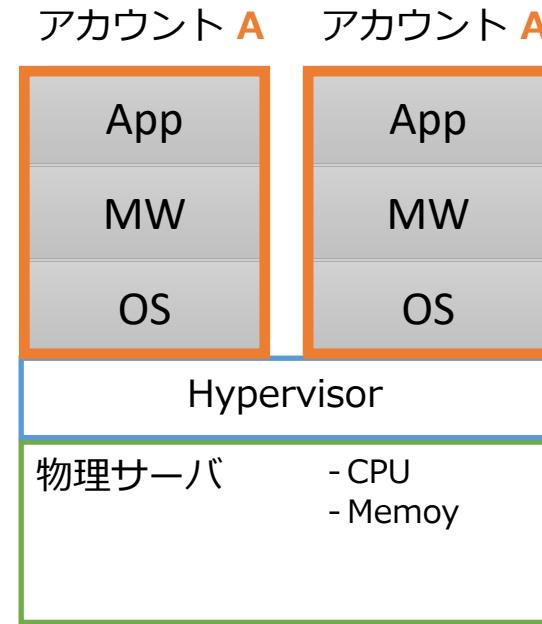


Amazon EC2 テナント

共有テナント



専有テナント

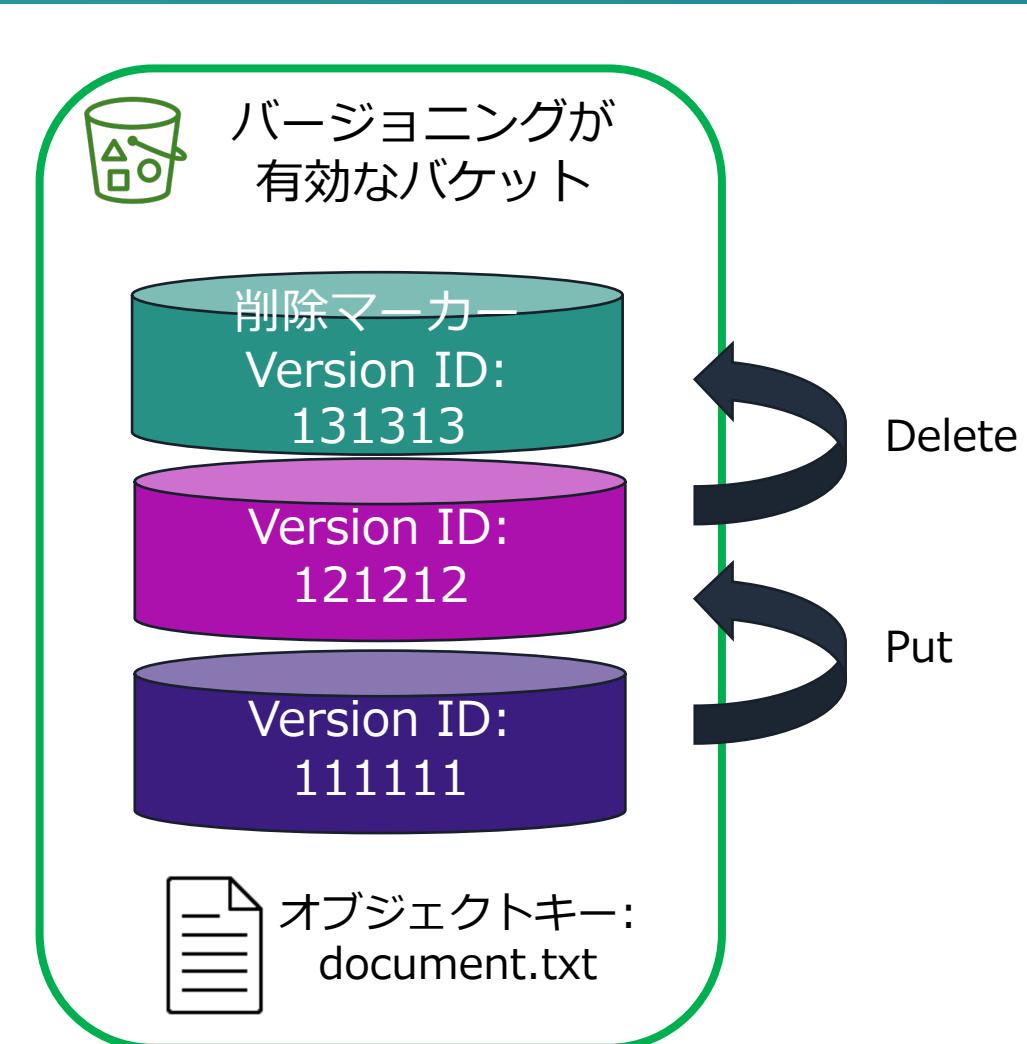


モジュール 5

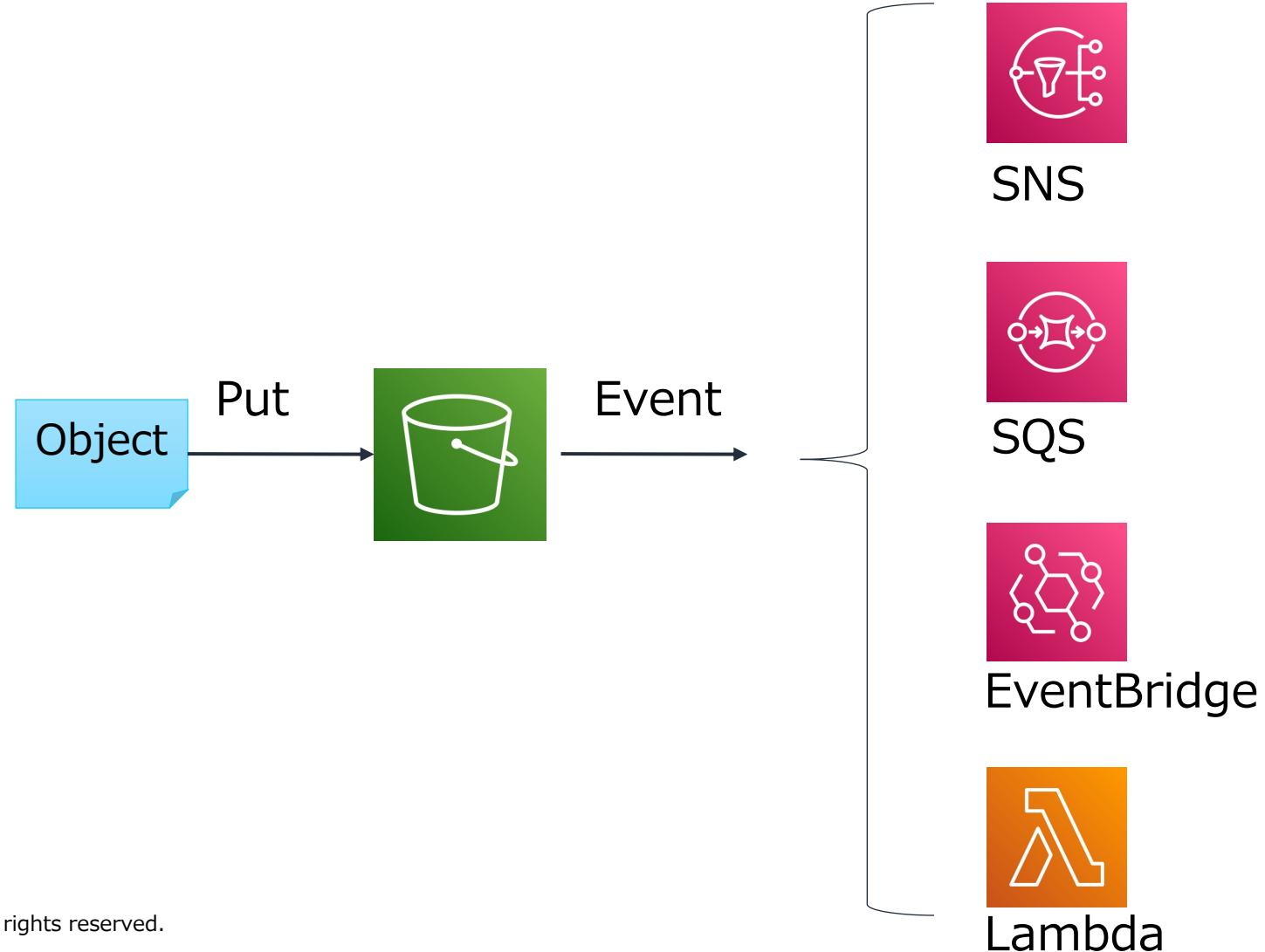
ストレージ

Amazon S3 バージョニング

- オブジェクトの複数のバージョンを同じバケットに保持する
- オブジェクトを以前のバージョンまたは特定のバージョンに復元する
- S3 オブジェクトロックを使用して、データの保持や保護を行う

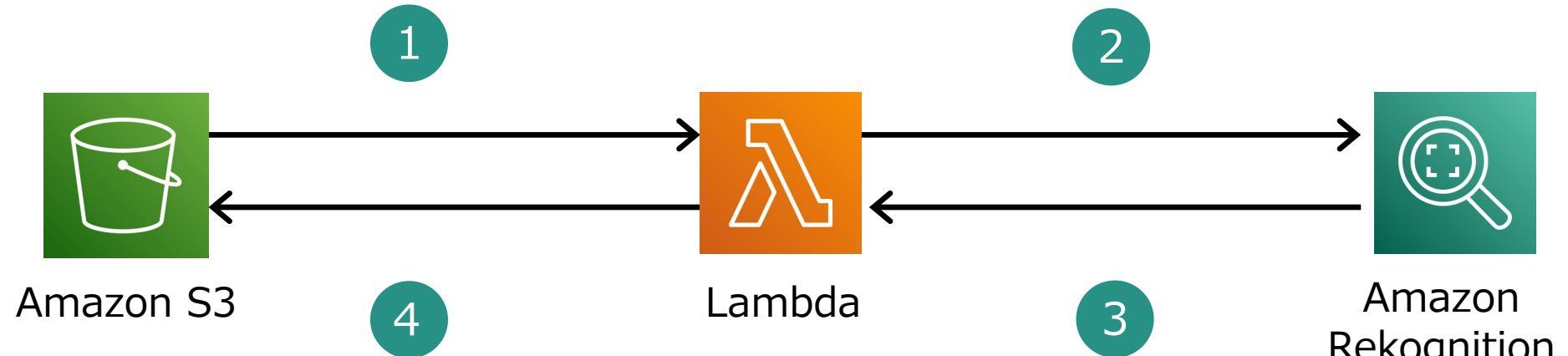


S3 イベント通知



Amazon S3 イベント通知: ユースケース: コンテンツのモデレーション解析

- 1.S3 にオブジェクトが登録されるとイベント送信
- 2.Lambda が画像データを取得し Rekognition の DetectModerationLabels API を実行
- 3.モデレーション結果を取得
- 4.オブジェクトのタグに結果を保存



モジュール 6

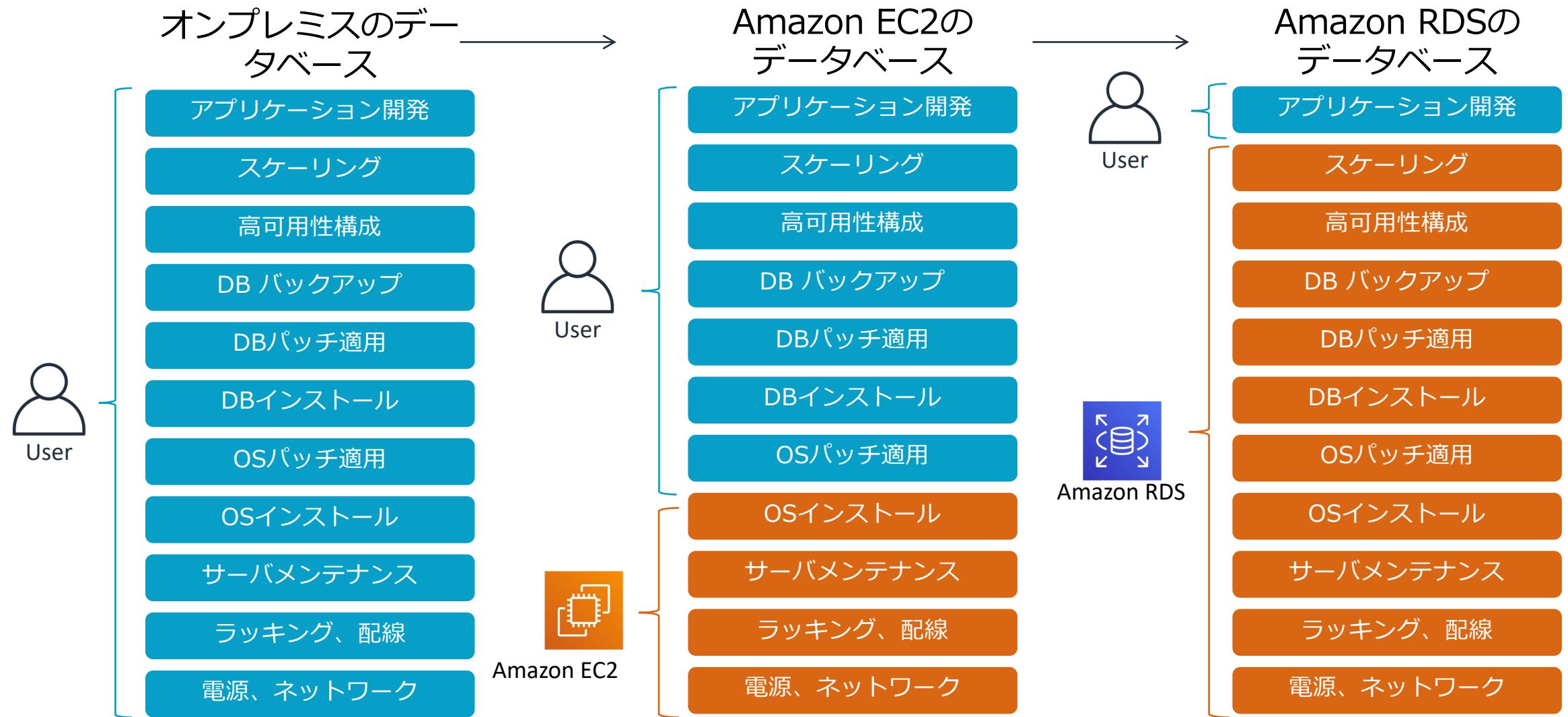
データベース

なぜデータベースが必要？

- ・ストレージにデータ保存してあればよいのでは？
- ・例：郵便番号一覧 csv から特定の郵便便番号の住所を探してみる
 - ・<https://www.post.japanpost.jp/zipcode/dl/oogaki-zip.html>
- ・リレーションナルデータベースであれば

```
SELECT 都道府県名, 市区町村, 番地 FROM 郵便番号  
テーブル WHERE 郵便番号 = '167-xxxx'
```

マネージドサービス

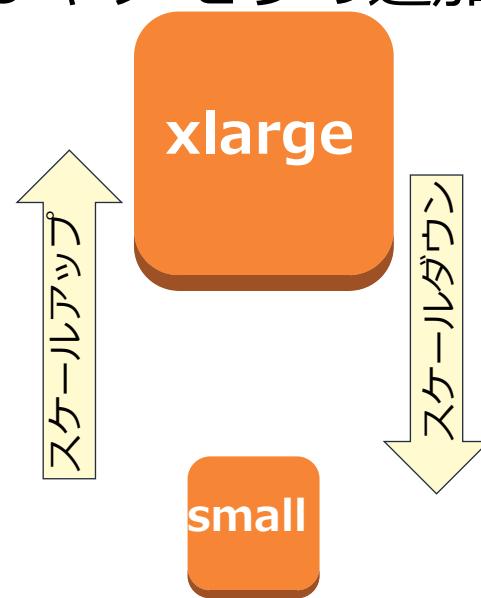


スケーラビリティとは？

- ・負荷に合わせてシステムの処理能力を柔軟に増減できる能力のこと

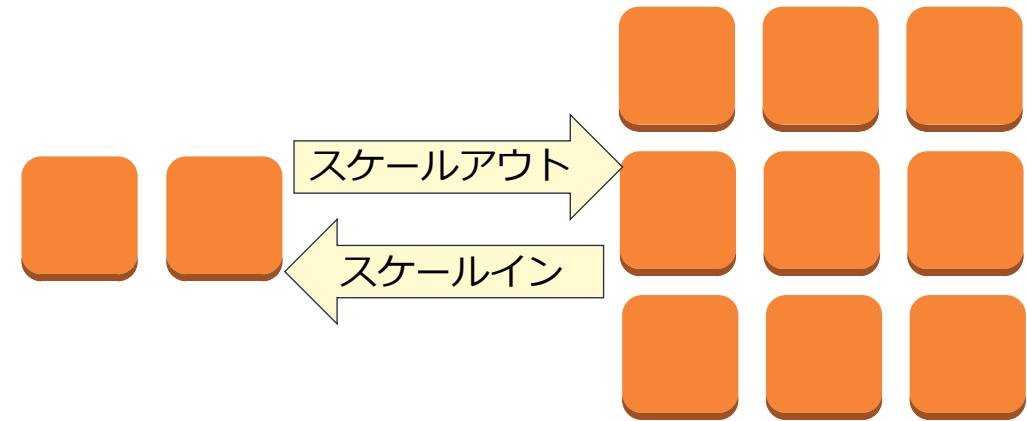
垂直方向のスケーラビリティ

インスタンスのスペックの変更
(CPU やメモリの追加など)

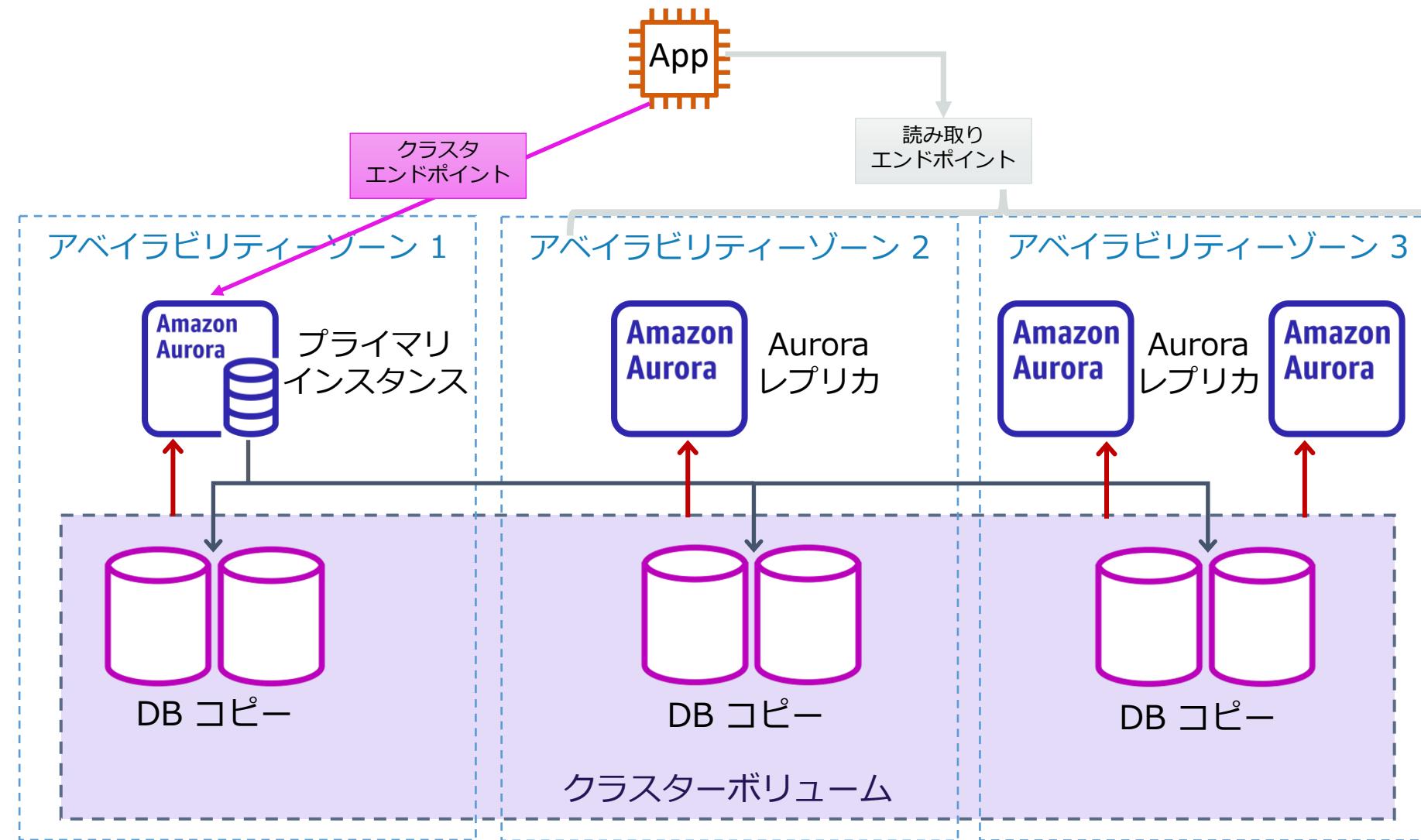


水平方向のスケーラビリティ

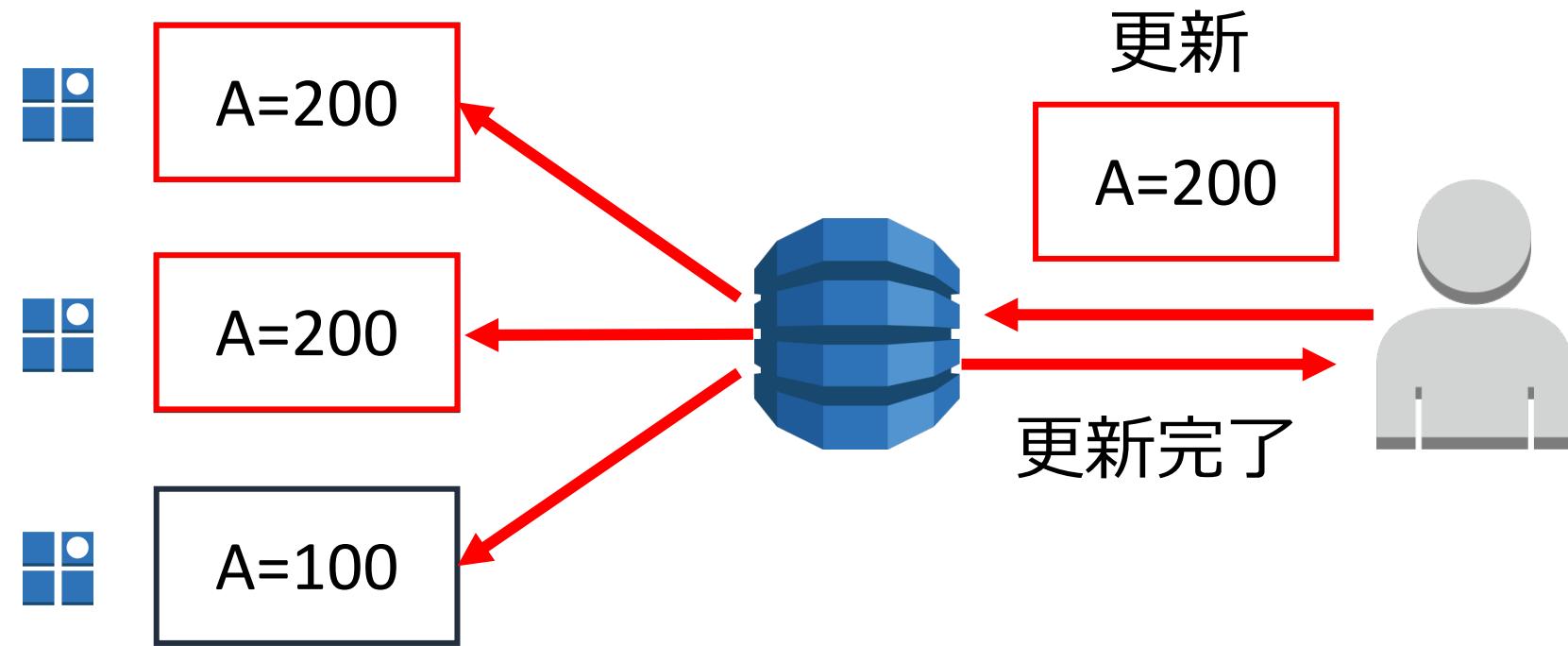
インスタンス数の変更



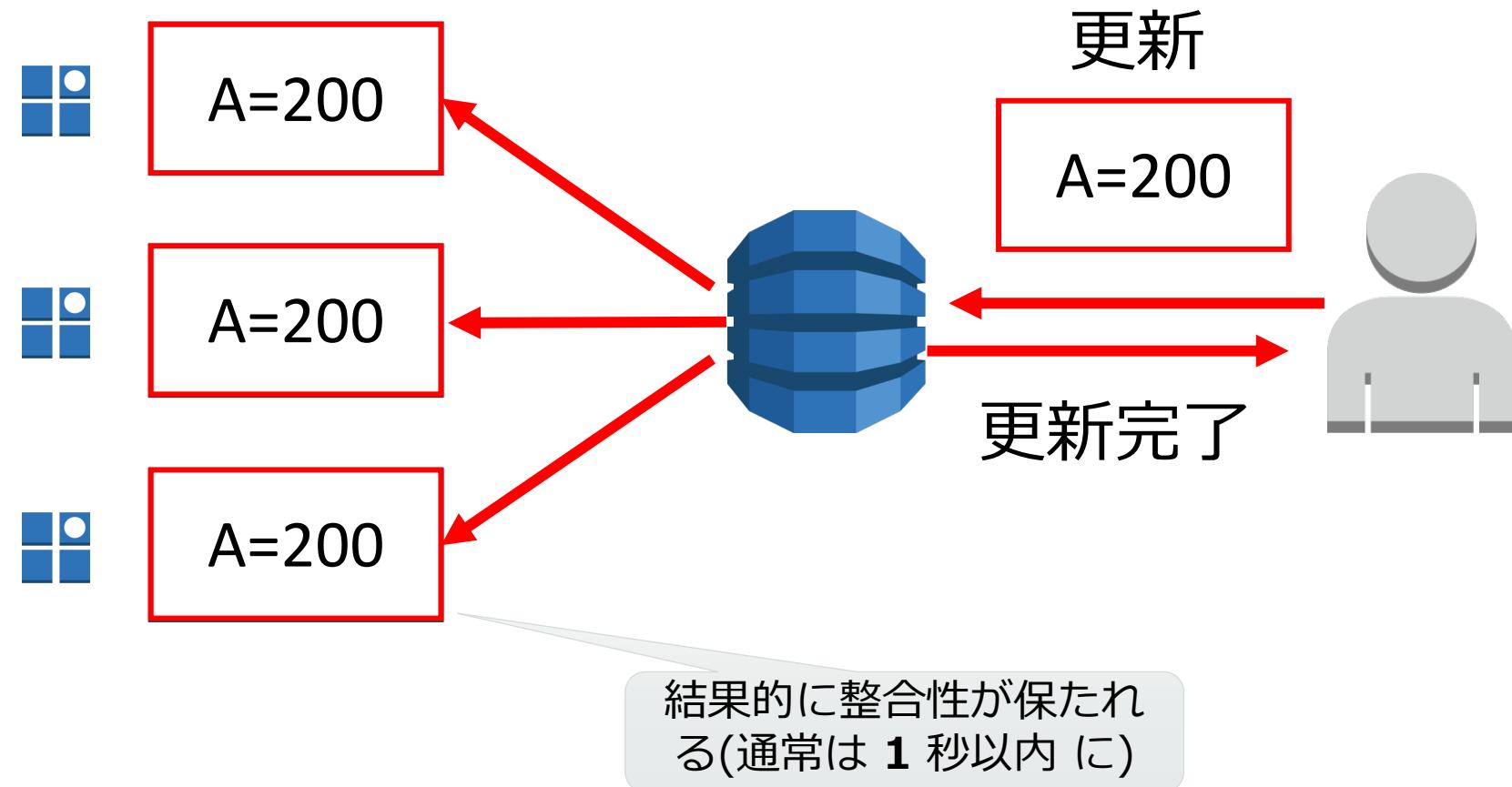
Aurora DB クラスターへのアクセス



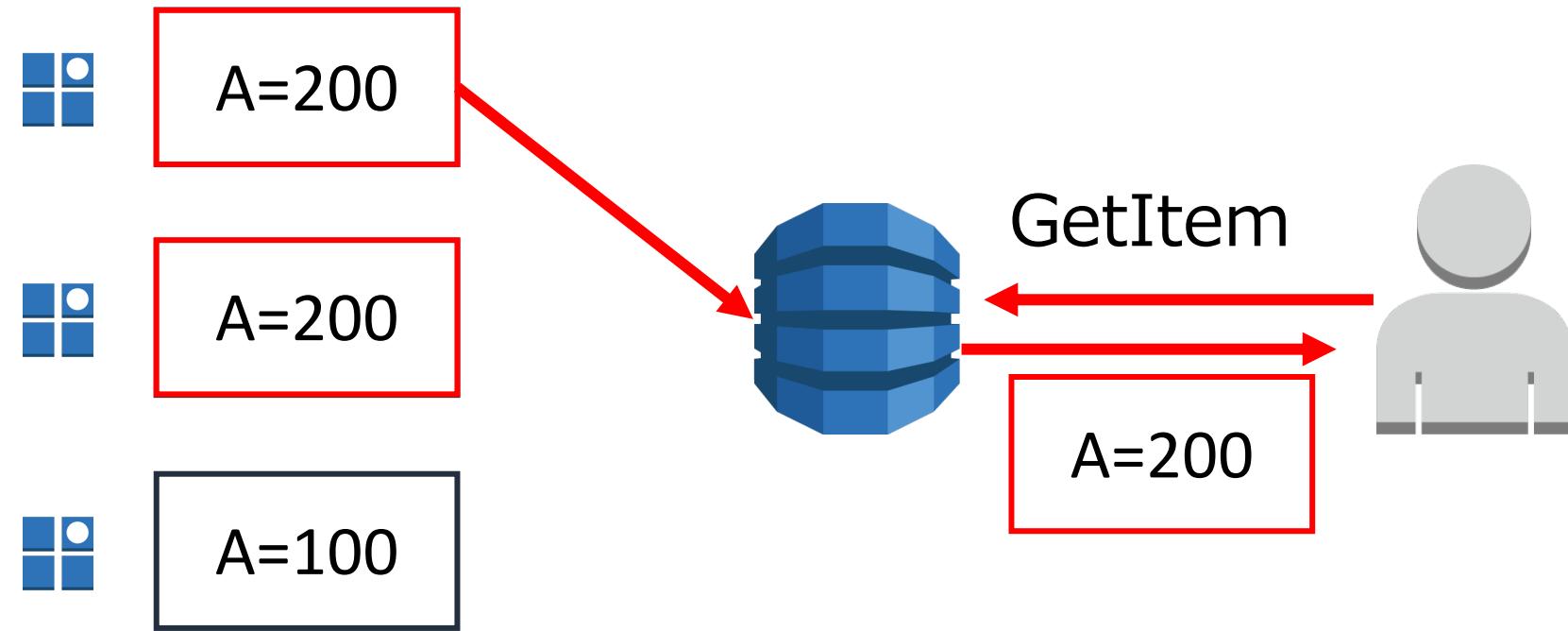
DynamoDB の更新処理



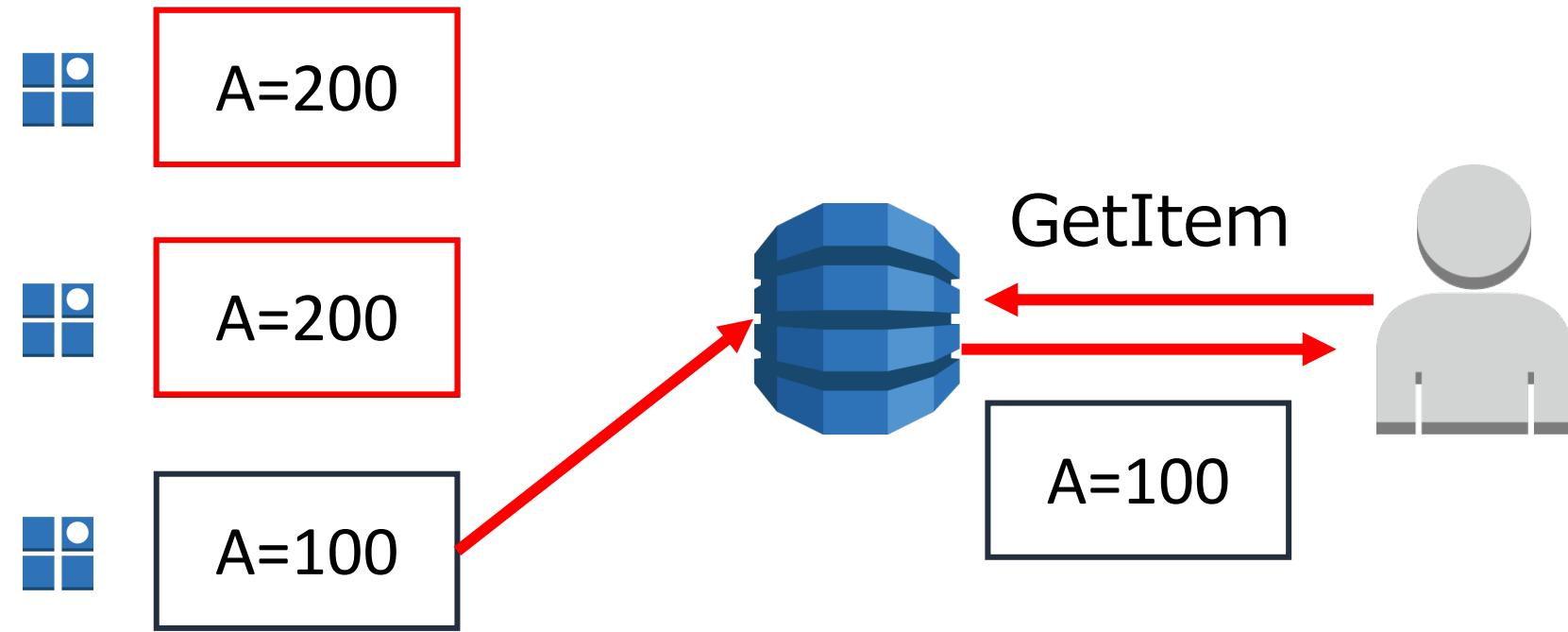
DynamoDB の更新処理



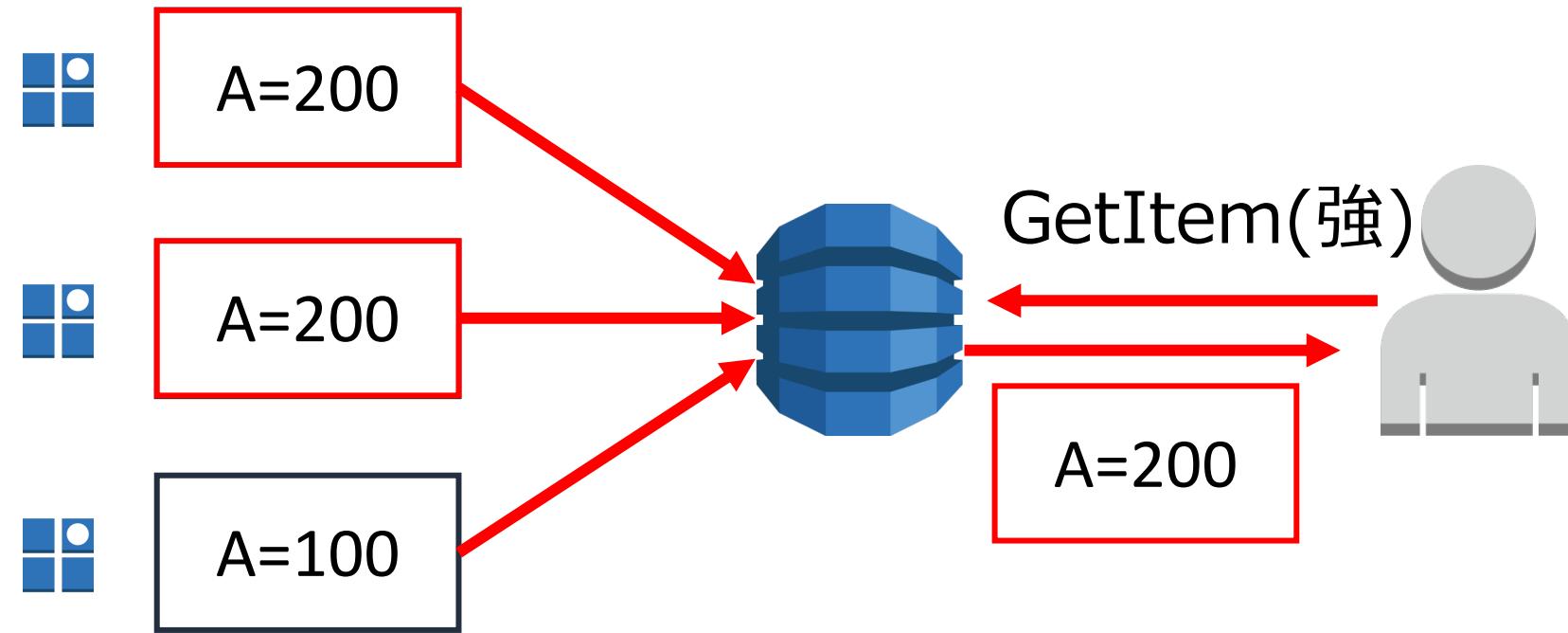
結果整合性の読み込み



結果整合性の読み込み



強力な整合性の読み込み



DynamoDB のユースケース

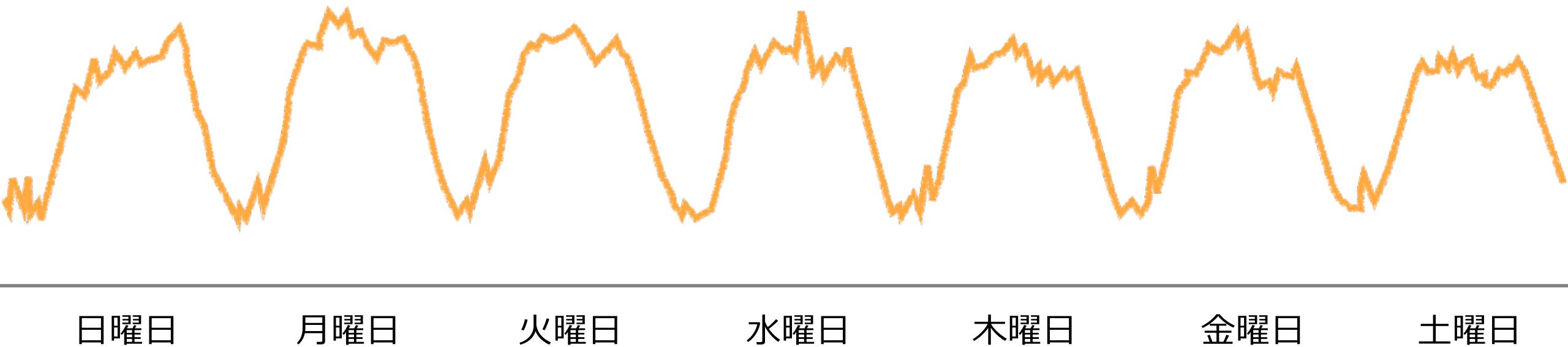
- プライムデー 2021 – トップチャートの 2 日間
 - <https://aws.amazon.com/jp/blogs/news/prime-day-2021-two-chart-topping-days/>
 - 数兆の API コール、ピーク時には 1 秒あたり 8,920 万件のリクエストを処理



モジュール 7
モニタリングとスケーリング

例: Amazon.com

プロビジョニングされたキャパシティー



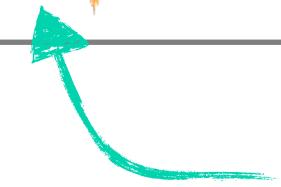
Amazon.com の 11 月のトラフィック

プロビジョニングされたキャパシティー

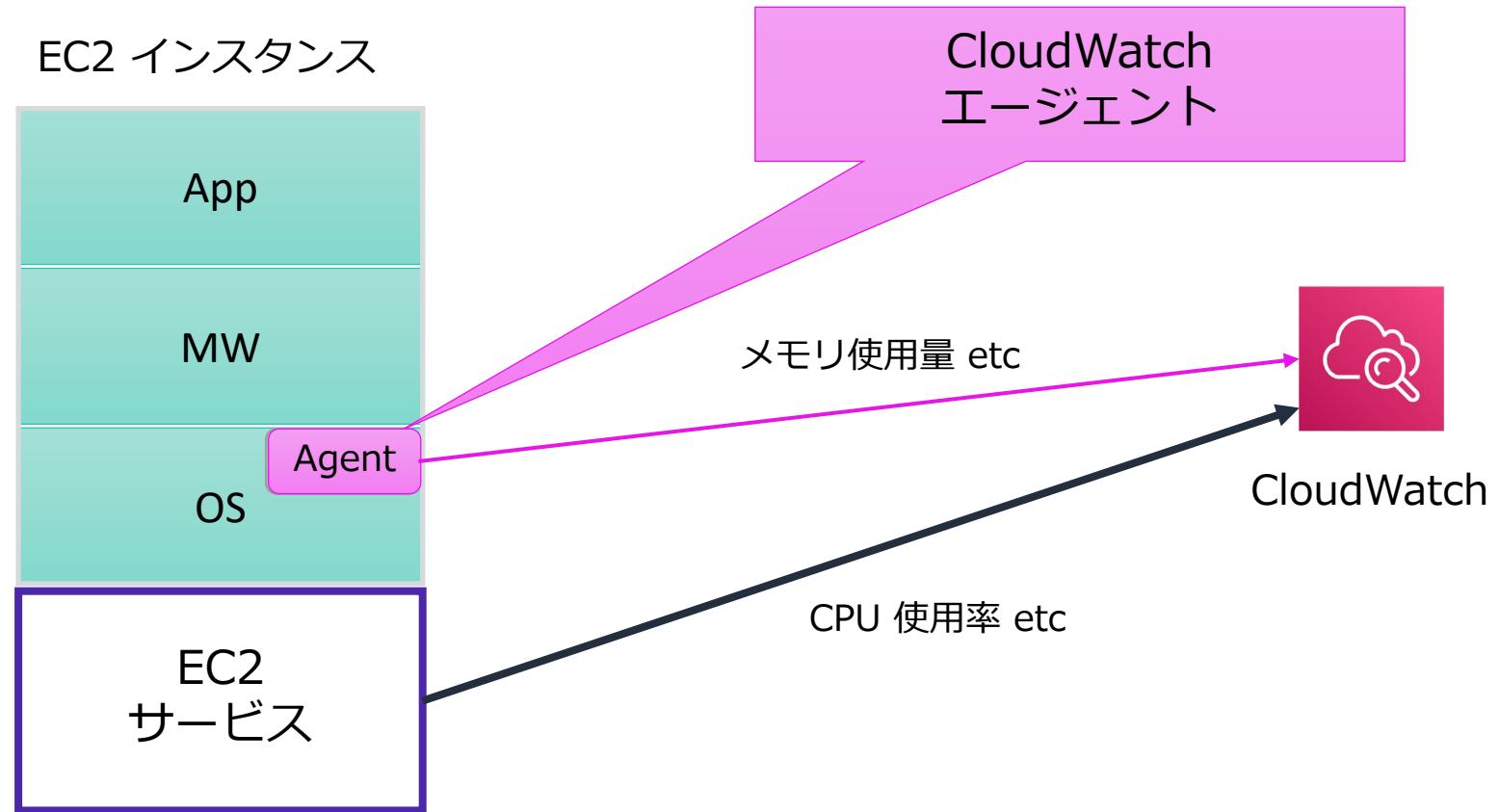
76%

課題は、今後必要になるコンピューティング性能を
効率的に推測すること

24%



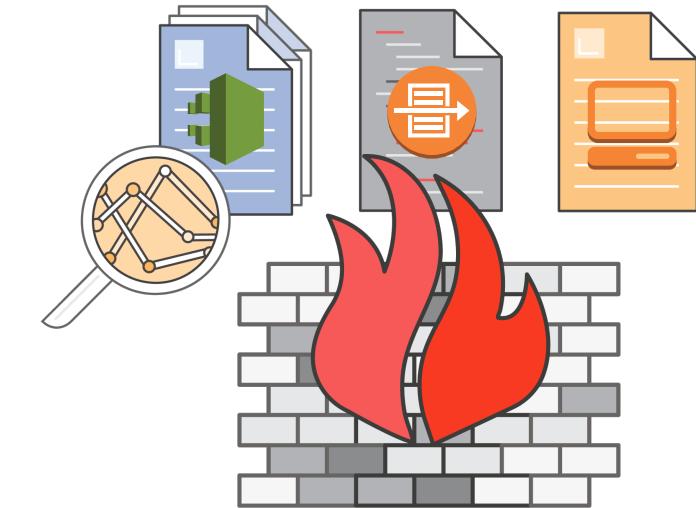
EC2 メトリクスの収集



https://docs.aws.amazon.com/ja_jp/AmazonCloudWatch/latest/monitoring/Install-CloudWatch-Agent.html

継続的なセキュリティ監視と脅威の検知を実現する Amazon GuardDuty

- CloudTrailやVPC Flow Logs、DNSのログ等のデータから疑わしいアクティビティを検知する
- GuardDutyはAWSが管理する基盤で動作し、エージェント等の導入は不要。性能影響もない
- サービスが検知したイベントは重要度に応じて3レベルにラベリングされ、推奨される対策とともに提示される
- 処理したログ量に応じた課金体系。30日の無料試用により実績量を測定できる
- 東京を含む各リージョンで利用可能に

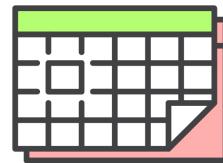


Current findings			
Actions		Saved filters	
Include and exclude filter options are available on certain finding attributes in the details			
	Finding	Last seen	Count
<input type="checkbox"/>	● [SAMPLE] Bitcoin-related domain queries from EC2 instance i-99999999 communicating with known Xor... ● [SAMPLE] EC2 instance i-99999999 communicating with known Xor... ● [SAMPLE] Bitcoin-related domain name queried by EC2 instance i-99... ● [SAMPLE] IAM User GeneratedFindingUserName logged into the AW... ● [SAMPLE] API GeneratedFindingAPIName was invoked from a Kali Li... ● [SAMPLE] Credentials for instance role GeneratedFindingUserName ... ● [SAMPLE] EC2 instance involved in RDP brute force attacks. ● [SAMPLE] Reconnaissance API GeneratedFindingAPIName was invo... ● [SAMPLE] Blackholed domain name queried by EC2 instance i-99999... ● [SAMPLE] API GeneratedFindingAPIName was invoked from a known... ● [SAMPLE] Unusual EC2 instance i-99999999 type launched	2017-11-09 16:00:04 (9 days ago) 2017-11-09 16:00:04 (9 days ago)	1 1 1 1 1 1 1 1 1 1
		Showing 59 of 59	26 31 2

自動スケーリングの方法

スケジュール

予測可能なワークフローに最適



時間や日に基づいて
スケーリングする

ユースケース: 夜間は開発インスタンスと
テストインスタンスをオフにする

動的

一般的なスケーリングに最適



Target Trackingを
サポート

ユースケース: CPU 使用率に基づく
スケーリング

予測的

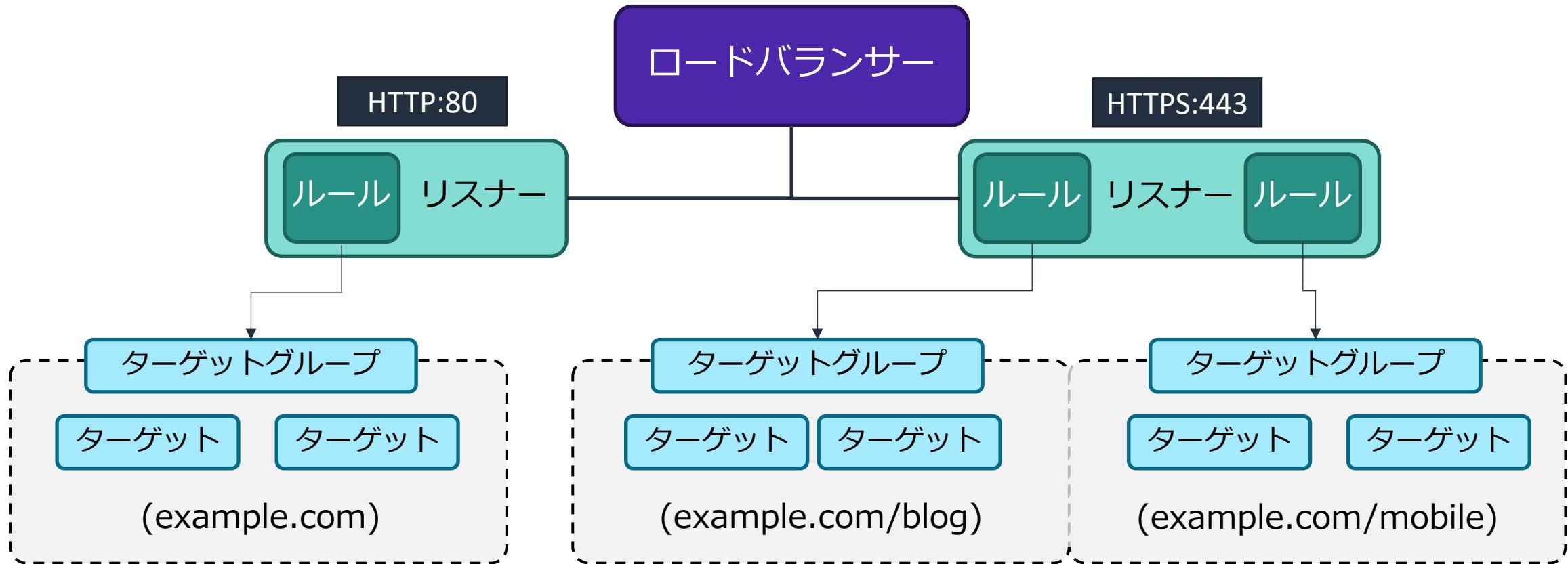
定期的なスパイク対応



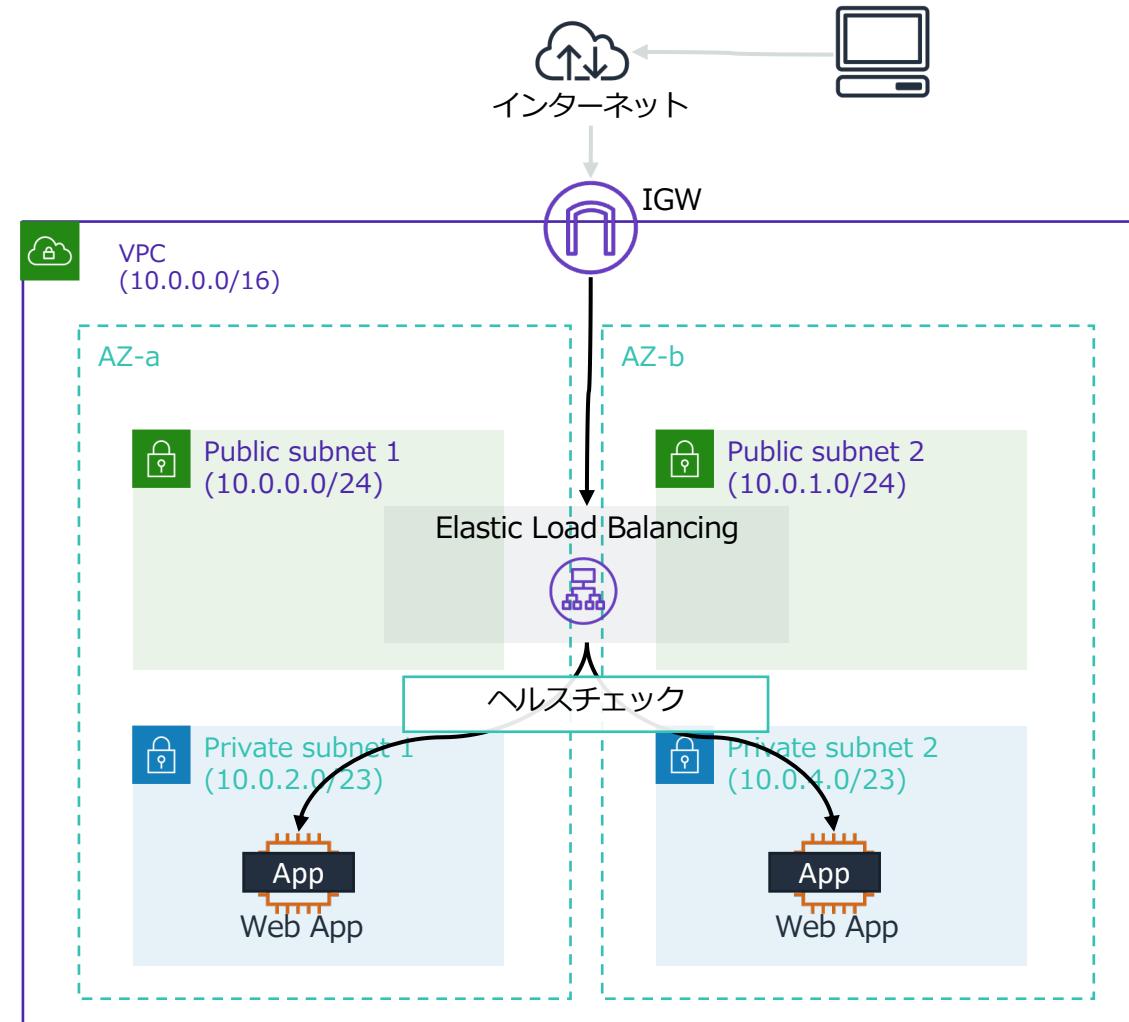
機械学習に基づく
スケーリング

ユースケース: ルールを
手動で調整する必要がなくなる

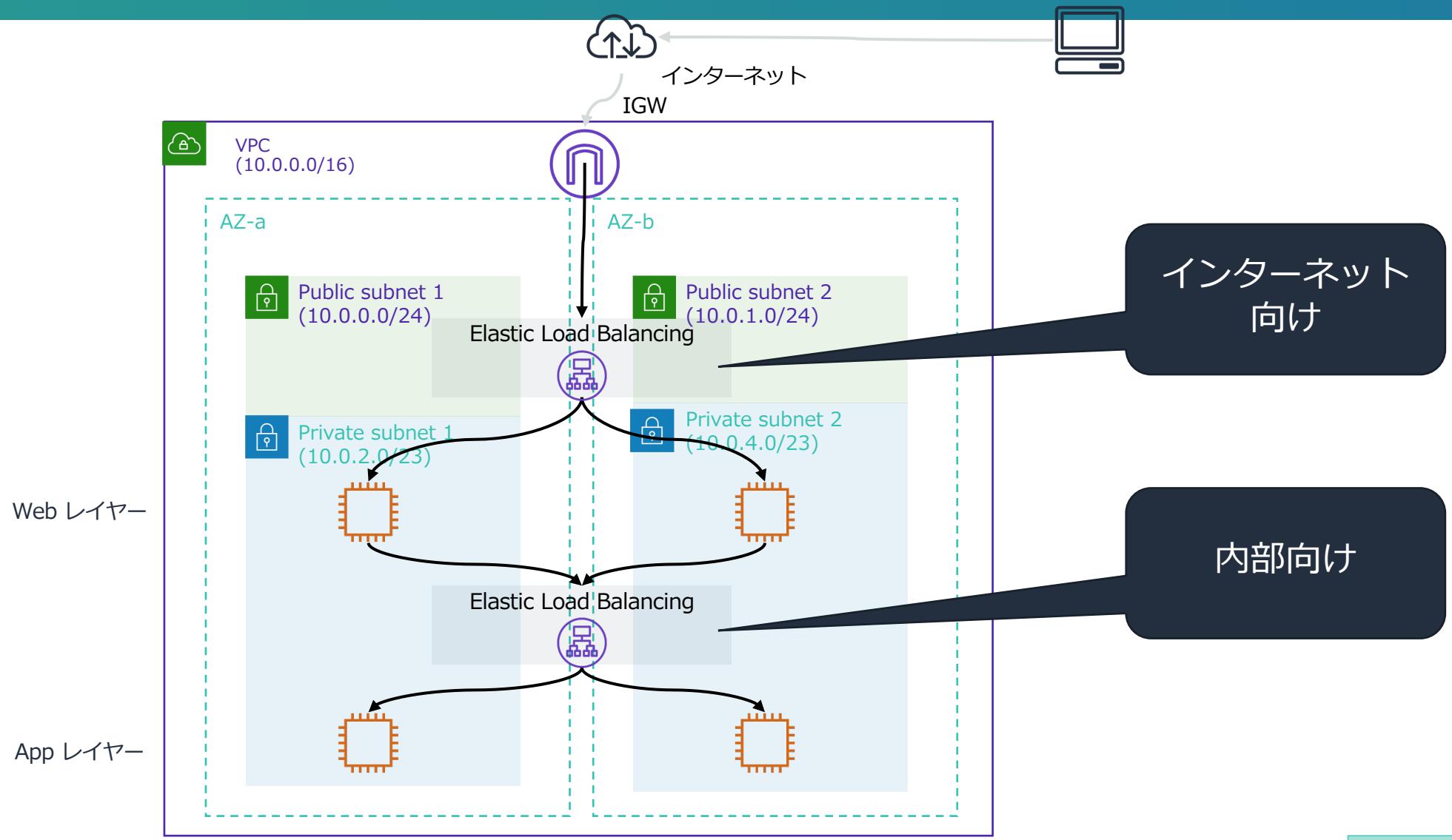
ロードバランサーのコンポーネント



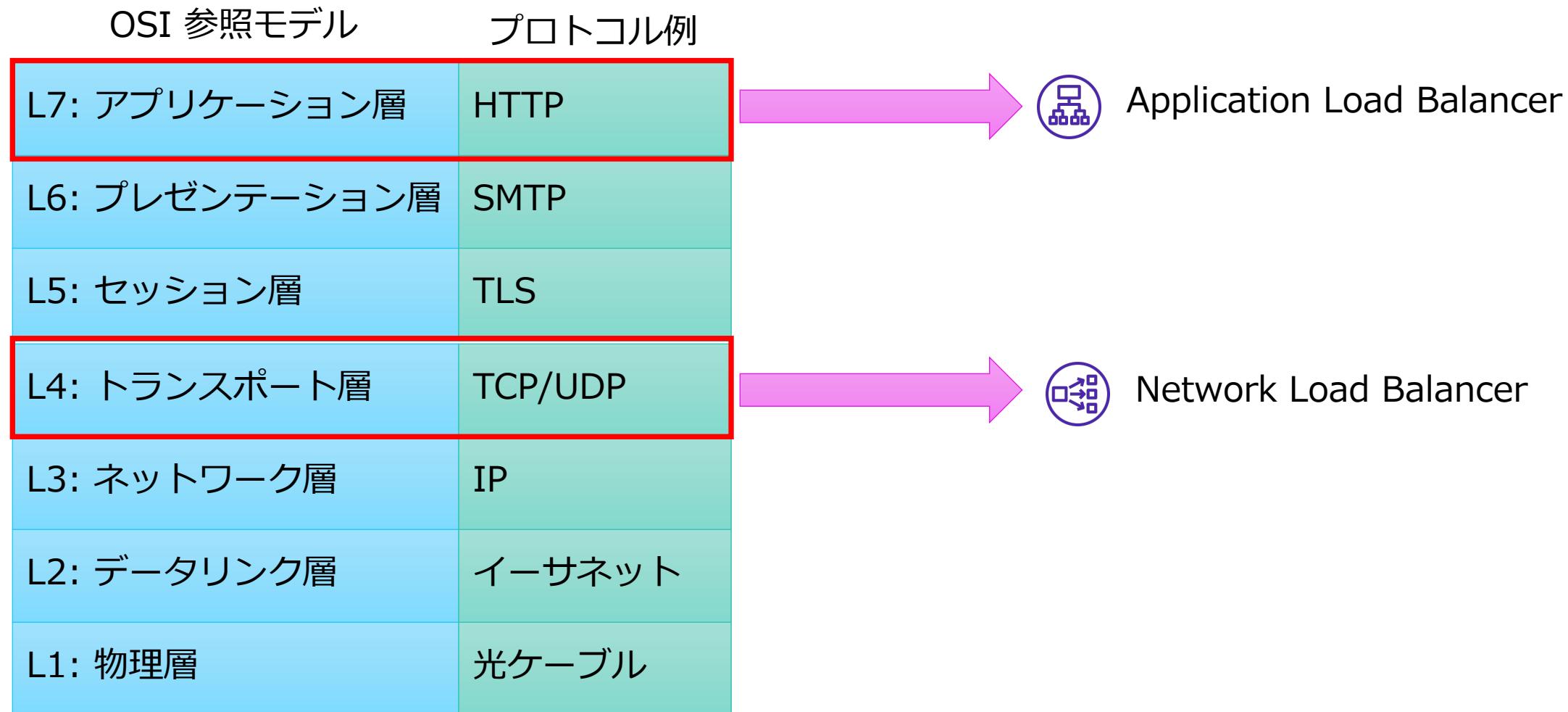
ヘルスチェック



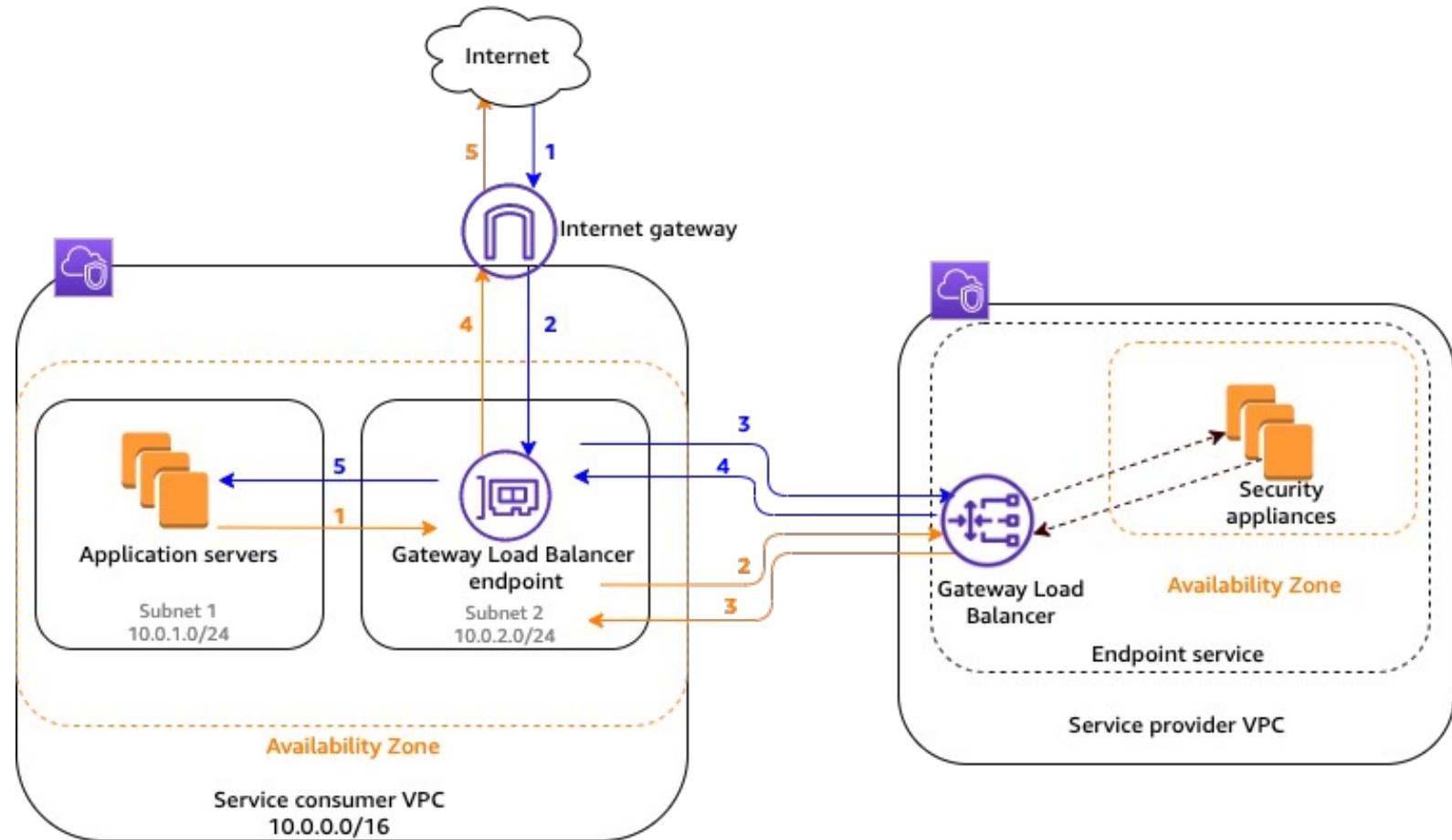
インターネット向け/内部向け



ロードバランサーの種類



Gateway Load Balancer



出典:

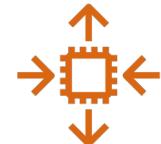
https://docs.aws.amazon.com/ja_jp/elasticloadbalancing/latest/gateway/getting-started.html#create-endpoint

ASG: 希望するキャパシティ

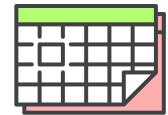
希望する容量が変更される契機



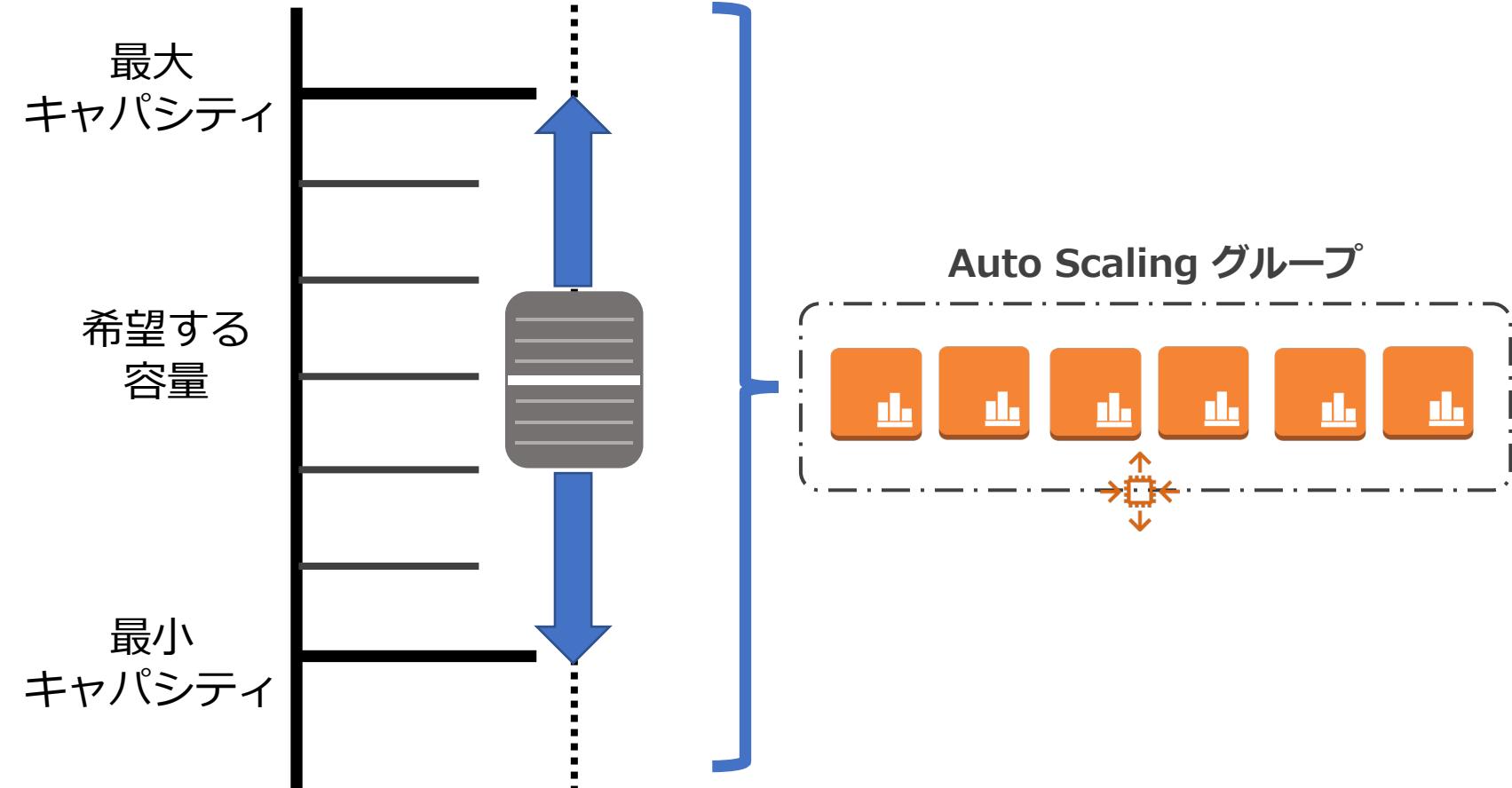
管理者が直接変更



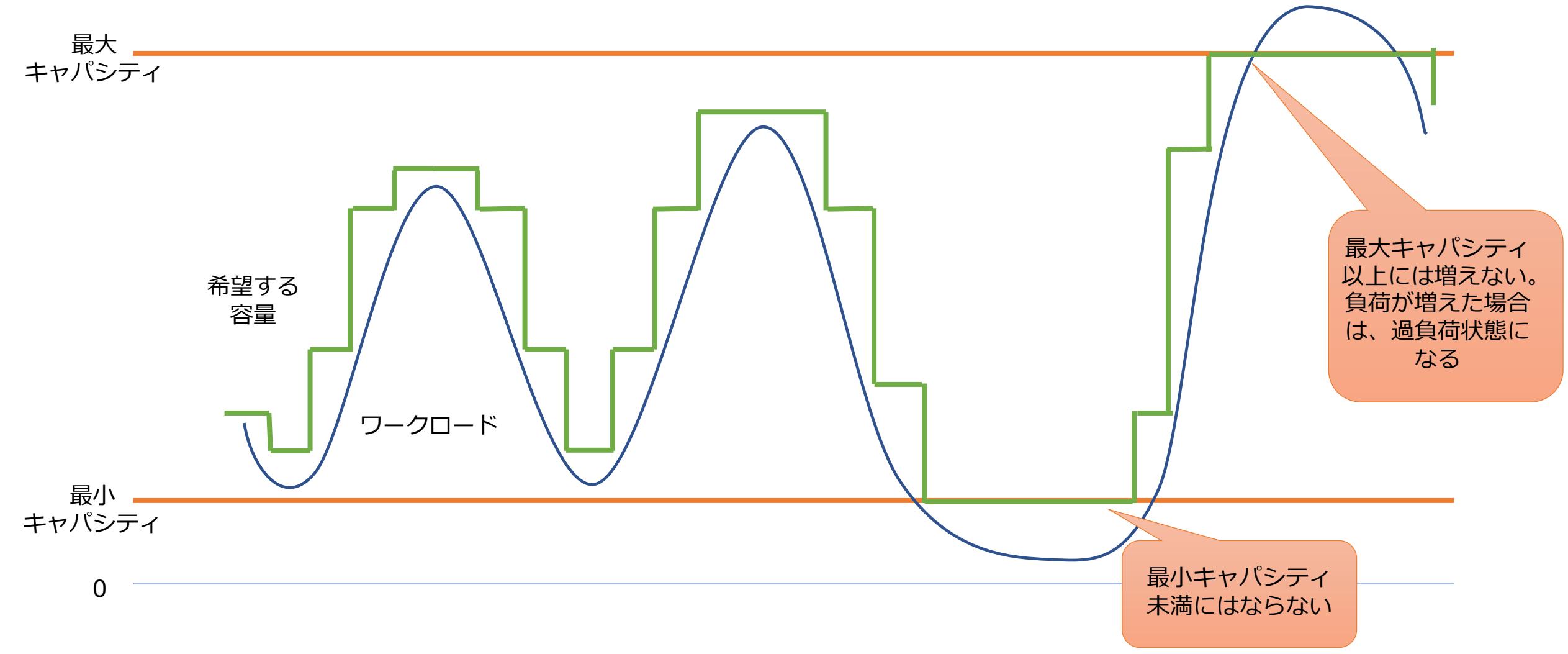
動的スケーリングにより変更



スケジュールにより変更



動的スケーリングの動作イメージ



モジュール 9

コンテナ

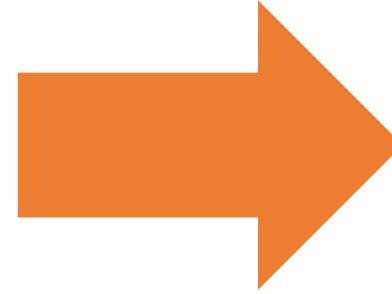
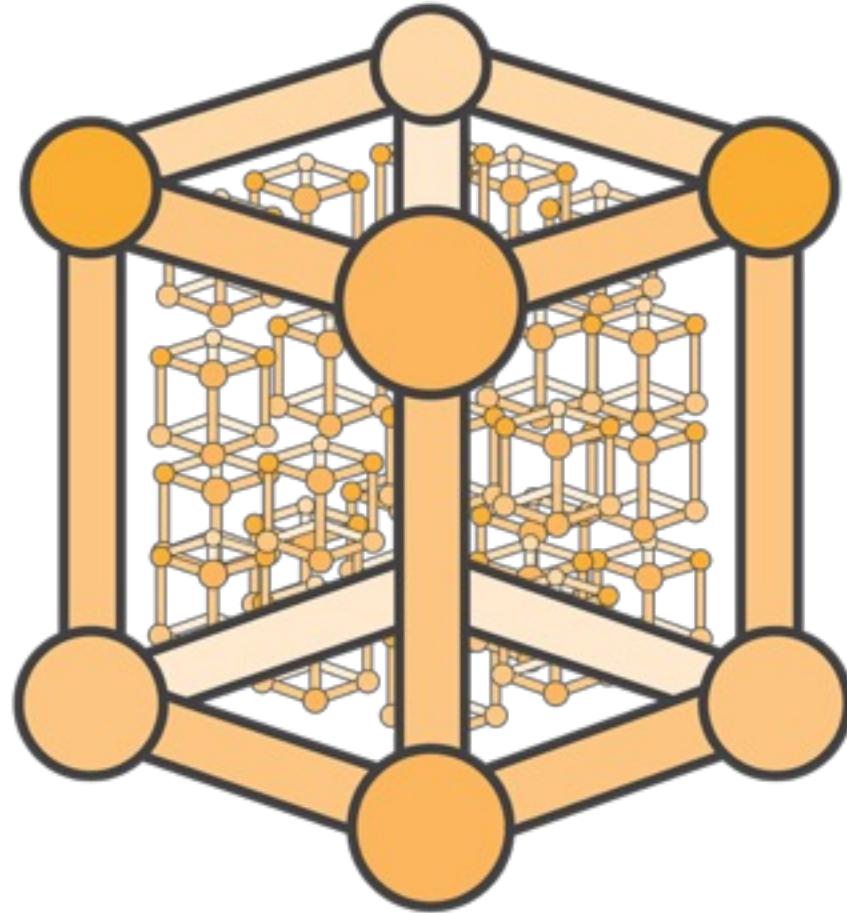
マイクロサービスが必要となる背景 (Amazon.com の事例から)



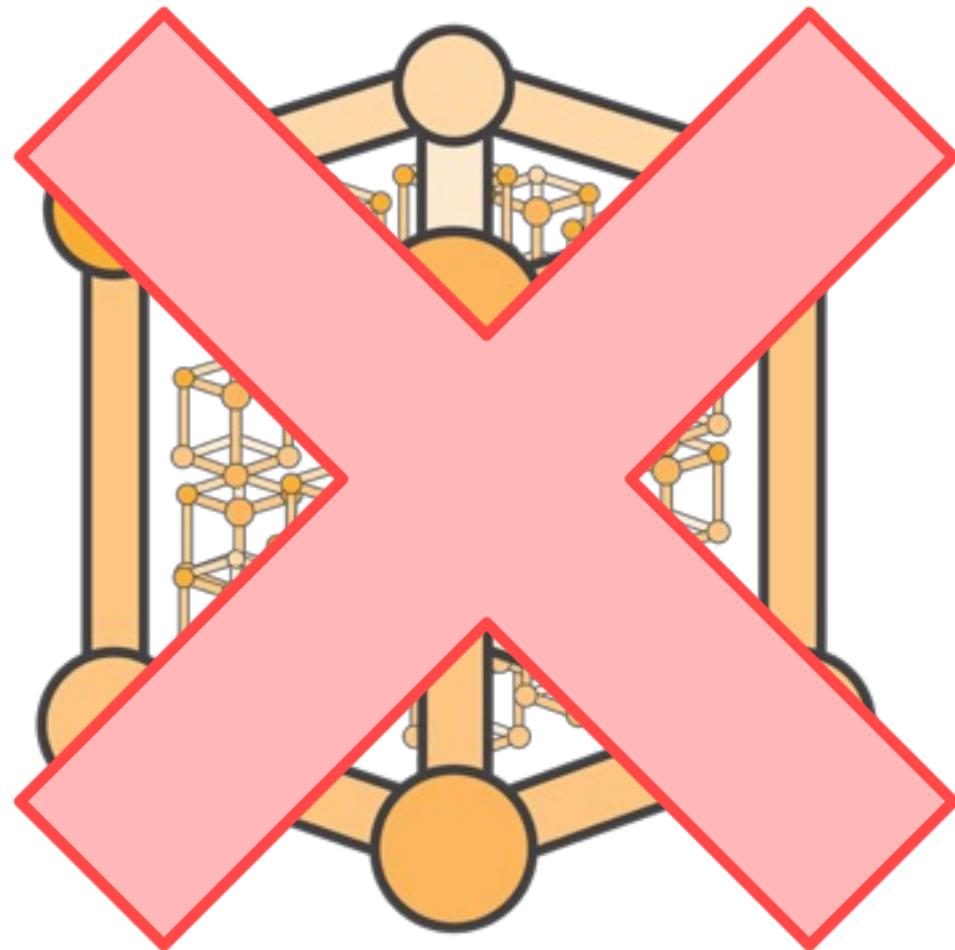
Amazon.com in 2001

補足

モノリシックアーキテクチャ

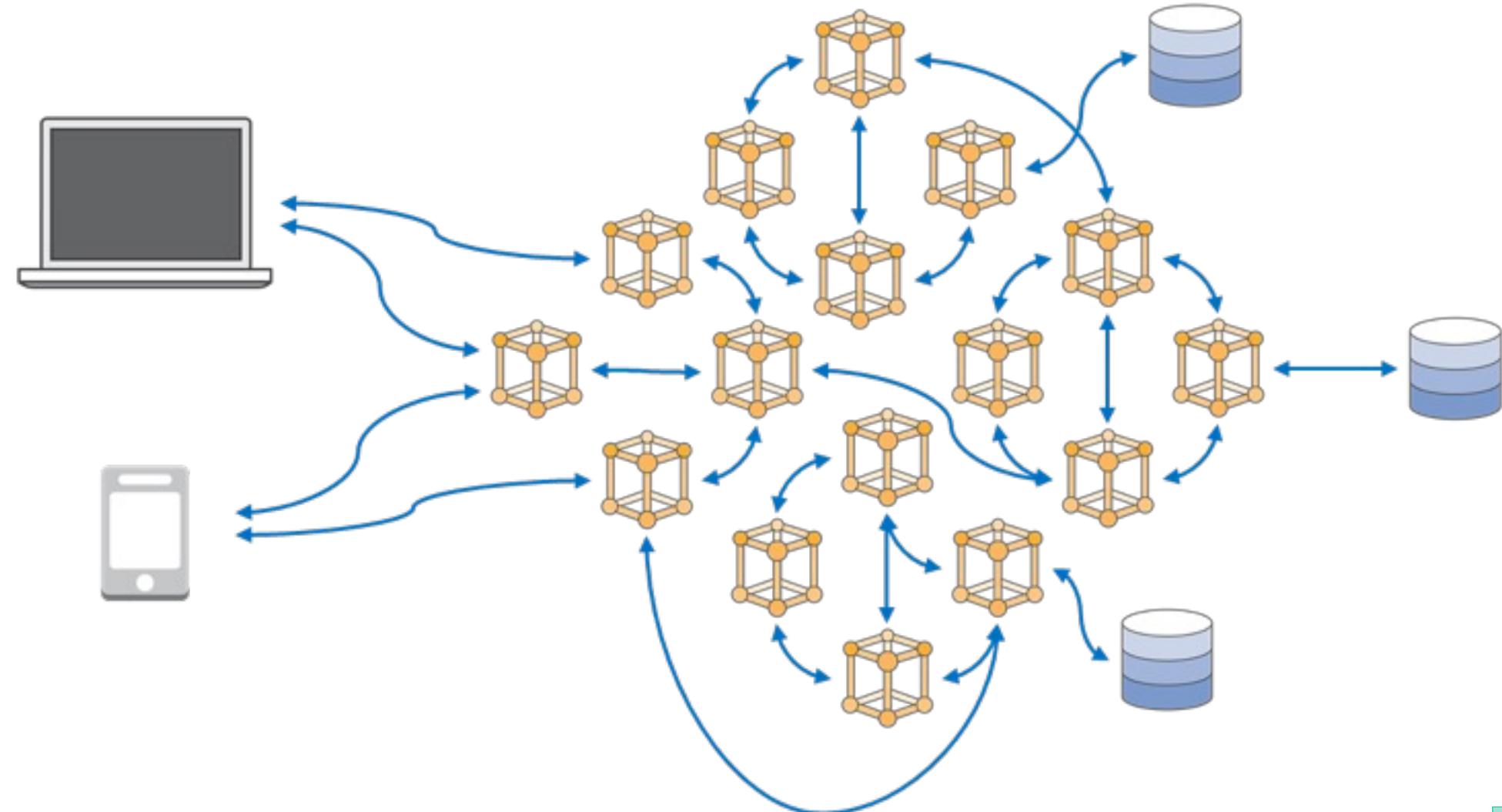


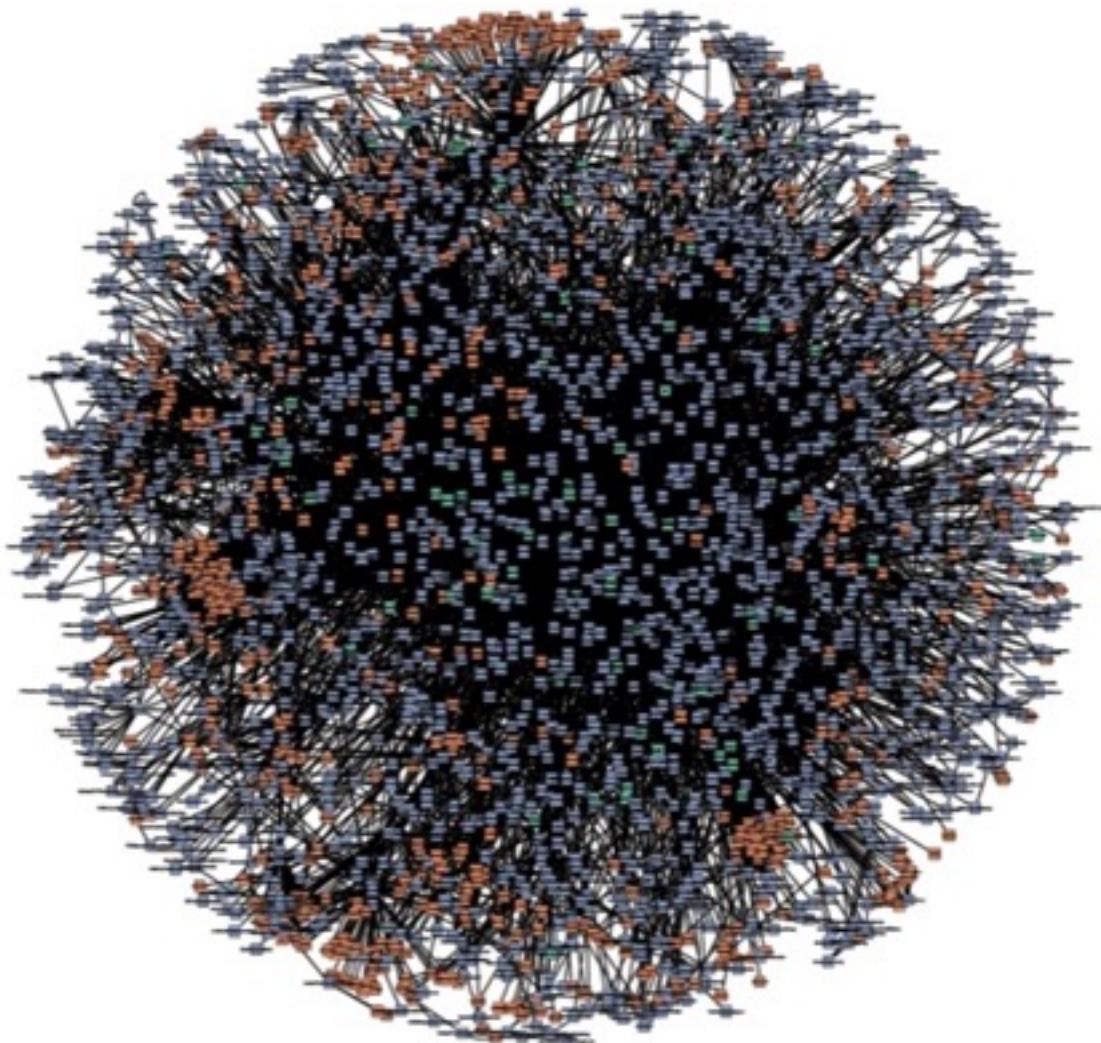
モノリシックアーキテクチャ → ある段階からの問題



- 密結合
- メンテナンスと維持が難しい
- ビルド・テストに時間がかかる
- デプロイがボトルネック
 - デプロイが一大イベント
- スケールが難しい
- 共同作業が難しい

マイクロサービスアーキテクチャ





- 単一の目的
- HTTPSのAPIでのみ連携
- お互いはブラックボックス
- “**Microservices**”



- ◆ Two-pizza teams
- ◆ 全ての責任と権限
- ◆ 良くしようとする動機
- ◆ “DevOps”

数千のチーム

× Microservices アーキテクチャ

× 繙続的デリバリ

× 複数の環境

= 5000万回/年のデプロイ

マイクロサービスの考慮点

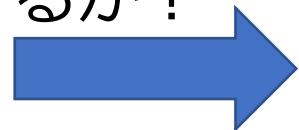
- 小粒のアプリケーションが大量に存在するようになる
 - 個々の異なるアプリケーションの構築/可用性/拡張性の考慮
 - 小さいアプリケーションの実行環境の分離

 コンテナ / コンテナオーケストレーション

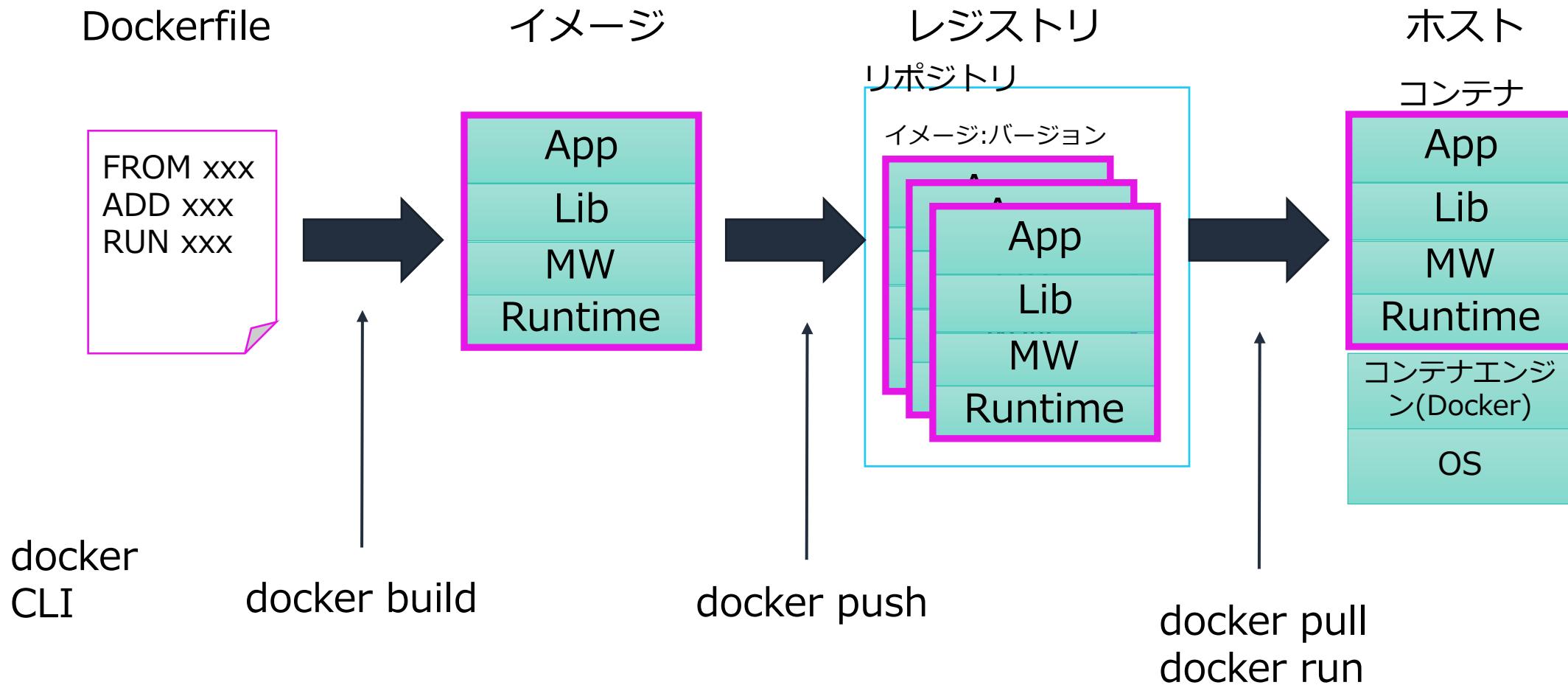
- 複数のサービスにまたがって処理が行われる
 - 障害時のエラーの特定や分析をどのように素早く行うか？

 AWS X-Ray

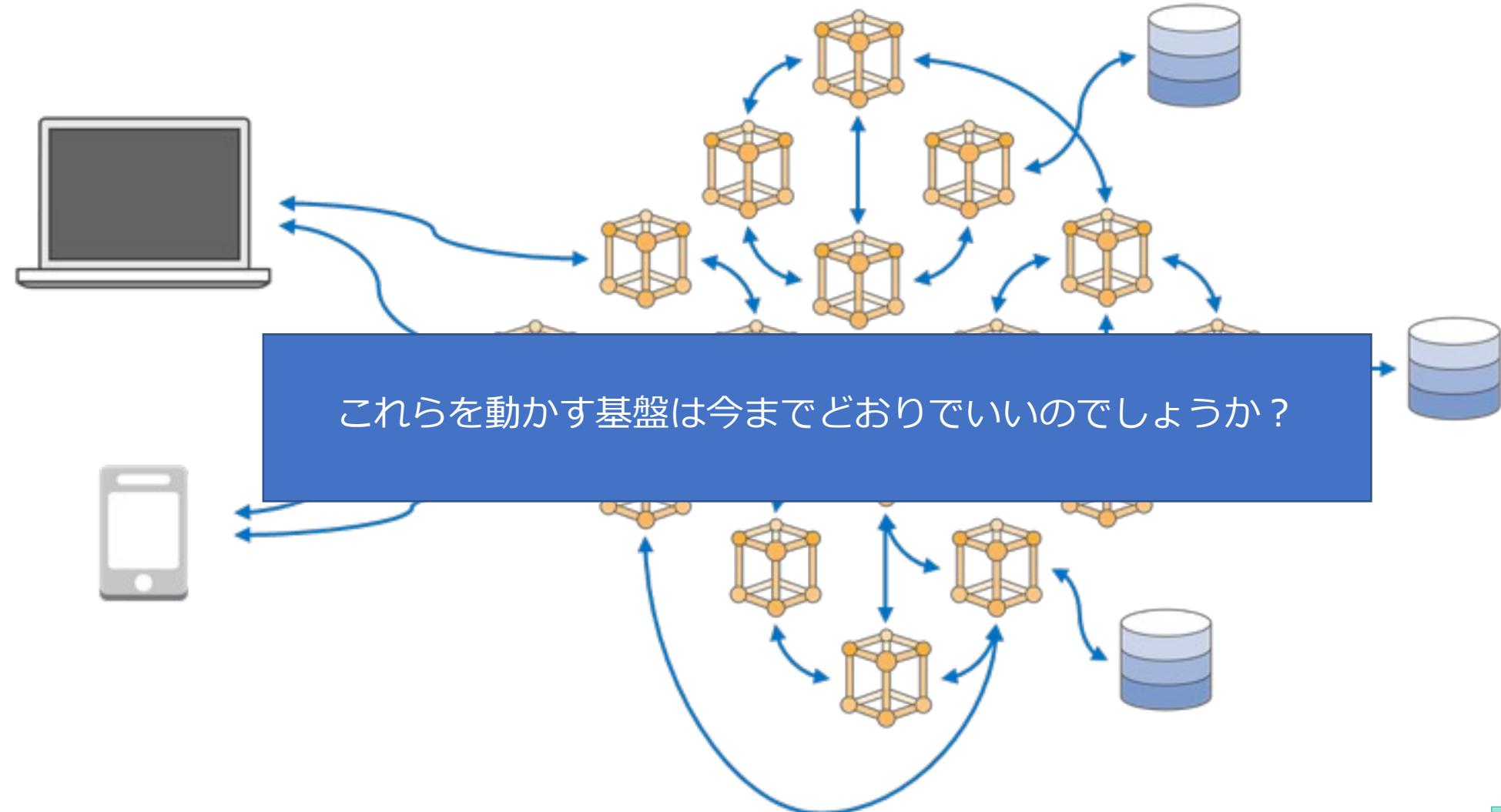
- サービス間で連携時の横断的関心事(Cross-cutting concerns)をどう実装するか？

 AWS App Mesh

Docker コンポーネントと用語



マイクロサービスアーキテクチャ



モジュール 10

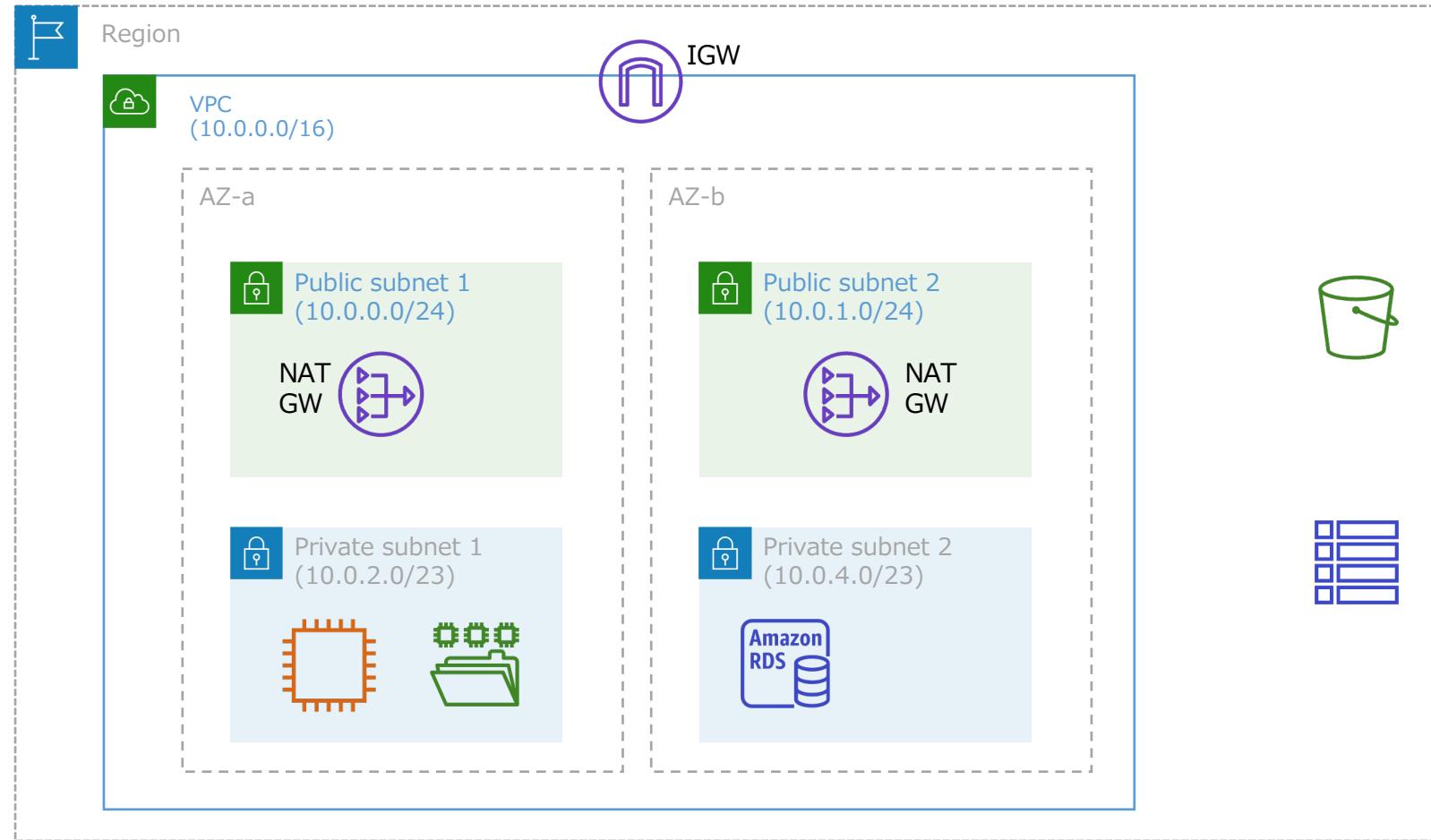
ネットワーク 2

モジュールの目標

前提: 複数ネットワークを相互に繋げる場合

- ネットワークの安全性を保つには?
- サービスを相互に接続するには?(VPC 内と外のサービス間の通信)
- ハイブリッドネットワークの設定に必要なオプションは?
- VPC の数が急増したときにトラフィックとセキュリティを管理するには?
- DNS 解決に使用できるものは?(オンプレミスとAWS での相互名前解決)

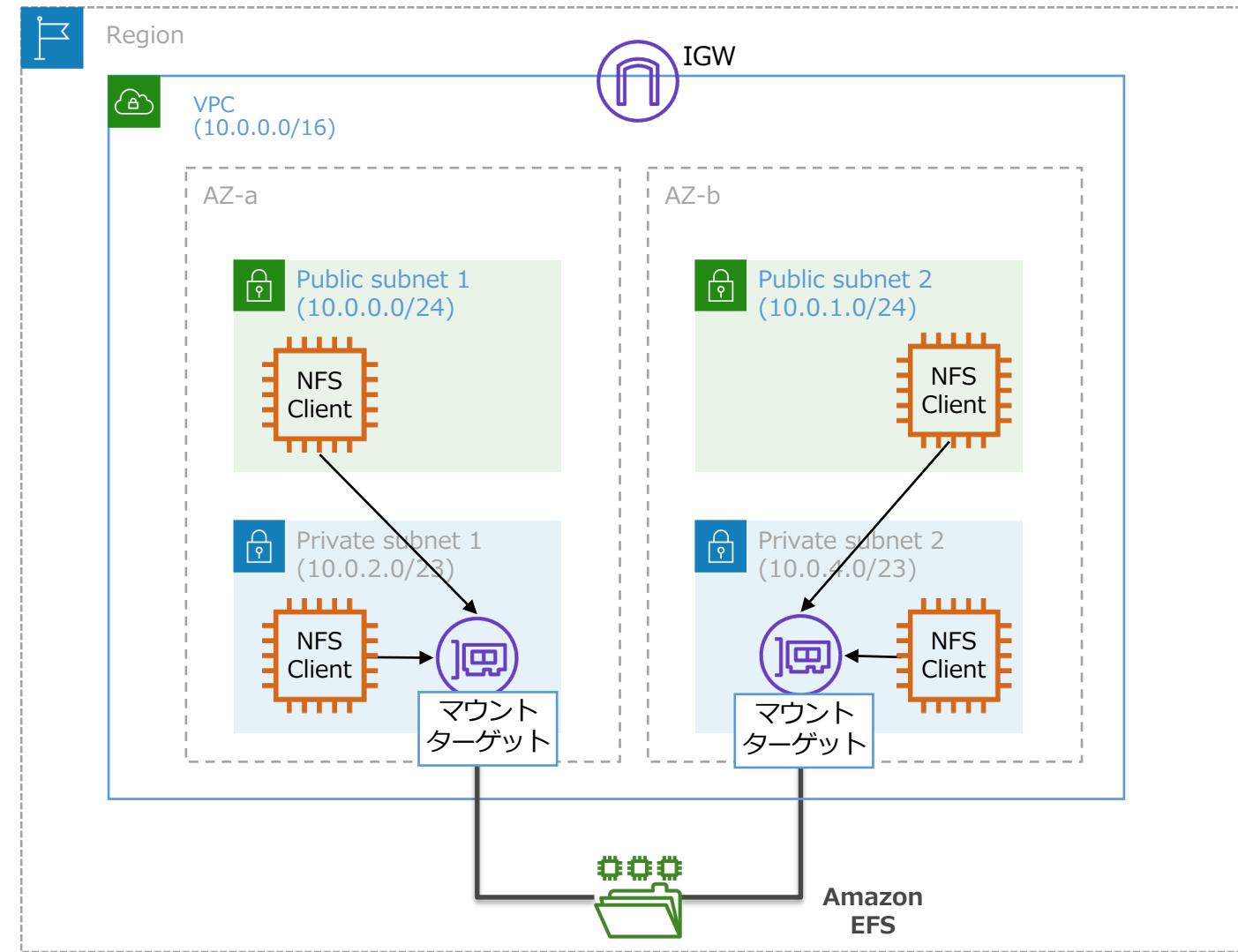
VPC の内か外か？



VPC の内か外か？

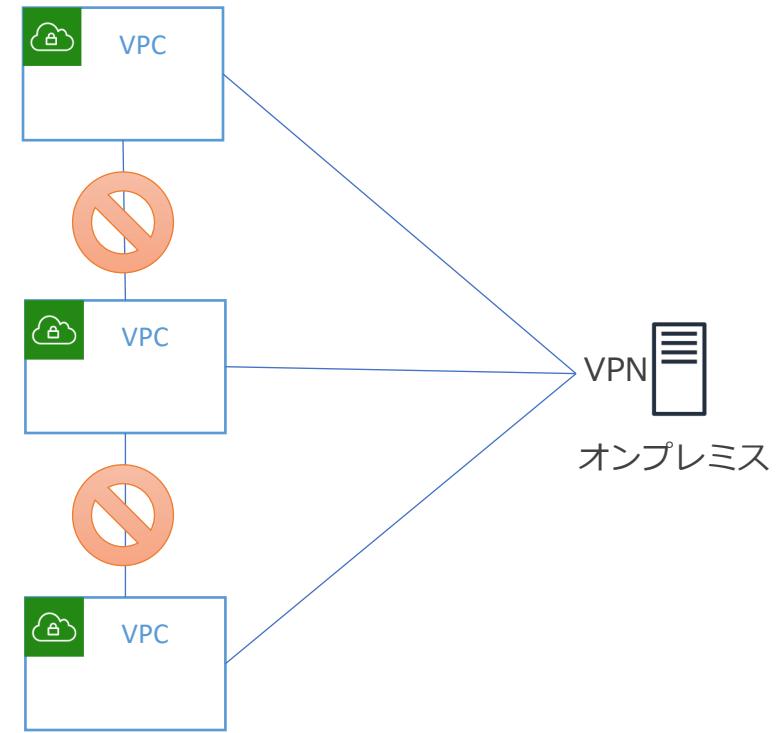
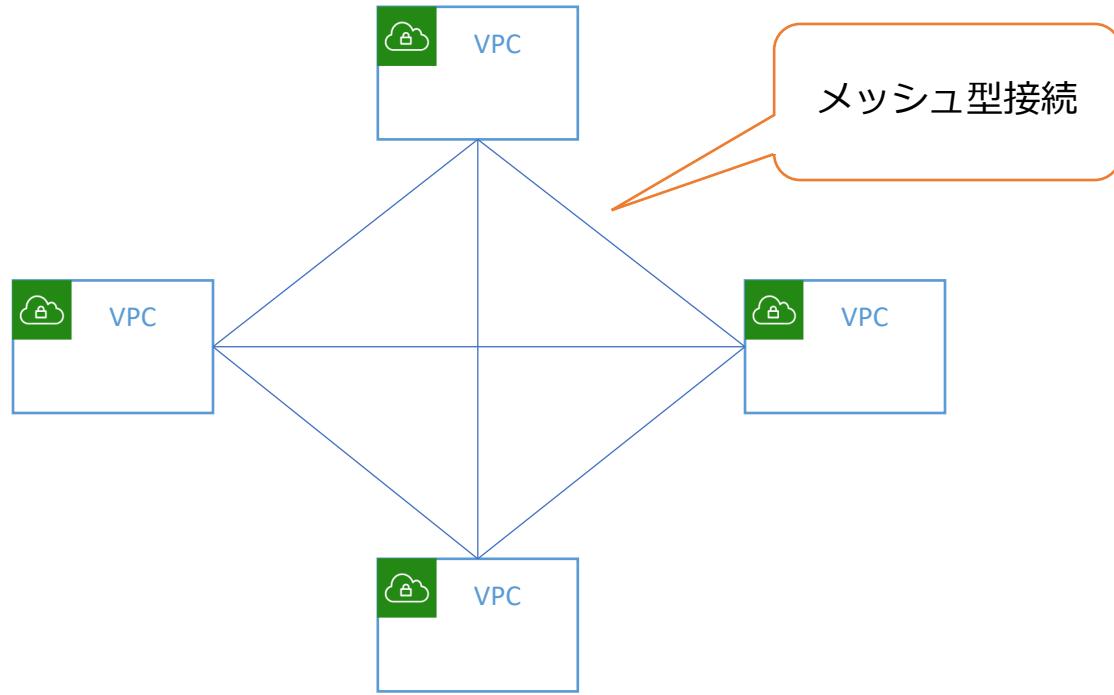
- EC2 インスタンス
- S3 バケット
- RDS インスタンス
- DynamoDB テーブル
- EFS ファイルシステム

Amazon EFS のアーキテクチャ



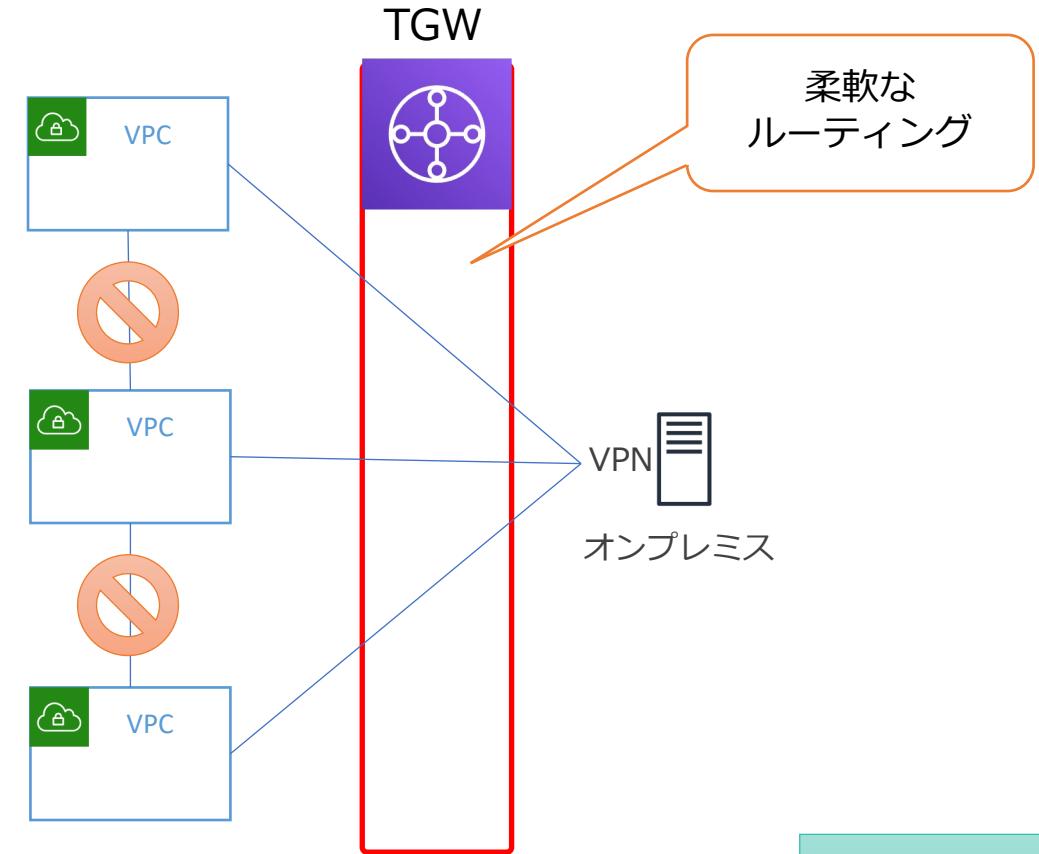
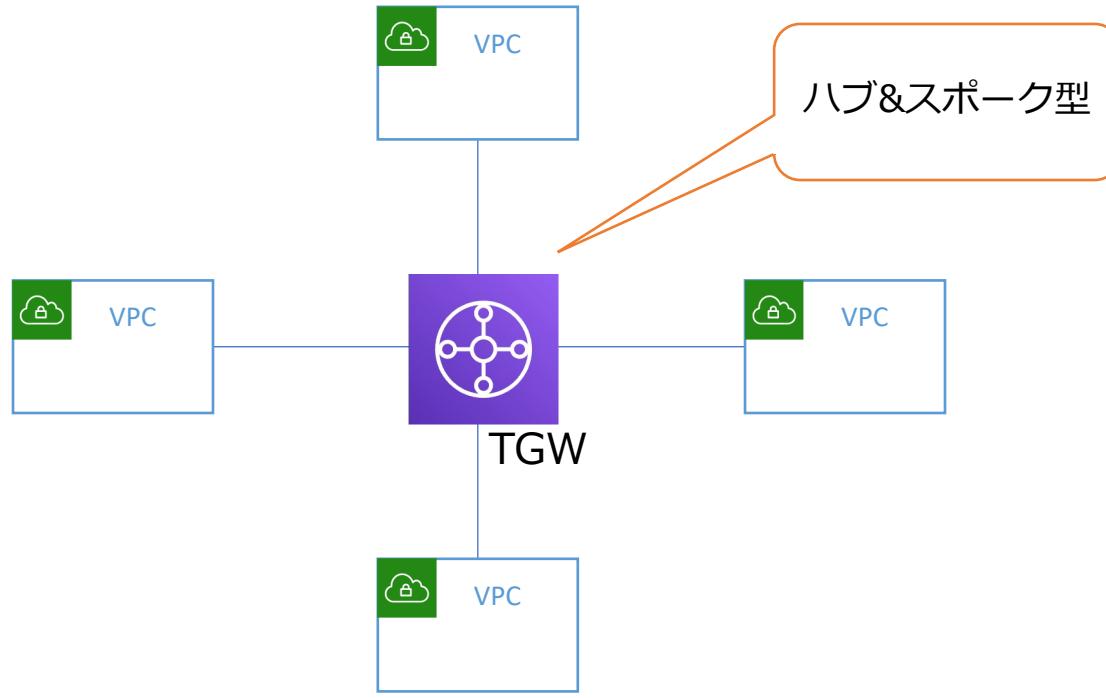
VPC ピア接続の考慮点

- 多数のVPC 間を相互につなぐ
- VPC 接続を柔軟にコントロールしたい



Transit Gateway(TGW)による解決

- 多数のVPC 間を相互につなぐ
- VPC 接続を柔軟にコントロールしたい



モジュール 11

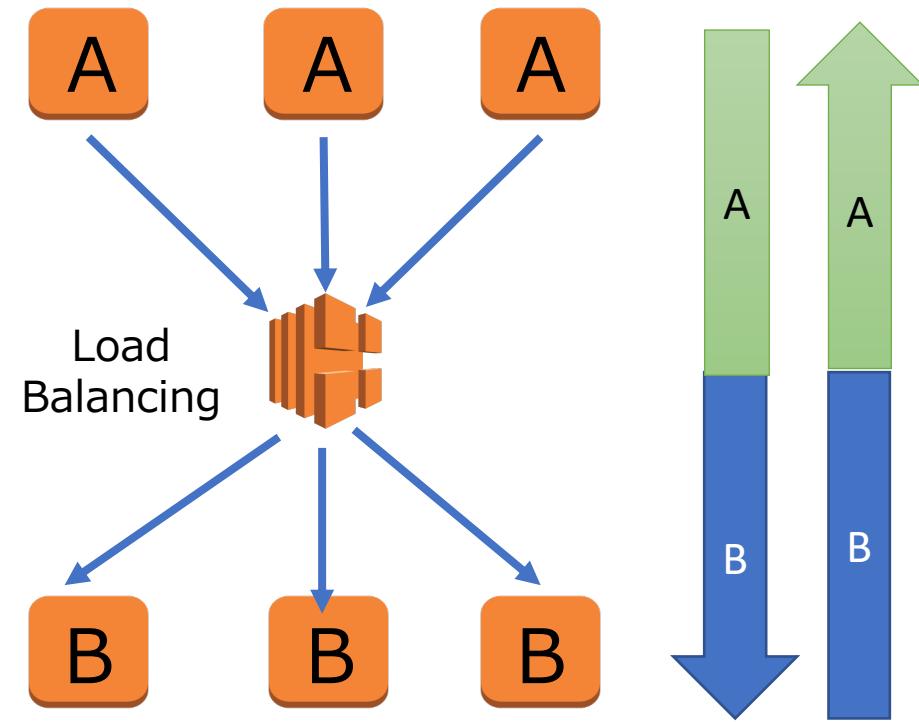
サーバーレス

モジュールの目標

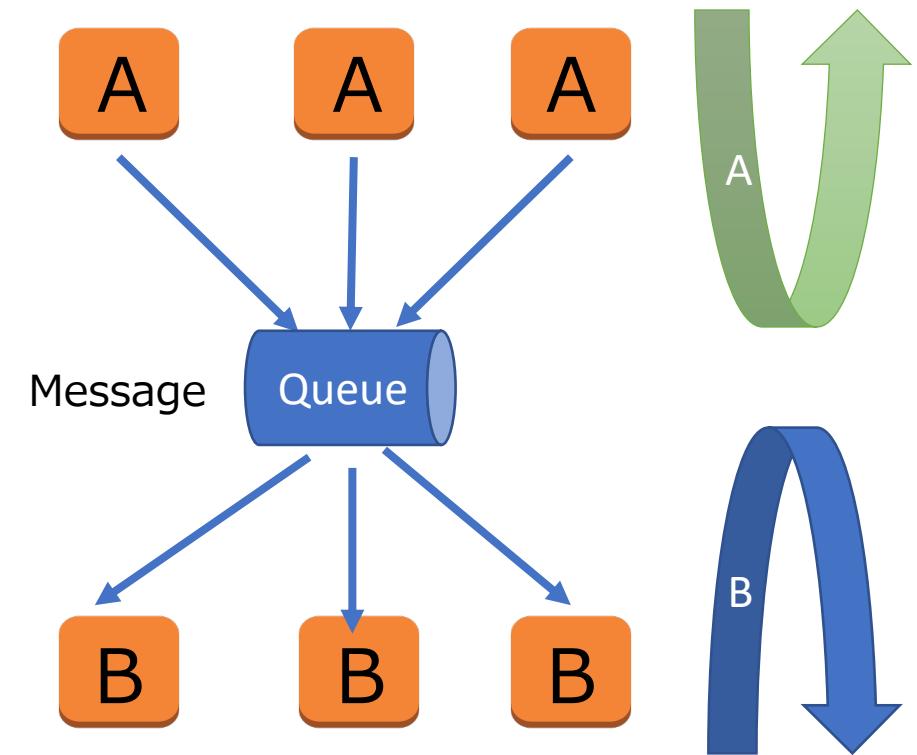
- アプリケーションを AWS のサービスに接続するにはどうすればよいですか
- アプリケーションにテキストメッセージを送信させるにはどうすればよいですか
- イベント駆動型アーキテクチャを構築するにはどうすればよいですか
- 大量のストリーミングデータをほぼリアルタイムで処理するにはどうすればよいですか
- 複数ステップのワークフロー内で API コールをどのように調整すればよいですか

同期処理と非同期処理

同期処理



非同期処理



モジュール 12

エッジサービス

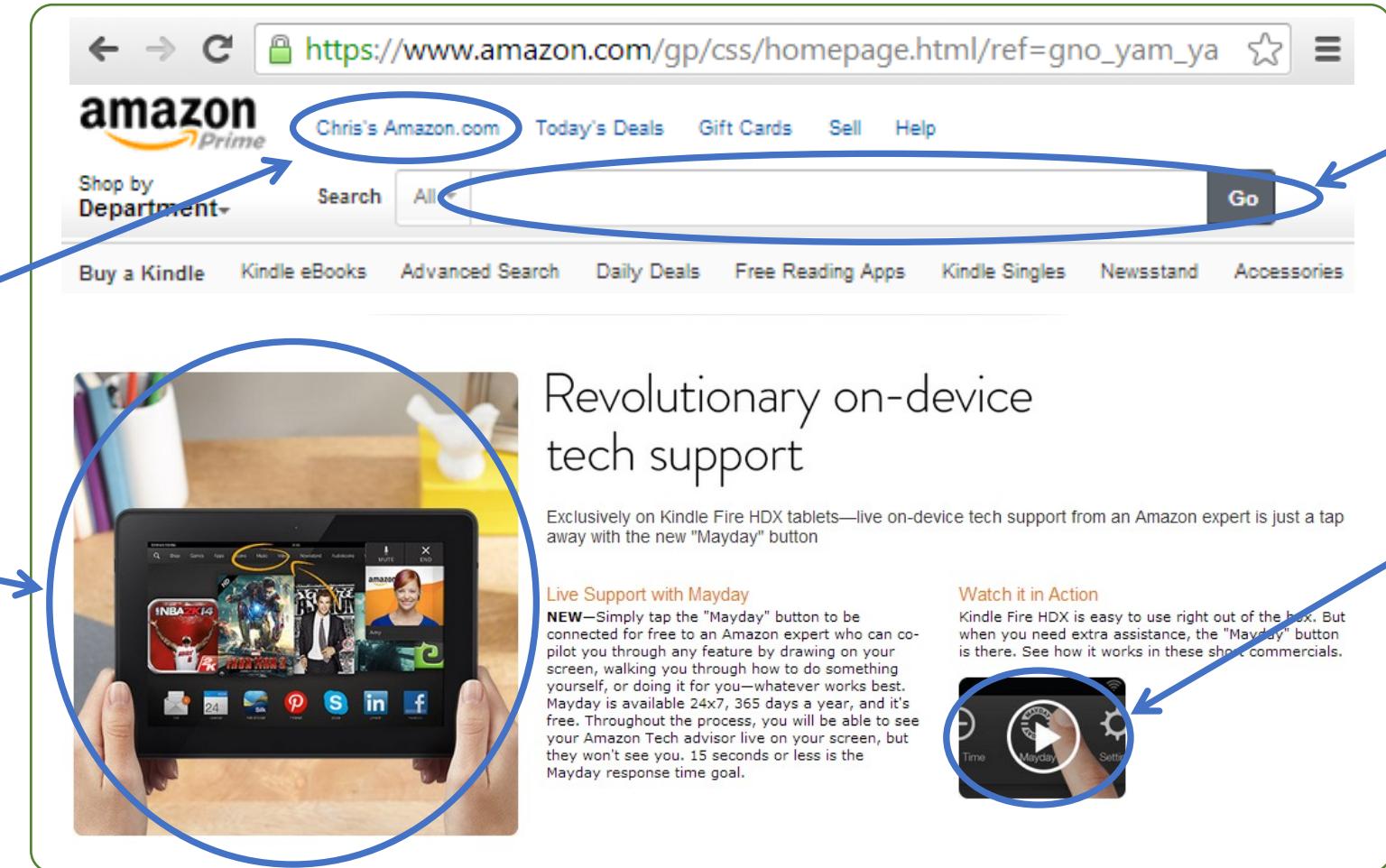
キャッシュできるコンテンツのタイプ

動的
(TTL が 0)

静的

ユーザーの
入力項目

動画



まとめ

復習のキヤップストーンプロジェクト

- Online Course Supplement: Architecting on AWS (Japanese)
 - <https://explore.skillbuilder.aws/learn/course/external/view/elearning/11372/architecting-on-aws-online-course-supplement-japanese>
- キヤップストーンを実施する場合は、ご自身でAWS アカウントを用意して実施してください

お疲れ様でした