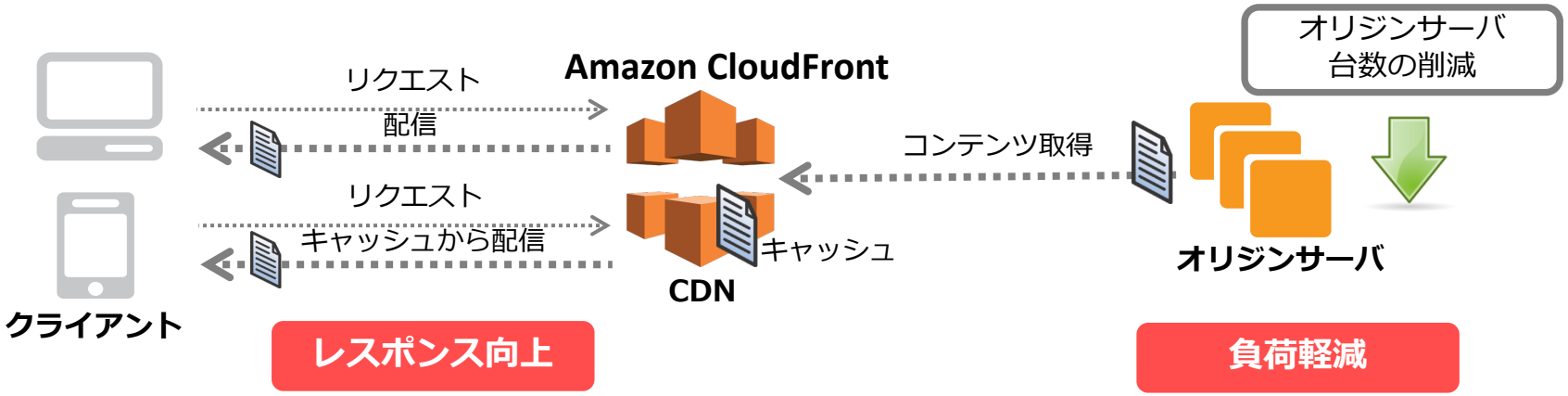
ワーク

- **以下のアーキテクチャのメリット・デメリットには何があると思いますか？**

  - (1) ひとつの AZ にサーバーを集約する、シングル AZ 構成
  - (2) 複数の AZ にサーバーを分散する、マルチ AZ 構成
  - (3) 別のリージョンにもサーバーを配置する、マルチリージョン構成

Contents Delivery Network

- 大規模なアクセスでも世界中にあるエッジロケーションを活用して効率的かつ高速にコンテツ配信が可能なサービス
  - ユーザからのアクセスを最も近いエッジロケーションに誘導することで**ユーザへの配信を高速化**
  - エッジロケーションでは、コンテンツのキャッシングを行い、**オリジンに負荷をかけず効率的に配信**



投影のみ

モジュール 1 のまとめ

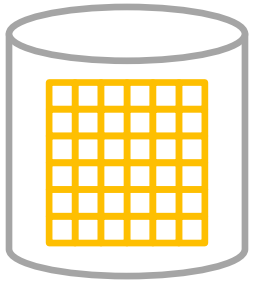
- **Well-Architected フレームワーク (W-A)**
  - AWS の設計原則と質問から構成されるベストプラクティス集
  - 5つの柱の観点からアーキテクチャを考えることが大事

- **AWS のグローバルインフラストラクチャ**
  - AZ … DC群 (近すぎず、遠すぎず)
  - リージョン … AZ群、地理的ロケーション
  - エッジロケーション … ユーザーに近い低レイテンシーでサービスを提供する場所
    - サービス: Route53 (DNS), CloudFront (CDN) etc.

ストレージの種類

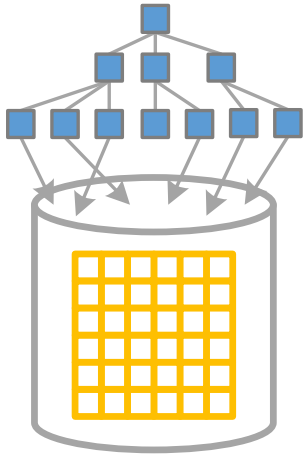
aws training and certification

ブロックストレージ



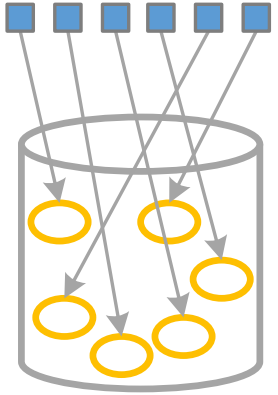
- データは単一または複数のディスクにブロックで格納される
- インスタンスにローカルアタッチされるストレージ

ファイルストレージ



- データはディレクトリ階層に ファイルとして格納される
- ネットワーク越しに共有され るストレージ

オブジェクトストレージ



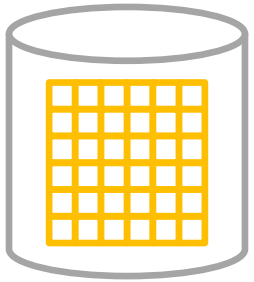
- データはフラットな空間にオブジェクトとして格納され、 キーで識別される
- キー・ベースでシンプルに GET と PUT で API アクセスするストレージ

投影のみ

ストレージの種類

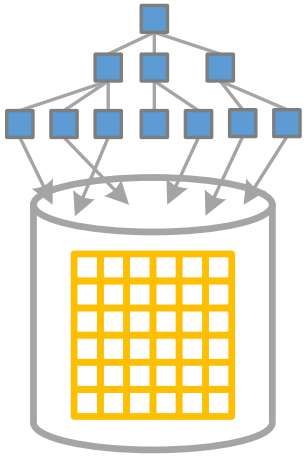
aws training and certification

ブロックストレージ



EBS

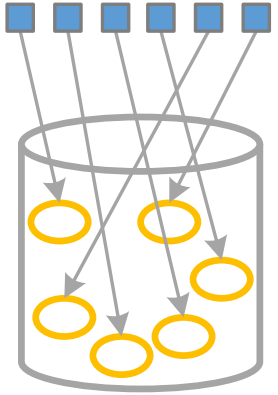
ファイルストレージ



EFS

FSx for Windows

オブジェクトストレージ



S3

S3 Glacier

投影のみ

モジュール 2 のまとめ（1/2）

- **S3 の特徴とユースケース**
  - オンライン　・・・　コンテンツ配信、ウェブサイト
  - 容量無制限　・・・　データ置き場
  - 高い耐久性　・・・　バックアップ

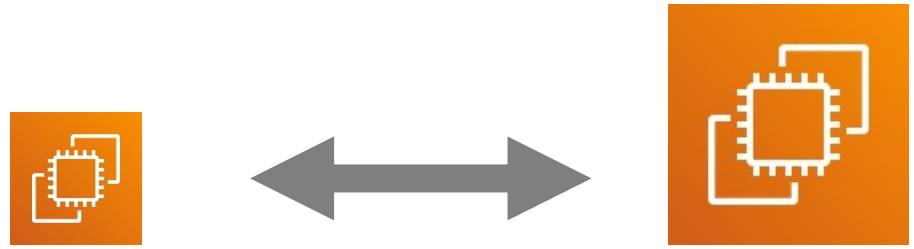
- **S3 を使いこなすためのキーワード**
  - セキュリティを高めたい　・・・　バケットポリシー
  - コストを最適化したい　・・・　ストレージクラス、S3 Glacier
  - 管理を楽にしたい　・・・　バージョニング、ライフサイクル管理、Access Points
  - パフォーマンスを高めたい　・・・　マルチパートアップロード、Transfer Acceleration

モジュール 2 のまとめ（2/2）

- **リージョンの選択基準**
  - 法令　・・・　国外に持ち出していいデータか？
  - レイテンシ　・・・　ユーザーに近い位置にサーバーがある方が遅延が少ない
  - 対応サービス　・・・　新しいサービスは少しずつ公開されていく
  - コスト　・・・　価格設定はリージョン毎に異なっている

デモ：Amazon EC2

- **デモ構成**
  - EC2 １台
- **デモ内容**
  - EC2のインスタンスタイプを変更する



t2.micro

vCPU: 1
Memory: 1GB

t2.large

vCPU: 2
Memory: 8GB

投影のみ

ワーク

- **インスタンスタイプ毎の料金を調べてみよう！**
  - 「東京リージョン」の「オンデマンド料金」「Linux」で計算
  - それぞれの値を見比べてインスタンスファミリーの特徴を考察

インスタンスファミリー		vCPU 数	1vCPU の料金	メモリ (GiB)	1GiB の料金
汎用	m5.large				
コンピューティング最適化	c5.large				
メモリ最適化	r5.large				

モジュール 3 のまとめ

- **EC2**
  - 短時間で取得・起動できる、サイズ変更可能な仮想サーバー
  - カスタマイズのキーワード　・・・　AMI、ユーザーデータ
  - ストレージの選択肢　・・・　EBS、EFS

- **最適化のポイント**
  - パフォーマンス　・・・　インスタンスタイプ、Compute Optimizer
  - コスト　・・・　購入オプション（RI、Savings Plans、スポットインスタンス）
  - コンプライアンス　・・・　専有オプション（専有インスタンス、専有ホスト）

マネージド機能の一例

- **自動バックアップ**
  - 1 日 1 回、自動で取得
  - 最大 35 日間の保持期間を選択可能
  - 5 分毎にトランザクションログ取得
  - 任意の時点にリストア可能（PITR）
  - 保持期間が過ぎると自動で削除
- **手動スナップショット**
  - ユーザーのタイミングで取得
  - 取得した時点にリストア可能
  - ユーザーが削除するまで残る

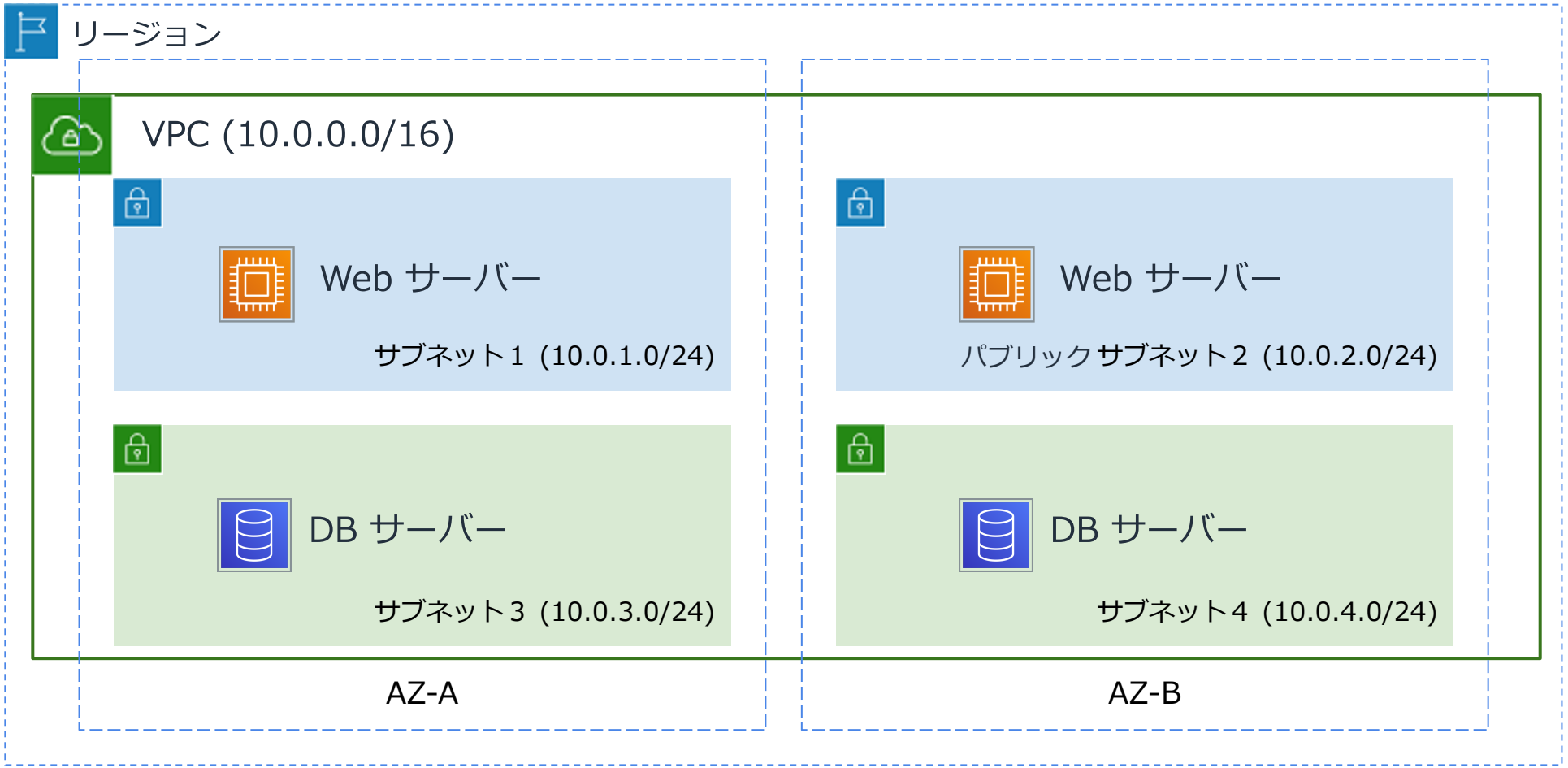
マネージド機能の一例

- **マルチ AZ 配置**
  - 元インスタンス（プライマリ）と別 AZ にインスタンス（スタンバイ）を起動
  - プライマリからスタンバイへ同期レプリケーションを実施 ➔ 耐久性の向上
  - 障害発生時にスタンバイ側へ自動でフェイルオーバー ➔ 可用性の向上

モジュール 4 のまとめ

- **データベースの選択方針**
  - リレーショナル or 非リレーショナル　・・・　データや性能要件などを考慮し適材適所
  - アンマネージド or マネージド　・・・　自由度と運用負荷の軽減のトレードオフ
- **RDS: マネージド・リレーショナル**
  - Aurora or その他　・・・　まずは Aurora から検討してみる
- **DynamoDB: マネージド・非リレーショナル**
  - マルチマスター、結果整合性に注意
- **DMS & SCT**
  - データベースの移行をサポート

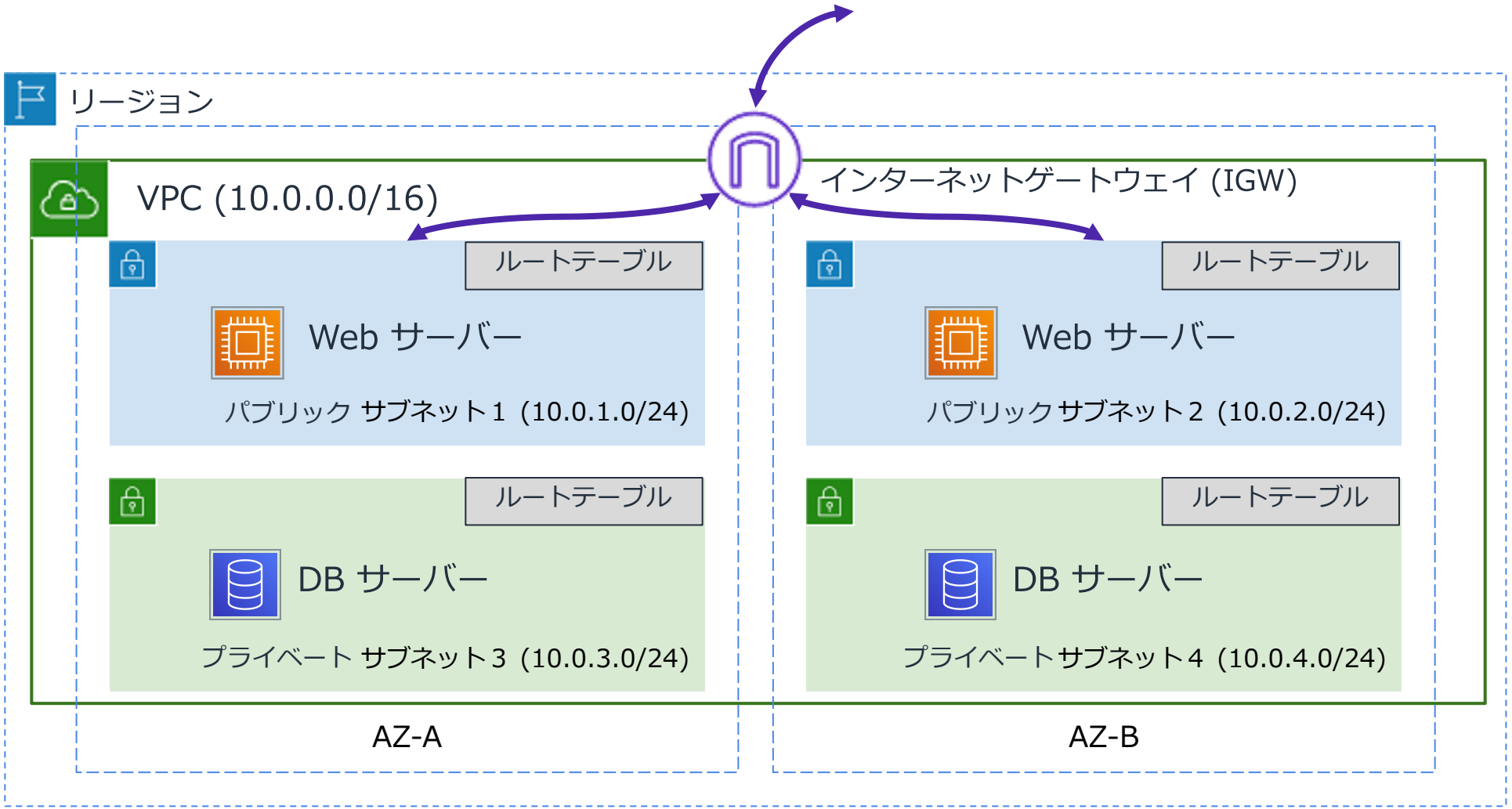
VPC の概念図



- VPC はリージョン内に作成する
- サブネットは AZ 内に作成する
- インスタンスの配置はサブネット内

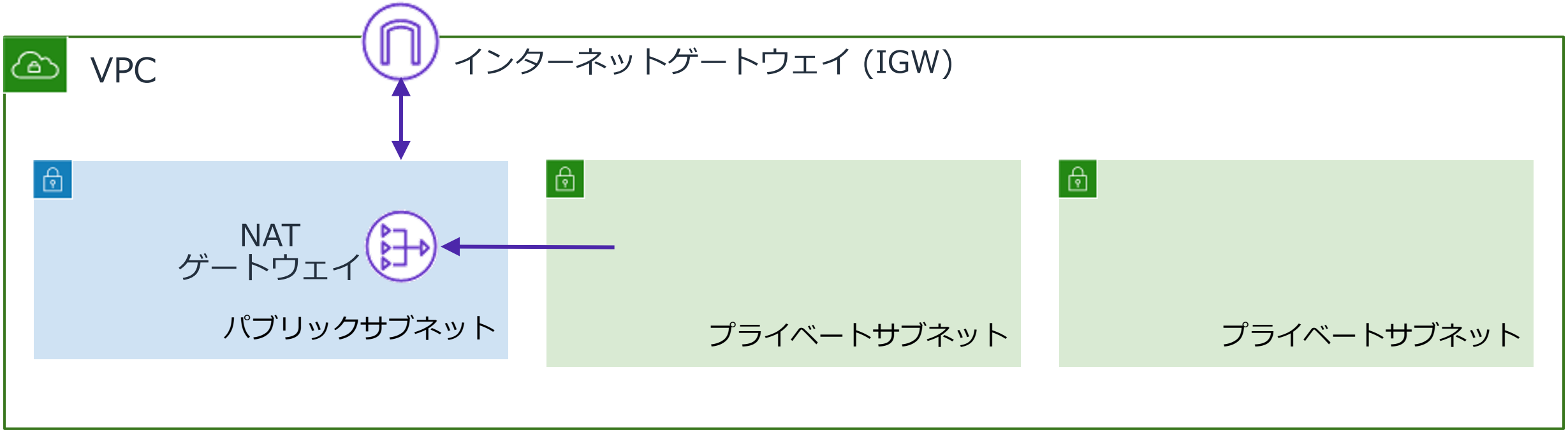
投影のみ

VPC の概念図



- VPC はリージョン内に作成する
- サブネットは AZ 内に作成する
- インスタンスの配置はサブネット内
- サブネット毎の経路表（ルートテーブル）
- IGW 経由で外部とt直接通信できるのがパブリック、できないのがプライベート

サブネットの構成パターン



送信先	ターゲット
10.0.0.0/16	Local
0.0.0.0/0	IGW

送信先	ターゲット
10.0.0.0/16	Local
0.0.0.0/0	NATGW

送信先	ターゲット
10.0.0.0/16	Local

モジュール 5 のまとめ

- **設計のポイント**
  - **Amazon Virtual Private Cloud (VPC)**
    - シングル VPC 構成、マルチ VPC 構成、**マルチアカウント構成でネットワークやアクセス権限を分離**
  - **サブネット**
    - パブリックサブネットとプライベートサブネットに分割し**できるだけプライベートに寄せる**
  - **ゲートウェイとルーティング**
    - IGWとNAT-GWとルーティングテーブルでインターネットへの経路設定
  - **ネットワークセキュリティ**
    - 基本セキュリティグループ、NACLで補完

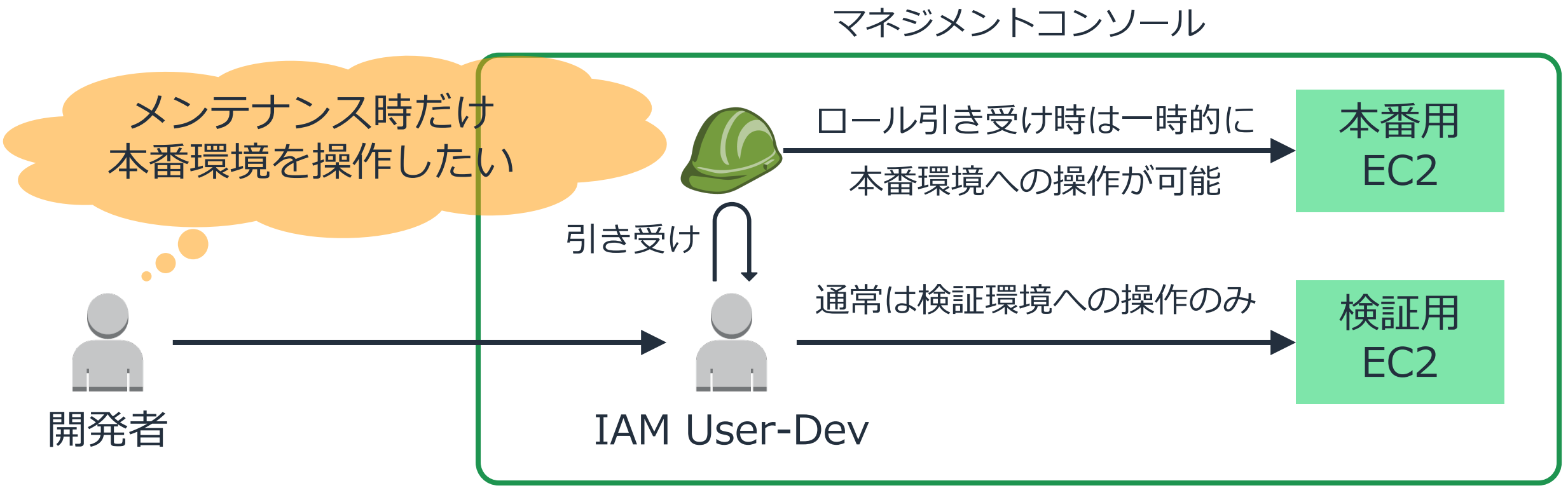
投影のみ

モジュール 6 のまとめ

- **VPC と他のネットワークの接続**
  - オンプレミス環境　・・・　VPN or Direct Connect
  - 他の VPC　・・・　VPC ピア接続 or Transit Gateway
  - 他の AWS サービス　・・・　VPC エンドポイント

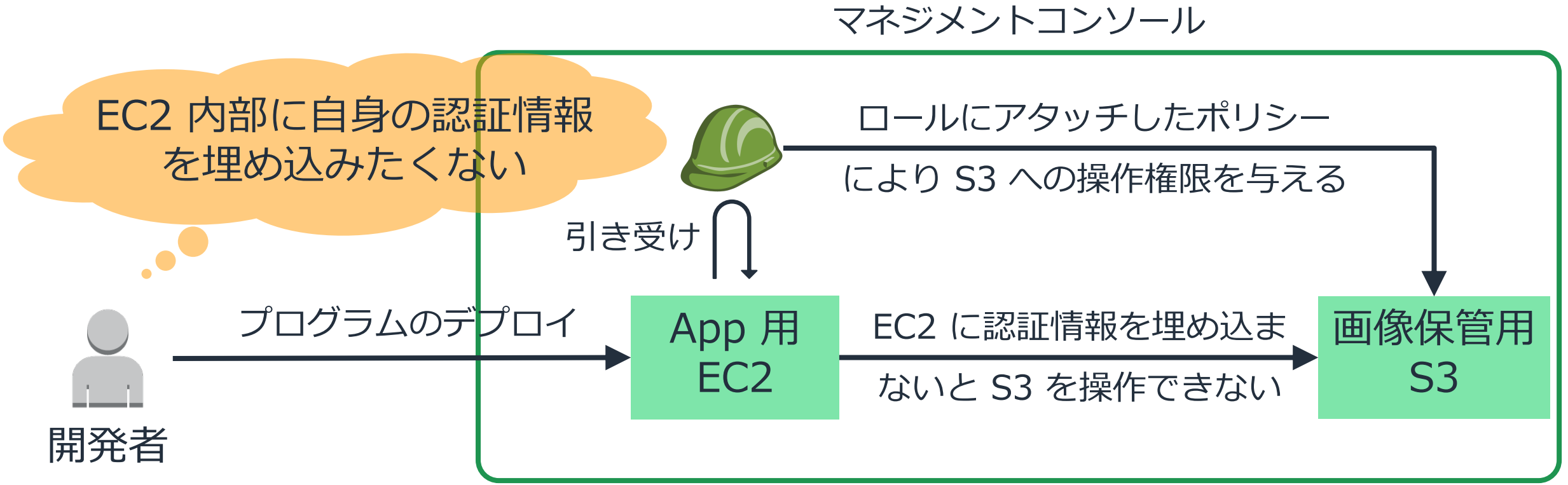
- **可用性を高めるアーキテクチャ**
  - マルチ AZ 構成　・・・　ELB
  - マルチリージョン構成　・・・　Route 53
  - ※ 可用性はアーキテクチャ全体を見渡して考える

ロールの例）一時的に権限を切り替え

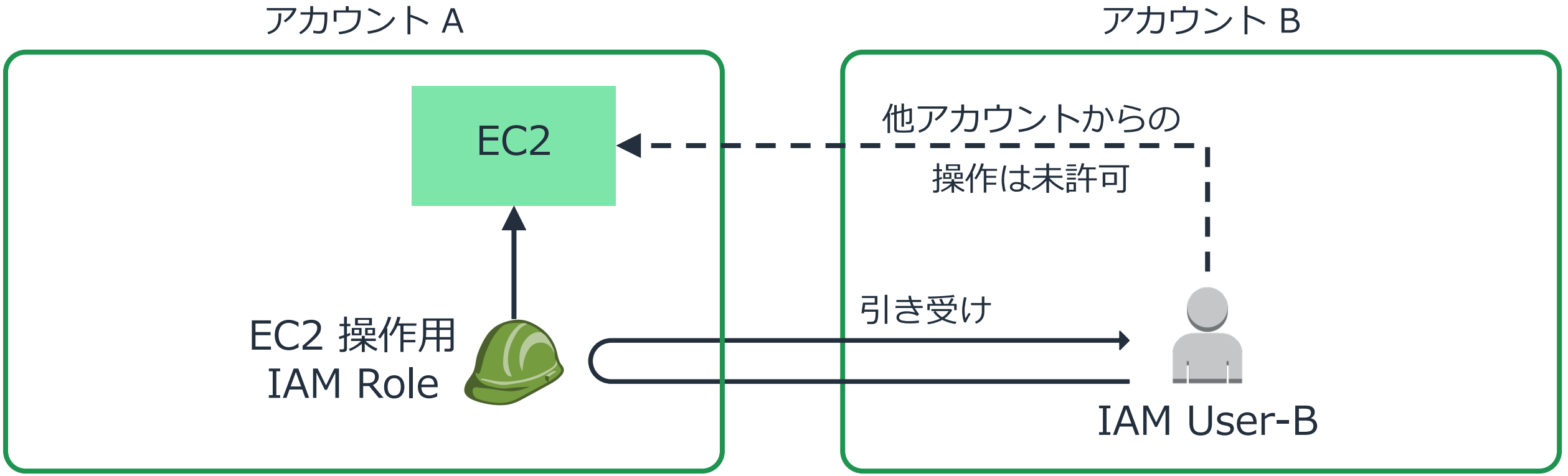


投影のみ

ロールの例）EC2 から S3 を操作



ロールの例）クロスアカウントアクセス



モジュール 7 のまとめ

- **IAM による認証＆認可**
  - 基本は IAM ユーザー（認証）と IAM ポリシー（認可）
  - 運用負荷を低減　・・・　IAM グループ

- **より効率的に利用するために**
  - IAM ロール　・・・　一時的な認証情報の発行で AWS で管理する認証情報を減らす
  - Cognito　・・・外部へ認証を移譲することで AWS で管理する認証情報を減らす
  - Organization　・・・　アカウントを分けることで独立した環境を用意

ワーク

- **Auto Scaling の最小・最大はどう決めるのがよい？**
  - 1) 最小台数について
  - 2) 最大台数について
  - ※ 正解があるわけではありません

モジュール 8 のまとめ

- **伸縮性とは**
  - 需要の変化に応じて “伸び縮み（スケーリング）” する性質
  - 必要なだけリソースを用意することでサービスを継続しつつ、コストも最適化
  - 3つの伸縮性 → 時間、ボリューム、予測
- **モニタリグ**
  - システム監視 → CloudWatch
  - コストの管理 → Cost Explorer
  - 操作ログ → CloudTrail
- **スケーリングの実現**
  - ３つの伸縮性を実現 → Auto Scaling
  - RDSの読み込み → リードレプリカ, DynamoDB
  - DynamoDB → Auto Scaling or オンデマンド

※ アーキテクチャ全体を見渡してスケーリングしていくことが重要

aws training and certification

投影のみ

モジュール 9 のまとめ

- **自動化する理由**
  - 手動作業のデメリット
    - スケールしない, バージョン管理できない, 証跡が取れない, 運用ミス
  - 自動化して手動作業のデメリットを解決

- **自動化を実現する方法**
  - インフラの自動化, IaC ・・・ CloudFormation
  - インフラより上のレイヤーの自動化 ・・・ Systems Manager
  - インフラとアプリケーションの自動化 ・・・ Elastic Beanstalk

モジュール 10 のまとめ

- **キャッシュの効果**
  - エッジ　・・・　通信遅延の短縮
  - ウェブ　・・・　ステートレス化
  - データベース　・・・　負荷の低減
  - ※ これらの効用と、キャッシュ管理のコスト増を天秤にかける

- **キャッシュのための AWS サービス**
  - エッジロケーション活用　・・・　CloudFront
  - DynamoDB にキャッシュ機能を追加　・・・　DAX
  - キャッシュ用のデータベース　・・・　ElastiCache

モジュール 11 のまとめ

- **疎結合化＆非同期処理の効果**
  - いつでもメッセージを送ることができ、いつでもメッセージを処理することができる
  - 送り手は完了まで待たずに済み、受け手は都合のいい時間にまとめて処理ができる
  - ※ 同期処理のほうが適しているものもあり、採用はケースバイケースで考える

- **疎結合化のための AWS サービス**
  - Amazon SQS　・・・　１：１ の非同期連携。メッセージキュー
  - Amazon SNS　・・・　１：N の非同期連携。Pub/Sub モデル
  - Amazon MQ　・・・　既存のシステムとの連携

モジュール 12 のまとめ

- **マイクロサービスとは**
  - マイクロサービス … 独立した複数サービスで構成され、API で連携・自律型、専門的
  - コンテナ … 再現性、自己完結な実行環境、VMより高速
  - サーバーレス … サーバー管理を意識する必要がない

- **マイクロサービスと相性の良いAWSサービス**
  - Amazon ECS　・・・　コンテナ
    - AWS Fargate の併用により運用負荷を低減
  - AWS Lambda　・・・　サーバーレス
    - 様々なAWSサービスと連携、API Gateway により公開
  - AWS Step Functions により複数 LambdaやECSからなるサービスの実行フロー制御

モジュール 13 のまとめ

- **災害対策計画の考え方**
  - 無闇な実施はコスト増を招く
  - RPO … 目標復旧時点。バックアップの方法等で管理
  - RTO … 目標復旧時間。復旧オペレーション等で管理

- **検討のポイント**
  - アーキテクチャの工夫（MAZ / NW 冗長化 / など）
  - マネージド機能の活用（CRR / グローバルテーブル / など）
  - 採用する復旧戦略
    - バックアップ＆復元 / パイロットライト / 低キャパシティスタンバイ / アクティブ
  - 復旧オペレーションの自動化（CloudFormation などの活用）