

# クラウドセキュリティ (SECKUN講義)

第1回

松本照吾 (アマゾン ウェブ サービス ジャパン株式会社)



# 松本 照吾

- Head of Security Assurance, Japan, Amazon Web services Japan
- CISA, CISSP, CAIS-Lead Auditor, MBA
- 自転車、ジョギング、スポーツ観戦、歴史、かぶりもの









# 講義にご参加いただく皆さんへのお願い

- 感想、質問はslackへ（参加しなかった方も読みやすい）
- 特に質問は、質問： って書いてくれると見つけやすいです
- AWS講座ではないので、AWSのサービス名とかは別におぼえなくていいです。
- AWS講座ではないので、サービスの詳細は答えません。（しらべてくださいませ）
- 参加型プレゼンツールのmentimeterを使っています。よろしければ上記からコードをいれてログインを。



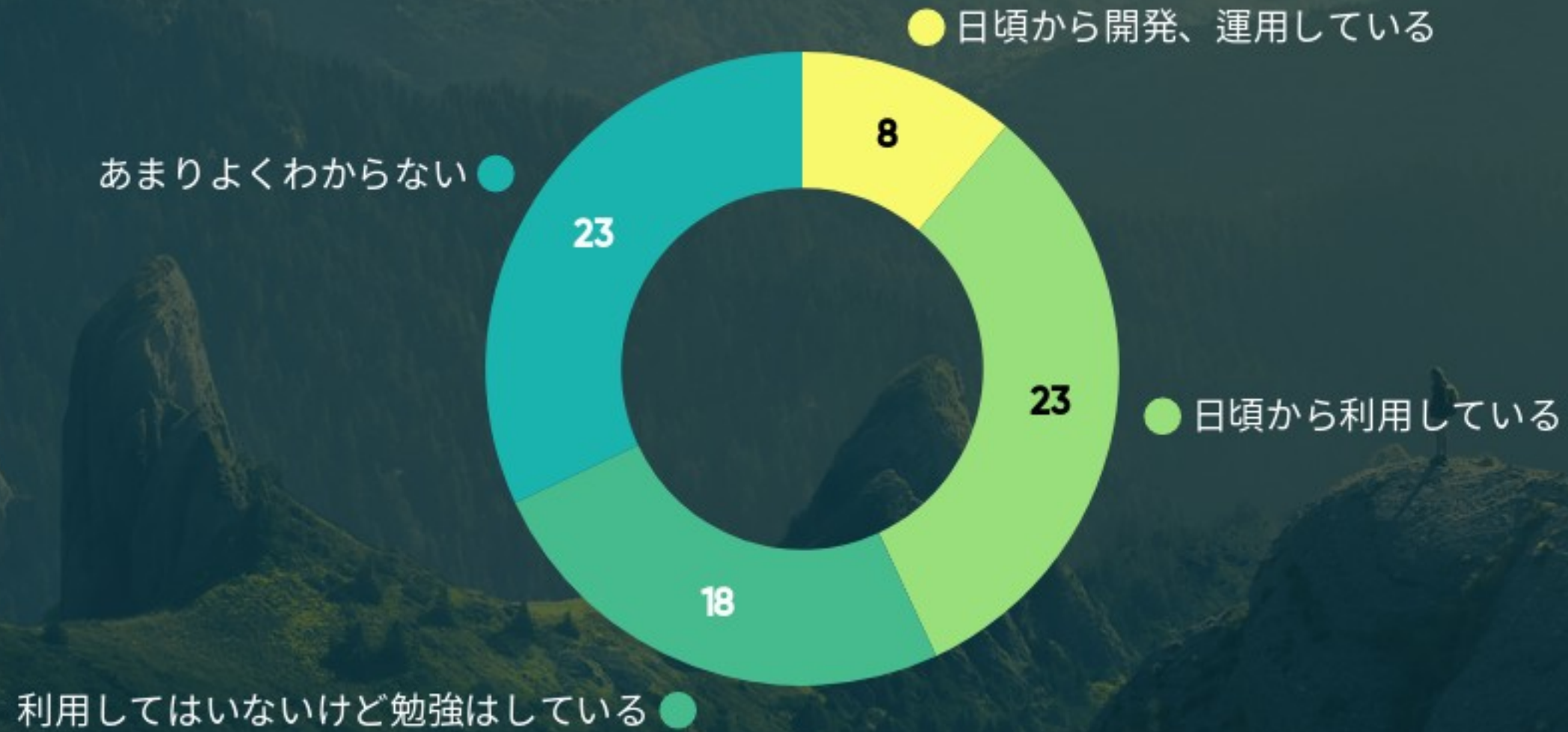
# 本講義の狙い

- クラウドの基本的な特性と影響
- クラウドにおいてサービスはどのように設計・運用されるのか（ここまで今回）
- クラウドを生かしたセキュリティの在り方（二回目）
- クラウドとは？を理解して、クラウドのセキュリティを理解する。



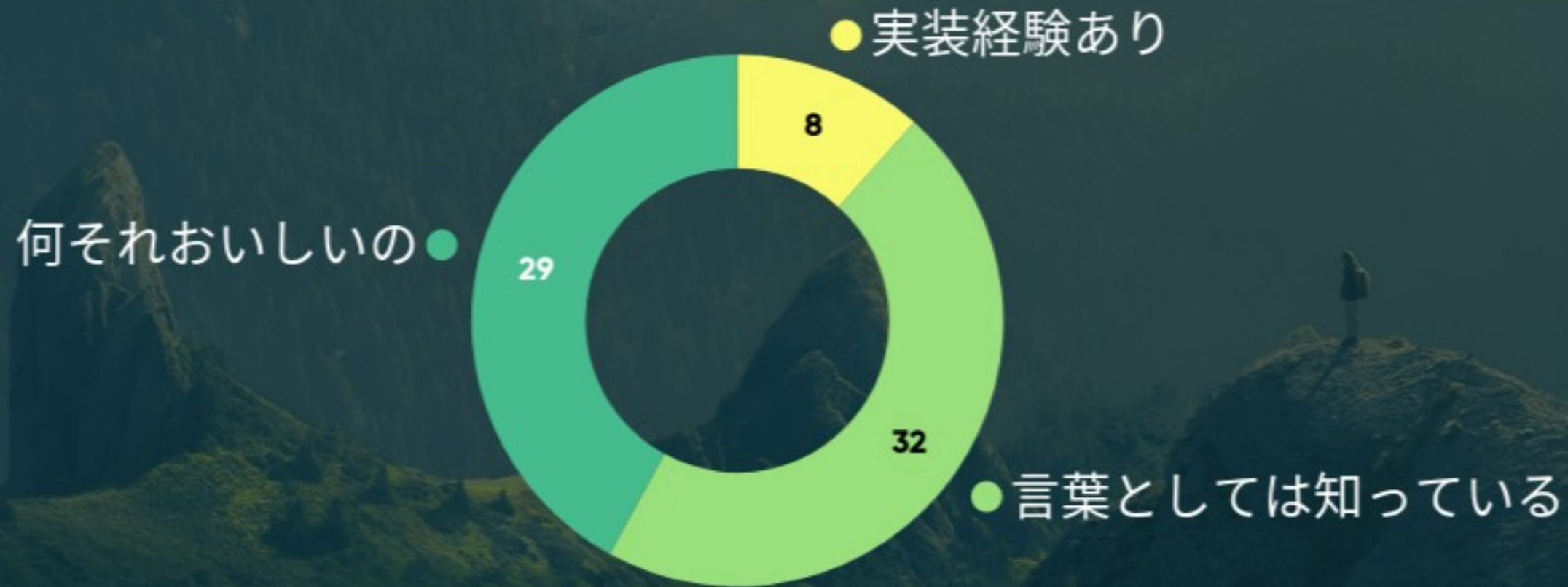


# 皆さんに質問：クラウドのご経験は？



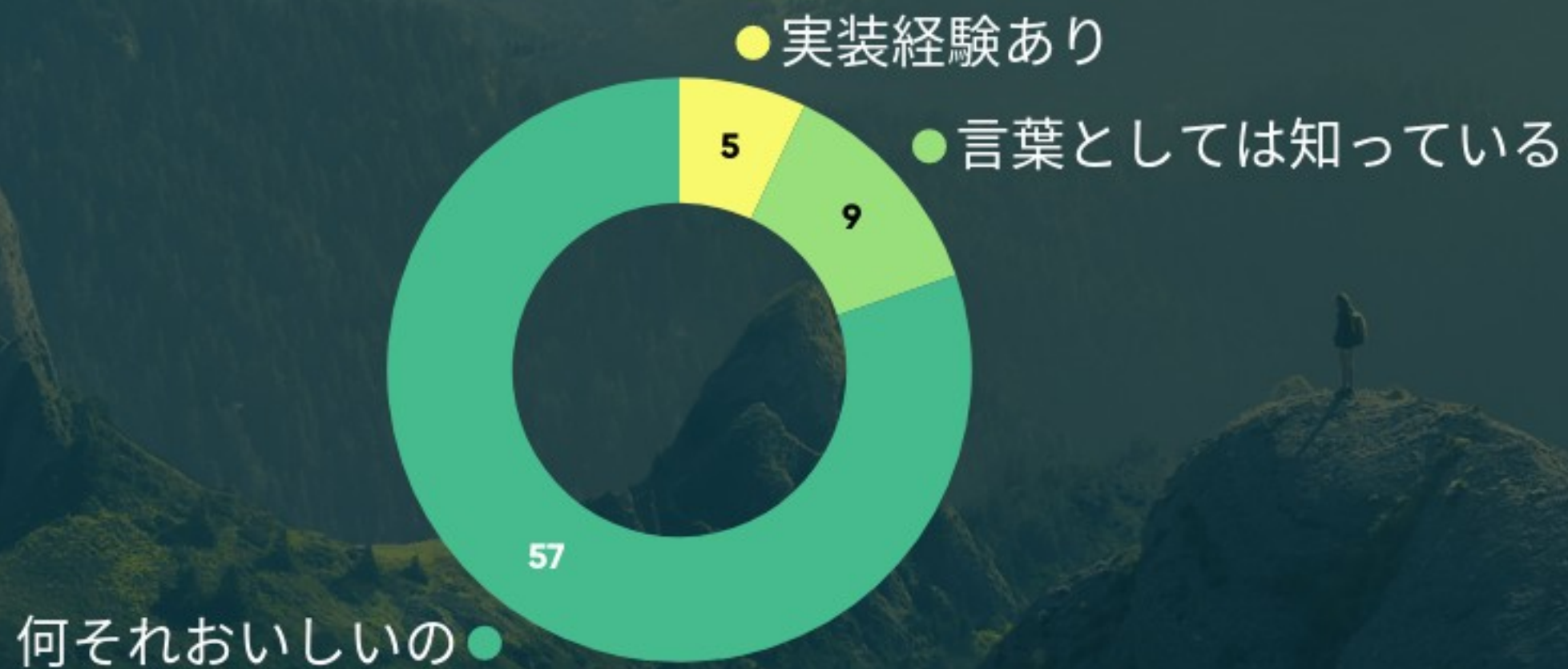


# 皆様に質問：サーバレスアーキテクチャを知っていますか

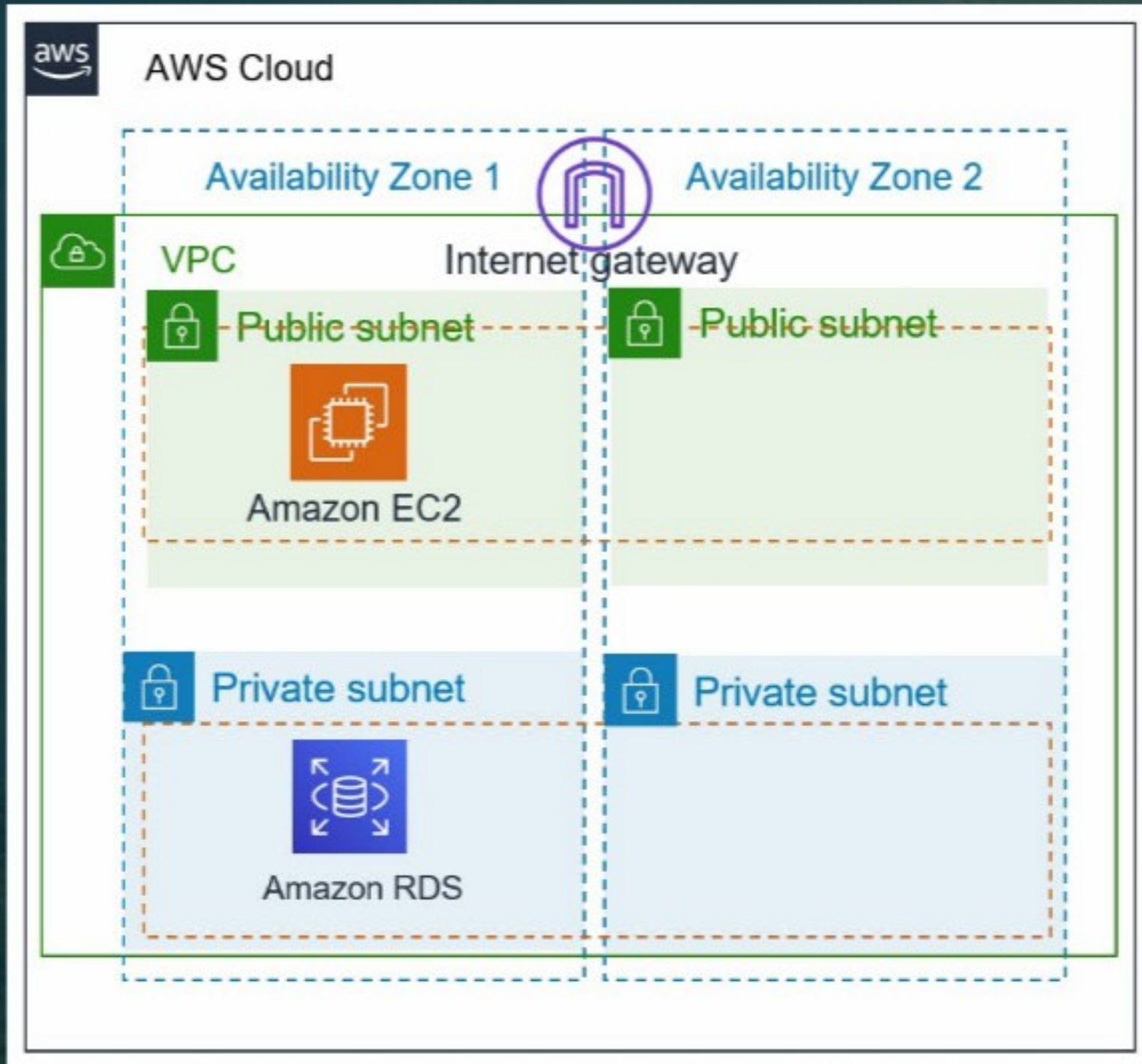




# 皆様に質問：イミュータブルインフラストラクチャを知っていますか







# ラボ（事前学習）の振り返り

- VPC（仮想ネットワーク）内のWebアプリケーション
- フロントにEC2によるWebサーバ
- バックエンドにRDSによるMySQL



# 終わった人のためのおまけ

- 本手順を実行した場合、Webサーバへのログインはできません。また、pingコマンドも到達しません。さて、ログインするために必要な設定はなんでしょう。
- EC2とRDS、作成する手順の中での違いはなんでしょう。か。  
（もし、MySQLをEC2上に構築する場合、どのような手順が必要でしょう）
- 本ネットワークの構成図を書いてみてください。また、講師はセキュリティ上、本構成はあまり推奨しません。なぜでしょう。





# オンプレミスで設計する場合、どの程度、構築に時間がかかる？





## サービスの設計と運用（環境の構築にかかる時間は？）

- 複雑な構成要素
- 繰り返しの作業
- 構成、環境のバックアップ、リストア、BCPの策定
- 環境の可視化



<https://aws.amazon.com/jp/quickstart/architecture/compliance-nist-high-impact/>



## 2. NISTによるクラウドコンピューティングの定義

クラウドコンピューティングは、共用の構成可能なコンピューティングリソース（ネットワーク、サーバー、ストレージ、アプリケーション、サービス）の集積に、どこからでも、簡便に、必要に応じて、ネットワーク経由でアクセスすることを可能とするモデルであり、最小限の利用手続きまたはサービスプロバイダとのやりとりで速やかに割り当てられ提供されるものである。このクラウドモデルは5つの基本的な特徴と3つのサービスモデル、および4つの実装モデルによって構成される。

### 基本的な特徴:

オンデマンド・セルフサービス (On-demand self-service)	ユーザは、各サービスの提供者と直接やりとりすることなく、必要に応じ、自動的に、サーバーの稼働時間やネットワークストレージのようなコンピューティング能力を一方向的に設定できる。
幅広いネットワークアクセス (Broad network access)	コンピューティング能力は、ネットワークを通じて利用可能で、標準的な仕組みで接続可能であり、そのことにより、様々なシンおよびシッククライアントプラットフォーム（例えばモバイルフォン、タブレット、ラップトップコンピュータ、ワークステーション）からの利用を可能とする。
リソースの共用 (Resource pooling)	サービスの提供者のコンピューティングリソースは集積され、複数のユーザにマルチテナントモデルを利用して提供される。様々な物理的・仮想的リソースは、ユーザの需要に応じてダイナミックに割り当てられたり再割り当てされたりする。物理的な所在場所に制約されないという考え方で、ユーザは一般的に、提供されるリソースの正確な所在地を知ったりコントロールしたりできないが、場合によってはより抽象的なレベル（例：国、州、データセンタ）で特定可能である。リソースの例としては、ストレージ、処理能力、メモリ、およびネットワーク帯域が挙げられる。
スピーディな拡張性 (Rapid elasticity)	コンピューティング能力は、伸縮自在に、場合によっては自動で割り当ておよび提供が可能で、需要に応じて即座にスケールアウト／スケールインできる。ユーザにとっては、多くの場合、割り当てのために利用可能な能力は無尽蔵で、いつでもどんな量でも調達可能のように見える。
サービスが計測可能であること (Measured Service)	クラウドシステムは、計測能力 <sup>1</sup> を利用して、サービスの種類（ストレージ、処理能力、帯域、実利用中のユーザアカウント数）に適した管理レベルでリソースの利用をコントロールし最適化する。リソースの利用状況はモニタされ、コントロールされ、報告される。それにより、サービスの利用結果がユーザにもサービス提供者にも明示できる。

# クラウドとは何か (NIST SP 800-145)

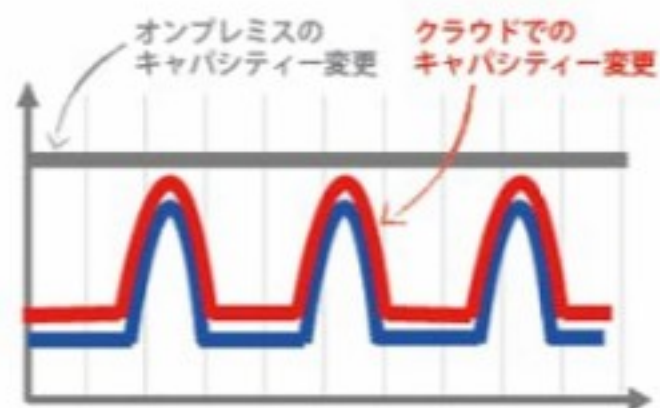
→ Amazonの場合、IT資産に係るコスト効率化を目的としてオンデマンドリソースに注目

→ クラウドを実現するための基礎技術が仮想化基盤

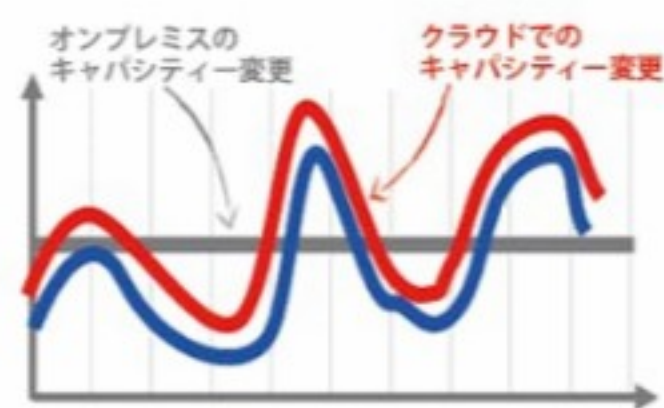
→ <https://www.ipa.go.jp/files/000025366.pdf>



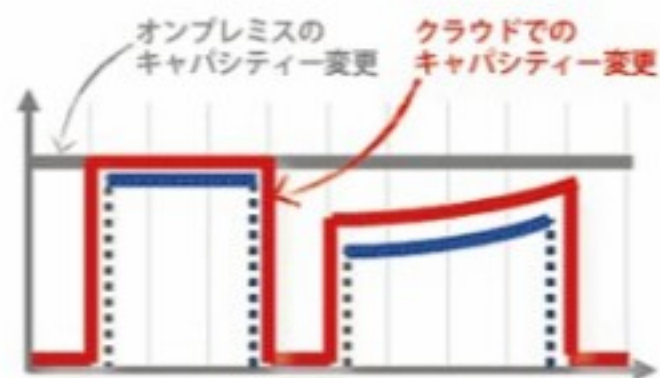
## 📦 オンプレミスとクラウドの比較



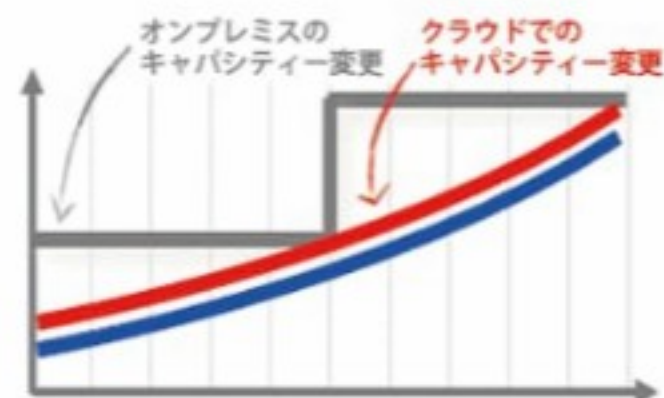
アクセス変動の予測がしやすいサーバー



アクセス変動の予測ができないサーバー



時間帯によってサーバーのオン・オフを行う場合



急成長する事業向けに利用するサーバー

# ITの需給に対する変化

- 季節性変動 (Amazon.com)
- 利用者の急激な増減 (配信、ゲーム)
- リソース利用の効率化 (ハウスキーピング)





# 仮想化技術がもたらしたもの

- H/W 一度調達したら変更は困難
- S/W 変化に関する柔軟性、拡張性
- 仮想化技術によってインフラをソフトウェア化
- 共有環境によるコスト効率化、セキュリティ管理の分離（事業としてのクラウド）







# 変わらないITから“変化”を前提としたITへ

- 本来ITはビジネスのニーズにあわせるもの
- ウォーターフォールからアジャイル
- DevOps (DevSecOps)
- Amazonは年間に5000万件デプロイ (1秒に1回、サービスが更新)



# 仮想化のもたらすもの：実装までのリードタイムと品質



容量予測や調達のリードタイムは大幅に削減  
テストの反復は品質の向上に。



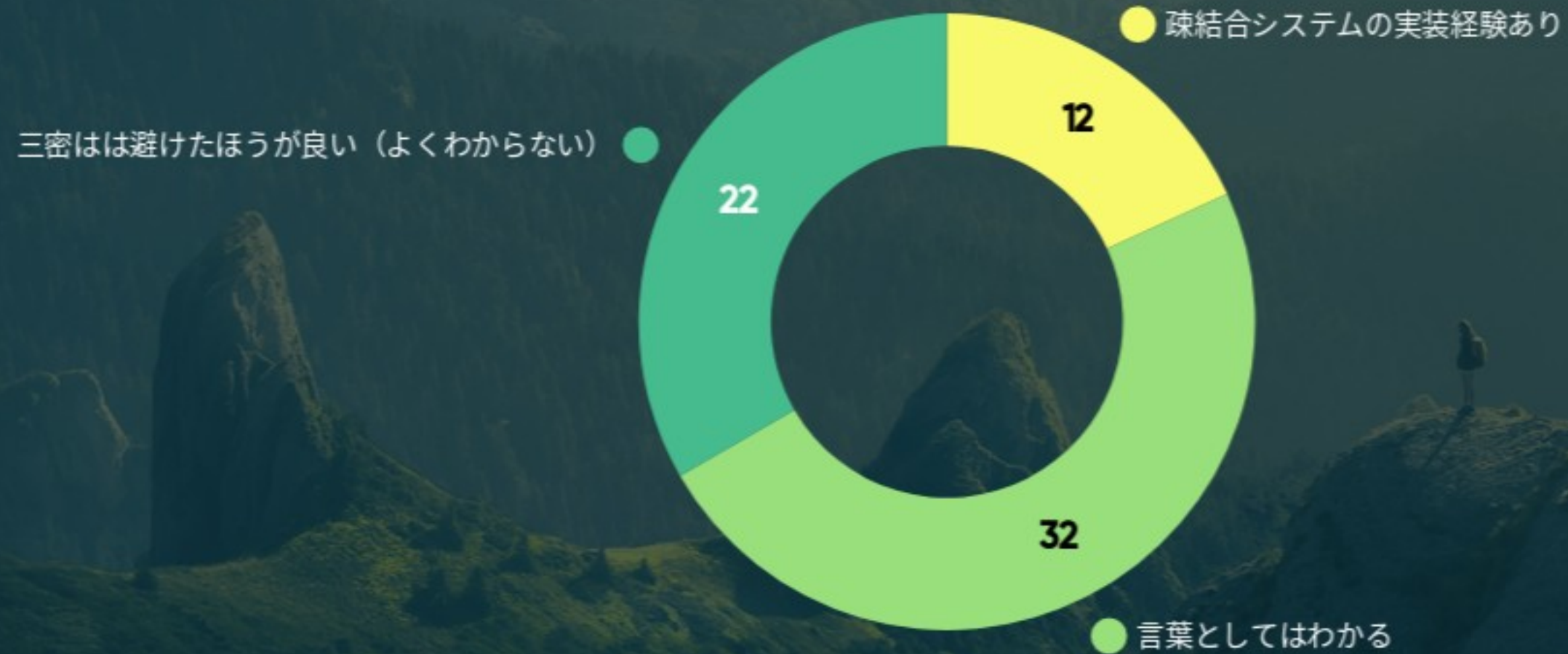


# クラウドにおけるサービス設計の現在



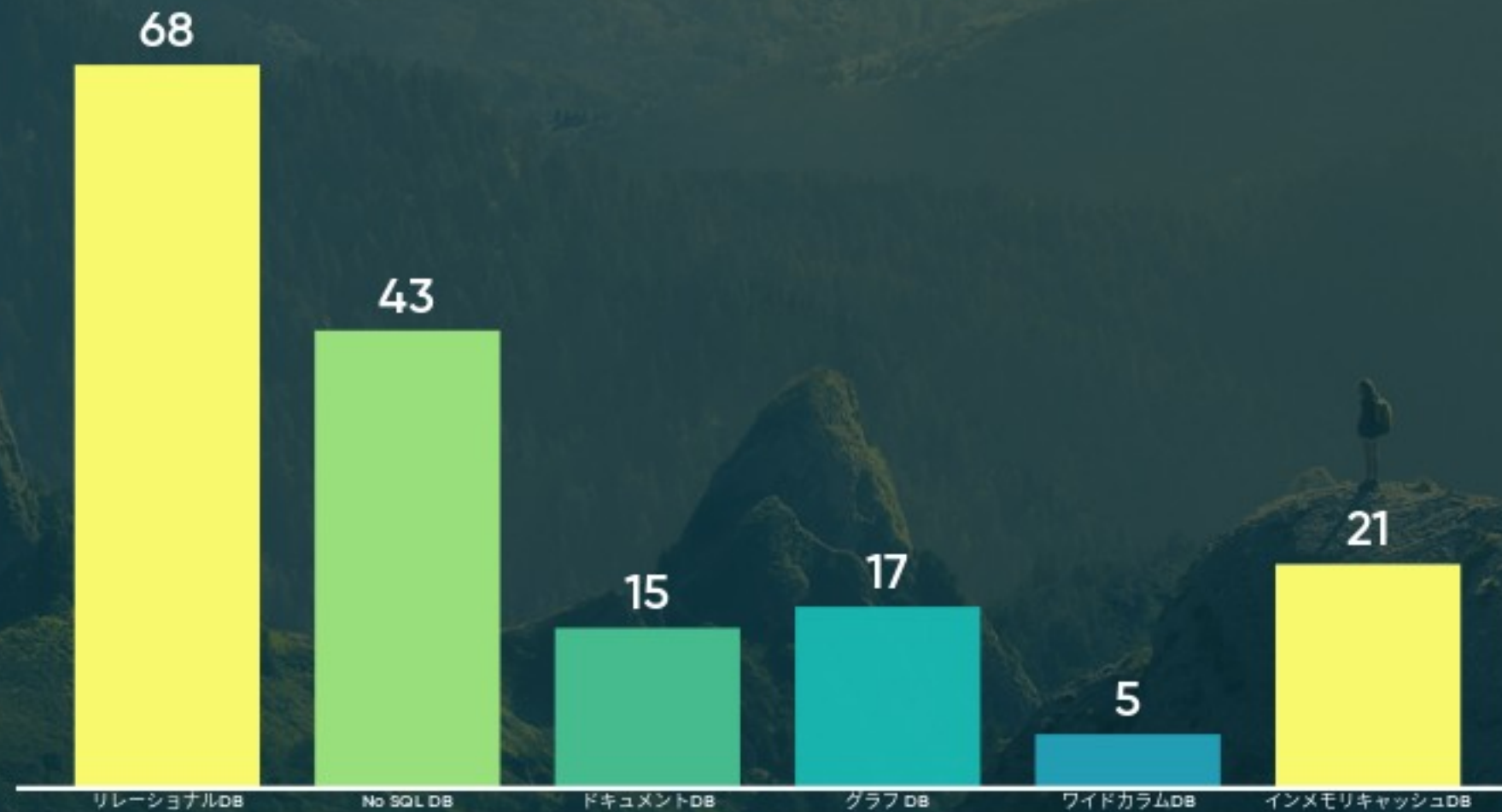


# 皆様に質問：疎結合ってわかりますか

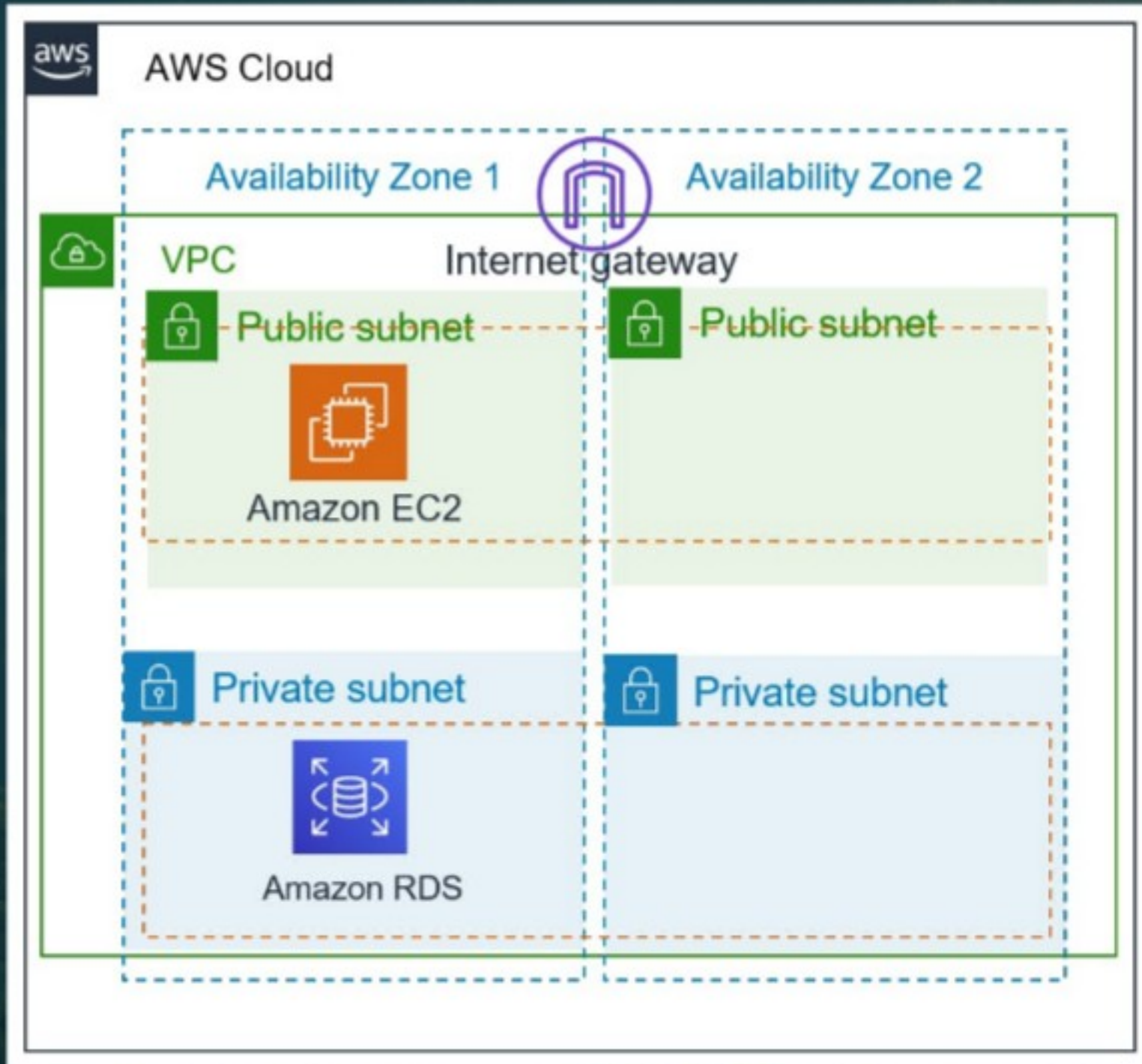




# 皆様に質問：知っているデータベースの種類を選んでください（複数回答）







# ラボ（事前学習）の振り返り

- VPC（仮想ネットワーク）内のWebアプリケーション
- フロントにEC2によるWebサーバ
- バックエンドにRDSによるMySQL



# 皆様に質問：なぜWebサーバとDBをわけ るのですか

DBサーバをパブリックに晒さないように  
するため。

Webサーバを複数立てて負荷分散できる  
ようにするため。

DBには直接アクセスしてほしくないた  
め。

実はよくわからない。

サービスを切り分けて、疎結合にするた  
め

外部から直接DBにアクセスさせないた  
め

不正アクセス、あるいは障害によって  
Webサーバに異常があった際に、DBへの  
影響を防ぐため。あるいはその逆。

DBサーバに対して直接利用者にアクセス  
させたくないため。

障害・セキュリティ等の影響範囲を区切  
るため



# 皆様に質問：なぜWebサーバとDBをわけ るのですか

CPUやメモリなどのリソースの観点と、  
外部NWと接続する境界セキュリティ観点

機能を分けることによって、それぞれの  
拡張性が上がる。

よくわからない。

WebとDBでは利用者が違うため

ハードウェアに求められる性能の種類が  
異なるから。

DBサーバをグローバルに晒さないように  
するため。

アプリケーションサーバとDBサーバはリ  
ソースの使い方の特性が異なる

Webサーバはインターネットに公開され  
ているため、攻撃されやすい。DBと一緒  
にしてしまうと、DB内の情報がインター  
ネット上に流出してしまう可能性が出て  
しまう。

データを保存する環境を公開することは  
セキュリティ的に好ましくない。データ  
の改ざんや破損の確立を下げるため。



# 皆様に質問：なぜWebサーバとDBをわけ るのですか

管理ポリシーが異なる

外部からDBサーバへ直接接続する必要がないため

サービス提供とデータの蓄積という機能が異なるためかな。

webサーバの変更や破損の影響をDBに及ばさないようにするため

DBを分けることで、外部アクセスを遮断し直接データを取得するなどを防ぐため。

1つが楽ですがセキュリティ上分けています。データが持ち出されにくいように。

・機能の分離

壊れた時に交換しやすいから。

Webからアクセスを受けるWebサーバ上にデータを格納した場合、もしも攻撃を受けた際にデータを抜き取られるリスクが高まるため、WebサーバとDBサーバは分けて構築する方がよいと考えます。



# 皆様に質問：なぜWebサーバとDBをわけ るのですか

サーバへの影響を避ける

・コンピューターリソースの利用状況を  
モニタリングしやすくする・動かすソフト  
ウェアの適正に合わせたスケール戦略  
を適用するため・

Webサーバが乗っ取られても重要なデー  
タが集まっているデータベースサーバは  
守ることができるから。

社外などからのアクセスはWebサーバに  
て実施し、セキュリティ面からDBサーバ  
は分ける必要がある

WebサーバとDBサーバが一緒だと、外部  
からWebサーバが乗っ取られると、DBサ  
ーバもそのまま乗っ取られるから。

不正アクセス、障害が発生した際に、DB  
へ影響を与えないため

リソースをそれぞれのサービスに最適化  
するため

DBの堅牢化、役割の違い、スペック検討

セキュリティ対応 / 構成管理上の都合 /  
運営者の権限管理の都合などを念頭にお  
いて、構成を組むことによると考えます



# 皆様に質問：なぜWebサーバとDBをわけ るのですか

データベースとWebサーバとはアクセスすべき者の範囲が異なり、セキュリティ上の設定を分け、リスク分離をしたいから。

グローバルネットワークにデータベースを起したくない。グローバルだと攻撃対象になり、脆弱性による問題でデータへのアクセスされる可能性がある。

DBへの不正アクセスなどを防ぐため

データベースサーバを外部に晒したくない、データを保存する場所と実際に動くところを分けることで、障害があってもデータだけは守れる可能性が挙げられる、など

それぞれの役割が違うので、分離した方が管理しやすい。スケールアウト、セキュリティ等々

・DMZと内部ネットワークに配置するため  
・障害時の影響を分離するため

データ保護の観点から見ると、格納されている情報のゾーニングが明らかに異なることが多く、公開レベルが異なるため。リソースの観点からは、アクセスのピークが異なる可能性があるため。障害発生時のデータ破損を防止するため。等々

セキュリティ上、ハードを分けておいた方がいい。Webサーバは外に開いている分、危険度が高くなる。

必要とする性能に違いがあるから。Web



# 皆様に質問：なぜWebサーバとDBをわけ るのですか

- ・ 障害が局所的になる ・

障害が発生した時のリスクをていげんするため

分離しておくことで、メンテナンスしやすくする。負荷がかかったときに、スケールアウト・スケールアップしやすくする。

セキュリティの確保。WebとDBを同居させたら、インターネットからDBにアクセスを許すことになる。

個別の拡張性を確保したい、依存性をなくしたい（疎結合としたい）

User Interface 部分とData を分ける事によりSecurityを担保

安全性も関係があります。情報交換と情報保存、ネット層も違います。

- ・ リソースを分けることで、処理を高速化、構造のシンプル化、管理の効率化を行う。
- ・ セキュリティ上、DBはインターネットに接するWebサーバなどに対して、もっと守られるべきであるから。

外部アクセスのフロントとなるWebサーバにデータベースを持たせるのは危険であるため



# モノリシック（単一）アーキテクチャの限界

- スケーラビリティ（拡張性）
- 耐障害性
- 変化への柔軟性（サービスの進歩）



# コンポーネントの分離

- 環境の混在による障害の低減
- 障害の分離
- リソースの効率化





サーバをペットにしない



# リソースによる役割の違い

- ストレージ：永続的なデータの保持
- コンピューティング：需要に応じた処理の効率化
- 実際にはストレージ自体も役割が多様化





# データベースサービス

データベースのタイプ	ユースケース	AWS のサービス
リレーショナル	従来のアプリケーション、ERP、CRM、e コマース	 Amazon Aurora  Amazon RDS  Amazon Redshift
キー値	トラフィックの多いウェブアプリ、e コマースシステム、ゲームアプリケーション	 Amazon DynamoDB
インメモリ	キャッシュ、セッション管理、ゲームのリーダーボード、地理空間アプリケーション	 Amazon ElastiCache for Memcached  Amazon ElastiCache for Redis
ドキュメント	コンテンツ管理、カタログ、ユーザープロフィール	 Amazon DocumentDB (MongoDB 互換)
ワイドカラム	高スケーラの異界アプリ、設備のメンテナンス、多数の装置の管理、ルートの最適化	 Amazon Keyspaces (Apache Cassandra 向け)
グラフ	不正検出、ソーシャルネットワーク、レコメンデーションエンジン	 Amazon Neptune
時系列	IoT アプリケーション、DevOps、産業テレメトリ	 Amazon Timestream
台帳	記録システム、サプライチェーン、銀行トランザクション	 Amazon QLDB

## データベースも多様化





「疎結合」とは「結合度を緩める」という意味で、言い換えると「独立性を高める」と表現することもできます。反対語は「密結合」となります。

現実世界の例を出してみましよう。例えば「年賀状など、手紙を送る場面」を想像してみるのはいかがでしょうか？手紙を郵便ポストに投函することを「疎結合」だとすると、郵便局で配達員に直接手渡しすることを「密結合」と考えることができます。



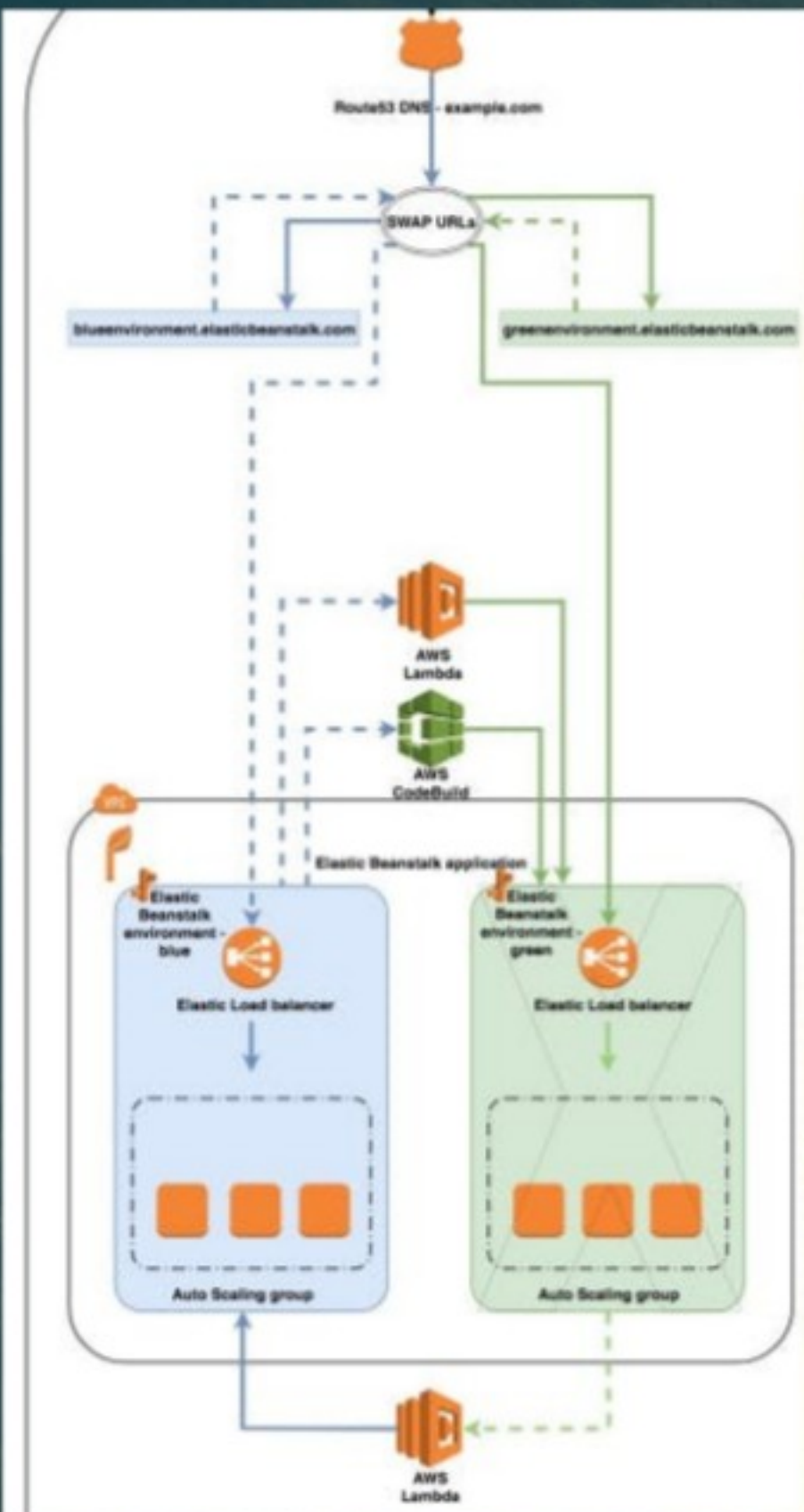


# システム設計のベストプラクティスとしての疎結合

- 処理リソース（可変）と保持リソース（永続）の分離
- 柔軟な処理の受付（バックエンドの処理能力によらない）
- キュー（一時的な保持、待ち行列の形成）
- ストレージやデータベース（永続的な保持）
- ロードバランサーなどを活用した処理リソースの柔軟な増減





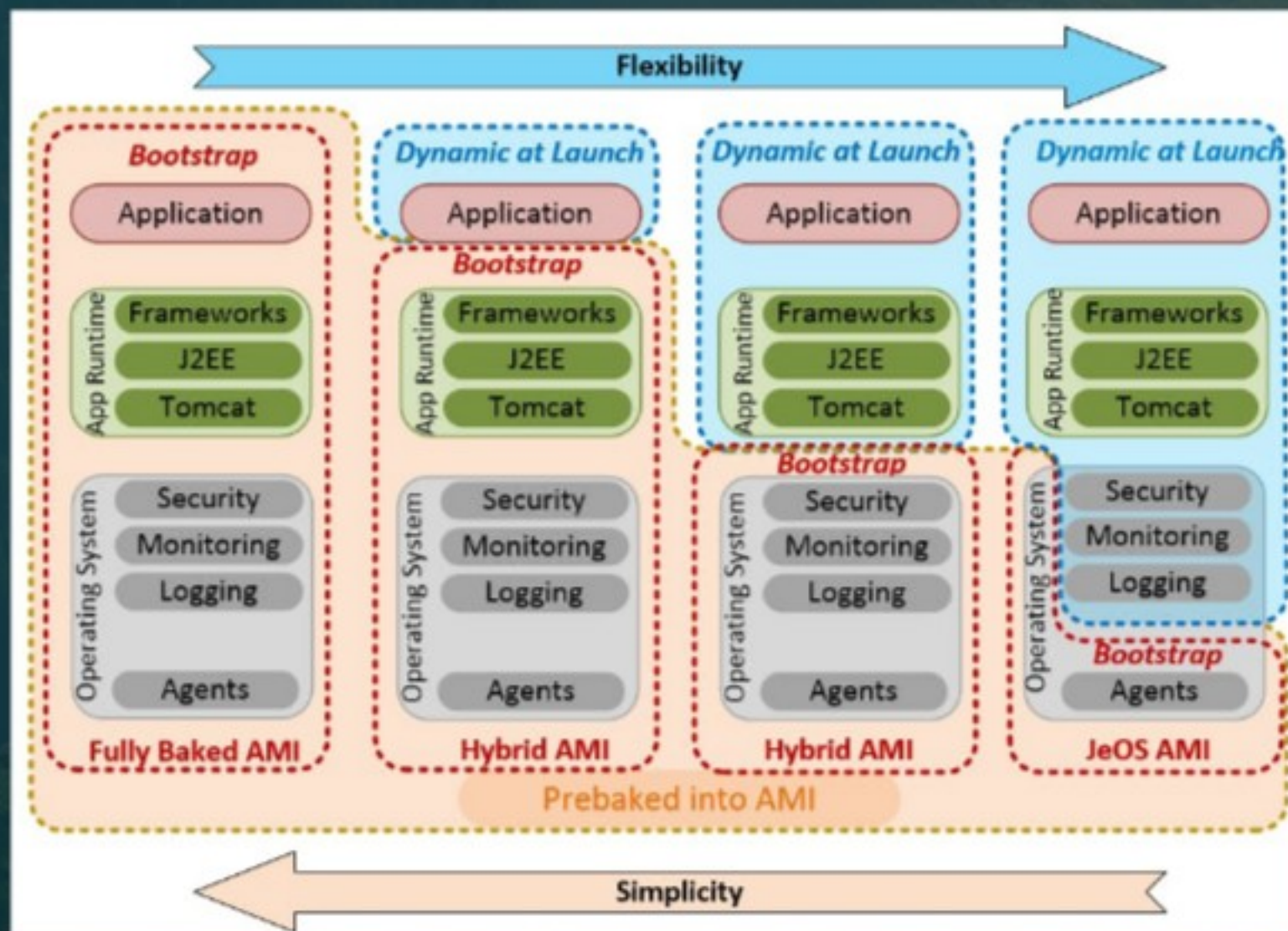


# イミュータブルインフラストラクチャ

- 本番環境の管理をしない（パッチ、バージョンアップ）
- 変更はサービス停止というビジネスリスク、環境の不安定化というセキュリティリスクを伴う
- ブルーグリーンデプロイメントなどによる環境自体の移行







# OSのイメージ化も管理にあわせて選択ができる

- 最低限のイメージ（利用時に構成、パッチ適用）
- 完全なパッケージ化（利用者による変更不可）
- 組織の戦略により管理手法は異なる





# クラウドにおける設計

- 疎結合を前提としたコンポーネント分離
- コンピューティングリソースは“捨てられるもの”
- データの保持には永続性が求められる





# AWSにおけるサービスの多様性

なぜ、多くのサービスが存在するのか





# AWSの現在

- 190を超えるサービス
- 24のリージョン、77のアベイラビリティゾーン
- 90%のサービスは利用者のフィードバックをもとに





# EC2とRDS、何が違いました？

マネージドサービスとそうではないサービスの違い

操作するレイヤが違った。

汎用コンピュータとDBの違い

OSが選択できるできないか

EC2は何でもソフトウェアをインストールできるけど、RDSはデータベースに特化したマネージドサービス（バックアップとかもある）

カスタマイズ性??

PaaSとSaaS

OSのレイヤーから選択するか、アプリから選択するかの違い

EC2はOSが選べ、構築したOS上に自由にミドルウェアがインストールできるが、RDSはOSを意識することなく、データベースを利用することができる。



# EC2とRDS、何が違いました？

EC2 OSやミドルウェアを自分で選択する。RDS マネージドサービス DBのエンジンだけ選び、管理はAWSにお任せ。

ユーザの責任分界点のレイヤーが異なる

EC2⇒サーバOSから選択。RDS⇒ミドルウェアから選択。

IaaSとSaaS

EC2は仮想マシンを提供するものに対して、RDSは管理されたRDBのエンドポイントを提供するもの。

EC2のほうが、ユーザ側で管理しないといけない領域が多い。OSやプラットフォームなど

EC2：OS以降の管理が必要なのか、  
RDS：データ(利用したい機能)のみの管理で済むのか。

提供しているサービスレイヤーの違い

マネージドサービスか否か（OSの選択が無い、パッチ当て等の保守が必要ない）



# EC2とRDS、何が違いました？

EC2はOSの入ったインスタンスを提供するのみでその後の処理は顧客に依存するが、RDSはある程度AWS側で管理する

RDSはOSを意識しなくてもよい。DBサービスのみを利用しているようなイメージである

IaaS PaaS

契約者の管理範囲が異なる

RDSは構築が速くOSのことを考えなくていい。

運用の負荷が違う（OSの選択など。IaaSとPaaS）

サーバとDBの違い？

RDSのほうが運用の負荷が少ない。EC2のほうが自由度が高い。細かいことができる。

EC2はサーバー機器を仮想的にしたもので、ここにいろんなアプリを自分でインストール・設定する。データベースも入れることができる。RDSはデータベースに特化したサービス。そのほかのアプリは入らない。



# EC2とRDS、何が違いました？

EC2-OSを選択。RDS-クラウド責任者が  
ある程度管理する。利用者の運用の負荷  
が減る。



# EC2とRDS（インフラストラクチャサービスとマネージドサービス）

- EC2：OSの選択からお客様責任（今後のパッチなども）
- RDS：OSは意識しない。バックアップなども自動化、ミドルウェアの設計から
- もちろん、EC2の上にMySQLをインストールもできる（その分、運用負荷）



アプリケーション作成	アプリケーション作成	アプリケーション作成	アプリケーション作成
スケールアウト設計	スケールアウト設計	スケールアウト設計	スケールアウト設計
定形運用設計	定形運用設計	定形運用設計	定形運用設計
ミドルウェアパッチ	ミドルウェアパッチ	ミドルウェアパッチ	ミドルウェアパッチ
ミドルウェア導入	ミドルウェア導入	ミドルウェア導入	ミドルウェア導入
OSパッチ	OSパッチ	OSパッチ	OSパッチ
OS導入	OS導入	OS導入	OS導入
HWメンテナンス	HWメンテナンス	HWメンテナンス	HWメンテナンス
ラッキング	ラッキング	ラッキング	ラッキング
電源・ネットワーク	電源・ネットワーク	電源・ネットワーク	電源・ネットワーク
オンプレミス	独自構築 on EC2	マネージドサービス	サーバーレス アーキテクチャ
開発者が担当	AWSが担当		

サービスによって管理レイヤが異なる



## AWS DeepRacer とは

1/18スケールの  
自律走行カー学習と評価のための  
シミュレータ世界中での  
レースリーグ

aws SUMMIT

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

# 新サービスのパターン

- 実現するものは同一、責任範囲が異なるもの(ストレージ=何らかのデータを保持するためのもの、コンピューティング=プログラム処理の実行基盤)
- まったく新しいワークロード
- サービスの継続的な機能強化





# サービス多様性の基礎



- リージョン：法などの規制を考慮した地域（複数のアベイラビリティゾーンからなる）
- アベイラビリティゾーン：レイテンシを考慮したデータセンター群
- エッジロケーション：キャッシュコンテンツデリバリーネットワークの拠点（WAFなど）



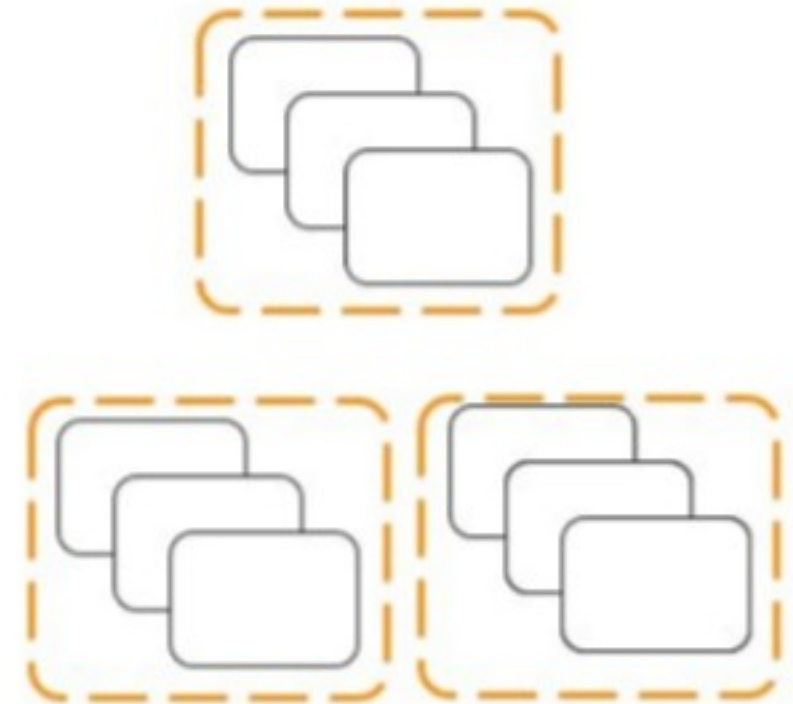


- 従来のインフラストラクチャでは、**単一の物理設備内**でサービスが稼働することを前提としたうえでの可用性の向上がはかれてきました。
- AWSの設計においては、**アベイラビリティゾーン**と呼ばれる**複数データセンターによる冗長化設計**のインフラストラクチャへの組み込み、仮想化技術によるコンピューティング資源の増減による可用性の確保など、**AWS自体が高可用性を実現するための機能を有しています。**

単一のデータ  
センター内での  
可用性確保



複数のデータ  
センターを前  
提にした可  
用性確保



可用性の組み込み例：アベイラビリティゾーン（AZ）  
物理的に隔離された複数のデータセンター上でサービスを提供  
することを前提としたインフラストラクチャ設計







## Amazon Simple Storage Service



Object



Bucket



General  
access  
points



Bucket with  
objects



VPC  
access  
points

## スケーラビリティ確保

- EC2であれば可用性は利用者の設計に依存（インスタンスを複数のAZに配置し、分散処理）
- サービスによっては複数のAZ展開が自明のものも
- EX. Amazon S3 (オブジェクトストレージサービス)
- 容量は無制限
- 99.999999999%の耐久性（S3に1万個のデータを保存した場合でもそのうちの1データが障害によって失われる可能性は平均で、1000万年に1度）





# サーバレスアーキテクチャ



Upload your code to AWS Lambda or write code in Lambda's code editor



Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity



Just pay for the compute time you use

- インフラ運用からの解放
- Lambda：イベントドリブンとコスト効率化
- マイクロサービスの実現



👉 ACM で証明書を作成 → HTTPS 接続する



© 2020 Amazon Web Services, Inc. or its affiliates. All rights reserved.



# 応用例

- S3は静的ウェブサイトホスティング環境になる
- 利用者はWebサーバの管理不要
- サーバレス≡サーバを“意識”しないアーキテクチャ







# 応用問題

皆さんが実施したハンズオン、EC2上の静的なコンテンツはS3に配置することもできます。

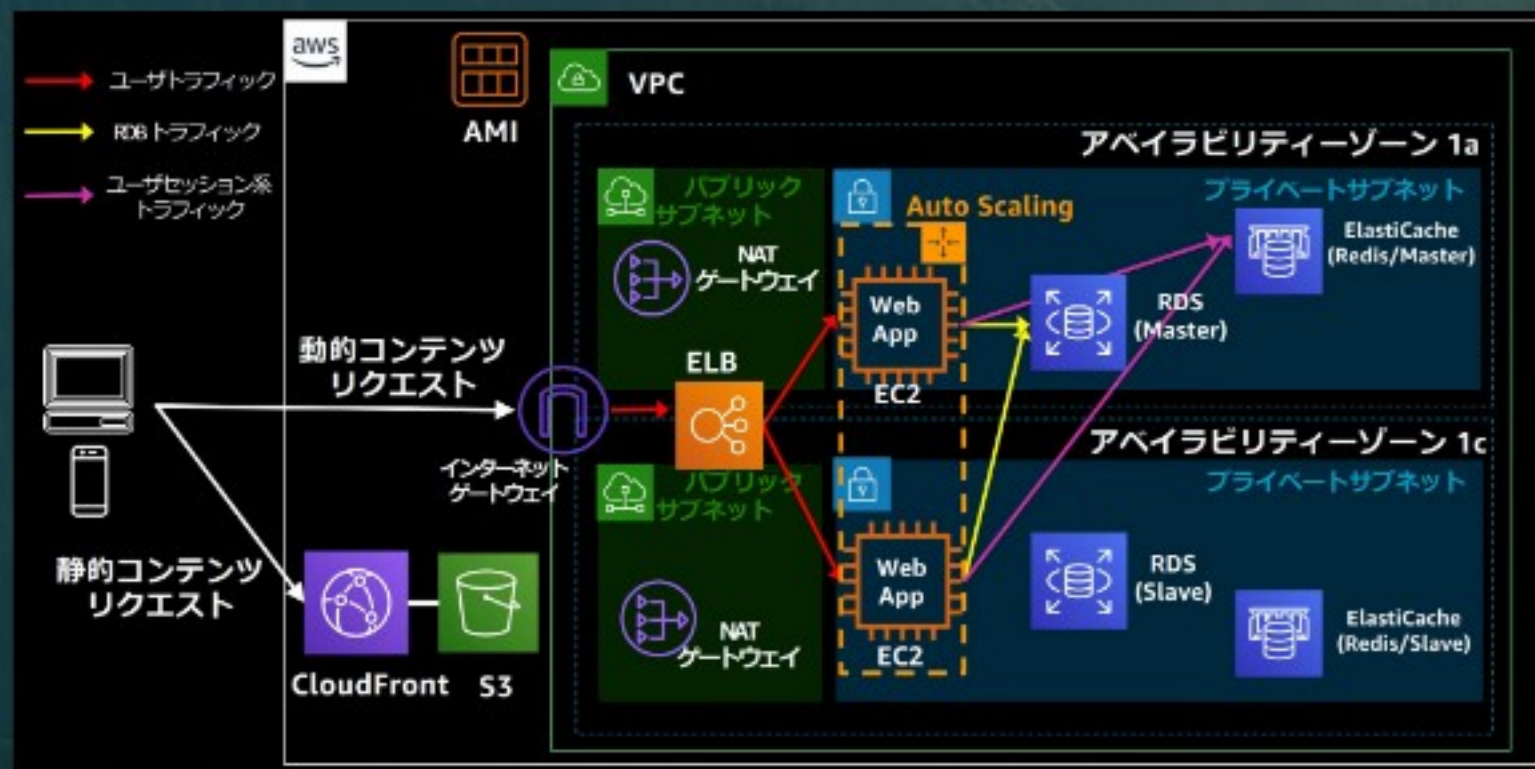
しかし、S3は3つのAZを前提として作られているサービスであり、VPC内には存在しません。

その場合、どうやってアクセスするのでしょうか。そのリスクは？





# プライベートなネットワークの意味



- S3やLambdaは基本的にはVPC外に存在
- アクセスコントロールはAWSのIAMで制御
- 経路は暗号化
- さて、リスクは大きいのか？







# 適切なサービス選択による設計のメリット

- パッチ管理、可用性管理などをAWSにオフロード
- サービスの改善やデータセキュリティに注力
- 性質に応じたサービスを組み合わせる“Building block”の発想
- 内外のネットワークの意味？





# \*\*\* as Codeのもたらす世界

Infrastructure as Code, Configuration as Code, Compliance as Codeなど、Code化されたインフラストラクチャがもたらすものは何かを考えていきます。





# マニュアルオペレーションの課題

- 再現性・一貫性
- 操作ミス
- 変化への対応





# クラウドがもたらしたものの

- APIによる共通化（異なるインフラ要素）
- ログによる活動の可視化
- コード化による再現性、拡張性





**Resources:****Ec2Instance:****Type:** 'AWS::EC2::Instance'**Properties:****SecurityGroups:**

- !Ref InstanceSecurityGroup

**KeyName:** mykey**ImageId:** ''**InstanceSecurityGroup:****Type:** 'AWS::EC2::SecurityGroup'**Properties:****GroupDescription:** Enable SSH access via port 22**SecurityGroupIngress:**

- IpProtocol: tcp
- FromPort: '22'
- ToPort: '22'
- CidrIp: 0.0.0.0/0

# CloudFormation (AWSのIaCの基本)

- AWSリソースのコード化によるデプロイ
- 変数による環境構築の柔軟性
- コードベースでのテスト





# コード化された世界とは？

- すべての命令はプログラムとして検証可能
- 変数や条件を変えることによる再利用（開発/本番）
- 同じAPIだから異なるアーキテクチャを統合できる（プログラムからプログラムを呼び出すことも）



# 100の環境があるとして監査戦略をどうやってたてますか？

グループ分けして監査対象を減らす

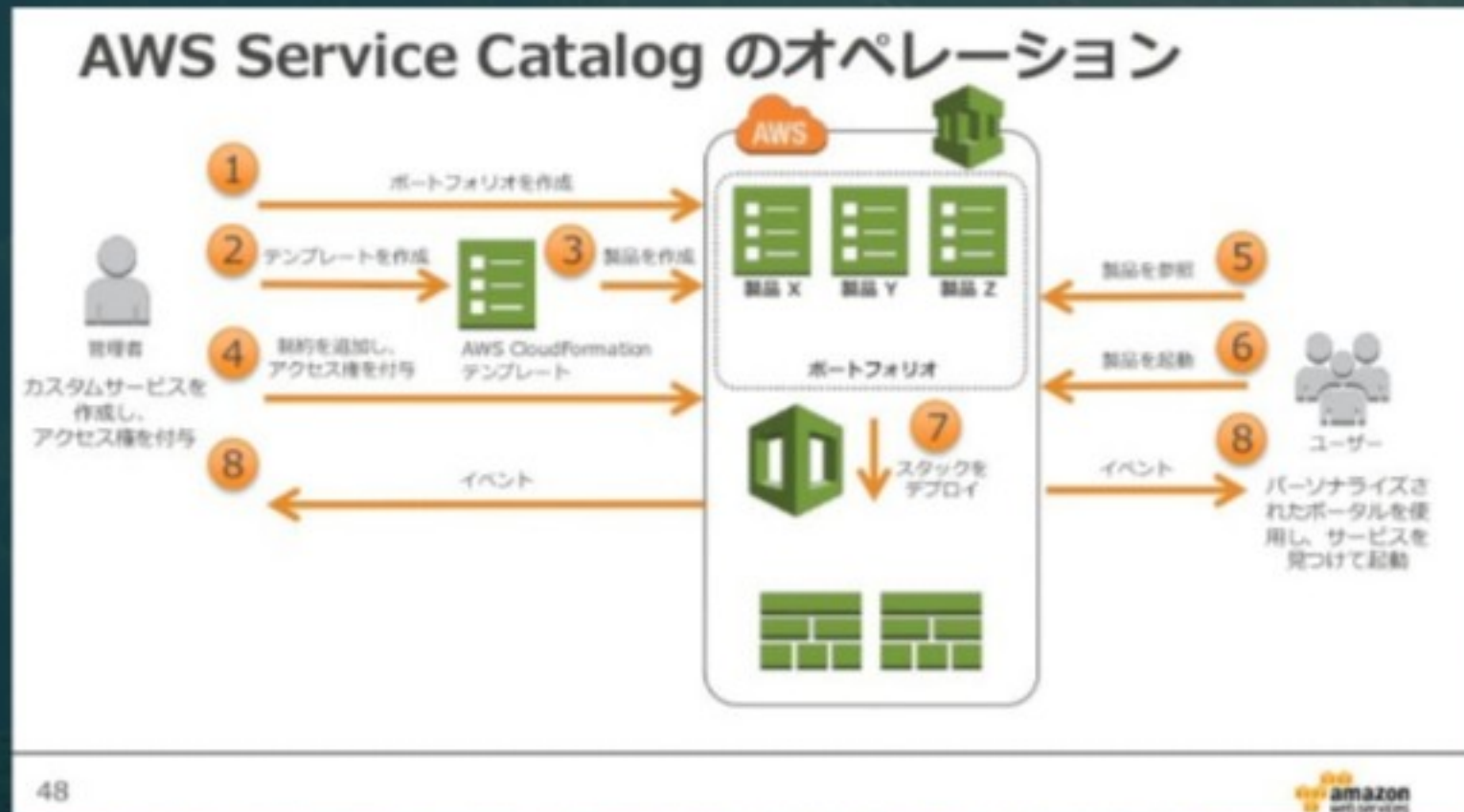
難しいですね。

環境をコード化して、差分を比較し一定の基準をもとに差分のデータを分析評価し、最適かつ効果的に監査できるサンプリング対象を抽出する



# ゆるやかな中央集権

- 利用者に一定の自由度を許容（サービスアジリティ）
- セキュリティをテンプレートに組み込み（Security By design）
- 継続的な監視、ガードレール化
- 環境の一貫性とモニタリングがあればリアルタイムに全件監査も現実的





# 今日はここまで

- 本日はあまり“セキュリティ”を出さずに、その背景を伝えてきました。
- クラウドならではの技術や設計の考え方の延長に“クラウドセキュリティ”が存在します。
- 次回はクラウドによるセキュリティを中心にお伝えします





# 次回に向けて

- Lab2,3の実施をお願いします。
- 本日実施もOKですが、移動がある方はどちらか一つを（どちらも1時間程度の制限）
- Labは後回しでも言い方で雑談をしたい方はWebExのBreakoutroomを用意しますので、クラウドに関するディスカッションを。





# Lab 2

- **Monitoring Security Groups with Amazon CloudWatch Events (日本語版)**
- 目的：サーバレスアーキテクチャを活用したイベント駆動型発見的（訂正的）コントロール
- 概要：Security Groupの変更をCloudTrail（AWSリソースのログサービス）で記録し、Lambdaによる対策を実施





# Lab 3

- Creating an Amazon Virtual Private Cloud (VPC) with AWS CloudFormation
- すみません。英語版しかないかも。
- CloudFormationを使ってVPCを作成、変更
- CloudFormationの立ち上がりに時間がかかることも。