AWS SAP

・Kinesis Data Stream
　ログのヒューリスティックに役立つ（EMRと併用）
　ストリームレコードの時間：**24時間～365日間**

・AD Connector
　既存のオンプレミスMicrosoft Active Directoryのユーザーやグループも割り当てることができる。

・Simple AD
　一般的なディレクトリ機能を備えた低価格のActie Directoryの互換サービス
　モバイルアプリやウェブアプリのサインアップやサインインについて、アクセスコントロール機能を追加できるサービス

・IAMクロスアカウントアクセス
　（アカウント間でIAMロール権限を委任すること）
　ITガバナンスの提供に役立つ
　ロールを使わせてもらう側は、Assume Roleを、ロールを渡す側は、マネージメントコンソールから渡す側のAWS IDを指定して譲渡を許可する。

・Amazon CloudSearch
　AWSクラウドにおけるマネージドがたサービスであり、ウェブサイトまたはアプリケーション向けの検索ソリューションを簡単かつコスト効率よく設定、管理、スケールできる。

・Amazon MQ
　クラウド内のメッセージブローカーへの移行を容易にするマネージドメッセージブローカーサービス。
　SQSと一緒。宅配ボックスのような役割。メッセージを受け取る側はそのタイミングを考慮しなくて良い。
　●SQSとの違い
　　用途：オンプレミス上でメッセージングサービスを利用しているシステムをそのまま移行する場合は、Amazon MQ、クラウドネイティブなサービスを構成する場合は、SQS

・S3の署名付きURL
　署名付きURLの作成者がS3への書き込み権限を持っている場合、署名付きURLを受け取った人はS3への書き込みができる。

・インスタンスプロファイル
　インスタンスプロファイルは IAM ロールのコンテナであり、インスタンスの起動時に EC2 インスタンスにロール情報を渡すために使用できます。

・Amazon Elastic Transcoder
　HLSファイルに変換して配信する機能
　（HLS：Http Live Streamingの略。動画をストリーミング送信するためのプロトコルの一つ）

・S3の暗号化

●SSE-S3

256ビットの高度暗号化規格（AES-256）を使用してデータを暗号化する

●AWS KMS

SSE-S3の機能　＋　誰が、いつキーを使用したかの証跡情報が提供される

アクセス監査ログには使えない。

●SSE-C

Amazon S3にデータを送信する前に暗号化する。

・AWS Storage Gateway ファイルゲートウェイ

インターネット経由でもDirect Connect経由でもどちらでも可能。

オンプレミス側のネットワークからS3へ接続するI/Fを提供するプロトコルのようなサービス。

●ゲートウェイキャッシュ型ボリューム

データはオンプレミス環境に設置したAWS Storage Gatewayキャッシュ内に一時的に書き込まれてそのあとクラウド上のストレージに書き込まれる

●ゲートウェイ保管型ボリューム

ユーザが書き込んだデータはオンプレミス環境に設置したAWS Storage Gatewayのストレージ内にすべて書き込まれ、その後クラウド上のストレージにも書き込まれます。

●仮想テープライブラリ

オンプレミス環境に構築されたAWS Storage Gateway上に仮想テープを構築し、クラウド上でライブラリ管理します。

S3を永続ストレージとしてオフサイトのバックアップを提供する。

ファイルゲートウェイはファイルレベルのストレージを提供し、ライフサイクルポリシーを使用して1年後にバックアップを削除できる

S3へのファイルインターフェースをサポートし、サービスと仮想ソフトウェアアプライアンスを組み合わせる。

・IDS/IPSの導入

インスタンスまたはリバースプロキシ層にIDS/IPSエージェントを実装する。

●AWS Config

AWSリソースの構成変更を取得。

（Amazon VPC、S3、EC2、ALB、AWS X-Ray、CloudFront、WAF、AWS Sheild、DynamoDB、CodeBuild、CodePipelineなど）

・ストリーム：リソースが変更/作成/削除されるたびに作成

・ヒストリー：任意の期間における各リソースタイプの構成要素の集合

・スナップショット：ある時点でのコンフィグレーションアイテムの集合

〇AWS Configダッシュボード

リソースの情報が確認できる（EC2Instanceなど）

〇設定タイムライン

その時点での構成の詳細情報とアタッチされていたAWSリソース

変更のビフォーアフターを確認できる（ボリュームの追加、インスタンス起動、インスタンスタイプ変更など）

イベント時間、ユーザー名、イベント名、イベントの表示など

〇リソースのインベントリ

ターミネート済みのEC2インスタンスの情報を確認

・リソース間の関係（リレーションシップ）

アカウント内のAWSリソース間の関係を管理

双方向の依存関係が自動的に割り当てられる

例：特定のセキュリティグループを使用しているリソース関係をクエリで表示させることができる。

〇AWS ConfigRules

AWS Configで管理する構成情報をConfig Rulesによって評価する。

（EBCボリュームの暗号化がされているか、EC2インスタンスが適切にタグ付けされているか。）

・マネージドルール：AWSにより定義され_ている

マネージドルールのカテゴリ：コンピューティング、データベース、マネジメントとガバナンス、ネットワークとコンテンツ配信、ストレージ

・カスタムルール：**AWS Lambdaをベースにルールを作成可能**、管理自体は作成者で実施

事前にLambda関数を作成（自由にルールを設定することが可能、作成した関数のARNをルールに紐づける、トリガータイプを選択）

①ルール評価実行（AWS Configによって、ルールに基づいたLambda関数が実行される。Lambda関数に対しイベントパラメータがセットされる。）

②評価の通知（Lambda関数の実行結果をAWS Configに引き渡す）

AWS Config Rule Development Kit(RDK)：カスタムルール作成を支援する開発キット

・トリガータイプ：ルール評価実行のタイミング

設定変更：関連リソースが作成、変更された時

定期的：任意のタイミング

・修正アクション：コンプライアンス違反のリソースに対して、**ルールに関連づけられた修正アクション**を実行

（事前入力されたリストから修正アクションを選択、**AWS System Manager Automationドキュメントを使用したカスタム修正アクション設定**）

コンプライアンス違反の検出ログ（Cloud Watch Events）から**Lambdaをトリガーし、より細かい修正アクションを実行可能**

・マネージドルールのユースケース#1：

AWS Config Rules Repositoryにより、Config Rulesを確認できる。

approved-amis-by-id：実行中のインスタンスで使用されているAMiが指定したものかをかくにん

required-tags：リソースに指定したタグがあるかどうかを確認

encrypted-volumes：アタッチ済みのボリュームが暗号化されていることをかくにん

ec2-instance-managed-by-ssm：EC2がAWS Systems Managerによって管理されていることを確認

s3-bucket-public-read-prohibited：S3バケットでパブリック読み取りが許可されていないことをかくにん

access-keys-rotated：アクセスキーが指定日数内にローテーションされるかどうかを確認

・SSMと連携したOS構成情報の管理例：
SSM Inbentoryでソフトウェアの導入状況を確認
AWS Config/Config Rulesでソフトウェアの導入状況を記録・監視
違反を検知したら、通知やサーバーを止めるなどの対処を実施
Config Rulesの修復アクションとしてSSM Automationの実行
Config Rules → SSM Automation → Lambda → SNS

・ベストプラクティス
１.全てのアカウントとリージョンでAWS Configを油口に
２.全てのリソースタイプについて、設定変更を記録する
３.グローバルリソースは１リージョンで記録を有効にする。
５.安全なS3バケットにヒストリーとスナップショットを保存する
１９.Data aggeregation機能を使って、管理アカウントから集中管理する。
２０.Organizationベースのaggregationを使う

・料金
AWS Config：リソースに対して記録された設定項目　１記録あたり：0.003USD
AWS Config Rules：実際に記録されたConfgルールの評価数（リージョンごと）
最初の10万ルールの評価につき：0.001USD
次の40万ルールの評価につき：0.008USD
次の500,001以上のルールの評価につき：0.005USD
追加料金：S3（ログ保存）、SNS（通知）、Lambda（カスタムルール）、SSM Automation（修復）

〇Amazon Simple Notification Service（SNS）
publisher(発行者）からのメッセージをsubscriber(購読者）に送信する

●Mobile Push（プッシュ通知）
モバイルアプリが起動していなくても通知
モバイルユーザーは通知を受け取るか否か選択可能
通知をきっかけにアプリを起動してもらう
●pub-sub
通知もできるが、分散アプリケーションの統合用途に用いられる。
●Topic Owner(所有者）
Topic OwnerがTopicを作成、管理する
Topicは、Publisherがメッセージを送信し、Subscriberが通知を受信するための通信チャネルとして機能する。
●Subscriber(購読者）
どのトピックを受け取るかをフィルターにより制限することができる。
●Publisher(発行者）
発行したいTopicにmessageを発行する
●配信プロトコル
http/https：購読登録時にURLを指定する。
Email：テキストがEメールとして、登録されたアドレスに送信される

Email-JSON：通知がJSONオブジェクトに送信される
Amazon SQS：指定されたキューにmessageを投入する。(FIFOキューは対象外)
AWS Lambda：AWS Lambda関数にmessageを配信する。
Platform application endpoint：サポートされているプラットフォームにpush通知する。
SMS：SMSテキストメッセージとして、登録された電話番号に送信される。

●シンプルなAPI
✔Topic Ownerのオペレーション
CreateTopic：新規Topicの作成
DeleteTopic：以前作成されたTopicの削除
ListTopics：所有されたTopicのリスト
✔Subscriberのオペレーション
Subscribe：特定のTopicに新規Subscriptionを登録
ConfirmSubscription：Subscription確認メッセージへの応答
UnSubscribe：登録済みのSubscriptionの解除
✔Publisherのオペレーション
Publish：Topicに新しいmessageをpublish

●SubscriberがFilter Policyを設定する。
WhiteListとBlackListがある

●raw messageの配信
Subscriberは、Amazon SQSおよび、Http/sへの配信にraw形式、つまりPublishされた通りにmessageが配信されるようにするオプトインすることができる
default、raw message

●購読プロトコルに合わせた発行
publisherはsubscriberの購読プロトコルごとに、messageをカスタマイズして発行できる。

●SNSのRetry Policy
SNSに送信された全てのメッセージは、直ちに配信される

〇AWS Step Functions
●モダンなアプリ開発のベストプラクティス
プロセスはステートレスかつシェアードナッシングである。
●概要
ステートマシンと呼ばれる仕組みでオーケストレートできる
ワークフローという形式で見やすく可視化できる
複数種類の同時に実行が可能
●ステートマシン
処理プロセスの流れ

●ステートマシンの実行方法
・Amazon CloudWatch Events
・Amazon API Gateway：あるAPIが呼ばれた際のバックエンドとして実行
・マネジメント　コンソール：コンソールから手動で実行

・AWS CLI：コマンドラインから実行
・各種SDK：LambdaやEC2に実装したアプリケーションから実行
●ステートマシンから呼び出すことが可能なAWSサービス
・AWS Lambda：Lambda関数の実行
・Amazon DynamoDB：既存のアイテムの取得、新規アイテムの登録
・AWS Batch：ジョブの起動、ジョブ完了の待機
・Amazon Elastic Container Serice：ECS/Fargateタスクの実行など
●Activity
サーバーやコンテナなどに実装したアプリケーションからポーリングすることで、独自定義の処理を実行する仕組み。
（1）独自に実装したアプリケーションからStep Functionsを定期ポーリングする
（2）ステートマシンを実行する
（3）Activityの Stateに達するとポーリングに対して応答が返却される
（4）アプリケーションで処理を実行し、結果をステートマシンに通知する。
●データの入力
・inputPath：渡された入力のうち、何をState内で使用するかを指定
・Parameters：渡された入力から、「キーと値」のペアを生成してStateに渡す

●ユースケース
・データプロセシング：複数のデータストアを利用する分析や機械学習
・eコマース：在庫追跡や注文処理
・動画処理：サムネイル作成、ビデオのエンコーディング
・バッチ処理：ゲノム情報解析のような学術領域
・ウェブアプリケーション：複雑なユーザー登録プロセス
・管理者承認：管理者が承認した場合に限り処理を行う

〇Amazon API Gateway
●APIのチェックポイント（窓口）
●Application Programming Interface：プログラムやソフトウェア同士がやりとりするための取り決め・仕様
●Web API提供時の課題
1. インフラの管理
2. APIの管理：設定やデプロイの制御
3. 認証と認可：アクセスの制限
4. 流量制限と保護（スロットリング）

●用途
1. インターネットからアクセス可能なパブリックなWeb APIの基盤を提供する
2. 自社企業・企業グループ内でのプライベートなWeb APIの基盤を提供
3. AWSサービス（例：Amazon DynamoDBなど）を独自のWeb API化する手段として利用する。
4. サーバレスアーキテクチャを実現する手段として利用する。

●API Gatewayが扱うAPI
・REST(Representational State Transfer)（ステートレス）
単一のHTTPメッセージで1つの操作に関する情報を含む
扱う情報をURIで表現する「リソース」として定義し、それらをHTTPメソッド

（PUT,GET,POST,DELETE,・・・）の表現で操作
　　３種類のエンドポイントタイプから１つを選択（**エッジ最適化（キャッシュを保持）**、リージョン、プライベート）

・WebSocket（ステートフル）
　HTTPの上で、クライアントとサーバー間の双方向通信を実現するための通信使用
　　1つのコネクションで継続的なデータ送受信が可能
　　URIスキームはセキュアWebSocket用の「wss://~」を利用

●REST API作成の流れ（新規作成）
「既存APIのクローン」または「Swagger(OpenAPI)ファイルのインポート」による作成が可能
①APIの設計（仕様の確認）：作成するAPIの要求・仕様を確認
②プロトコル：API仕様に基づきREST or WebSocketを選択
③-a：リソース＆メソッド設定：プロトコル種別に応じたAPIの設定を実施
③-b：ルート設定：設定後、APIとしてのテスト呼び出しが可能
④デプロイとステージ設定：デプロイ操作によりクライアントから呼び出し可能な状態へ、ステージ（APIが動作する環境）を指定
　　ステージ名はエンドポイントURL（パス）の一部として利用される。

●REST API設定（REST）リソースとメソッド
「/」を最上位としたツリー構造にて、「りそーす」を定義
　各リソースに受け付けるHTTPメソッドを指定

●API設定(WebSocket)-ルート
WebSocketの状態に関するイベントを「ルート」として定義
3つの事前定義ルートに加えてメッセージに応じたルートを指定可能
API（WebSocket)
　ルート選択式（例）：$request.body.action

●API設定 -認証・認可
①IAMアクセス権限：AWS署名 v4による認証
②Lambdaオーソライざー：認証処理を委任するLambda関数を指定してAPIへのアクセスを制限
③Cognitoオーソライざー：指定したCognitoユーザープールをもとに事前に認証

●API設定 -統合タイプ
Lambda関数：Lambda関数を指定して呼び出す
HTTP：インターネット経由で呼び出し可能なURLとメソッドを指定して呼び出す
Mock：モックとしてAPI Gatewayで直接、固定的な応答を返す
AWSサービス：AWSのサービスを直接呼び出して、各サービスで用意されているアクションを実行
VPCリンク

○Amazon CloudFront

●Edge Service

エッジロケーションから提供されるサービス群
（Lambda, Route53, AWS Shieldなど）
187PoPs（176エジロケーション＋11リージョナルエッジキャッシュ）

●Webアクセスの課題

✔インターネット経由でのアクセスにおけるネットワーク遅延の影響
・距離に依存n
・コンテンツの元サーバーが遠いと、応答に時間がかかる。
・応答時間の多くが、ネットワーク転送の待ち時間を占める場合も

✔大量のアクセスへの対応
・変化しない静的なデータが多く含まれる（画像、動画、CSS、JavaScriptなど）
・同じデータを何度も酒盗するのは、ネットワーク帯域、サーバーリソースの無駄

●特徴

大容量キャパシティを持つ地理的に分散したサーバー群からコンテンツをキャッシュしたり代理配信するサービス
DNSを応用した仕組みで最適なエッジロケーションを割り当て
・高性能な分散配信
・高いパフォーマンス
・キャパシティアクセスからの解放
・ビルトインのセキュリティ機能（AWS WAF連携、DDoS対策）
・設定が簡単で即時利用可能
・充実したレポーティング(ログ、ダッシュボード)
・完全従量課金製

●リージョナルエッジキャッシュ

オリジン　→ Regional Edge Cache　→ エッジロケーション

●CloudFrontコンテンツ配信設定の流れ

distribution
・ドメインごとに割り当てられるCloudFrontの設定
・AWS Management ConsoleもしくはAPIで即時作成可能

１. Amazon S3バケット、ALB、EC2、オンプレミスにある独自のHTTPサーバーなどのオリジンサーバーに設定
２. ファイルをオリジンサーバーにアップロード
３. CloudFront distributionを作成
４. CloudFrontがドメイン名を割り当て
５. distributionの構成を全てのエッジロケーションに送信

○AWS Database Migration Service

●概要

DBのデータ連英を支援するサービス
セットアップ・利用が安易
使った分だけの従量課金製
テーブル定義を補助するツール提供
低負荷で継続的なレプリケーション

●一般的なレプリケーション方式

✔物理レプリケーション

・トランザクションログファイルをターベットに転送

・データベース単位

✔論理レプリケーション

AWS Database Migration Serviceはこっち

・トランザクションログファイルからDMLを作成し、ターゲットに適用

・テーブル単位

・差が発生する可能性がある。（Ex. ターゲット側への更新エラー、文字の文字化け）

・参照、更新が可能

●DMSの連携対象

・テーブル、制約（インデックス、プロシージャ、ビューは連携されない）

・DMSで作成されるオブジェクトはテーブルと主キーなどの一部の制約のみ

・DRのような同一構成を維持するためには、ソース、ターゲット両方のデータベースにオブジェクトを作成する必要がある。

●DMSアーキテクチャ

✔フルロード

既存データの連携

①ソースのデータベースに対してselectを実行し、データを暗ロードする。テーブルは複数テーブルを同時に暗ロード

②ターゲットのDBに対してデータをロード。

✔CDC(Change Data Capture)

更新差分データの連携

①ソースで発生したデータ変更は受信バッファにキャプチャされる

②CDCにおいて、ソースからターゲットへキャプチャされたデータ変更を仲介するバッファ

③送信バッファにキャプチャされたデータ変更をターゲットへ適用

●DMSアーキテクチャ（ソースがOracleの場合）

✔LogMiner

・オンラインRedoログとアーカイブRedoログファイルを読み取るOracle API

・DMSのデフォルト

・ログの解析はソース側のDBサーバー

・ネットワークの転送量は指定したテーブルのみ

・ASM仕様有無に依存しない

✔Binary Reader

・ORDER_LINEテーブルを指定した例

・①要求、②転送、③更新情報取得

・ログの解析はレプリケーションインスタンス

・ネットワークの転送はログファイル全体

・ASM仕様有無で内部処理が異なる

●レプリケーションインスタンスのサイジング
テーブル数、データ量、変更差分の量など様々な要因に依存するため、事前に決定することはむずい。
（CPU、メモリ、ディスク、ネットワーク）
・CPU、メモリ
CPU：FULL Loadによるテーブルデータの並列転送、大量のトランザクションログの更新
メモリ：FULL Loadの結果キャッシュ、CDCのキャッシュ
・ディスク：メモリ不足時に、バッファ情報をファイルに出力、FULL LoadのデータをCSVに出力、出力したCSVをターゲットにImport
・ネットワーク：Full Loadのデータ転送、CDCの転送

●DMSの機能
DMSによるレプリケーション対象の制御
・選択ルール：行のフィルタ、列の削除（remove coloum:stocksなど）
・変換ルール：テーブル名の変更、列名の変更、ターゲットのテーブルに式を使用して列を追加
移行後のデータ比較
・タスクの検証（Full Load時、CDC時どちらでも可能）
Full Load時の検証：データをパーティションという単位で分割し、分割したパーティションごとに検証
パーティションはデフォルトで50000
CDC時の検証：変更された行のみ検証

●トラブルシューティング
CloudWatchログを有効化することを水晶

●AWS CodeDeploy

〇デプロイ手段
・Push型：Deploy先サーバーを知っている必要がある
・Pull型（推奨）：各インスタンスが必要な変更をPull

〇主要コンポーネント
・アプリケーション：アプリケーションを一位に識別する
・コンピューティングプラットフォーム：アプリケーションがデプロイされるプラットフォーム（Lambda、EC2など__）
・デぷろいグループ：デプロイ環境の定義、AutoScalling、
・デプロイタイプ：In-Place、Blue/Green
・デプロイ設定：どのようにデプロイするかを定義したもの、デプロイする割合
EC2：One-at-a-time、Custom、Half-at-a-time、All-at-once
Lambda/ECS：Linear、Canary、All-at-once
・リビジョン：
EC2：ソースコード、Webページ、スクリプトなどとAppSpecファイルをまとめたアーカイブ
Lambda：Lambdaデプロイ用のAppSpecファイル

ECS：ECSデプロイ用のAppSpecファイル
・ターゲットリビジョン：リポジトリにアップロードした直近のリビジョン
・サービスロール：CodeDeployに付与するIAMロール
管理ポリシー：AWSCodeDeployRoleなど
・IAMインスタンスプロファイル：EC2インスタンスに付与するIAMロール、S3から配布物を取得できるようにする。

〇EC2/オンプレミスのデプロイ
・**AWS CodeDeploy Agentの導入が必要**
・オンプレミスインスタンスへは、In-Placeデプロイのみ
・デプロイグループにAuto Scalling Groupを指定することで、スケールアウト時に最新のリビジョンが自動でデプロイされる
・ライフサイクルイベントへHookを指定し、スクリプトを実行可能
・Hookが失敗した場合やAmazon CloudWatchアラームを検知した場合は、数秒で迅速にロールバック

〇リビジョンの構成
✔フォルダ構成
・appspec.yml(必須)
・ビルド済みの成果物
・そのほかの配布物
・Hookスクリプト
✔アップロード先
・**Amazon S3**
・**GitHubのリポジトリ**

〇CodeDeploy Lambdaのデプロイメント
AWS Lambdaの関数重みつけエイリアスを利用したトラフィックのシフト
カナリアデプロイやリニアデプロイを選択可能
Validation Hookは各ステージへのデプロイ時のテストを有効化
Hookが失敗した場合やAmazon CloudWatchアラームを検知した場合は数秒で迅速にロールバック

〇機能
・Validation Hook Lambda
・CloudWatchアラームによるデプロイ停止
メトリクスやログを監視し、デプロイを停止することできる
・ステータスの通知
通知ルール、デプロイトリガーを設定してデプロイ状況の通知を受け取ることできる
・VPCエンドポイントをサポート
EC2へのデプロイの場合は、codedeploy、code deploy-commands-secureの両方が必要
Lambda、ECSの場合は、codedeployのエンドポイントのみ必要

〇りょうきん
・**EC2/Lambda/ECSのデプロイは無料**

・オンプレミスのインスタンスに対するデプロイは、0.02USD/インスタンス/デプロイ

●AWS CodeBuild

継続的なスケールと同時複数ビルドプロセス
サーバーの管理は不要
利用した分のみの支払い
CloudWatchによるモニタリング可能
・一貫したイミュータブルな環境のために個々のビルドを新規Dockerコンテナで実行

〇AWS CodeBuildプロジェクトの作成
・プロジェクト名
・ソース選択（S3,CodeCommit,Githubなど）
・Build環境：OS、ランタイム、Dockerイメージなど
・buildspec
・ログ設定：CloudWatch Logsへログを送信する
・ソースの作成：buildspec.yml（buildspecバージョン、ランタイムにjavaを指定、jarファイルを作成、ビルド出力を定義）

〇buildspec.ymlについて
version（必須）：buildspecのバージョン
0.2の使用を推奨
run-as：コマンドを実行するLinuxユーザ（指定したユーザーに読み取り及び実行権限付与、指定しない場合、全てのコマンドがrootユーザで実行される）
env：環境変数（AWS Systems Manage Parmeter Storeの値、AWS Secrets Manager値を指定可能）
proxy：プロキシサーバー（アーティファクトのアップロード時、CloudWatch Logsへのログ送信時にプロキシサーバーを利用するか個別に定義可能）
batch：バッチビルド設定
batch/build-graph：バッチ内の他のタスクに依存する一連のタスクを定義（前のタスクが終わらないと次のタスクにすすまない）
batch/build-list：同時に実行するタスクを定義
batch/build-matrix：様々な環境と並行して実行されるタスクを定義
phases（必須）：実行するコマンド
ビルドの各段階でCodeBuildが実行するコマンドを記述する。
（install、pre_build、build、post_build、Docker）
reports：テストレポート作成
artifacts：AWS CodeBuildの出力
アーティファクト名、アーティファクトに含めるサブディレクトリとファイルを指定
cache：キャッシュ設定
キャッシュタイプはビルドプロジェクトに対して設定（キャッシュなし、S3、ローカル）

〇機能
・ビルドをローカルで実行する
AWS CodeBuildエージェントを使用して、ローカルマシンでビルド可能
（GitとDockerが必要）

・AWS Session Managerでビルド環境へアクセスできる
マネージメントコンソール、CLIを介してLinux、Windowsのビルド環境にアクセス
・VPCないリソースへのアクセス
固定IPアドレスを必要とする外部サービスへの通信にNATゲートウェイ、NATインスタンスのEIPを使用する。
※インターネットゲートウェイを選択することはできない

●AWS CodeStar & AWS CodePipeline

○AWS CodeStarの概要

・数分間で開始
・チームを跨った開発をセキュアに
・ソフトウェアデリバリーの管理を簡単に
・豊富なプロジェクトテンプレート
✔プロジェクトダッシュボード
統合メニュー、Repositoryエンドポイント、デリバリーパイプライン、Appエンドポイント、CloudWatchメトリクス、開発ツールとの連携
Source、Build、Deploy→ステージとして、ソース・ビルド・デプロイが構築される
✔開発ツールとの連携
Eclipse、Visual Studio、AWS Cloud9などとの連携
✔セットアップ内容
プロジェクトテンプレートの選択（EC2 or Beanstalk or Lambda）
プロジェクトの設定（レポジトリ（CodeCommit、Github））
プロジェクトの編集（プロジェクトダッシュボード、CodePipeline継続的デプロイメントパイプライン、CloudWatchメトリクス）
✔料金
サービス自体に料金は発生しない

○AWS CodePipelineの概要

・継続的デリバリーサービスを手きょう
・ソフトウェアリリースプロセスをモデル化する
・AWSサービスやサードパーティーのツールと統合
・開発スタイルに合ったパイプラインを自在に構築

○定義
ソース（ステージ）→ビルド（入力アーティファクト、アクション）→検証環境デプロイ（出力アーティファクト）→テスト実行（トランジション）→本番環境デプロイ
○パイプラインの作成
・パイプライン名を指定する
・任意のサービスロールを指定する
・高度な設定
・ソースプロバイダーを指定する
・ソースプロバイダーに紐づく情報を指定する（例：AWS CodeCommitのリポジトリ名やブランチ名をトリガーとする）
・ビルドステージを追加する（オプショナル）

・デプロイステージを追加する（オプショナル）
・レビュー（パイプライン構成を確認）
〇パイプラインの実行履歴（最大12ヶ月間確認可能）
〇パイプライン実行結果の通知
通知はSNSで送信される（例：CodeCommit → SNS → AWS Chatbot → Slack）
〇AWS Lambdaファンクションの実行
利用例：AWS CloudFormationを使用して、任意のタイミングでリソース作成・削除
LambdaファンクションでCNAMEをスワップすることで、ゼロダウンタイムでのバージョンアップをElasticBeanstalkで実行
AMIスナップショットを作成することで構築、デプロイする前にリソースのバックアップを作成
IRCクライアントにメッセージをポストするなど、サードパーティーの製品を統合

〇Step Functionの実行
利用例：条件付き分岐、エラー処理、非同期タスクを行う処理
他のAWSのサービスとの連携
シンプルなリソースパイプラインを維持し、複雑なワークフローの動作をStepFunctionsステートマシンエンジンに委任する
〇Approvalアクション
パイプラインに承認アクションを追加し承認されたら次のステージへ
〇実行時のアクション間の変数引き渡し
利用例：CodeCommitソースアクションが出力するコミットメッセージを元に、手動認証操作のためのカスタムメッセージを定義する
〇構成例
Amazon ECSでのCI/CDパイプライン
サーバーレスでのCI/CDパイプライン
Github、Jenkinsとの連携
〇料金：
アクティブパイプライン1つにつき：1.00USD/月
実験を奨励するため、パイプラインは作成後の最初の30日間は無料

●CloudFront Deep Dive
〇リージョン別エッジキャッシュ（REC）
ユーザーからのリクエストをRECで集約し、オリジンにリクエスト
エッジロケーションに近いRECを利用
〇オリジンインフラの保護
同時に大量のリクエストが発生した場合に、最初のリクエストのみをオリジンに送り、負荷低減を実現する仕組み
〇用語集
✔viewer：クライアント/Webブラウザ
✔Distribution：コンテンツ配信の設定単位
✔CloudFront：ドメイン名
・Origin：コンテンツ提供元のWebサーバーごとに作成

・Behavior：キャッシュ動作設定、URL パスパターンごとに作成

〇CloudFront 設定

1 . Distribution に関連するリソースの準備と設定

Route53 ホストゾーン、AWS Certificate Manager(ACM)SSL/TLS 証明書

HTTP/1.0、HTTP/1.1、HTTP/2m、WebSocket に対応

TLS1.3 クライアント接続に対応、デフォルトで有効化

CNAME エイリアスを利用して代替ドメイン名の指定が可能

AWS WAF 連携（WAF で定義した Web ACL を Distribution に適用）

✔AWS Certificate Manager との統合

新規の SNI(Server Name Indication) 証明書を数分で発行し、CloudFront コンソールから直接デプロイ

生成された SNI 証明書は無償利用が可能、生成された SNI 証明書は自動更新が可能、既存証明書のインポートも可能

（※SNI:一つのサーバーで、複数のドメイン名を使用する場合に、複数の SSL 証明書を運用できるようにする SSL/TLS の拡張子）

✔レポート / ログ / モニタリング

CloudFront → Management Console
→ CloudWatch
→ S3 → Amazon Athena → Amazon QuickSight

2 . Origin に関連するリソースの準備と設定

・カスタムオリジンの Web サーバー（オリジンのドメイン名を指定）

・S3 オリジンの S3 バケット (OAI)

・Origin Shield：オリジンへのリクエストを低減する機能

・オリジンの通信ポリシー：CloudFront エッジとオリジン間の通信方式

✔オリジンサーバーの保護

S3 は OAI

カスタムオリジンの場合は、ALB のカスタムルールにて、HTTP ヘッダーのチェックが可能

✔オリジンフェイルオーバー

オリジングループを作成し、プライマリ・セカンダリオリジンを指定

3 . Behavior に関連するリソースの準備と設定

✔Cache Policy、Origin Request Policy、Behavior 概要

Min TTL、Max TTL、Default TTL、Headers、Cookies などの指定が可能

✔キャッシュコントロール機能

GET/HEAD/OPTION のリクエストがキャッシュ対象

Cache Policy/ Origin Request Policy

オリジンに転送さうるリクエストとキャッシュキーを分離

4 . Distribution に関連する機能の設定

●AWS Security Hub

✔Security Hub セキュリティ基準の有効化

業界標準やベストプラクティスに基づいた自動コンプライアンスチェック

・AWS Foundational Security Best Practices v1.0.0

AWSセキュリティ専門家により定義された統制項目
・CIS AWS Foundational Benchmark v1.2.0
Center for Internet Securityが定義した要件の一部に対してチェックをする
・PCI DSS v3.2.1
クレジットカード情報を保存・処理・転送する組織が従うセキュリティ標準であるPCI DSS要件の一部に対してチェックする

✔カスタムアクションによる自動化
・Security Hubカスタムアクション
・Lambda →

●AWS Cloud Formation

EC2やELBといったAWSリソースの環境高地s区を、設定ファイルを元に自動化できるサービス
JSONやYAMLフォーマットのテキスト形式で記述する
〇基本機能
・作成：依存関係がある場合は自動で解決
・変更：Change Setを作ることで差分内容を事前に確認可能
・削除：依存関係を解決しつつリソースを全て削除

〇StackSets
一回の操作で複数のアカウントやリージョンへスタックを作成、更新、削除できるCloudFormationの機能

〇運用
・更新：直接更新と変更更新
・スタックとリソースの保護：スタックやリソースが誤って変更/削除されないように保護するための機能
スタックポリシー：Denyルールなど
リソースのDeletion Policy：スタックが削除されたときにリソースをほぞ又はバックアップ（Delete , Retainm, Snapshot）
・ライフサイクル別のテンプレート管理：規模が大きくなると単一のテンプレートでの管理は手間と時間がかかり、変更時の影響範囲も大きくなるため、スタックを分割
スタック分割：Application Layer（EC2, RDS、SQS、CICDサーバなど）、Security Layer(IAMロール、セキュリティグループなど)、Network Layer（VPC、サブネットなど）
・CloudFormationリソースインポート
手動で作成したAWSリソースヲCloudFormationスタックにインポートして管理可能。
✔ヘルパースクリプト
・cfg-init
EC2インスタンス内でパッケージの椅子トールなどを実施できるヘルパースクリプト
・can-get-metadata
・cfn-signal
EC2が正常に作成、更新されたかをCloudFormationに送信する
・can-hub

CloudFormationで作成されたリソースのメタデータの変更を検知し、変更が検出されたときにカスタムフックを実行するために使用するデーモン

✔Dynamic References

AWS Systems Managerのパラメータストアと AWS Secrets Managerに格納されたデータを動的に参照

✔各アカウントにベースラインを展開する

・スタックセットの作成

（リージョン、展開先、デプロイオプションなどを指定（OUIへのデプロイができるがアカウント単位でも指定可能））

作成後：ドリフトステータスの確認、パラーメーターの更新もできる

アカウント作成時にスタックを指定できる、

・ベースラインの例

・ ControlTowerの設定（CFnテンプレートから自動でスタックが作成される）

・ロギングの有効化

・各アカウントの管理用IAMロールを作成

Organizations SCPで以下を制限

Role名にbaseline-*がついたロールの変更禁止、CloudTrailおよびConfigの設定変更禁止

✔StackSetsの実行権限（マネージド）

Organizationsで信頼されたアクセスを有効にするとStackSetsのマネージドロールが自動生成される

・ StackSets管理アカウント側：AWSServiceRoleForCloudFormationStackSetsOrgAdmin

・ターゲットアカウント側：stack sets-exec-*

✔セルフマネージド

それぞれ名前固有のロールを作成して必要な権限を渡す（Organizations不要）

・ StackSets管理アカウントがわ：AWSCloufFormationStackSetsAdministrationRole（名前固有）

・ポリシー：AWSCloufFormationStackSertsExcectionRoleを引き受けられるように設定

・信頼関係：cloud formation.amazonaws.comを受信

・ターゲットアカウントがわ：AWSCloudFormationStackSetsExcectionRole（名前固有）

・ポリシー：ターゲットアカウントでリソース作成に必要な権限を付与

・信頼関係：管理側アカウントIDを信頼

✔任意の処理を追加する（Custom Resources）

CFnの中からLambdaを実行する

課題：CFn上に状態が記録されないため、できればリソースプロガイドによる実装が必要

✔スタック作成時にテンプレートを加工する（マクロ）

テンプレートの記述量を減らすため、Lambdaでテンプレートを加工する

CFnスタック作成処理実行前に処理

例：AWS::Serverless(SAM)、AWS::Include

✔スタック作成権限とリソースの保護

CFn実行時の権限はcreate-stack APIを呼び出したユーザ/ロールの権限と同じ
・サービスロール指定により、別権限でStackを作成させることが可能
リソースの保護
・スタック削除保護
・スタックポリシー
・リソースの保護

●Amazon Kinesis Video Streams

●ストリーミング形式
✔メディア形式で収集
・数百万台大規模のデバイスからのセキュアなデータ取り込み
・クラウド保存できる
・プロデューサーから受け付けたデータ（ストリーム）からコンシューマーへ再生
・プロデューサーとストリーマーは１対１
・ストリーマーとコンシューマは１対N
・フォーマット：コンテナ、コーデック
・コンテナ：映像データと音声データをまとめて動画データにするためのファイルフォーマット(MP4,AVI)
・コーデック：コンテナに入れるデータを圧縮するためのアルゴリズム
・メディア取り込み
１. PutMedia API
MKVフォーマットのフラグメントをPayloadとして、HTTPS POSTする
２. Amazon Kinesis Video Streams Producer SDK利用
デバイス上のハードウェアメディアパイプラインと統合するためのSDK
Platform Independent Layer、Wrapper Layer、Dockerイメージ、GStreamer
GStreamer：オープンソースのマルチメディアアプリケーション開発用フレームワーク
でもアプリケーションsyん：Producer SDK C++（映像ソースからAmazon Kinesis Video Streamsに動画を送信）
C Producer

・フラグメント：短時間のフレームをまとめたシーケンス
・チャンク：ストリーム内でのデータの格納形式
・フレーム：動画の元となる1コマの静止画像で、フラグメントに含まれる
✔通信のセキュリティとアクセス許可
SSLでの暗号化、IAMでのアクセス許可
✔データの暗号化
常にサーバーサイド暗号化が有効になっている
ストリーム作成時に暗号化に使用するAWS KMSカスタマーマスターキーを指定できる（後から変更はできない）
ストリーム作成時にユーザー指定のキーが指定されていない場合は、規定のキー（Kinesis Video Streamが提供）が使用される
✔保存期間
０～１０年
✔HTTPライブストリーミング(HLS)再生機能
ライブストリーミングや動画ビューワーをコンシューマーの実装なしで実現

・流れ：プロデューサー→Kinesis Video Streams→HLSストリーミング→スマホ

✔Amazon Kinesis Video Streams MEdia Viewer
HLSもしくはDASHを利用したメディア再生のイメージ
✔同時に再生できるセッション数の上限
✔ライブ再生時のレイテンシー
必ずテストを行なって評価する
✔レイテンシーの考え方
デバイスのメディアパイプライン：イメージセンサーからのデータ読み取り
✔メディア分析
GetClip APIでのMP4位っぷの出力
Amazon Rekognistion など

●WebRTC
リアルタイム双方向メディアストリーミング
クラウド保存で機内
低遅延

○AWS Single Sign-On(SSO)

・IDフェデレーションをもっと簡単に使用できるサービス
・Organizationsとの連携などの仕組みあり。
・AWSアカウント以外のサービスとの連携もできる。

○シングルサインオン（SSO)
●マルチアカウントのアイデンティティ管理
✔シングルサイトオンの考え方：
・利用者目線：一つのIDで複数のAWSアカウントを利用できるため利便性が向上
・管理者目線：IDが1つに統合されたため管理がシンプルに
●シングルサインオンとIDフェデレーション
シングルサインオン：1度のユーザ認証処理によって、独立した複数のソフトウェアシステム上のリソースが利用可能になる特性
IDフェデレーション：シングルサインオンを実現する方式の一つ。一つの組織を超えて他の管理ドメインのサービスにもログインできるようにする処理のこと。SAMLなどの標準技術を適用して実現することが多い。
パスポートのようなもの。IDプロバイダーでアサーションを発行して、サービスプロバイダーにアサーションを提示する。
●AWSにおけるシングルサインオン
IDプロバイダー（IdP）でユーザーを認証し、サービスプロバイダー内のIAMロールのセッションを用いてアクセス許可をする

●SSO導入に必要なタスク
1. アイデンティティストア
IDプロバイダー向け被参照設定

2. IDプロバイダー（IdP）
アイデンティティストアの参照設定

アクセス権限セットの設定

●AWS Fargate

・コンテナネイティブ：仮想マシンを意識しないシームレスなスケーリング
・Amazon ElasticContainer Service → Fargate → ECR を起動

●メリット

以下全てを AWS 側にアウトソース可能
・EC2 インスタンスのプロビジョニングや管理
・状態異常が発生した EC2 インスタンスの再起動や入れ替え
・ホストレベルのスケーリング管理

●Fargate の基本

✔コンピュート

柔軟な選択肢：50 の CPU、
タスクレベル（必須）
コンテナレベル（オプション）
コスト：Fargate の方が利用状況に対して最適化されることが多い。
→Fargate ではより細かい単位でスケール可能（最小で 0.25vCPU、0.5GB でタスクを実行可能）

✔ストレージ

書き込み可能レイヤストレージ
・レイヤストレージとは
Docker イメージは例やで構成されており、最上位のレイヤは書き込み可能で、実行中のコンテナのファイル変更をキャプチャ
揮発性を持ち、タスク停止後には利用不可
・ボリュームストレージ
タスクごとに 4GB のボリュームを利用可能
タスク定義ないでボリュームマウントとして構成
複数のコンテナ間での共有
揮発性があり、タスク停止後は利用不可

✔ネットワーク

none：タスクのコンテナの外部接続なし
bridge：タスクは Docker の組み込み仮装ネットワークを使用
host：EC2 インスタンスのネットワークインターフェースにコンテナポートを直接マッピング
awsvpc：タスクごとに Elastic Network Interface が割り当てられる
セキュリティグループを Task ごとに割り当てられる
Task 内のコンテナは localhost インターフェースを共有
VPC 内の他リソースへ Private IP で通信が可能

●AWS Glue

●データ分析のプロセス

収集 → 保存 → 分析 → 活用
ニーズ：大量のデータが保存でき、かつ必要なときに必要な分のデータを取得して、

活用できる保存場所が求められる
そこから：データレイク（生データをそのまま保存する）
→生データを分析するための前処理を実施する。
→この処理をAWS上で実施するのが"AWS Glue"

●特徴
サーバレス
AWSサービスとの連携
セキュア
VPCからのアクセスなど

●Workflow Managementにより以下の一連の流れを自動化
①クローラーが、データをクロール
②クローラーが取得したデータをデータカタログに登録・更新
③トリガーにて、ジョブの実行タイミングを定義
④データカタログのメタデータをもとにデータソースからデータを抽出
⑤サーバレスエンジンにてジョブを実行し、ターゲットに出力

●Glueの構成要素
①データカタログ
Apache Hiveメタストア互換nのメタデータリポジトリ
メタデータをRedshift Spectrum, Athena, EMRに連携可能
（Apache Hiveメタストアとは、実データとは別に表の定義だけ格納する仕組み）
（クローラーとは、Glueのデータカタログにメタデータを作成するプログラム）
・接続管理
S3,DynamoDB, JDBC接続のアクセス方法
S3：AWS IAMでアクセス制御
S3バケットを指定
DynamoDB：IAMでアクセス制御、テーブル名指定
JDBC：事前に接続設定を追加する、セキュリティグループでアクセス制御

②サーバレスエンジン
・ジョブ作成
ETLの処理単位をジョブと良い、ジョブの種類にApache SparkとPython Shellがある。
Glueが自動生成したコード、自身が作成するスクリプト、既存のコードが実行可能
ジョブの状態を追跡できるブックマーク機能がある。
・Worker Type
Glue内のSparkジョブに「メモリ大量使用ワークロード向けのWorker Typeが指定可能に
・Dynamic Frame
SparkSQL DataFrameと似たGlue特有の抽象化の概念
複数の方の可能性を残して、後で決定できるようにする（Choice型）
・Choice型
DynamicFrameの列で複数の方を発見したときに両方の型を持つことができる。

③オーケストレーション
ワークフロー

●Glue Studio
従来：コードベースのJOB作成
進化：ETLジョブの作成、実行、監視を簡単にする視覚的なインターフェース

○Amazon EMR

・データ活用のためのツール
・従来のリレーショナルデータだけでは扱いきれなくなってきた。
・リアルタイム処理を実現するために、データレイクという考えが一般的になってきた（ETL）

●Apache Hadoopとデータレイク
リアルタイム処理、NoSQLデータ処理、ビックデータ活用

●Amazon EMR概要
・大幅なコスト節減を可能にする、クラウドを利用したマネージドなhadoopとSpark
・20のオープンソースプロジェクトがサポートされており、AWSは30日以内に追従しているため品質が高い

✔簡単
・Hadoop/Sparkクラスターを数分で起動
・Hadoopのインストールやメンテナンス不要
・クラスタのチューニングと構成を自動実行
・スけーリンポリシーに基づくクラスタ自動拡張
・処理完了時にシャットダウン
・手動による介入不要

✔低コスト
・リザーブドインスタンスとスポットインスタンスを活用可能
・クラスタ内のノードの役割に合わせて選択可能
・オンプレミスでのデータ活用と比較して、低いTCO

●AWS SageMaker

●開始流れ
①ノートブックインスタンス作成
必要：ノートブックインスタンス名、インスタンスタイプ、IAMロール
②IAMロール
指定するS3バケットを選択して、簡単にロールを作成することができる。

●SageMaker Exampleにサンプルが格納されている。
（例：Chainerで画像認識を行い、作成されたモデルをエンドポイントにして推論を行うなど。）
（必要なもの：from sagemaker.chainer.estimator import Chainerなど）

●学習実行流れ
①SDK経由で学習ジョブを実行

②学習用インスタンスが起動
③コンテナ上に学習データとコードが読み込まれ、学習を実行
④学習済みモデルを保存
⑤学習が終了したらインスタンスも削除される

●推論実行流れ
①SDK経由で推論エンドポイントを作成
②推論用インスタンスが起動
③コンテナ上にモデルが読み込まれエンドポイントとして機能（APIのエンドポイントとして読み込まれる）

●バッチ推論の流れ
①SDK経由でバッチ推論ジョブを作成
②推論用インスタンスが起動
③コンテナ上にモデルが読み込まれる
④推論用データをS3から読み、結果をS3に出力
⑤推論が終了したらインスタンスも削除される。

●AWS Lambda

●基本
・1コンテナで複数イベントを同時に処理することはない
・コードは依存関係も含めてビルド、パッケージングした上でアップロードされる
ZIP形式、アップロードしたものはS3に保存され、実行時以外は暗号化される
ARNを指定して実行することも可能

●基本設定
・メモリ
64MBごとに128MBから3008MBの間で設定可能
メモリ容量に応じてCPU能力なども比例
メモリ容量が一定を超えると使用するコア数も増えるため、マルチコアを活用するようなコードを実装することでより効率的な処理が可能
・タイムアウト
Lambdaファンクションの実行時間に関するタイムアウト
最大900秒まで
・実行ロール
必要なAWSリソースへのアクセスを許可するIAMロール
指定されたIAMロールに沿ってLambdaファンクションからAWSのリソースへのアクセスが許可される

●イベントソース
・イベントの発生元となるAWSのサービス又はユーザが開発したアプリケーション
Lambda関数の実行をトリガーする
・イベントソースには3タイプある
・ポーリングベース
ストレームベースとそれ以外がある
Lambadaがポーリングをして処理するデータがある場合にLambda関数を実行
・それ以外

Lambda関数はイベントソースから呼び出される
・イベントソースによって呼び出しタイプが異なる
従ってリトライの動きも異なる

●呼び出しタイプ
カスタムアプリケーションによる呼び出し、もしくはAWS CLIなどを用いての手動実行の場合に呼び出しタイプを指定できる
・イベントソースがAWSのサービスの場合はサービスごとに事前に決められており、変更はできない
・非同期呼び出し
InvocationTypeはEvent
レスポンス内容はリクエストが正常に受けつられたかどうかのみ
・同期呼び出し
InvocationTypeはRequestResponse
実行完了時にレスポンスが返ってくる。レスポンス内容はLambdaファンクション内でセット

●Lambda関数のリトライ①
エラーの種類、イベントソース、呼び出しタイプによって異なる
〇ストリームベースではないイベントソース
・同期呼び出し
リトライなし
エラー発生時にはレスポンスのヘッダにFunctionErrorが含まれる
パーミッション、Limit、関数コードや設定の問題によるエラーの場合は特定のステータスコードが返る
AWSサービスからの呼び出しの場合、その設定に従う
・非同期呼び出し
自動的に2回までリトライされ、その後イベントは破棄される
リトライには遅延がある
Dead Letter Queueを設定sルコとで未処理のイベントをSQSキュー又はSNSトピックに移動させ確認が可能

〇ポーリングベースでストリームベースのイベントソース
データの有効期限が切れるまでリトライを行う
失敗したレコードの有効期限が切れるかせ処理が成功するまで、そのシャードからの読み込みはブロックされ新しいレコードの読み込みは行われない
〇ポーリングベースでストリームベースではないイベントソース
そのバッチのメッセージは全てキューに返り、Visibility Timeoutがすぎレバまた処理が行われ、その後成功すればキューから削除される
新しいメッセージの処理はブロックされない

〇VPCアクセス
・RDSやElasticacheなどVPC内のリソースへインターネットを経由せずにアクセス可能
・VPC内リソースにアクセスさせたいLambdaファンクションに対してVPCサブネットおよびセキュリティグループを指定
・AZごとに1つ以上のサブネットを指定しておくのがおすすめ

・新規作成時だけでなく既存のものを後から変更することも可能
✔Elastic Network Interface（ENI）を利用して実現
・作成・削除はLambdaによって完全にコントロールされる
・ ENIには指定したサブネットのIPがDHCPで動的に割り当てられる
・ファンクションに割り当てるIAM Roleに"AWS LambdaVPCAccessExecutionRole"というポリシーをアタッチしておくこと
✔注意点
・設定した段階からインターネットアクセスは不可となる
パブリックIPアドレスは割り当てられない
必要な場合はNATインスタンスを用意する、もしくはVPC　NATゲートウェイを利用する
・必要なENI又はサブネットIPがない場合は、リクエストすが増えた場合に失敗する
非同期呼び出しの場合、このエラーはCloudWatch Logsに記録されない
コンソールで実行するなど、同期実行することでエラー応答は取得できる

●実行ロール
・Lambdaファンクションがリソースにアクセスするためのアクセス許可
最低でもCloudWatch Logsへのアクセス許可が必要
Lambdaファンクションん作成時に指定することで、呼び出されたときにLambdaによってこのロールが引き受けられる
・AWS Identity and Access Management(IAM)を利用して作成
信頼されたエンティティとして"AWS Lambda"を指定
アクセス許可は定義済みのもの、もしくはユーザが用意したものを指定

・Lambdaリソースを使用するためのアクセス許可を他のアカウントやAWSサービスに付与するには、リソースベースのポリシーを使用
Lambdaファンクション、バージョン、エイリアス、レイヤーバージョンなど

●同時実行数
ある時点における実行中のLambda関数の数
・セーフガードとしてアカウントに対してデフォルトでは1,000で制限されている
実績に応じて制限緩和申請が可能
・許可された同時実行数を超えてリクエストされたとき、スロットリングエラー（エラーコード：429）が返却
非同期呼び出しの場合、15~30分程度はバーストとして許容されるがそれ以降はスロットリング対象
・アカウントに許可された値を上限として関数単位で任意の割合で割り振ることも可能
・ConcurrentExecutionsとUnreservedConcurrentExecutionsのメトリクスで確認可能

●同時実行数のみつもり
・ポーリングベースかつストリームベース
シャード数と同じ
・ポーリングベースだがストリームベースでない

同時実行数までポーリングを自動的にスケールアップ
SQSでは5つの同時関数呼び出しが最初のバーストでサポートされ、1分ごとに60の同時実行呼び出しで同時実行が増加

・それ以外
秒間呼び出し回数　✖　平均実行時間（秒）で算出

●Lambda関数のライフサイクル
1.(ENIの作成)
2.コンテナの作成
3.デプロイパッケージのロード
4.デプロイパッケージの展開
5.ランタイム起動・初期化
6.関数/メソッドの実行
7.コンテナの破棄

●料金体系
・リージョン
月間100万リクエストまでは無料
釣果ぶんは$0.20/100万リクエスト
・実行時間
100ms単位での課金となり、100ms以下は繰り上げ
メモリ容量により単価と無料時間が異なる

●プログラミングモデルの基本
・ハンドラー
利用する言語の関数もしくはメソッドを指定し、実行の際に呼び出すエントリポイントとなる
ハンドラーを呼び出すことで、Lambda関数のコードが実行開始される
呼び出しの際にパラメータとして渡されるイベントのデータ（JSON形式）にアクセスすることが可能
・コンテキスト
ランタイムに関する情報が含まれ、ハンドラー内部からアクセス可能
コンテキストオブジェクトが2つ目のパラメータとしてハンドラー関数に渡される
コールバックを使用する言語の場合、コールバックメソッドの振る舞胃を設定可能
・デフォルトでは全ての非同期処理の完了を待ってレスポンス
・falseにするとcallbackで呼び出された時点で即座に処理終了
・ロギング
Lambda関数ないにログ出力のステートメントを含めることができる
CloudWatch Logsの制限を受けるため、スロットリングによって失われることがあることに注意
・例外
Lambda関数として使用する言語によって正常終了方法は異なる
・注意点
Lambda関数はステートレスにする必要がある

永続化するにはS3、DynamoDB、又は別のクラウドストレージサービスなどへの保存が必要

●設計図

・PythonとNode.js向けのみ提供

・Lambda関数作成時に選択可能

・Serveless Application Repositoryというのもある

ユーザーによる追加登録が可能、設計図の場合は追加登録は不可

●コンソールエディタ

AWS Cloud9をベースとしたコンソールエディタが利用可能

マネージメントコンソール上で、Lambda関数のコーディング、テスト、実行結果の表示が可能

●デプロイパッケージ

✔コードと依存関係で構成される.zip又は.jarファイル

・フォルダに含めないフラットなzipファイルであること

・依存関係の含め方は言語による

・Lambda関数の実装コードとしてアップロード

・コードファイルおよびデプロイパッケージを構成する全ての依存ライブラリに対してグローバルな読み取り権限が必要

●Lambda関数の設定

同時実行数の管理

アカウント単位で許可された同時実行数をファンクション単位で任意に割り当て可能

例：アカウントの上限が1000の場合

関数A：上限100

関数B：上限200

その他：上限700

DBへの書き込みや外部APIの呼び出しなどダウンストリームの流量制御や、ENI/IPアドレスの利用料をコントロールしたい場合に便利

・Custom Runtimes

Linux互換のランタイムを持ち込むことで、任意の言語をLambda関数を記述できるようになった

使用時には関数のランタイムをprovidedに設定

AWSからはC++/RustをOSSでリリース

・Runtime bootstrap

Bootstrapとは、ウェブサイトやwebアプリケーションを作成できるフリーソフト。

ファンクションに"bootstrap"と呼ばれる実行ファイルを含める必要がある

デプロイパッケージに含めてもLayerとして登録しても良い

・Custom Runtimeの作り方

Custom Runtimeのエントリポイントはbootstrapという名前の実行ファイル

bootstrap内では初期化処理を実行したのち、ファンクションの実行処理を実装する

・Initialization Tasks

・Processing Tasks

●モニタリング

・メトリクスとその意味
Invocation：Lambdaの実行回数
Duration：Lambdaの実行時間を計測
Errors：Lambdaが正常終了しなかった回数を計測
Throttles：Throttleの発生回数（Lambdaの同時実行数が超過した数）
IteratorAge：ストリーム（kinesis/dynamoDB）でのみ利用。バッチサイズ分取得したレコード終端の時刻とLambdaがイベントとして受信した時刻差で表示される
DeadLetterErrors：DLQを設定し、そのDLQへの書き込みが失敗すると増加する
ConcurrentExecutions：特定の時点における特定の関数の同時実行数
UnreservedConcurrentExecutions」同時実行制限を指定していない関数の同時実行数の合計
・ストリームベースでの監視/運用のポイント
IteratorAgeの"Average"が大きくなっている場合は、処理遅れが想定される
Kinesis Streamsのshardを分割
shardを分割することで並列処理数が増える（shard数＝同時実行数）
東京リージョンの場合、デフォルトでは200shardまで（制限緩和可能）
Error metricsが上がった場合は、処理がロックされている場合がある
Lambdaのプログラム改修
kinesis streamsの読み込みをlatest/timestampベースで再設定
streamの場合、Errorを上げないように設計/テストすることが重要

・モニタリングで気をつけるポイント
Throttleを観測した場合
どのLambda関数が大量に使われているかを特定
CloudWatchのAll across accountでアカウント全体の使用状況と個別Lambdaの使用状況を確認
同時実行数の制限緩和申請を行う
Error metricsの増加
権限設定漏れによって実行が全てエラーになっているパターン
SQS/ストリームであり得るパターン、バッチサイズとプログラム処理時間とLambdaの対街の関係を確認
1処理時間　✖バッチサイズ　＞　タイマ値となるこれも処理が進まないパターンとなる
Dead Letter Queueが増え続ける
後段処理、連携システムのダウンなど、システム系としての障害の確認などを実施

・AWS X-Rayを使用したトラブルシューティング
デプロイパッケージにX-Ray SDKを追加し、アクティブトレースをONにする。
実行ロールへの権限追加も必要
・X-Rayデーモン
Lambda関数をトレースする場合、X-Rayデーモンが自動的に実行され、トレースデータが収集されてX-Rayに送信される
トレース時は、X-Rayデーモンによって最大16MBまたは関数のメモリ割り当ての3パーセントを消費
Lambdaは冠すのメモリ制限を声内容、X-Rayデーモンを終了しようとする

✔アプリケーションの管理
管理する対象はLambda関数、イベントソース、その他のリソース
管理するためのツールセット
・AWS Serverless Application Repository
・数クリックでアカウントにデプロイできるLambdaアプリケーションレポジトリ
AWS CloudFormation
・AWSリソースの定義とプロビジョニング自動化
AWS Serverless Application Model(AWS SAM)
・サーバレスアプリケーションの定義とパッケージング・デプロイ
・AWS CloudFormationテンプレート言語の拡張
AWS CLIおよびAWS SAM CLI

✔コールドスタート
Initializationがある
✔Warm Start
Invocationから始まる
✔最大のポイント
如何に関数を効率よく実行するか
コールドスタートを速くする
・コンピューティングリソースを増やす
Lambdaの設定値としてはメモリしかない
→メモリサイズと比例してCPU能力も割り当てられる
→メモリ設定＝パフォーマンス設定
・ランタイムを変える
AWS Lambdaに限らず、JVMの起動は遅い
・パッケージサイズを減らす
不要なコードは減らし、最小限のサイズにする
依存関係を減らす
コード最適化ツールを使って減らすというてもある
・ProGuard(Java)、UglifyJS(Node.js)
・VPCは必要でない限り使用しない
使うのはVPC内のリソースにどうしてもアクセスする必要があるときだけ
例：RDSインスタンスなど、パブリックに公開できないリソース
VPCアクセスを有効にしているとコールドスタート時に10秒から30秒かかる
同期実行が必要な箇所やコールドスタートを許容できない箇所にはなるべく使わない

・関数コードを効率化
コードの最適化
Fatでもモノリシックな関数にならない
・一つの関数はなるべく単機能にする
・関数内でのオーケストレーションは禁物
デプロイパッケージの依存関係をコントロールする
ハンドラとコアロジックは分離させる
・コンテナ再利用を有効活用する

グローバルスコープを賢く活用する
AWS SDKのクライアントやDBクライアントの初期化はハンドラの外側で行う
・必要なもののみ読み込むようにする
無駄に不要なデータを読み込まない
・Amazon S3 selectを利用して読み込むものを絞り込む

・非同期呼び出しを活用する
同期呼び出しの場合、同時実行数の制限に引っかかって詰まりがち
・非同期呼び出しの場合、許可された同時実行数内で順次処理をし、バーストも許容されている
・同期呼び出しの場合、許可された同時実行数を超えた時点でエラーが返されてしまう
非同期の場合はリトライされる
できるだけ非同期でInvokeするのがスケーラビリティの観点ではおすすめ
その処理、本当にレスポンスが必要ですか？

・冪等性を確保する
冪等性はお客さんのコードで確保する必要がある
AWS Lambdaで保証しているのは最低1回実行することであり1回しか実行されないことではない
実装例：イベントiDヲAmazon DynamoDBに保管し処理実行前にチェックする
処理前後でバケットを分け、処理後に消す
・Think Parallel
AWS Lambdaの最大限活かすためにはいかに並列で処理を実行するかが重要

✔Systems Manager - Parameter Store

設定データ管理と機密管理のための中央集権的ストレージ
・階層型
・パスワード、データベース文字列、ライセンスコードなどのデータをパラメータ値として保存
・値は平文もしくは暗号化して保存可能
・Amazon SNSを用いた変更通知が可能、自動化されたアクションの実行
・AWS KMS、Amazon CloudWatch、AWS CloudTrailとの統合
・追加料金不要
・API/ SDKから利用可能
環境変数やしークレット管理の一元化が可能

✔AWS Secrets Manager

DBの認証情報、パスワード、サードパーティーのAPIキーなどのシークレット情報を一元的に保存・制御可能
ライフサイクルを通じて簡単にローテーション、管理、取得
RDS、Redshift、DocumentDBとの統合が組み込み済み
・認証情報をユーザーに代わって自動的にローテーション
きめ細かいAWS Identity and Access Management(IAM)ポリシーと、リソースベースのポリシーを使ってシークレットへのアクセスを管理することが可能
有料

✔ストリーム型のイベントソースを利用する上でのベストプラクティス

・**ストリーム位のシャード数はLambda関数の同時実行数と同じ**

・バッチサイズはLambda関数の実行あたりの最大レコード数

バッチサイズが大きいとLambada関数の同時実行数は低下する

・処理が失敗するとデータが期限切れになるまでシャード全体がブロックされるため、Lambda関数は常に成功を返すようにして、

関数コードないでログ出力やDLQへ失敗したレコードの情報を送る

・ここで利用するDLQはLambda設定ではなく、自前で用意する

・スループットの問題が起こりがちになるため、複数のコンシューマーを用意するのは避ける

DynamoDB Streamsをサポートするのは最大で２プロセスからのアクセス

スケールさせるにはファンアウトパターンを検討

・Kinesis/DynamoDBのようリトライや順序が重要でない場合は、SNSの利用も検討する

・ファンアウトパターン

Kinesisを受け付けるLambdaは一つ（Lambda Dispatcher function）、その先にLambda Processor functionを複数個ぶら下げる

✔Lambdaで利用するデータベース

AWS Lambda + RDBMSガアンチパターントされる利用

・コネクション数の問題

Lambdaはステートレスなプラットフォームであるため、コネクションプールの実装は難しい

Lambdaがスケールすると各ファンクションからDBへコネクションが増えていき、やがて最大同時接続数を超えてしまう

・VPCコールドスタートの問題

実行数が少ない、コールドスタートによる遅延が許容できる環境であれば、RDBMSを使っても問題ない

✔選択肢

１.Amazon DynamoDBを使う（推奨）

分散データベースであり、コネクション数の問題から解放される

標準でVPCがいリソースとして利用可能

VPCエンドポイントを追加して利用することも可能

スループットがスケーラブル

・Capacity Unitのプロビジョニング

・On-Demand

・Auto Scaling

２.RDBMSへの反映は非同期にする

・DynamoDB StreamsとAWS Lambdaを利用して非同期にRDBMSへ反映

・Kinesis Data Streams や Amazon SQSとAWS Lambdaを利用して非同期に反映

・API Gateway との組み合わせの場合、サービスプロキシとしてLambda関数を非同期に呼び出し

・VPCレイテンシの問題も緩和される

３.同時実行数を制限する

✔ Lambdaのセキュリティ

・コントロールプレーン
ファンクション管理APIを提供
AWSサービスとのインテグレーションを管理
・データプレーン
Invoke APIのコントロール
Lambda関数がInvokeされると実行環境を新たに割り当てる、もしくは既存の実行環境を選択する

✔実行環境の隔離
・関数コード
・ Lambda layer
・組み込みもしくはカスタムのランタイム
・ Amaozn Linuxベースの最小限のユーザーランド
・定義
cgroup：実行環境ごとにCPU、メモリ、ディスク帯域、ネットワーク帯域へのリソースアクセスを制限
namespace：プロセスID、ユーザID、ネットワーク・インターフェースとそのほかのリソースをグルーピング各実行環境は専有のnamespaceで実行される
seccomp-bpf：実行環境内から利用されうるsyscallを制限
iptables/routing tables：実行環境をお互い隔離
chroot：ファイルシステムへの特定範囲でのアクセスを提供

✔MicroVMの隔離
単一のAWSアカウントにおいて、複数の実行環境を単一のMicroVM上で実行できるが、AWSアカウント間で共有や再利用されることはない
✔EC2モデルとFirecrackerモデル
EC2モデル：EC2InstanceとMicroVMが一対一
Firecrackerモデル：実行環境とMicroVMが一対一

✔ペイロード
Lambda関数の呼び出しタイプによってペイロードは異なる経路で処理される
・同期呼び出し
・ API Callerからロードバランサに渡され、その後Lambda invoke serviceに渡される
・非同期呼び出し
・ Amazon SQSを利用したキューにキューイングされる
・内部的なpollerプロセスがメッセージを取得しLambada invoke serviceへと渡す
✔ランタイム
Lambdaがサポートしているランタイムのメンテナンスは AWS責任
✔Lambda関数のAudit
・ Amazon CloudTrail
AWS Lambdaを含むAWSアカウント全体の統制、コンプライアンス、オペレーション監査、リスク監査を実装可能
・ AWS Config
Lambda関数、ランタイム、タグ、ハンドラ名、コードサイズ、メモリ割り当て、タイムアウト設定、同時実行数の設定に対する設定変更や削除を追跡可能

●AWS Systems Manager(SSM)

Enable（準備）、Provision(展開)、Operate(操作）
Systems Manager は Operate に分類される
✔役割
・グループか
。可視化
・対応

✔機能
・全体
クイックセットアップ：インスタンスを SSM で管理するよう自動構成
・オペレーション管理
Explorer：運用アイテム情報のダッシュボード
OpsCenter：運用アイテムの管理
・アプリケーションマネジメント
リソースグループ：タグによるサーバ軍のグループ管理
AppConfig：アプリケーション設定の管理
パラメータストア：設定パラメータの集中管理用データストア
・アクションと変更
Automation：AWS 環境全体に対する自動化処理の実行
Change Calender：実行可否を制御するカレンダー
メンテナンスウィンドウ：自動化処理のスケジュールと順序の管理
・インスタンスとノード
コンプライアンス：コンプライアンスの適合状態ダッシュボード
インベントリ：サーバ構成情報のインベントリを閲覧する
マネージドインスタンス：SSM 管理対象のサーバー一覧
ハイブリッドアクティベーション：オンプレミスサーバを SSM 管理下に入れる
セッションマネージャー：SSM を使ったサーバへリモートアクセスする
Run Command：サーバ軍の上でコマンドを実行する
ステートマネージャー：サーバー群の構成を指定した状態に維持する
バッチマネージャー：サーバー群に指定ルールに基づきバッチを適用する
ディストリビューター：サーバー群にパッケージをインストールする
・共有リソース
ドキュメント：SSM で実行する処理を記述したドキュメント

✔準備
Step.1 マネージドインスタンスにする
（SSM 管理下のインスタンス群、EC2 インスタンスの他、オンプレミスのインスタンスも含められる）
〇マネージドインスタンスにする手順
1 . SSM Agent を導入する（SSM Agent が SSM API と連携し、各種操作やコントロールを行う）
Amazon Linux や Windows、Ubuntu Server のオフィシャルイメージにはダウンロード済み
それ以外の AMI、およびオンプレミスサーバーは手動でインストール
2 . SSM API への経路確保

①インターネット経由で確保
②VPCエンドポイント経由（Private Link、VPN、Direct Connect）
3. IAMロール付与
IAMロールを作成し、EC2にアタッチ
（AmazonSSMManagedInstanceCore）
これらの設定を簡素化する機能が"クイックセットアップ"
できること：SSM設定の自動構成ができる。
いいところ：新規インスタンスを自動でマネージドインスタンスにできる、SSMのベストプラクティスに則って管理できる。

Step.2 インスタンスをグループ化する
「リソースグループ」は、AWSリソースをグルーピングすることで、整理・管理をしやすくする機能
タグエディターを使うことによって、一度に複数のリソースのタグを追加・編集・削除が可能

〇オンプレミスの場合
1. TLS証明書のインストール
2. ハイブリッド環境用のIAMロールを作成（初回のみ）
3. SSMでアクティベーションコードを生成
4. インスタンスにアクティベーションコードを設定

●リソースの今を把握しよう
〇SSM Explorer
・クロスアカウント、クロスリージョンで、"今"のリソース状況を可視化できる
・デフォルトのダッシュボードに表示されるOpsData
EC2情報
・EC2インスタンス数
・マネージドインスタンス数
・AMI別インスタンス
OpsCenter OpsItems
パッチコンプライアンス
〇SSM OpsCenter(Explorerの詳細)
・運用作業項目（OpsItems）を表示、調査、解決できるダッシュボード
・OpsItemsに対して修復を行なったり、対応の完了を記録してクローズするなど運用タスク管理に利用できる
・CloudWatch Eventsのルールとして登録する
・デフォルトでEC2やRDS、SSMなどのイベントが登録ずみ
・Amazon EventBridgeと連携でき、外部アラートも登録することができる
〇SSM コンプライアンと
コンプライアンスに準拠していないリソースを表示できるダッシュボード

●インスタンスの中身を把握する
〇SSM インベントリ
・OSバージョン、タグ、アプリケーション情報、MACアドレス、ファイルな

どを記録し、可視化する
Athenaと連携されているので、柔軟なクエリを行うことが可能
→Quick Sightにより可視化することが可能
（EC2→SSM Inventory→S3→Athena→QuickSight）
・AWS Configに構成情報を送信し、インベントリ情報の変更追跡が可能

●SSMでできる定型作業の整理
1。SSMでは、運用処理をSSMドキュメントにて定義し、実行する
・汎用的な処理は事前定義されたドキュメントにあり
・カスタマイズした処理を実現したい場合は、ドキュメントを自作する
2。事前定義ドキュメントの中でも、需要が多く複雑な処理は、ドキュメントの実行フレームワークをSSMの機能として提供
・サーバーの構成情報の収集：実行フレームワークはSSSインベントリ
・バッチ適用プロセスの自動化：実行フレームはSSMパッチマネージャー
・ソフトウェアパッケージの配布：実行フレームはSSMディストリビューター

〇SSMドキュメントで運用を定義し実行する
1。カレンダーを作成
2。ドキュメンと定義
SSMドキュメントを選択する
（Run Command（OS上でコマンドを実行、コマンドドキュメントを実行する、サーバログイン不要）、Automation（AWSサービス全体に渡ったワークフロー、自動化ドキュメントを実行する）
3実行
・一度きりの手動実行なら
Run Commandをそのまま「実行」
Automationをそのまま「実行」
・繰り返し実行したい定期実行なら
ステートマネージャー
メンテナンスウィンドウ
・実行できる日時を制御するなら
Change Calender
〇SSM ステートマネージャ
・サーバー群に対して定期的に処理を行うためのフレームワーク
サーバーの状態を確認・是正するための定期的な処理に向く
〇SSM メンテナンスウィンドウ
・サーバー群に対して定期的に処理を行うためのフレームワーク
・サービス停止を伴うような比較的重い処理に向け
・Lambda、Stem Functionsも実行可能
〇SSM Change Calendar
システム内で利用するカレンダー情報を集中管理するサービス

〇SSM パッチマネージャー
・パッチルール準拠の確認、インスタンスへのパッチ適用が可能
・Scanのみ、Scan&Installの2通り
・パッチベースラインをOSの種類ごとに作成
・注意点

インスタンスに指定されたパッチダウンロードサイトへのアクセス経路の確保が必要

パッチ適用後の再起動は、NoRebootオプションでタイミングを制御可能

〇SSM ディストリビューター

・独自のソフトウェアパッケージの配布、インストールが可能

・AWSの各種パッケージが事前定義されておりその導入・更新にも有効

●非定型なインタラクティブ操作も可能

〇SSM セッションマネージャー

・セッションマネージャー - RDPアクセスなど

・**通信ポートを開放せずにサーバへのシェルアクセスが可能**
踏み台サーバー入らずに

・セッションマネージャーで用意されている接続手段

１。SSM Agent経由で直接アクセス

・操作ログをCloudWatch LogsやS3に保存、暗号化も可能

・セッションマネージャーでの接続履歴や、CloudTrailにて接続情報を追跡可能

・インスタンスでSSHを起動させる必要はない。ポートの穴あけも不要

・AWS CLIからアクセスも可能

２。SSM Agentでトンネルを作成してSSHなどでアクセス
操作履歴は保管されない

・RDP（リモートデスクトップ）接続)

・SSH接続
現在のSSHクライアントにProxy設定を追加することで利用可能

●アプリケーションの設定管理もSSMで

〇SSM パラメータストア

・コンフィグレーションや設定値を顕現別の階層型で保存

・パスワードなど機密情報をKMSで暗号化

・パブリックパラメータあり

・CloudFormation、Lambda、ECS、CodeBuild、CodeDeployなどのサービスと統合済み

〇SSM AppConfig

・アプリケーションの設定情報を迅速に展開するための機能

・環境ごとに異なる設定情報をデプロイできる
設定情報はパラメータストアもしくはSSMドキュメントとして保管
アプリケーションコードからAppConfigのGetConfiguration APIでパラメータを取得。その値で動作を変えるよう開発する。

・展開前にバリデーションも実施できる

・デプロイ戦略として、カナリアリリースも利用可能

●セキュリティ～ベストプラクティス

・最小限の特権アクセス

・VPCエンドポイントを使用

・対話型コマンドのみを使用

・SSMツールを最新に保つ
・CloudTrail、Config、CloudWatchで監視する

●料金
基本的に無料

●Amazon EventBridge

✔イベントバス
・メッセージの送受信を管理する
・Push型で非同期で一度に単一のメッセージを処理する
・送信者が受信者を意識しない（Pub/Sub）
→これをAWS上で実装するサービスが、Amazon EventBridge

✔EventBridgeのメリット
・イベント駆動アーキテクチャを容易に実装
・運用負荷の低減
・実装量の削減
・SaaS由来のデータの利用

✔イベントバスの種類
・デフォルト
AWSサービス（例：EC2インスタンスが停止した）
・カスタム
独自のアプリケーション（例：注文機能が注文を受け付けた）
・パートナー
SaaS（SaaSのCRMで顧客情報が更新された）

✔アーキテクチャ

●構成要素

・イベントソース

〇AWSサービスが送信するイベント
・EventBridgeに直接送信されるイベント：AWSリソースの状態変化を起点
（例：EC2インスタンスの停止を検出など）
・CloudTrail経由で送信されるイベント：ユーザーのAWSリソースに対する操作を起点
（例：EC2インスタンスの停止リクエストを検出など）

〇カスタムイベント
ユーザーのPutEvents APIの呼び出しによって送信される
送信先のイベントバスは送信時に選択可能

〇SaaSイベント
SaaS専用のイベントバスに送信される

・パートナーイベントソース
SaaSアプリケーションからのイベントを有心を受け付けるためのリソース

・イベントパス
・デフォルト、カスタム、パートナーの３種類がある。

・ルール（後続処理に送信するイベントを送信）

どんなイベントを送信するか
どのイベントバスを対象にするか
どのターゲットに送信するか
・スケジュールに基づいてトリガーするイベントを作成できる

・ターゲット（Kinesis、Lambda、Step Functions、CloudWatch Logsグループなど）
1つのルールあたり、5つまでのターゲットに送信可能
・ターゲットのリソースは同じリージョンに存在する必要がある

●アクセス制御
〇イベントバスへのアクセス制御
誰がイベントバスにアクセスできるのか
イベントバスごとにOrganizationsの組織や他アカウントからのアクセスを許可できる

〇ターゲットへのアクセス制御
どのターゲットにアクセスできるのか
マネジメントコンソール上でターゲットを追加した場合は自動で権限が付与される
・LambdaはLambda関数ポリシー、SNSはトピックポリシー、SQSはキューポリシーにアクセス許可を記述する
・CloudWatch LogsはEventBridgeからのイベント送信がデフォルトで許可されているため、利用者による設定は不要
・IAMポリシー
上記以外のサービスはIAMロールを割り当ててIAMポリシーで許可する

●SaaSのAPIをポーリングする
１。cronでアプリケーションを起動し、APIを呼び出して取得したデータをイベントバスへカスタムバスとして送信する（EC2）
２。SaaSのAPIをポーリングする（Lambda）
３。SaaSのWebhook機能でPushしてもらう(API Gateway)
４。EventBridgeのSaaS連携機能の利用（パートナーイベントソース）←これが効果的
（SaaSアプリの例：DATADOG、KARTE、Auth0など）

●クロスアカウント連携
イベントバス間のアカウント間の共有
Publisherは自身が存在するAWSアカウントのイベントバスのみを意識すれば良い。
ルールで他アカウントのイベントバスをターゲットに指定する
アカウント間の依存関係がEventBridgeに閉じる。

●AWS Certificate Manager

✔TLS（SSL)とは
SSL / TLSトランスポート層のプロトコルであり、データを暗号化して送受信するためのもの

（HTTPやFTPの通信をセキュアに）

✔TLSの成り立ち

SSLのバージョンアップにより、SSL3.0をもとにTLS1.0が制定

✔TLSサーバー署名書

・脅威

なりすまし、盗聴、改ざん、否認

・対策

特定、暗号化、視認（ブラウザユーザーに鍵アイコンを見せる）

✔PKI(Public Key Infrastructure)

公開鍵暗号技術と電子署名を使ってインターネット通信を安全にするきばん

公開鍵暗号方式

ペアになった公開鍵と秘密鍵を用いて暗号化と復号化を行えるようにするもの

電子署名

秘密鍵で暗号化されたデータを公開鍵で複合化できることが秘密鍵を保有している本人であると特定できる。

電子署名書

PKIにおける鍵を持っている人を証明するもの（TLSサーバー証明書）

CA

インターネットの世界では、電子証明書を発行する組織を認証狂句と呼ぶ

ルートCA

証明書を発行するCAは、上位のCAに認証をしてもらうことで、その正当性を表明する。

✔機能

●発行機能

パブリックDV証明書の発行

プライベート証明書の発行

●展開更新機能

発行した証明書の展開

インポートした証明書の展開

●対象サービス

・Elastic Load Balancing(ELB)

・Amazon CloudFront distribution

・Amazon API Gateway上のAPIのカスタムドメイン

・EC2やオンプレサーバーなどの内部リソース（プライベート証明書）

・コンソール

●メリット

・数クリックで簡単に証明書を発行

・無料

・安全な鍵管理

・キーペアとCSR生成をしなくて良い

・更新もしなくて良い

・サードパーティ証明書のインポートと統合

●ライフサイクルマネジメント

ACMはパブリック・プライベート証明書ともに1つのインターフェースでの管理
ACMに統合されるサービスはマネージド証明書更新と自動適用
ベストプラクティスを用いたプライベートキーの安全な管理
コードからプライベート証明書のエクスポートや適用をAPI呼び出し可能

●ACMパブリック証明書/プライベート証明書

	ACMパブリック証明書	ACMプライベート証明書
検証方法	ドメイン所有検証が必要	外部検証は不要
アプリケーション、ブラウザ、OSからの信頼方法	標準で信用される	ルートCAをトラストストアに追加する必要があり
証明書のサブジェクト	有効なパブリックDNS名のみ	パブリックまたはプライベートのDNS名、または有効なX.509サブジェクト
ユースケース	公開サーバー	内部リソース（サーバー、クライアント、コンテナ、IoT）
ルートCA	パブリックルートCA	プライベートルートCA

●ACMの認証タイプ
・ドメイン認証（DV）

●ACMに統合されるサービス
証明書配置可能（無料）
・ELB
・Amazon CloudFront
・Amazon API Gateway
連携利用可能
・AWS Elastic Beanstalk
・AWS CloudFormation

●証明書のエクスポートやリージョン間でのコピー不可

●デモ（ALB）
事前準備：ドメインを取得していること、DNSレコードヲツイカデキルカコト、
①ドメイン名の追加（ワイルドカード指定も可能）
一枚の証明書でサブドメインを含めた証明書の利用が可能
②証明書のリクエスト（パブリック証明書、プライベート証明書）
③検証方法の選択
1。DNS検証（推奨）
DNSにCNAMEレコードを追加し、ACMが自動的に検証
2。Eメール検証
何かの理由でDNSケンショウヲ行えない場合に検証
④ドメインの検証
Route53にCNAMEレコードが登録されて、証明書の検証ができるようになる。
ここで証明書が発行される。
⑤ALBに証明書を設定
マネージメントコンソールで発行した証明書を選択する。

●

●CodePipelineによる接続
　CodeCommit,CodeBuild,CodeDeployを接続する。
　（CodePipelineをアップデートしたり管理するのが、CodeStar）
　作成したパイプラインファイルは、リージョン内のS3に格納される。
　ソース：.javaファイルなどのソースコードをチェックイン、新しいコードのぴあレビュー
　ビルド：コードのコンパイル、ユニットテスト、スタイルチェッカー、コードメトリック、コンテナイメージの作成
　テスト：他のシステムとの統合テスト、ロードテスト、UIテスト、侵入テスト
　運用：本番環境にデプロイ

●Route53のヘルスチェック判定基準
　・HTTP/HTTPSヘルスチェック：Route53がエンドポイントとのTCP接続を4秒以内に確立できること
　　接続2秒以内に、HTTPステータスコード2xxまたは3xxでエンドポイントが応答すること
　・TCPヘルスチェック：TCP接続を10秒以内に確立できること。
　・HTTP/HTTPSヘルスチェックと文字列の一致：TCP接続を4秒以内に確立できること。
　　接続2秒以内に、HTTPステータスコード2xxまたは3xxでエンドポイントが応答すること
　　ヘルスチェッカーはHTTPステータスコードを受信した2秒以内にそのエンドポイントからレスポンスボディを受信する必要がある。
　　Route53は指定された文字列をレスポンスボディで検索する。その際、検索文字列全体が、レスポンス本文の最初の5,120バイト内に出現している必要がある。

●Amazon Lex
　音声やテキストを使用した会話型インターフェースを様々なアプリケーションに構築するためのサービス。

●IAM Access Analyzer
　リソースのポリシーを確認して、意図せぬ公開設定などがされていないか検出し、可視化する機能になります。

●AWS LightSail
　Amazon LightSailは、AWSが提供しているVPS（Virtual Private Server:仮想プライベートサーバー）サービスです。
　Amazon LightSailには、コンピューティング環境だけでなく、ストレージ、スナップショット、ロードバランサー機能、ファイアウォール、DNS機能など、いくつもの機能が揃っています。
一方、Amazon EC2で提供しているのは、コンピューティング環境だけです。ストレージならAmazon S3、データベースならAmazon RDSなど、他のサービスを組み合わせて

必要な仕様を構成し、申し込む必要があります。
✔特徴
・低コスト
・サーバー作成が簡単
・サーバーの複製が簡単
・柔軟性が少ない
・停止していても課金される

●AWS Lake Formation

- **AWS でデータレイクを構築・運用するためのマネージドサービス**
  - 実体は、ほぼ AWS の各種サービスをラップしたもの (Glue, IAM, S3, etc..)
  - データレイク専用にアクセス制御を行うために、IAM とは別に独自の権限管理機構を持つ
- **実データも保持しセキュリティ向上と権限管理が簡単に行える AWS Glue** という印象
  - IAM や Glue を個別に駆使してデータレイクを構築・運用するよりデータレイクに特化していて扱いやす

●Amazon DocumentDB

✔概要
JSON ドキュメントと DB 内容を変換して整合性を取る必要性があった。
DocumentDB は JSON-like なデータ（ドキュメント形式）でデータ保管することができた。
ドキュメントはより自然な形でデータを表現可能
柔軟性のあるスキーマ、インデックス
Javascript で表現可能な柔軟なクエリ

✔ユースケース
・コンテンツ管理
・モバイルアプリ
・カタログ
・リテール＆マーケティング
・パーソナライゼーション
・ユーザプロファイル

●App Runner

特徴
1。ECR にコンテナイメージを push するか、GitHub にソースコードを push すると、それを検知し、App Runner がいい感じにデプロイしてくれる
2。コンテナがデプロイされると App Runner がマネージドの LB を設定してくれる
3。Web アプリを仕様に合わせデプロイするときに、コンテナ管理とか VPC 周り、ALB、NLB とか
Scalling、自動化するなら CodeBuild とかを組み合わせていたものが、App Runner がこれらをまとめて提供してくれる。

●Amazon Redshift

データウェアハウス
主に大容量データを高速に集計・分析する必要があるワークロードに活用
✔アーキテクチャ
オープンソースのPostgreSQLが元になっている。
●コンポーネント
✔リーダーノード
アプリケーションからクエリを受け付けるノード（サーバーのよなもの）
（クエリのエンドポイント）
SQLをコンパイルして、コンピュートノードに配信する
クエリ結果をクライアントにかえす
✔コンピュートノード
リーダーノードから受けたクエリの情報を並列で処理する。
キャッシュからの読み取り
✔マネージドストレージ
Redshift管理のS3バケット
●キャッシュ
データは永続ストレージとしてのS3とキャッシュとしてのローカルSSDがある
アクセス頻度の高いブロックはキャッシュにとどまり、あまりアクセスされないブロックは自動的にキャッシュアウト
✔特徴
・ハイパフォーマンス
列思考ストレージ（該当する列のみを読み書きすることができる。）：
データ圧縮(列の圧縮を行うことで一度のアクセスで読み込めるデータ量が多くなる)
ソートキー：
データ分散：
・フルマネージド
自動バックアップ、スケジューリング機能、パッチ適用の自動実行
機械学習ベースの自動最適化によるクエリ性能の向上
拡張性＆柔軟性

●Amazon Keyspaces
オープンソースソフトウェア(OSS)のNoSQLデータベースの一種である Apache Cassandra を互換できるマネージドデータベースサービスです。

●Amazon Timestream
フルマネージドな時系列データベース

●AWS CodeArtifact
Amazonから提供されているフルマネージド型アーティファクトリポジトリサービス。

●Amazon MSK（Amazon Managed Streaming for Apache Kafka）
Amazon MSK は、Apache Kafka をストリーミングデータの処理に使用するアプリケーションを簡単に構築および実行できようにする完全マネージド型のサービス
（Apache Kafkaとは、ログ、センサーなどIot機器のストリーミングデータを高速に中継保管し、処理サーバへ受け渡す機能というイメージとなります。）

●Amazon AppFlow
　Amazon AppFlowはFlow Salesforce や Slack Google Analytics などの AWS外部の SaaS アプリケーションと統合して Amazon S3、Amazon Redshiftなどのストレージへ安全にデータを転送するサービス

●AWS Directory Service for Microsoft Active Directory（AWS Managed Microsoft AD）

●Active Directory Migration Toolkit
　オンプレミスの Active Directory から AWS Managed Microsoft AD にユーザーを移行できる。

●AWS Global Accelerator
ALB、NLB、EC2の前段に置いてアプリケーションの可用性とパフォーマンスを改善するサービス。
cloudfrontと似たようなサービスですが、以下の差分がある。

- cloudfront
  - プロトコル：http/httpsのみ対応
  - キャッシュ：使う
  - クライアント証明書：利用しない
- global accelerator
  - プロトコル：TCP/UDPのトラフィック・ルーティングに対応
  - キャッシュ：使わない
  - クライアント証明書：利用する

●AWS Marketplace
　データベースやアプリケーションなどがインストールされた、AMIを提供しているオンラインストアです。
　そのAMIを利用することで、EC2を立ち上げてから手動でインストールする必要はないです。

●AWS Data Exchange
　AWS Marketplace に登録されている様々なサードパーティーデータをサブスクライブすることが出来ます

●ENA と EFA

● AWS Migration Hub
　既存のサーバーを検出し、各アプリケーションの移行状況を一括で進捗確認ができます

●OpenID Connect

● AWSAgentless Discovery Connector
　VMware 仮想マシン (VM) に関する情報のみを収集できる VMware アプライアンス

●エイリアスレコード
　一般的なDNSサーバでは、Aレコードを登録する際には値としてIPアドレスを指定することになる。
しかし、ELBなどのIPが可変であるAWSリソースをターゲットにする場合、IPを直接指定するのは得策ではない。
その解決策として、AWSリソースがデフォルトで持っているDNS名をIPアドレスの代わりに入力できるようにしてくれるのがエイリアスレコード。

通常のAレコード:：ホスト名 - IPアドレス
Route53のエイリアスレコード：ホスト名 - エイリアスレコード

● プライベートホストゾーン
　Route53の機能のひとつで、VPC内のみで名前解決をしてくれるもの
（VPC外で名前解決をするのはパブリックホストゾーンとなります）

●AWS Application Migration Service（MGN）

　仮想環境・物理環境やパブリッククラウドからAWSへの移行をサポートしている。
　✔特徴
　　・物理環境、仮想環境の両方に対応している
　　・エージェント (AWS MGN Replication Agent) を移行元サーバにインストールする必要がある。
　　・ダウンタイムを最小限に抑えられる。

●AWS Server Migration Service (AWS SMS)

　サーバー仮想マシン (VM) をクラウドホストの Amazon マシンイメージ (AMI) として段階的にレプリケートし、Amazon EC2 にデプロイする
　✔特徴
　　・**エージェントをインストールする必要がない**

●S3アクセスポイント

　1つの S3 バケットに対して複数のユーザーやアカウントからのアクセスが必要な場合、それぞれのアクセス許可レベルをコントロールする単一のバケットポリシーが必要です。アクセスするユーザーやアプリケーションが増えるにつれてバケットポリシーが複雑になり、テストや変更等が難しくなります。
S3 アクセスポイントは 1 つの S3 バケットに対し用途ごとのアクセスポイント作成し、個別にアクセスポイントポリシーを定義できる便利機能です。

- S3 Bucketへのアクセス権限を管理するための新しいBucket設定です。
- 新規、および既存Bucketに設定可能です。
- アクセスポイントという単位でカスタムアクセス権限を設定可能です。
- アクセスポイント経由でアクセスすることでカスタムアクセス権限が適用されます。

●S3ファイルゲートウェイ

　S3へのファイルインターフェイスをサポートし、サービスと仮想ソフトウェアアプライアンスを組み合わせる。
　ネットワークファイルシステム（NFS）やサーバーメッセージブロック（SMB）などの

業界標準のファイルプロトコルを使用して、S3でオブジェクトを保存および取得できる。

●ファイル転送サービス比較
AWS DataSync と AWS Snowball Edge
- 厳しい帯域制限 → Snowball
- 切断された環境 → Snowball
- その他ネットワーク環境が厳しい → Snowball
- 上記以外 → DataSync

AWS DataSync と AWS Storage Gateway
- 一度きりの移行 → DataSync
- 継続的なデータ更新 → Storage Gateway

●AWS DataSync
　AWS へのデータ移行を簡素化および高速化し、オンプレミスのストレージ、エッジロケーション、他のクラウド、および AWS ストレージ間でデータを迅速かつ安全に移行するのをサポートするオンラインデータ移行および検出サービス。
　Direct Connectを使用して、S3にデータを転送することができる。
　S3イベント通知は、LambdaとAWS Batchをトリガーとしてデータ処理ができる。

●Amazon Route53 Resolver
　オンプレミス-vpcの相互名前解決が簡単に実現できる。

●Amazon CloudSearch
　Webサイトやアプリケーション検索機能を実装することができるマネージド型サービス。

●試験対策でやったこと
〇AWS Black Belt Onlineで見たもの
【AWS Black Belt Online Seminar】AWS Service Catalog
【AWS Black Belt Online Seminar】AWS Config
【AWS Black Belt Online Seminar】Amazon Simple Notification Service (SNS)
【AWS Black Belt Online Seminar】AWS Step Functions(DVPの範囲？)
【AWS Black Belt Online Seminar】Amazon API Gateway
【AWS Black Belt Online Seminar] Amazon CloudFrontの概要
【AWS Black Belt Online Seminar】Amazon CloudFront deep dive
【AWS Black Belt Online Seminar】AWS Database Migration Service
【AWS Black Belt Online Seminar】AWS Direct Connect
【AWS Black Belt Online Seminar】AWS CodeDeploy
【AWS Black Belt Online Seminar】AWS CodeBuild
【AWS Black Belt Online Seminar】AWS CodeStar & AWS CodePipeline
【AWS Black Belt Online Seminar】AWS IoT Core（要らないっぽい？）
【AWS Black Belt Online Seminar】Amazon Route53 Hosted Zone
【AWS Black Belt Online Seminar】Amazon Route53 Resolver

【AWS Black Belt Online Seminar】Amazon Key Management System(全然分からない)
【AWS Black Belt Online Seminar】AWS Security Hub
【AWS Black Belt Online Seminar】AWS CloudFormation
【AWS Black Belt Online Seminar】AWS CloudFormation deep dive
【AWS Black Belt Online Seminar】Amazon Kinesis Video Streams(途中までしか見ていない)
【AWS Black Belt Online Seminar】AWSアカウント シングルサインオンの設計と運用
【AWS Black Belt Online Seminar】AWS Fargate
【AWS Black Belt Online Seminar】Amazon Elastic Container Service (Amazon ECS)
【AWS Black Belt Online Seminar】 AWS Site-to-Site VPN
[AWS Black Belt Online Seminar] AWS Glue
【AWS Black Belt Online Seminar】Amazon EMR
【AWS Black Belt Online Seminar】Amazon SageMaker Basic Session
【AWS Black Belt Online Seminar】 AWS Lambda Part1
【AWS Black Belt Online Seminar】 AWS Lambda Part2
【AWS Black Belt Online Seminar】 AWS Lambda Part3
【AWS Black Belt Online Seminar】AWS Lambda Part4
【AWS Black Belt Online Seminar】AWS Systems Manager
【AWS Black Belt Online Seminar】Amazon EventBridge
【AWS Black Belt Online Seminar】AWS Certificate Manager
AWS Application Migration Service Architecture
【AWS Black Belt Online Seminar】Amazon DocumentDB (with MongoDB Compatibility)
【AWS Black Belt Online Seminar】Amazon Redshift

〇AWS Skill Builderで見たもの
Architect Learning Plan(Japan)
https://explore.skillbuilder.aws/learn/lp/393/Architect%2520Learning%2520Plan%2520%28Japan%29

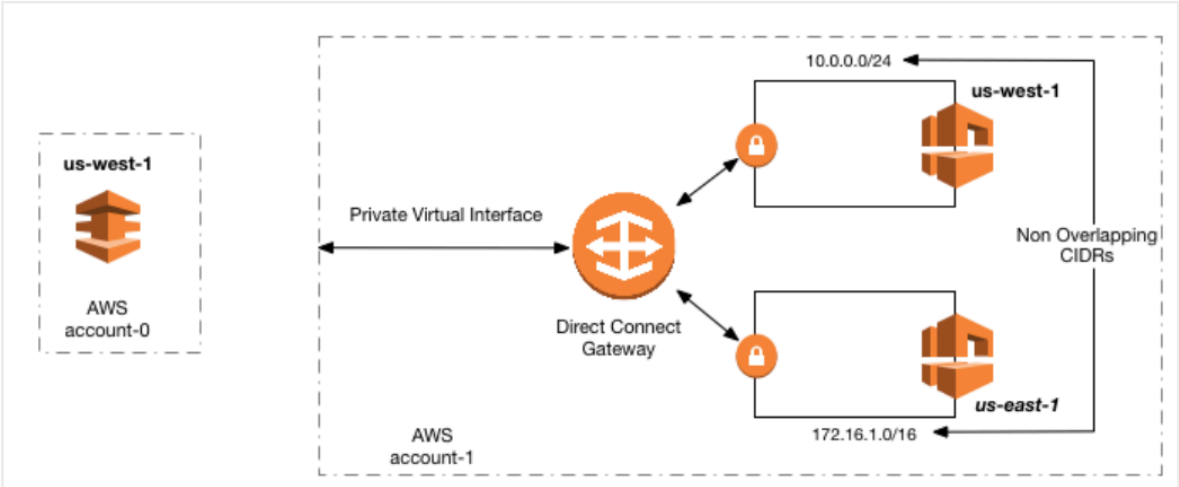
〇Udemy問題集
　Prepare for your SAP-C02 exam. 180 high-quality practice test questions written from scratch with detailed explanations!
　Be AWS Certified Solutions Architect Professional. Full Amazon Web Services Solution Architecture deep-dive for SAP-C02
　AWS Certified Solutions Architect - Professional Official Practice Question Set (SAP-C01 - Japanese)

〇AWS Web問題集

●試験メモ
　・S3は動画ファイルをホストするための最もコスト効率の高いソリューションを提供す

る。（静的だけではない。）
・S3はプレフィックスごとに1秒あたり3,500のPUT/COPY/POST/DELETEまたは5.500のGET/HEADリクエストを達成できる
10このプレフィックスを作成して読み取りを並列化させることで、読み取りパフォーマンスを1秒あたり55,000リクエストにスケーリングできる
・SAMLベースのフェデレーション中に、DevelopmentDeptの属性をAWS Security Token Serviceセッションタグとして渡すことができる。
・**AthenaはS3 Glacierに入ると分析ができなくなる。**
・外部IDはSTSのAssumeRole APIに渡すことができるデータの一部
・AWS Elemental MediaConvert は、ブロードキャストグレードの機能を備えたファイルベースの動画変換サービス
・Kinesis Data Streamsのレコードにアクセスできるのは、ストリームに追加された時点から最大24時間。
拡張データ保持を有効にすることにより、制限を最大7日間まで上げることができる。
・複数リージョンのcloudtrailログを1つのS3バケットに書き込むことができる。
・IPsec（Security Architecture for Internet Protocol）は、インターネットなどにおける通信で暗号化を担うプロトコル
・VPCでインスタンスがカスタマーゲートウェイに到達できるようにするには、VPN接続で使用されるルートを含め、仮想プライベートゲートウェイ(VGW)を指すようにルートテーブルを設定する必要がある。
仮想プライベートゲートウェイとは、1つのVPCとオンプレミス環境をVPN経由で接続する時にVPCに設置する通信の出入口
・ELBはトラフィックスパイクを処理するためにアプリケーションサーバーが4台増えて11個のIPアドレスを使用する。
・FGAC(Fine Grained Access Control)は、DynamoDBテーブルの所有者がテーブル内のデータに対して詳細なコントロールを行うための機能。
具体的には、テーブル所有者は誰がテーブルのどの項目や属性にアクセスでき、どのようなアクションを実行できるかを指定できる。
・**RDSはDNSエンドポイントのみを使用して接続することができ、IPアドレスを使用しない。**
・レイテンシーベースのルーティングに基づいて、地理的な場所に送信するためには、各リージョンにRoute53の①**加重ベースのレコードを定義し、②レイテンシーエイリアスレコード**を作成する。
・DynamoDB Streamsは、DynamoDBテーブル内の項目に加えられた変更に関する情報の順序づけされた情報。
・**ゲートウェイエンドポイントでアクセスできるのは、同じリージョンのリソースのみ**
・CloudFormationでOpsWorksを使うときは、「AWS::OpsWorks::Stack」とする。
・**EBSおよびRDSのスナップショットは、そのリージョンで作成したものを違うリージョンにコピーできるが、コピーしないで直接違うリージョンに作成できない。**
Lambdaなどを使って対応する必要あり。
・EBSボリュームの種類
gp2の容量を3TBに増やすと9000IOPS（3IOPS/GB)になり、300ドルで実現可能
・NAS(ネットワークHDD)とは、ネットワーク上に接続することができるハードディスク。
・AWS Application Discovery Serviceは、オンプレミスサーバーのホスト名、MACアドレス、ネットワークスループット、DNSサーバー、CPU、メモリ、ディスク使

用量などのメトリクスを収集できる。
　システム間のネットワーク接続を特定することもできる。
・プロキシ統合とは、API Gatewayでマッピングをしなくても、リクエストに入っているパラメータをまるっとLambdaに渡してくれる機能。
　リクエストパラメータを自動でいい感じに変数に詰めてLambdaに渡してくれる。
・汎用SSDストレージを4TBから6TBに増やしてもIOPSの向上にならないた。
・ALBのアクセスログは、クライアントIPアドレス、接続タイプ、ユーザーエージェントなどの詳細を含むアプリケーションアクセス情報を提供する。
・Route53がWebサービスのエンドポイントのヘルスチェックを合格にするパターンは、2xxまたは3xxでエンドポイントが応答する必要がある。
　また文字列の一致はレスポンス本文最初の5,120バイト内に出現している必要がある。
・RDSでLambdaを使用する
　RDSデータベースからのイベント通知をLambdaで処理できる。
　（RDS　→　SNS　→　Lambda）
・approved-amis-by-id
　実行中のインスタンスで使用されているAMIが指定したものかどうかを確認する。
　承認済みのAMI IDのリストを指定する。
・Directconnectゲートウェイは、複数のリージョンでの通信を可能にする。



・EMRFSの使用
　全てのAmazon EMRクラスターがEMRとAmazon S3の間で通常のファイルを直接読み書きするために使用するHDFSの実装
　EMRFSはHadoopで使用するために、S3で永続的なデータを保存できるようにしながら整合性のあるビューやデータの暗号化といった機能も提供する
・S3のイベント通知ターゲット
　・SQS
　・SNS
　・Lambda
・CloudTrailログはリアルタイムではなく、通常15分以内に配信される。
・1つのVPCでのぴあリング接続の上限は125個
・AWS Backupは、AWSサービス全体のデータバックアップを集中管理して自動化できる
　Organizationsのバックアップポリシーで簡単にバックアップポリシーを作成できる。

・S3 Glacier Deep Archiveの取得時間は12時間
・Amazon Auroraグローバルデータベース設定には、マネージドRPOがサポートされている
プライマリクラスターのリージョンとは別に、セカンダリクラスターを作成することで、低いRPOとRTOを実現することができる。
・EBS マルチアタッチ：複数のインスタンスを一つのEBSに接続することができるが、同じAZにある必要がある。
・cloudformationの**stacksetsを使うことで、複数リージョン・複数アカウントに一括でスタックをデプロイすることができる**
・**SQS標準キューをFIFOキューに変更することはできない。**
•S3バケット →Lambda→ES
•SCP は、管理アカウントのユーザーやロールには影響を与えません。SCP は、組織内のメンバーアカウントにのみ影響を与えます。
・IBM Db2は、SCTとDMSを使用してRDS for MySQLなどに移行できる。
・パブリック仮想インターフェースを利用したVPN over DXは、一貫したネットワークパフォーマンスとセキュリティを提供する。
・Direct Connect だけでは IPsec 暗号化は提供されないため、VPN がオンプレミスから AWS ネットワークへ IPsec 暗号化を提供します。プロキシサーバーはオンプレミスのトラフィックを受け入
・エッジ最適化エンドポイントはAPI Gatewayのキャッシュ
・Amazon Aurora ServerlessのData APIを使用すると、Aurora Serverless DBクラスターへのウェブサービスインターフェイスを操作できる。
エンドポイントを利用して、接続を管理せずにSQLステートメントを実行することができる。
・EFS：LinuxやUnix系サーバーで使う（NFS)、FSx：Windows系のサーバーで使う
・**VPCピアリングだと、通信が一度インターネットを経由する。**
PrivateLinkを使うと、通信が**AWS**のネットワーク内で完結してインターネットに出ない。（**事前にNLBを作成することが必須)**
・S3バケットのリクエスタ支払い問題は、クロスアカウントロールはNG。ロールが属するアカウントにリクエスト料金が発生するため。
・**CloudTrailはCloudWatchLogsではなく、CloudWatchEventsと統合する。**
・GuardDuty は、API に不正にアクセスしようとする悪意のある試みをブロックできません。むしろ、そのような試みを監視/検出することしかできません。
・Amazon Kinesis Data Firehose は、DynamoDBに直接書き込むことはできない。
・VPN CloudHub：VPCに対して、複数のオンプレミスサーバーを接続することができる。
・SQSの遅延キューとメッセージタイマー：特定のメッセージの処理を遅らせるのがメッセージタイマーで、キュー全体の処理を遅らせるのが遅延キュー
・EFSはオンプレミスとEC2の両方から同時アクセスが可能（オンプレミスからは、DirectConnectやVPNを介して通信する）
・DirectConnectなどの10Gbpsはビットを指している。問題文の転送量（150TBなど）はバイトを指しているので、転送速度を8割る必要がある。
・S3のサーバーアクセスログは、SSE-S3のみで暗号化できる。KMSはサポートされていない。
・**S3 Standard-IA と S3 One Zone-IA の最小ストレージ期間料金は 30 日間**

●作りたいもの
　キーワードを絞って、そのキーワードの動画を自動抽出してくれるアプリ（高評価数、再生回数など）

●新旧比較

【比較】AWS SAPの旧試験と新試験の違いまとめ【SAP-C02】｜フリーランスネットワークエンジニアの技術ブログ

nokonokonetwork.com