

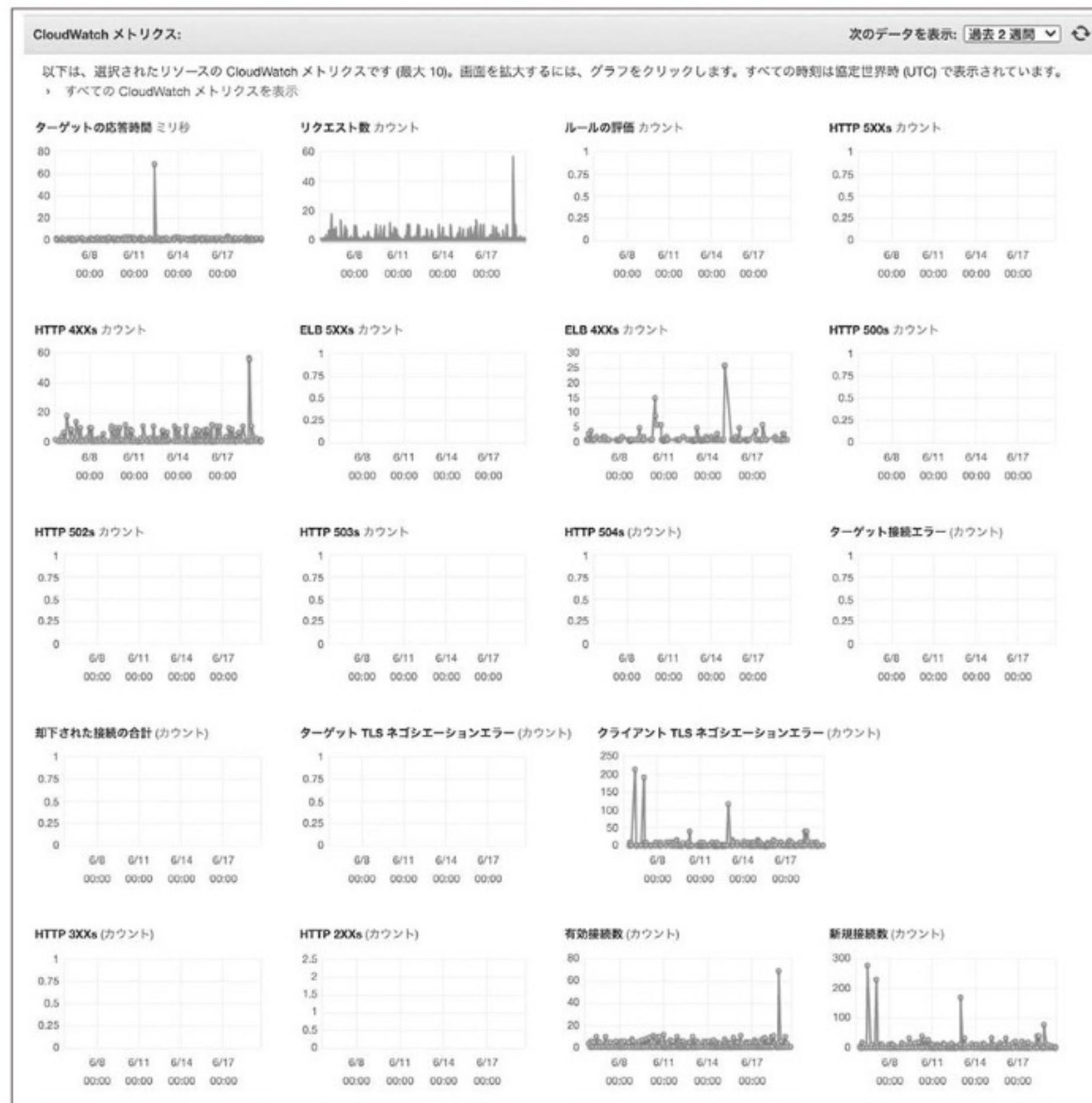
スケーリングする機能を有しています。ELBは複数のアベイラビリティゾーンをまたいで構築されるようになっており、高い可用性と信頼性を保ちながらパフォーマンスを確保できることを特徴としています。ELBは一般的なロードバランサーが持つ機能は維持しながらも、ソフトウェアのアップデートなどはAWSマネージドで行ってくれるため、非常に運用もしやすくなっています。以降、ELBの特徴を順次説明していきます。

●ヘルスチェック / リクエストモニタリング・ログ

ELBでは、ヘルスチェックとしてロードバランシング先となるターゲット（EC2インスタンスやコンテナなど）に対しリクエストを送信して死活監視を行い、正常実行されているターゲットにルーティングします。死活監視するリクエストは、プロトコルやポート、ターゲットとなるURLパス、タイムアウト時間、正常・異常とみなすしきい値、リクエスト送信間隔など詳細な設定が可能です。

また、CloudWatchを利用して、下記のイメージにあるように、正常/異常リクエストの記録やパフォーマンスのモニタリングができます。メトリクスは60秒間隔で収集され、統計情報として平均や合計、最大・最小値などの詳細を表示することも可能です。

【ELBのメトリクスを表示するコンソール画面】



ELBでは、最短5分間隔でアクセスログを取得でき、S3バケットを指定してログを自動保管することもできます（デフォルトでは無効化されています）。ロードバランサーライアーアクションによって、ログの形式は異なりますが、ALBでのHTTPSプロトコルでは、以下のような形式で出力されます。

```
https 2018-07-02T22:23:00.186641Z app/
my-loadbalancer/50dc6c495c0c9188
192.168.131.39:2817 10.0.0.1:80 0.086 0.048 0.037 200 200 0 57
"GET https://www.example.com:443/ HTTP/1.1" "curl/7.46.0" ECDHE-RSA-
AES128-GCM-SHA256 TLSv1.2
arn:aws:elasticloadbalancing:us-east-2:123456789012:targetgroup/
my-targets/73e2d6bc24d8a067
"Root=1-58337281-1d84f3d73c47ec4e58577259" "www.example.com"
"arn:aws:acm:us-east-2:123456789012:certificate/
12345678-1234-1234-1234-123456789012"
1 2018-07-02T22:22:48.364000Z "authenticate,forward" "-" "-"
10.0.0.1:80 200 "-" "-"
```

ログのフィールドにはリクエストの詳細が表示されます。具体的な内容については割愛しますが、記録される内容はELBの公式ガイド^{※24}を参照してください。また、アクセスログの記録はベストエフォートで行われるもので、すべてのリクエストを完全に記録するものではありません。完全なアクセスログが必要な場合はCloudTrailでのログ記録を使用できます^{※25}。使い分けとしては、URLごとのアクセス状況の解析する場合には前者のアクセスログを、監査記録などは後者のCloudTrailを使用するとよいでしょう。

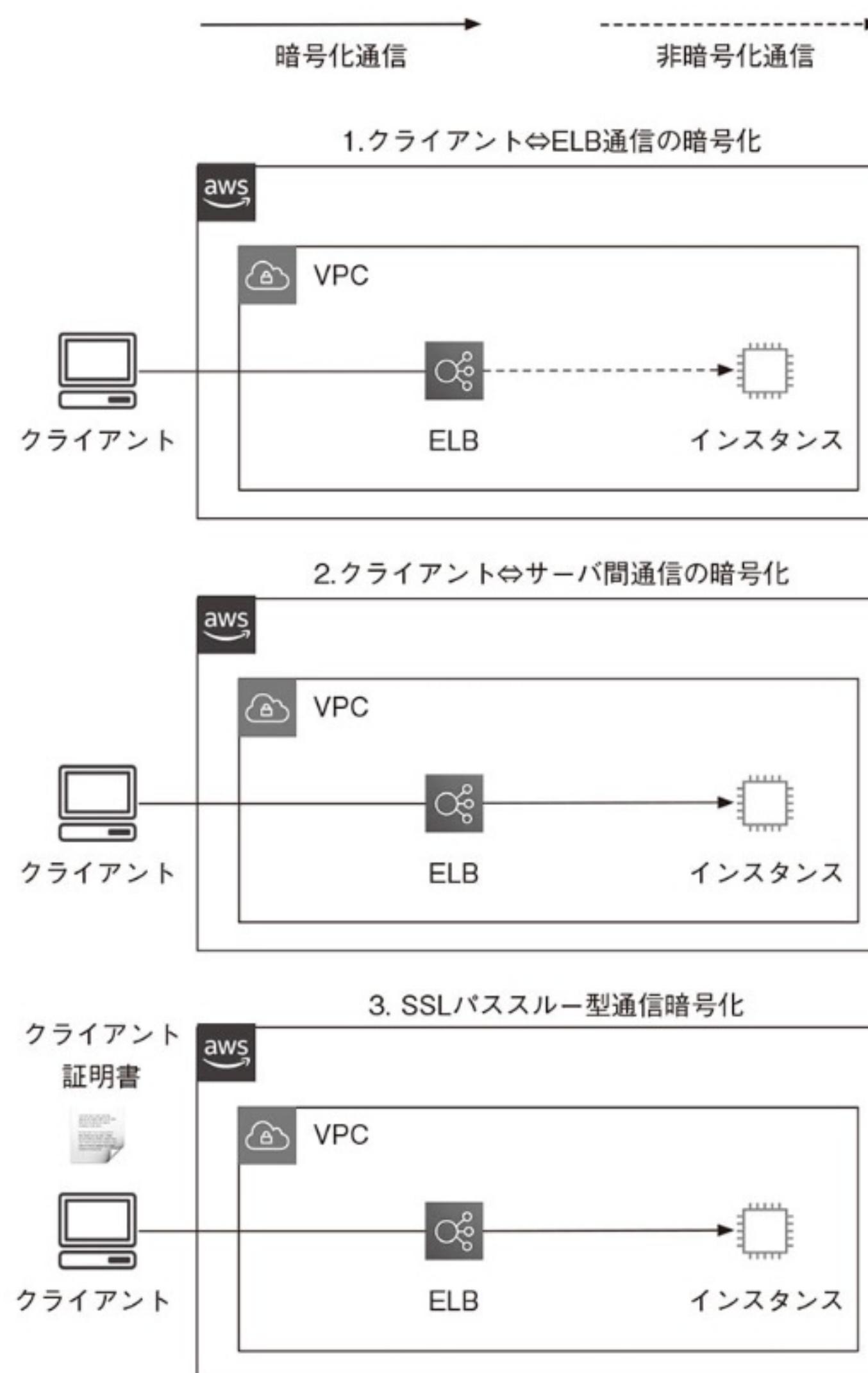
●SSL/TLS ターミネーション

ELBでは、SSL/TLSを使った暗号化通信を利用でき、以下のようないくつかのパターンで、クライアント・サーバ間の通信を暗号化できます。通常、ELBが暗号化の終端として復号処理をするSSL/TLSターミネーションが行われます。

^{※24} https://docs.aws.amazon.com/ja_jp/elasticloadbalancing/latest/application/load-balancer-access-logs.html

^{※25} https://docs.aws.amazon.com/ja_jp/elasticloadbalancing/latest/application/load-balancer-cloudtrail-logs.html

【クライアント・サーバ接続の暗号化通信のパターン】



1. クライアント ⇄ ELB通信の暗号化

ELBでSSL/TLSターミネーションを行い、ELBからバックエンドターゲットへの通信は暗号化を行わないパターンです。

2. クライアント ⇄ サーバ間通信の暗号化

ELBでSSL/TLSターミネーションを行ったのち、ELBからバックエンドターゲットも新たなSSL/TLS暗号化通信を行うパターンです。暗号化処理のオーバーヘッドがかかる反面、よりセキュアに通信を保護しながらバックエンドサーバとアクセスできます。

3. SSLパススルーパターン

ELBではなく、バックエンドのEC2などでSSL/TLSターミネーションを行うパターンです。クライアント側でも証明書が必要なユースケースなどで適用されます。なお、AWSでは後述するNLBでSSLパススルーパターンを実現できます。詳細はこの後の「NLB」の項を参照してください。



SSL/TLSターミネーションでユースケースに応じたパターンについて問われる場合がありますので、対応方法をしっかり押さえておくようにしましょう。

● Connection Drain

ロードバランサーを終了する際に、残ったリクエストの処理のために遅延時間を設けて終了します。デフォルトでは300秒待ってから終了処理に入りますが、最大1時間まで延長できます。

●ステイッキーセッション

同じクライアントからリクエストを常に固定したターゲットに振り分けます。WebSocket通信や、ショッピングカートのような機能を実現する際にサーバのセッションデータを利用するなど、サーバとステートフル通信が必要な場合に利用されます。リクエストを振り分けるための識別子にはELBによって生成されるCookieが使用されます。



ステイッキーセッションを利用する際のユースケースについて問われる場合があります。用途を理解しておきましょう。

● WebSocket 対応

プッシュ通信などの双方向通信を実現するWebSocketをサポートします。

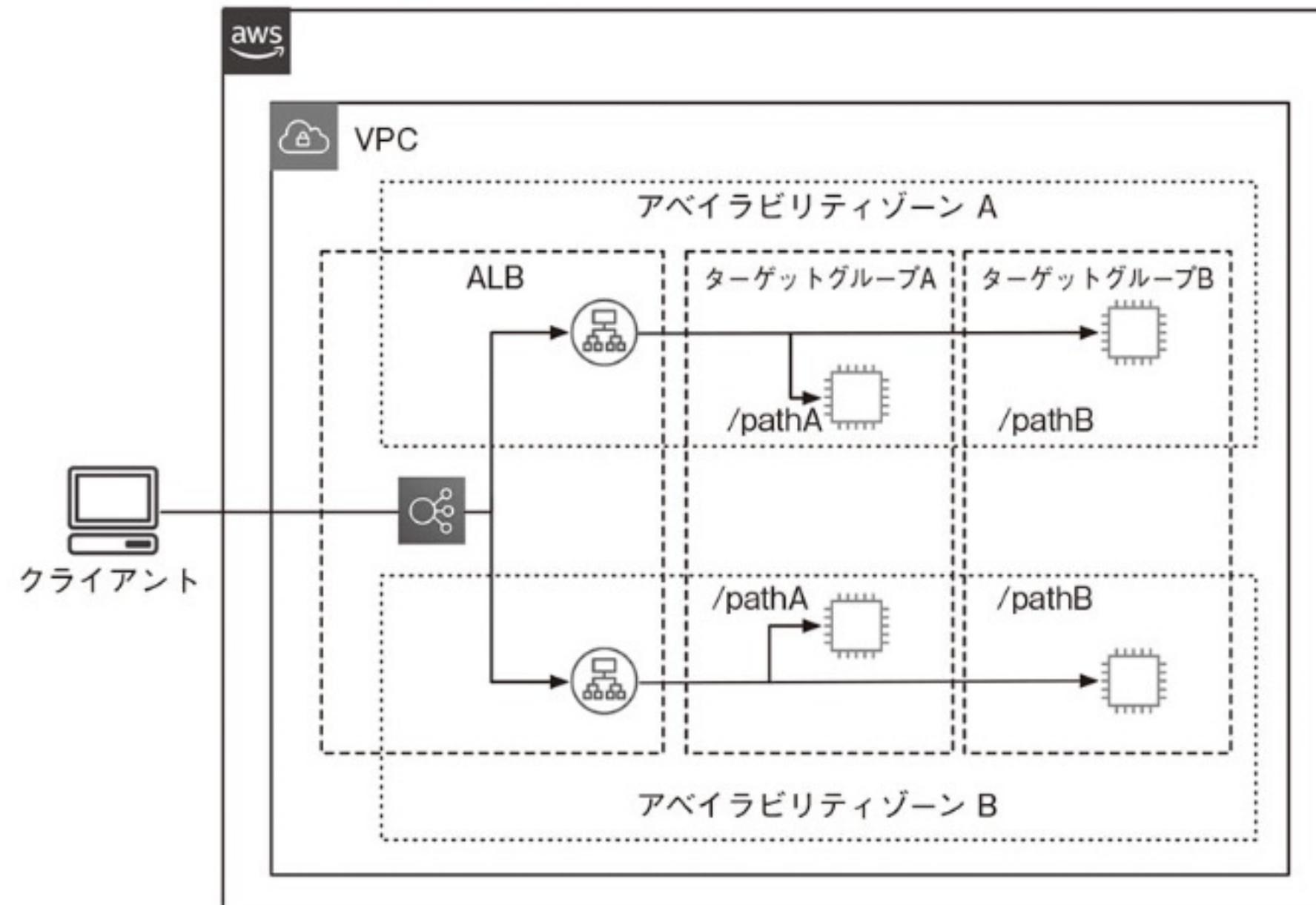
2 ALB

ALBはリバースプロキシ型のロードバランシングサービスです。負荷分散はOSI 7層モデルでの第7層（アプリケーション層）レベルでルーティングが可能です。リスナーと呼ばれるコンポーネントを作成し、使用するプロトコルとポートを設定して、ルーティングルールを設定します。以下は、リクエストURLのパスパターンに応じたルーティングを行うパスベースルーティングのイメージです。



リバースプロキシ型とは、応答通信もロードバランサーを経由してクライアントへ届くタイプのことです。

【ALBのルーティングイメージ】



例えば、「<https://example.com/pathA>」や「<https://example.com/pathB>」といったURLのパス単位で、ターゲットグループ（アベイラビリティゾーンをまたいでインスタンスやコンテナをまとめた仮想的なグループ）を振り分け指定し、より細かく負荷分散することができます。また、ALBではさまざまなルーティングオプションを選択できます。ターゲットグループには、以下の図のように優先順位をつけて適用条件を設定でき、複数のルーティングを組み合わせることもできます。

【ALBのルーティング設定イメージ】

This screenshot shows the AWS CloudFormation configuration for an Application Load Balancer (ALB). It defines four routing rules for port 80:

- Rule 1:** If path is /aws-code-pipeline/*, then forward to ecs-sample-codepipeline-service:1 (100% weight).
- Rule 2:** If path is /backend/*, then forward to ecs-sample-backend-service:1 (100% weight).
- Rule 3:** If path is /frontend/*, then forward to ecs-sample-frontend-service:1 (100% weight).
- Default Rule (Last):** If none of the above conditions are met, then forward to sample-ecs-target-group:1 (100% weight).

ALBでは、以下のルーティング設定を組み合わせることができます。

●パスベースルーティング

リクエストURLのパスパターンに応じてルーティングします。

●ホストベースルーティング

HTTPヘッダのhostフィールド（URLのドメイン）にもとづいてルーティングします。1つのロードバランサーで異なるドメインのアプリケーションを振り分ける場合などに使用されます。

●HTTPヘッダベースルーティング

HTTPヘッダの値に応じてルーティングします。対象となるヘッダや適用値は任意に設定できます。

●HTTPメソッドルーティング

HTTPメソッド種別に応じてルーティングします。

● クエリ文字列ベースルーティング

HTTPリクエストのクエリパラメータの値に応じてルーティングします。対象となるクエリパラメータや適用値は任意に設定できます。

● ソースIPアドレスCIDRベースルーティング

リクエスト元のソースIPアドレスCIDRに応じてルーティングします。

ALBにはそのほかにも以下のような機能があります。

● OIDC プロバイダ認証

AWS Cognitoや、Amazon、Facebook、GoogleなどのソーシャルなOIDC IDプロバイダと連携するユーザ認証をサポートしています。認証が成功したリクエストは、ALBによりアクセストークンやIDトークン、ユーザクレーム（ユーザ情報）がHTTPヘッダに付与されてリクエストが転送されます^{※26}。こうしたトークンを使用して、リクエストが転送されたアプリケーション側で、IDプロバイダを通じた認可制御などに利用することができます。

● ネイティブな HTTP/2・gRPC 通信サポート

HTTP/2ではHTTPヘッダの圧縮などでデータサイズが軽減されており、より高速・低コストなエンドツーエンド通信が可能です。gRPCは、Googleが高速なサービス間通信技術としてオープンソース化したリモートプロシージャコール（Remote Procedure Call）と呼ばれる技術です。ALBはHTTP/2とgRPCでの通信をネイティブで利用できるほか、gRPCステータスコード、リクエストメトリクス、アクセスログ、ヘルスチェック機能をサポートします^{※27}。

● AWS サービスとの連携

ALBでは、以下のAWSサービスと連携できます。

● EC2 / ECS / AutoScaling

EC2とECSは、トラフィックのルーティング先となるVPC内のコンピューティングサービスです。AutoScalingと組み合わせることにより、需要に応じて自動的にリソースを増減できます。

● Lambda

ALBではLambda関数をターゲットグループに設定できます。また、次のような状況下でヘッダ構文の検証を有効化できます。

- a. クライアントからのリクエストまたはLambda関数からのレスポンスに複数の値を持つヘッダが含まれている場合
- b. 同じヘッダが複数回含まれている場合
- c. 同じキーに対して複数の値を持つクエリパラメータが含まれている場合

● Route 53

Route 53では、エイリアスコードを用いたドメイン名とロードバランサーDNSの関連付けを行えます。

● Certificate Manager

Certificate Managerでは、ALBで使用するSSL証明書の発行・設定を簡単に行えます。

● AWS WAF

Webアクセスコントロールリスト（Web ACL）のルールにもとづいてリクエストを許可またはブロックできます。

● AWS Global Accelerator

Global Acceleratorは、AWSのエッジロケーションを使ってALBへアクセスする高速なグローバルネットワークロードバランサーです。フロントにGlobal Acceleratorを配置して、複数のAWSリージョンのALBにトラフィックを分散することで、可用性・パフォーマンスを向上させます。



ALBからLambda関数を呼び出す際に使用する、ヘッダを用いたパラメータの受け渡し方法等を問われる場合があります。少々細かい設定方法ですが、ターゲットとしてLambda関数を設定する場合の注意点は公式ガイド^{※28}も参考にするようしてください。

3 NLB

※26 https://docs.aws.amazon.com/ja_jp/elasticloadbalancing/latest/application/listener-authenticate-users.html

※27 <https://aws.amazon.com/jp/blogs/news/new-application-load-balancer-support-for-end-to-end-http-2-and-grpc/>

NLBは主にOSI 7層モデルでの第4層（トранスポート層）での負荷分散用途で使用されるL4ロードバランサーです。ALBのようなL7ロードバランサーでは、アプリケーション

※28 https://docs.aws.amazon.com/ja_jp/elasticloadbalancing/latest/application/lambda-functions.html

ン層レベルでロードバランスするために、一般的にロードバランサー内でTCPコネクションを一度終端して振り分けします。それに対し、NLBのようなL4ロードバランサーではIPアドレスとポートをもとに振り分けるため、より低負荷で負荷分散が可能な特性を持っています。

NLBには次のような特徴・機能があります。

●高可用性・高スループット・低レイテンシ

单一もしくは複数のアベイラビリティゾーン構成が選択でき、毎秒数百万のリクエストを低レイテンシで処理できます。

●クライアントのIPとポートの保持

NAT変換型L4ロードバランサー同様、通信はロードバランサーをすべて経由しますが、クライアントのIPアドレスとポートが保持され、DSR型L4ロードバランサーのように直接クライアントと通信しているかのように振る舞います。

●ロードバランサーのIPアドレスが固定

NLB作成時に割り当てられたIPアドレスか、Elastic IPアドレスのいずれかを設定すると、その後はロードバランサーのIPアドレスが固定となります。オンプレミスネットワークのファイアウォールの制約でIPアドレスの固定が必要なユースケースなどでの使用が想定されています。

●暖機運転（Pre-Warming）申請が不要

ALBやCLBでは通常のスケーリングで間に合わない急激なトラフィックアクセスの増加が見込まれる場合、事前に暖機運転申請を行い、ロードバランサーをスケールアウトさせておく必要がありますが、NLBでは必要ありません。固定のIPアドレスが維持されたままで動的にスケールされます。

●VPCエンドポイント（PrivateLink）のサポート

PrivateLinkを使用してAWSネットワーク内でNLBをサービスエンドポイントとして公開できます。異なるVPCのWebサービスにインターネットを経由せずアクセスしたい場合に使用されることを想定しています。

NLBはALBと同じく、リスナーやロードバランスされる対象（EC2やECSなど）をターゲットグループとして設定します。

●NLB利用時の考慮点

NLBそのものにはセキュリティグループ（34ページ参照）を設定することはできません。ターゲットグループ側には送信元のIPアドレスがクライアントのまま届くことになります。34ページでも説明したとおり、ALBのターゲットグループにはアクセス元を制御するセキュリティグループを設定しますが、NLBでは、そのセキュリティグループ内の通信許可ソースとしてNLB自身（正確にいえばNLBに適用するセキュリティグループ）を指定することができません。そのため、ターゲットとしてECSを選択して動的ポートマッピングの使用を検討する場合、アクセス可能なポートを絞るなどの留意が必要です。

●SSLパススルー通信

前項のSSL/TLSターミネーションの中でも触れたとおり、セキュリティ要件が高く、クライアントの証明書による認証が必要なユースケースがあります。こうした要件を満たすためにバックエンドのEC2でエンドツーエンドのSSL/TLSターミネーションを行う場合、NLBでTCP転送するようにリスナーを設定し、SSL通信をパススルーします。

【NLBのリスナー設定】

Protocol	Port	Default action
TLS	: 443	Forward to Select a target group Create target group
TCP	1-65535	
TLS		
UDP		
TCP_UDP		

4 その他のロードバランサー

ALB、NLBのほかにも、ロードバランサーにはGLBおよびCLBがあります。ロードバランサーとしての用途は限られるため、ここでは簡単に紹介します。

●GLB

GLB（Gateway Load Balancer）は、2020年11月に発表された4番目のロードバランサーですが、これまで紹介してきた、不特定多数のクライアントからのリクエストを受け付けるロードバランサーとは少々用途が異なります。

Gateway Load Balancerでは、インターネットゲートウェイまたは仮想プライベートゲートウェイにおける送受信ネットワークトラフィックを別のVPCネット

ワークへ中継します。中継する意図は、主にゲートウェイを通じて入ってきたトラフィックに不正な攻撃が含まれていないか、サードパーティ製のセキュリティミドルウェアがインストールされたEC2インスタンスで構成されるネットワークへ転送し、検出・対応するためです。

このような構成をとる場合、従来は複雑な設定で構成する必要があり、ルーティングの冗長化やパフォーマンスの確保が課題となるケースがありました。しかし、Gateway Load Balancerを導入することで、こうした課題をクリアしながらサードパーティのセキュリティミドルウェアに透過的にアクセスすることが可能になります。

●CLB

CLBは2009年にリリースされて以降、Elastic Load Balancingという名称でサービス提供されてきましたが、2016年にALB、2017年にNLBが登場したことで、CLB（Classic Load Balancer）に改称されました。CLBはALBとNLBの項で解説したL4およびL7双方のロードバランサー機能を提供してきたサービスです。ただし、L7としての利用ではALBのような詳細なルーティングの設定はできません。ALBとNLBの登場以降はメンテナンスマードに近い位置付けであり、後継サービスのほうが低コストかつ機能もほぼ代替可能です^{※29}。古くから使用されているレガシーなプラットフォームであるEC2-Classicを使用しなければならないなどの制約がない限り、ALBとNLBを使用するほうがよいでしょう。

2-7 Amazon ECS

「Amazon Elastic Container Service (Amazon ECS)」は、Dockerコンテナをスケーラブルかつ簡単に実行・停止・管理できるコンテナ管理サービスです。さまざまなAWSサービスと連携しながら、多数のコンテナを高い可用性を保ちつつ高速に実行することができます。

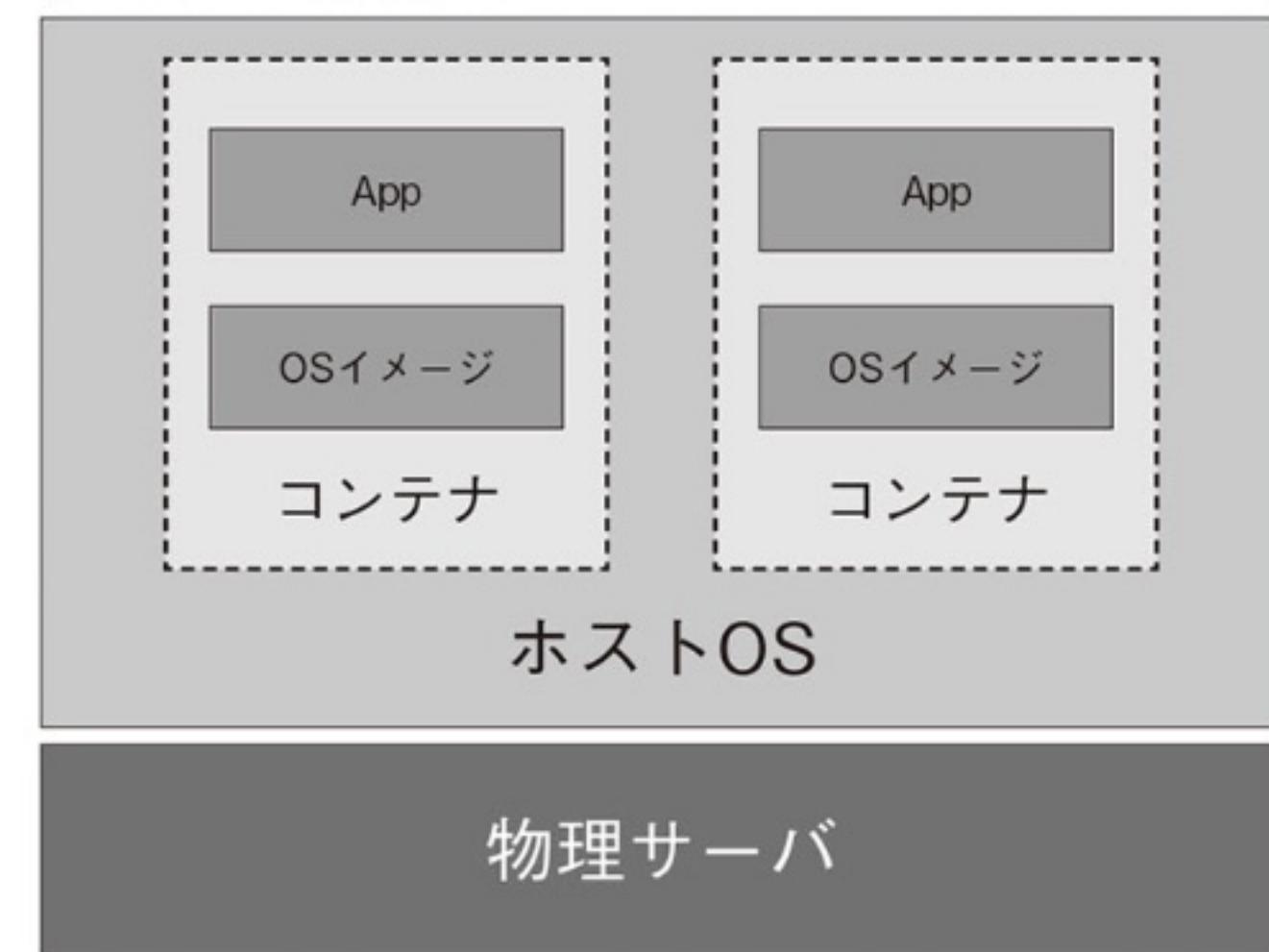
1 コンテナとオーケストレーション

ECSはDockerコンテナを扱うサービスです。ECSを解説する前に、まずコンテナ自体とコンテナを管理するためのオーケストレーションツールについて説明します。

●コンテナ

コンテナはホストとなるLinux OS上で、OSイメージとアプリをひとまとめにし、1つのプロセスとして動作します。下図の点線で示した範囲が1つのコンテナとなります。コンテナにはOSのイメージが割り当てられますが、OSイメージ自体がゲストOSとして動作するわけではありません。イメージ内の指定されたプロセスのみが実行されます。図では、Appとして示されたアプリケーションが「実行されているプロセス」に相当します。

【コンテナの概念図】



※29 <https://aws.amazon.com/jp/elasticloadbalancing/features/#compare>

コンテナでは、ハイパーバイザ上の仮想マシンのようにゲストOS自体のさまざまなプロセスが実行されるわけではありません。そのため、より軽量・高速に仮想イメージを起動できます。また、OSイメージと起動プロセスをまとめたコンテナイメージは可搬性があり、別のホストOS上でもそのまま実行することができます。

コンテナを実行する代表的なソフトウェアとしてDockerが挙げられます。Dockerは、ホストOSとなるLinux上でデーモンサービスとして動作し、主に以下のような処理を実行します。

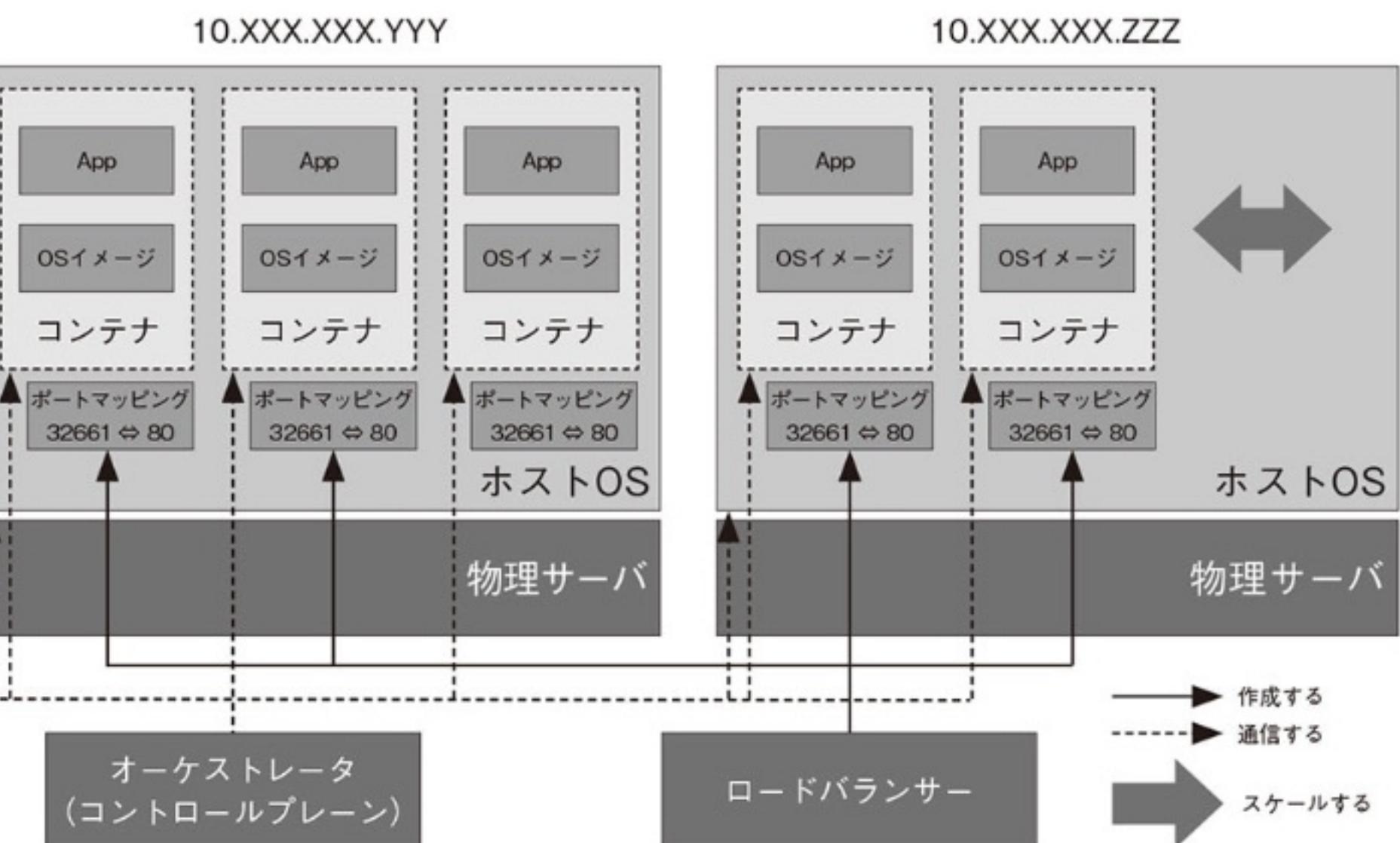
- ・「Dockerfile」という設定ファイルを使って、OSイメージや実行プロセスをコンテナイメージとして作成する。
- ・ホストOS上でコンテナイメージをプロセスとして実行する。
- ・ブリッジ（Linuxが提供する仮想のネットワークスイッチ）を使って、ホストOSのポートとコンテナのポートをマッピングし、コンテナ内部と外部のネットワーク通信を行う。
- ・Linuxが提供するControlGroup機能を使って、コンテナ上で実行されるプロセスにCPUとメモリを割り当て制御する。
- ・ネットワーク上のレジストリに対してコンテナイメージを送受信する。

●コンテナオーケストレーション

コンテナは仮想マシンと同様、1つのホストOS上で、複数のアプリケーションやデータベースなどのミドルウェアプロセスを動かせます。ただし、実行するコンテナが増え、かつ動的に作成・破棄する場合には、前述したようなDockerを単純に用いるだけの方法では運用が困難になってきます。

例えば、Webサービスを提供するアプリケーションがアクセス数に応じて、アプリケーションを実行するコンテナを増やす（スケールする）例を考えてみましょう。リクエスト数が増えてきたら、それを処理するためのアプリケーションコンテナの数を増やすことで対応します。ただし、1つのサーバでアプリケーションを複数実行している場合は、ホストOSのIPアドレスやその上で動くコンテナへ接続するためのポートを、ロードバランサーが知っておかなければなりません。コンテナの増減に応じて、手動でロードバランサーの設定を変えることもできますが、コンテナの数が増えてくるとそれも現実的ではありません。そこで、以下の図のように、コンテナの実行や接続するIPアドレスやポートを管理する役割を持つオーケストレータが必要になってきます。オーケストレータはホストOSの作成のほか、コンテナを作成・プロビジョニング・実行し、マッピングされるホストOSのIPアドレスやコンテナのポートを管理して、適宜動的にコンテナへ割り当てます。

【オーケストレータの役割】



上記の図の例では、IPアドレス10.XXX.XXX.YYYおよび10.XXX.XXX.ZZZを持つ2つのサーバで5つのコンテナが実行されています。コンテナには、オーケストレータによって適切なポートが割り当てられます。このときロードバランサーに必要なのは、コンテナが実行されているホストのIPアドレスと、コンテナとの通信がマッピングされるポートといった接続情報です。オーケストレータは有効なコンテナの接続情報を動的にロードバランサーと連携し、自動的にコンテナを増減するオートスケーリングを実現します。

なお、コンテナが実行されるリソースはデータプレーン、コンテナを管理するオーケストレータはコントロールプレーンとも呼ばれます。

2 ECSの構成要素と概念

ECSでは、複数のDockerコンテナが実行されているEC2インスタンスを、「クラスタ」と呼ばれる単位で扱います。Dockerコンテナを単純にEC2インスタンス上で運用する場合と比較して、ECSではリージョン内の複数のアベイラビリティゾーンをまたいでコンテナを実行できるため、可用性の高い運用が可能です。

2023年12月時点では、データプレーンとして、EC2にクラスタを構築し、コンテナをプロジェクトする「EC2起動型」と、実行するクラスタ自体はマネージドとしてAWSが自動で管理し、コンテナだけを意識する「FARGATE」、ECS Anywhere^{※30}を使って、オンプレミスなどAWS外のサーバリソースを利用する「EXTERNAL」に分類できます。コンテナの元となるDockerイメージはECRやDocker Hubなどのレジストリから取得されます。

※30 <https://aws.amazon.com/jp/blogs/news/getting-started-with-amazon-ecs-anywhere-now-generally-available/>

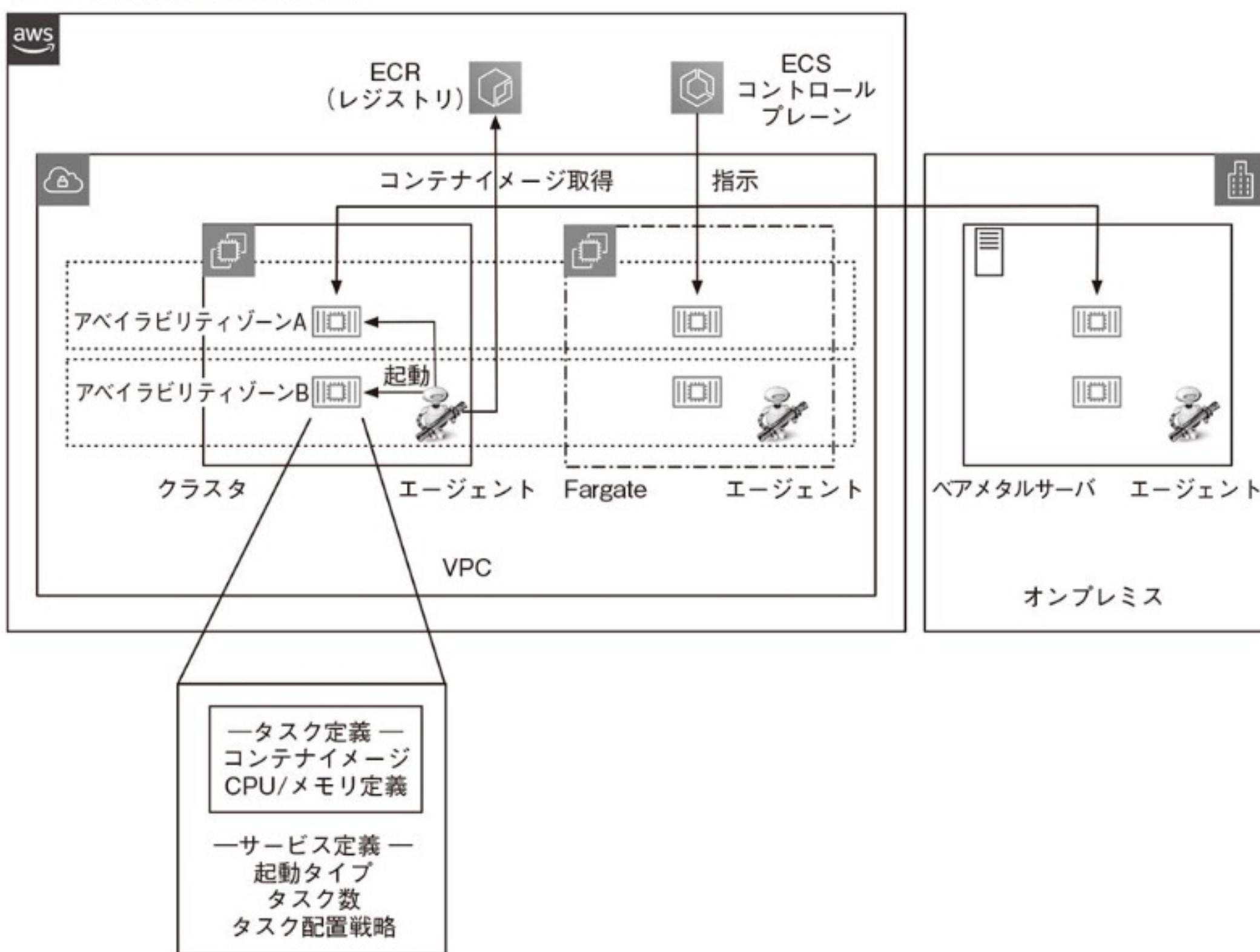
ECSでは、以下が主要な構成要素として定義されています。

【ECSの主な構成要素】

構成要素	説明
クラスタ	Dockerコンテナを実行するためのデータプレーンの実体となる1つ以上のEC2インスタンス群。Fargateではクラスタ自体がAWSのマネージドサービスとして管理される。
タスク	Dockerコンテナイメージ設定と、イメージから生成されるコンテナにおけるクラスタ上の実行設定をセットにした概念。
サービス	タスク定義された内容にもとづき、実際に実行されるコンテナを指示する概念。

以下の図は、ECSコントロールプレーンがDockerコンテナを実行するイメージを表したもので
す。ECSコントロールプレーンはマネジメントコンソールやCLIからAPI経由で要求を受け付け、コンテナのライフサイクル全体を管理します。データプレーンとなるクラスタやオンプレミスサーバには ECSのエージェントがインストールされており、コンテナイメージの取得やコンテナの起動・破棄は、コントロールプレーンからの指示を受けたエージェントが実行します。

【ECSの構成要素と概念】



ECSを実際に使用するには、ローカルの端末などで実行するコンテナのイメージを作成し、ECRやDocker Hubといったレジストリに登録しておきます。そして、マネジメントコンソールやCLIなどを使ってクラスタ起動やタスク定義を行い、サービス条件を定義して実行します。



Amazon ECS マネジメントコンソールは、2023年から新しいUIがデフォルトとして使用されるようになっています。2023年12月時点では、新コンソールでは使用できない設定が含まれています（切り替えることは可能）。今後のアップデートにより改善されていくことがアナウンスされていますが、制約については「新しいAmazon ECSコンソール」^{※31}を参考にしてください。

また、新しいコンソールでは、クラスタを作成する際、インフラストラクチャの設定として、キャパシティプロバイダ^{※32}や外部インスタンスをチェックする項目が追加されています。Fargateではなく、EC2起動型やECS Anywhereを使用した外部インスタンスを利用する場合は、それぞれチェックを入れてクラスタを作成する必要があります。

【ECSクラスタ作成時のインフラストラクチャ設定画面】

▼ インフラストラクチャ 情報

クラスターは、2つのキャパシティープロバイダーを使用する AWS Fargate (サーバーレス) 用に自動的に設定されます。Amazon EC2 インスタンスを追加するか、ECS Anywhere を使用して外部インスタンスを追加します。

AWS Fargate (サーバーレス)
利用ごとのお支払い。ワークロードが小さい、バッチまたはバーストしている場合、またはメンテナンスのオーバーヘッドがない場合に使用します。クラスターには、デフォルトで Fargate および Fargate スポットキャパシティープロバイダーがあります。

Amazon EC2 インスタンス
手動設定。リソースの需要が一定である大規模なワークロードに使用します。

ECS Anywhere を使用した外部インスタンス
手動設定。データセンターコンピューティングの追加に使用します。

※31 https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/new-console.html

※32 https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/cluster-capacity-providers.html

3 ECSタスクとサービス

ECSを使ってコンテナを実行する上で必要な設定について解説します。

●ECS タスク定義

ECSタスクはコンテナイイメージの定義に相当する概念です。コンテナイイメージ自体とAWS環境で実行するために必要な定義を行います。ECSタスクで設定する重要な設定内容は以下のとおりです。

【ECSタスクの定義項目】

カテゴリ	設定要素	説明
タスクの定義設定	タスク定義ファミリー	タスク定義を登録するときに、ファミリー（複数バージョンのタスク定義の名前のようなもの）を指定する。
インフラストラクチャの要件	起動タイプ	「FARGATE」、「EC2」、「EXTERNAL」のいずれかを選択する。
	タスクロール	アプリケーションの処理内容に応じて必要なIAMロールを設定する。
	タスク実行ロール	タスク実行用エージェントのIAMロールを指定。ロールには最低AmazonECSTaskExecutionRolePolicyを付与しておく。
ネットワークモード		コンテナとクラスタ間のネットワーク変換の方式を指定する。none、bridge（EC2起動型デフォルト）、host、awsvpc（Fargate型デフォルト）の4つの方式が選択できる。awsvpc型ではセキュリティグループをタスクごとに設定でき、VPCフローログのモニタリングなどセキュリティ面で強化される ^{※33} 。
ランタイムプラットフォーム		タスクを実行するオペレーティングシステムとCPUアーキテクチャを定義する。Linux/X86_64およびWindows Serverが利用可能。
タスクサイズ	タスクメモリ	コンテナに割り当てるメモリを指定する。
	タスクCPU	コンテナに割り当てるCPUユニット数を指定する。
コンテナの定義	コンテナ名	コンテナの定義名を最大255文字で設定する。
	イメージ	コンテナイイメージの識別子を<repository-url>/<image-name>:<tag>形式で指定する。
	プライベートレジストリの認証	Secrets Managerの認証情報を利用したプライベートレジストリの認証設定を行う。オンにした場合、Secrets Manager ARNを併せて指定する。

カテゴリ	設定要素	説明
コンテナの定義	メモリ制限	コンテナが使用するメモリのハード/ソフト使用制限を指定する。ハード制限を指定した場合、コンテナはその制限を超えると強制終了される。ソフトリミットを指定した場合は、ECSによってコンテナ用にメモリの容量が予約される。
	ポートマッピング	ホストOSのポートとコンテナに割り当てるポートのマッピング設定を行う。例えば、Webサーバだと任意のホストOSのポート番号とコンテナが使用する80番ポートを設定する。動的にホストOSのポートを割り当てる場合、ホストポートに0を設定する。
ストレージ	ボリューム	コンテナにアタッチするストレージのタイプとマウントするディレクトリを指定する。Dockerボリュームオプションで複数のコンテナでボリュームを共有するには、単一のタスク定義で複数のコンテナ定義を行い、共有ボリュームをマウントする必要がある。なお、DockerボリュームオプションはEC2起動タイプのみでサポートされる。



ホストOSのポート番号を動的にコンテナに割り当てる場合の設定方法を押さえておくようにしましょう。



複数のコンテナでストレージを共有する場合の設定方法を押さえておくようにしましょう。



コンテナの定義では、その他オプションとして環境変数やログ記録、ヘルスチェックやタイムアウト、リソース制限、モニタリングなどカスタマイズ可能です。詳細は「詳細コンテナ定義パラメータ」^{※34}を参考にしてください。



さまざまな環境で同一のコンテナイイメージを使用するには、環境変数の設定でSystems Manager Parameter Storeから値を取得できるようにし、コンテナ実行時に値を挿入できるよう設定しておくのがポイントです。コンテナの実行時に環境変数のパラメータを入れ替えることで、アプリケーションの挙動も切り替えることができます。ステージングや商用の場合などでも、パラメータだけ入れ替えればよく、タスク定義やアプリケーションをビルドし直す必要はありません。

※33 https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/bestpracticesguide/networking-networkmode.html

※34 https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/task_definition_parameters.html#advanced_container_definition_params

●ECS サービス実行

ECSサービスでは、実際にデータプレーン上でコンテナを実行するための詳細な条件設定を行います。ECSサービス定義で設定する重要な設定は以下のとおりです。

【ECSサービスの定義項目】

カテゴリ	設定要素	説明
環境	既存のクラスタ	サービスを実行するためのクラスタを選択する。
	コンピューティングオプション	「FAGATE」を選択する場合は、キャパシティープロバイダー戦略を、「EC2」、「EXTERNAL」のいずれか選択する場合は、起動タイプを選択します。なお、キャパシティープロバイダー戦略を選ぶ場合、次の「起動タイプ」を選択する必要はありません。
	起動タイプ	「FAGATE」、「EC2」、「EXTERNAL」のいずれかを選択する。
デプロイ設定	アプリケーションタイプ	サービスとして実行するアプリケーションのタイプを指定します。Webアプリケーションなどは「サービス」を、バッチジョブは「タスク」を設定する。なお、ここでのタスクはスタンダードアロンで実行されているタスクを示しており、コンテナ定義する用語としてのタスクとはまた別と解釈してよい。
	タスク定義	タスクのリビジョンやタスク定義ファミリーを設定する。
	サービス名	サービスに割り当てる一意の名前を設定する。
	サービスタイプ	REPLICASかDEMONを設定する。REPLICASは、クラスタ全体で「タスクの数」として指定した数のコンテナを実行するオプション。DAEMONは、ECSクラスタ1台につき、1つのサービスを実行し、クラスタインスタンスの増減に合わせて実行コンテナのサービスを増減させるオプション。
	タスクの数	クラスタで実行するコンテナ数を設定する。
	デプロイオプション	コンテナをアップデートする場合の戦略を選択する。ローリングアップデートか、AWS CodeDeployを使用したBlue/Greenデプロイメントが選択可能。
	最小実行タスク	コンテナの最小実行数の割合を設定する。
	最大実行タスク	コンテナの最大実行数の割合を設定する。
	デプロイ不具合の検出	デプロイの不具合を検出する方法と、不具合が発生した場合に取るべきアクションを指定する。

カテゴリ	設定要素	説明
ネットワーク構成	クラスターVPC	コンテナを配置するクラスターのVPCを選択する。
	サブネット	コンテナを配置するサブネットを選択する。
	セキュリティグループ	ECSクラスタに割り当てるセキュリティグループを設定する。
	パブリックIP	タスクのElastic Network InterfaceにパブリックIPを割り当てるかを設定する。
タスクの配置	配置テンプレート	コンテナを配置する戦略を選択する。詳細は次項「タスク配置戦略とタスク配置制約」を参照。
ロードバランシング	ロードバランサーの種類	ECSコンテナに処理を分散させるロードバランサーやその詳細な条件を指定する。
AutoScaling	Service Auto Scaling	AutoScaling機能を有効化する場合、オプションを設定する。

2

4 タスク配置戦略とタスク配置制約

ECSでコンテナを実行する場合、ワークロードによってはコンテナ実行環境のカスタマイズが必要になるケースがあります。ここではコンテナの実行環境の設定戦略について解説します。

●タスク配置戦略

ECSで起動タイプ「EC2」を選択する場合、指定されたタスク数にもとづいて、クラスタ内でのコンテナの配置をある程度コントロールすることができます。これをタスク配置戦略と呼び、大きく分けると以下の3つのタイプがあります。

1. binpack

ECSクラスタのCPUやメモリ状況にもとづき、可能な限り最小のインスタンスでタスクを配置します。クラスタに使われるインスタンスが最小になり、コスト削減に効果的ですが、可用性が失われる可能性があります。

2. random

ECSクラスタに対し、タスクをランダムに配置します。

3. spread

指定した値にもとづいて、タスクを均等に配置します。

サービス定義で設定するタスク配置は、上記の戦略を複数組み合わせて設定することができます。設定はJSONテンプレートの形式で記述します。デフォルトで

は、以下の「アベイラビリティゾーンバランススプレッド」が使用されます。この戦略では、アベイラビリティゾーン全体でタスクを均等に分散し、次に各アベイラビリティゾーン内でインスタンスを均等に分散します。

```
"placementStrategy": [
  {
    "field": "attribute:ecs.availability-zone",
    "type": "spread"
  },
  {
    "field": "instanceId",
    "type": "spread"
  }
]
```

「アベイラビリティゾーンバランスbinpack」では、アベイラビリティゾーン全体でタスクを均等に分散し、次に各アベイラビリティゾーン内でメモリにもとづいてタスクをbinpackします。

```
"placementStrategy": [
  {
    "field": "attribute:ecs.availability-zone",
    "type": "spread"
  },
  {
    "field": "memory",
    "type": "binpack"
  }
]
```

各タイプごとに利用できるフィールドは公式ガイド「Amazon ECSタスク配置の制約事項^{※35}」の組み込み属性を参照してください。



タスク配置戦略のオプションやその内容を押さえておくようにしましょう。

^{※35} https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/task-placement-constraints.html

●タスク配置制約

タスク配置戦略の属性を組み合わせて、タスク配置に関わる制約を「タスク配置制約」としても設定できます。タスク配置制約では、以下の制約条件が利用できます。

1. distinctInstance

各タスクを別々のクラスタのインスタンスに配置します。

2. memberOf

特定の条件を満たすクラスタのインスタンスにタスクを配置します。これは「クラスタクエリ言語^{※36}」という固有の条件式を使って、クラスタコンテナインスタンスをグループ化できるものです。

例えば、特定のアベイラビリティゾーンを指定してタスクを配置する場合は、クラスタクエリ言語を使って以下のように表現します。

```
attribute:ecs.availability-zone in [ap-northeast-1a, ap-northeast-1c]
```

これらの条件式は、前述のサービス定義でオプションとして指定できます。可用性が落ちることを許容する代わりに、なるべく近いロケーション配置で高速にデータを共有しながら、コンテナ間で並列バッチ処理を行いたい場合などに活用するとよいでしょう。

^{※36} https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/cluster-query-language.html

2-8 Amazon RDS

Amazon RDS (Relational Data Service) は、構築や運用を容易にするリレーショナルデータベースサービスです。

1 RDSの概要

Amazon RDSは、時間単位の従量課金制で利用するフルマネージドなリレーショナルデータベースです。冗長化構成によるフェイルオーバーや自動バックアップ機能など、高い可用性を保つつも、簡易なセットアップで運用が容易な点を特徴とします。RDSでは2023年12月現在、以下のデータベースエンジンを選択可能です。

- Amazon Aurora
- PostgreSQL
- MySQL
- MariaDB
- Oracle
- Microsoft SQL Server

いずれのデータベースエンジンを選択しても、RDSではマネージドサービスとしてサポートされます。マネージドサービスとしてサポートされる内容は次のとおりです。

1. 拡張性

- a. スケールアップ・スケールダウンが容易に実行できます。EC2同様、RDSを実行する際は、インスタンスタイプを指定して起動します。実際に稼働した際のパフォーマンスに応じて、適宜スケールアップ・スケールダウンすることができます。
- b. 実際にデータを保存するストレージは、データベースを停止することなく増やすことができます。サイズを自動的にスケーリングするオプションも用意されています^{※37}。

2. 高可用性

RDSはアベイラビリティゾーンをまたいだ冗長化構成による自動フェイルオーバーをサポートしています。データベース構築時に「Multi-AZ配置」というオプションでスタンバイデータベースを使った冗長化構成を選択できます。このオプションではプライマリとなるデータベースノードと、別のアベイラビリティゾーンにスタンバイとなるデータベースノードが配置されます（次項の図「RDSのMulti-AZ配置とリードレプリカ」を参照）。スタンバイデータベースノードはプライマリデータベースノードからデータを複製され、常に同期的に更新されていますが、プライマリに障害が発生すると自動的にフェイルオーバーし、スタンバイノードが自動的にプライマリに昇格します。なお、プライマリ・スタンバイの双方とも同一のDBエンドポイントでアクセスされるため、フェイルオーバーが発生しても、参照側が切り替える必要はありません。



Amazon RDS Multi-AZ DB Cluster の発表

2022年の3月にRDS Multi-AZ DB Clusterという新機能が発表されました^{※38}。上述した従来のMulti-AZ配置が、アクセス不可能なスタンバイデータベースとして、別のアベイラビリティゾーンで1つだけ待機していたのに対し、この機能では同時に2つの読み取り可能なスタンバイデータベースをそれぞれ異なるアベイラビリティゾーンで構成し、より可用性やパフォーマンスを高めることが可能になっています。2023年12月時点では、MySQLおよびPostgreSQL互換のRDSで利用可能です。各データベースエンジンごとで可能なMulti-AZ構成の違いの最新状況は「Amazon RDS マルチ AZ」に記載の比較表^{※39}も参考にしてください。

3. DB バックアップ

RDSには自動バックアップ機能があります。バックアップ対象として次の2つのデータをS3へ保存します。障害発生時はバックアップをもとにリストア可能です。

- 1日一度、バックアップウィンドウで指定した時間でのDBスナップショット
- 5分間隔のトランザクションログ

4. DB パッチ適用

メンテナンスウィンドウで指定した時間帯に、データベースエンジンのソフトウェアアップデートが自動で行われます。パッチには必須のものと、任意で適用するか選択できるものがあります。

※37 https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/UserGuide/USER_PIOPS.StorageTypes.html#USER_PIOPS.Autoscaling

※38 <https://aws.amazon.com/jp/blogs/news/amazon-rds-multi-az-db-cluster/>
※39 <https://aws.amazon.com/jp/rds/features/multi-az/>

5.OS パッチ適用

DBパッチ適用と同様、メンテナンスウィンドウで指定した時間帯に、OSのソフトウェアアップデートが自動で行われます。RDSの起動構成にもよりますが、メンテナンス時はデータベースが停止されます。ダウンタイムを最小化する方法は公式ドキュメント^{※40}を参照してください。



re:Invent 2022で、MySQLおよびMariaDBをデータベースエンジンとするRDSおよびAuroraにおいて、Amazon RDS Blue/Green Deploymentという機能が発表されています。ダウンタイムを伴うメンテナンスについて、こちらの機能で柔軟な切り替えを行うことが可能です。

6.OS インストール、サーバメンテナンス、ハードウェア資産管理、電源/ネットワーク/空調

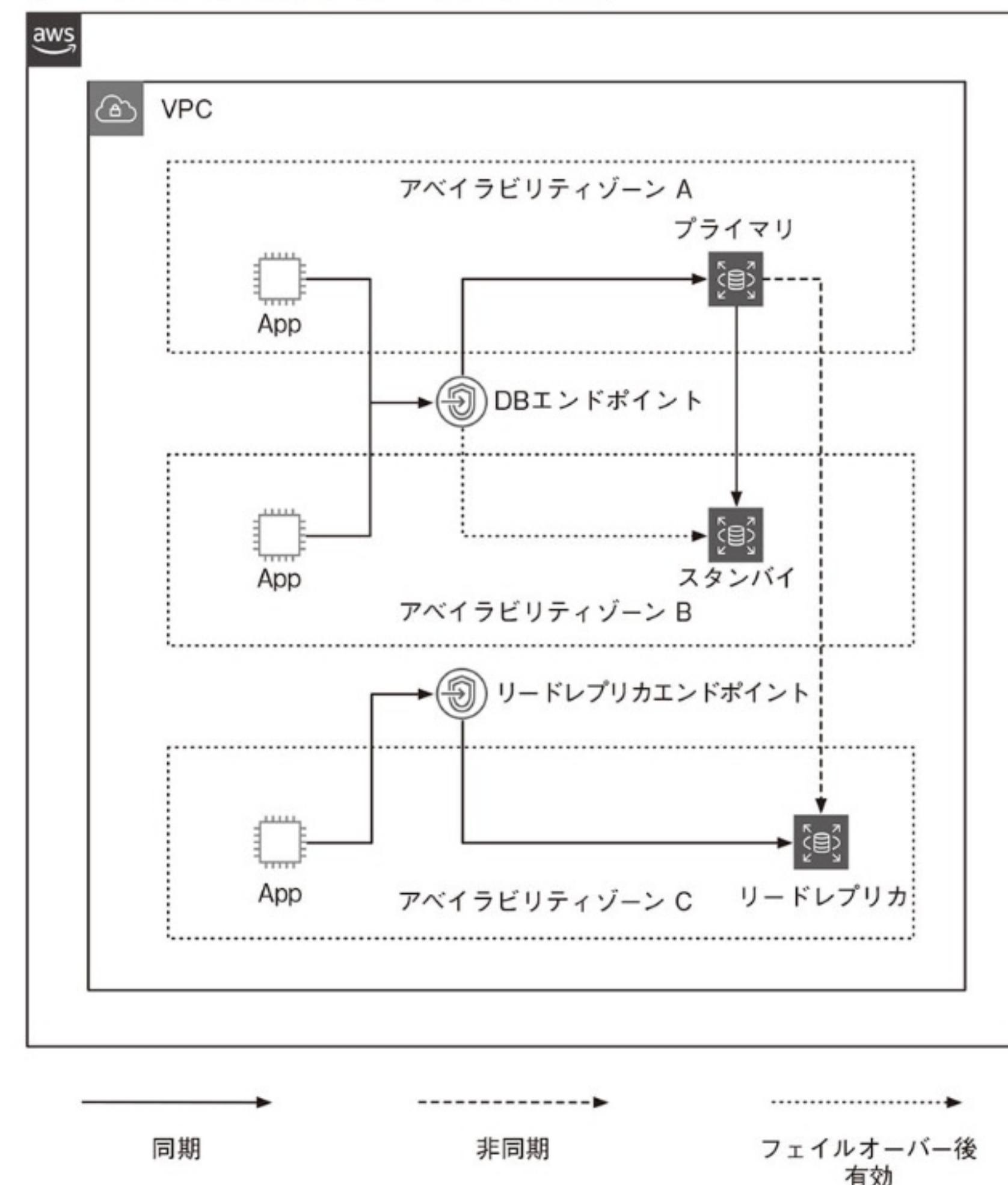
EC2におけるマネージドサービスのサポート範囲と同様です。DBインスタンスは要求を受けて起動し、オプションに応じて指定したVPC・サブネット内の各アベイラビリティゾーンに配置されます。ハードウェアのメンテナンスは意識する必要はありません。

2 リードレプリカ

RDSには読み取りリクエストをオフロードするためにリードレプリカを作成するオプションがあります。リードレプリカでは、プライマリとなるデータベースを非同期で複製します。読み込み専用のデータベースを作成することで、読み込みアクセスを分散し、全体のパフォーマンス向上に寄与します。

次の図は、リードレプリカ構成時におけるアーキテクチャの全体像を示しています。

【RDSのMulti-AZ配置とリードレプリカ】



※40 <https://aws.amazon.com/jp/premiumsupport/knowledge-center/rds-required-maintenance/>

リードレプリカの特徴は以下のとおりです。

- 読み込みアクセスが多い場合、最大5台までリードレプリカを構築して、リクエストをオフロードすることができます。ただし、リードレプリカはプライマリのデータが非同期に反映されるため、常に最新のデータが返されるとは限りません。加えてリードレプリカは独立したDBインスタンスとして機能し、プライマリのDBエンドポイントとは異なります。書き込み処理はプライマリDBエンドポイントを指定しておく必要があります。
- オプションとして、任意のタイミングでリードレプリカをスタンダードアロンDBインスタンスへ昇格することができます。データ分割のためのシャーディング用途で使用したり、プライマリDBインスタンスに障害が発生した場合のデータ復旧スキームの手段として採用したりすることも可能です。ただし、昇格した後に、アプリケーションの参照先をリードレプリカのエンドポイントに向ける必要があります。RDSのエンドポイントはリネーム機能に対応しているため、リードレプリカエンドポイントをこれまで使用されていたDBエンドポイントに変更するとよいでしょう。ただし、リネームにはDBの再起動が必要となります。
- 災害対策等の用途で別のリージョンにクロスリージョンレプリカを作成することができます。

なお、リードレプリカの制約として、Microsoft SQL Serverや一部のリージョンでは使用できないという制約があります。リードレプリカの制約やデータベースエンジンごとの違いは公式ガイド^{*41}を併せて参照してください。

3 Amazon Aurora

RDSのデータベースエンジンの1つであるAuroraは、クラウドの普及に伴って、Amazonがその内部アーキテクチャを再設計したデータベースです。

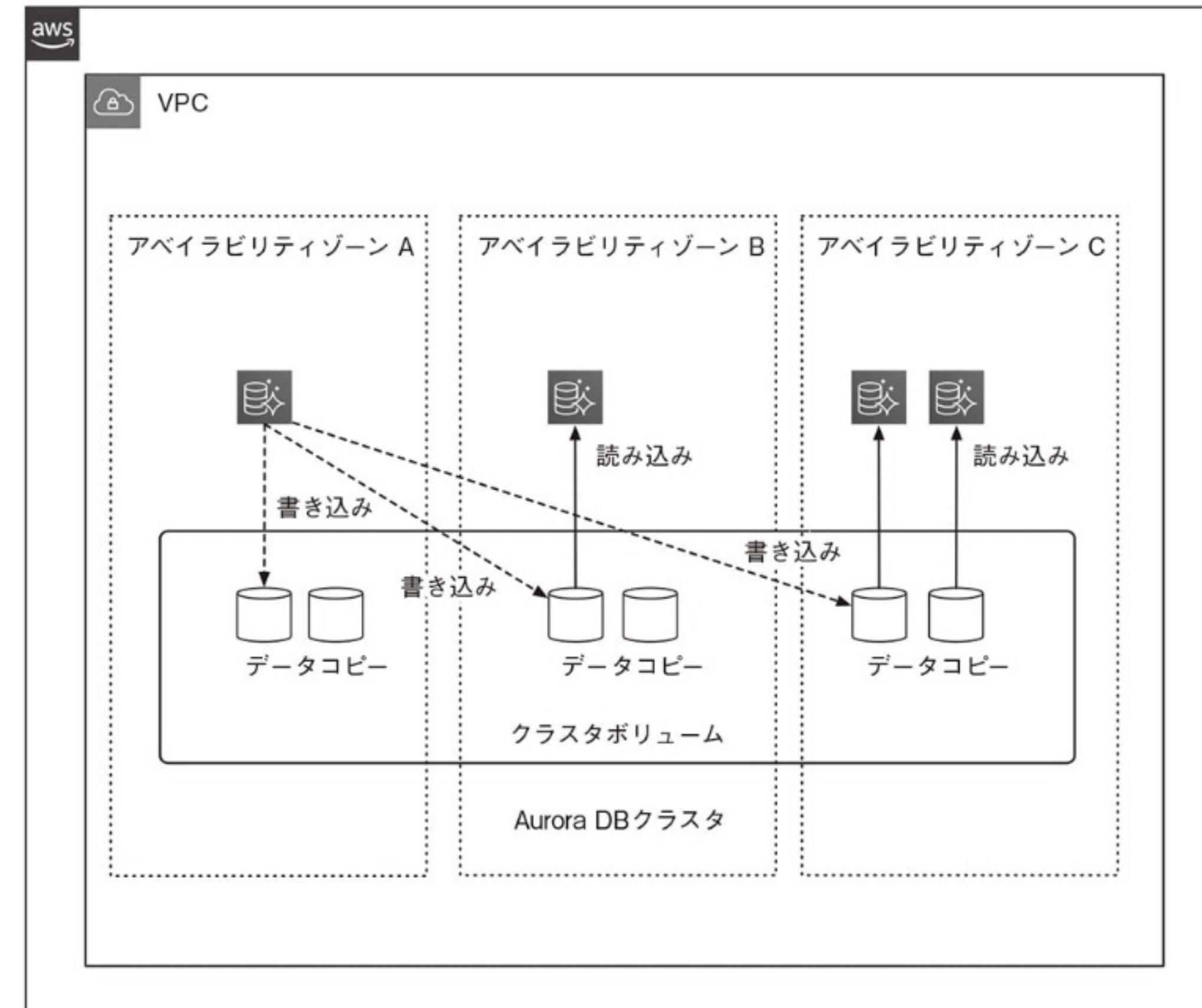
●Auroraの特徴

AuroraはRDBの特性である一貫性を持ち、RDSが持つリードレプリカといったハイパフォーマンスのための特徴を維持しながら、Quorumにもとづく結果整合性でデータを分散して保存するストレージクラスノードクラスタを採用して高可用性を実現しています。そのため、あるアベイラビリティゾーンで障害が発生した場合でも、RDSと違い、マルチアベイラビリティゾーン配置でスタンバイデータベースを作成せずとも、データベースを継続して運用することができます。

*41 https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/UserGuide/USER_ReadRepl.html

次の図は、Auroraのアーキテクチャを示しています。

【Amazon Auroraのアーキテクチャ^{*42}】



Amazon Auroraは以下の特徴を備えています。

●より高速なデータ同期・フェイルオーバー

プライマリインスタンス、リードレプリカとともに、3つのアベイラビリティゾーンでレプリケート（複製）された同じクラスタボリュームのデータコピーを参照するので、レプリカの非同期更新時間が小さくなっています。プライマリインスタンスに障害が発生した場合は、より高速にリードレプリカへフェイルオーバーすることができます。

●自動ストレージ拡張

データは10GBずつ「protection groups」と呼ばれる論理的なグループに保存され、64TBまで自動的にスケールアップします。

*42 https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/AuroraUserGuide/Aurora.Overview.html

● さまざまなエンドポイント

RDSと同様、プライマリインスタンスを指し示す「クラスタエンドポイント」とリードレプリカを指し示す「読み取りエンドポイント」があります。また、ユーザーがワークロードに応じて任意にレプリカを設定できる「カスタムエンドポイント」の定義もできます。

● リードレプリカのオートスケーリング

Auroraのリードレプリカは、メトリクスに応じてAutoScalingにより自動増減する機能もあります。これにより、リードレプリカへの読み取りクエリの分散や、リクエストの増減分に応じたコスト最適化が可能です。

● DBバックアップ機能の拡充

RDSと同様、自動バックアップが常に有効になります。バックアップ中でもパフォーマンスに影響を与えることなく、各セグメントごとにAmazon S3へ継続的にスナップショットが保存されます。これは増分バックアップで行われ、またバックアップウィンドウで処理時間を指定することはできません。データのある時点に戻すBackTrack機能はサポートされます。

● クロスリージョンデータベースの性能向上・低価格化

RDSと同様、クロスリージョンでのデータベースレプリケーションがオプションとしてサポートされます。クロスリージョンでデータベースを展開する場合は、1つのプライマリAWSリージョン(データを管理)と最大5つのセカンダリAWSリージョン(読み取り専用)で構成されます。展開先のリージョンでは、DBインスタンスがなくストレージクラスタボリュームのみになるため、より高速で低価格でのレプリケーションが可能です。

● パラレルクエリ

ストレージノードに搭載されたCPUを活用し、多数のストレージノードで並列にクエリを実行し、高速にスキャンすることができます。

データベースエンジンは2023年現在は、MySQLとPostgreSQL互換のバージョンが提供されています。これまで説明してきたとおり、その内部アーキテクチャは従来のオープンソースのものと大きく異なりますので、あくまで互換です。サポートされているバージョンや機能制約はそれぞれ公式ガイド「Amazon Aurora MySQLの概要^{※43}」および、「Amazon Aurora PostgreSQLの更新^{※44}」を参照してください。

※43 https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/AuroraUserGuide/AuroraAuroraMySQLOverview.html

※44 https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/AuroraUserGuide/AuroraPostgreSQLUpdates.html

● Aurora Serverless

Aurora Serverlessは、リードレプリカのオートスケーリングと似た機能で、リクエストの負荷に応じて、インスタンスタイプの変更や起動停止を自動実行することができます。Aurora Serverlessはデータベースサーバをプロビジョニングして管理するのではなく、キャパシティユニットと呼ばれる使用量を指定します。キャパシティはプールされており、指定値を満たすようにルーターフリートがクライアント接続を追加します。不定期、断続的、または予測不能なワークロードに対して、比較的シンプルでコスト効率のよい方法として使用できます。Aurora Serverlessの機能は以下のとおりです。

● 自動起動停止

オンデマンドで起動し、使用していないときはシャットダウンする

● インスタンスタイプ変更

リクエスト負荷に応じて、自動でスケールアップ・スケールダウンする

● Data API

Aurora DBクラスにVPC外からアクセスするためのエンドポイントを作成する機能

なお、通常のAuroraと比べて、Aurora Serverlessは実行可能なバージョンなどに制約があります。これらの制約は、re:Invent 2020で発表されたAurora Serverless v2で一部解消されていますが、利用する前に確認しておくことをお勧めします。詳細は公式のユーザガイド「Aurora Serverless v1の制約事項^{※45}」を参考してください。

2-9 Amazon S3

「Amazon S3 (Amazon Simple Storage Service)」は、容量無制限で高い耐久性・性能を備えるスケーラブルなオブジェクトストレージサービスです。

1 S3の概要

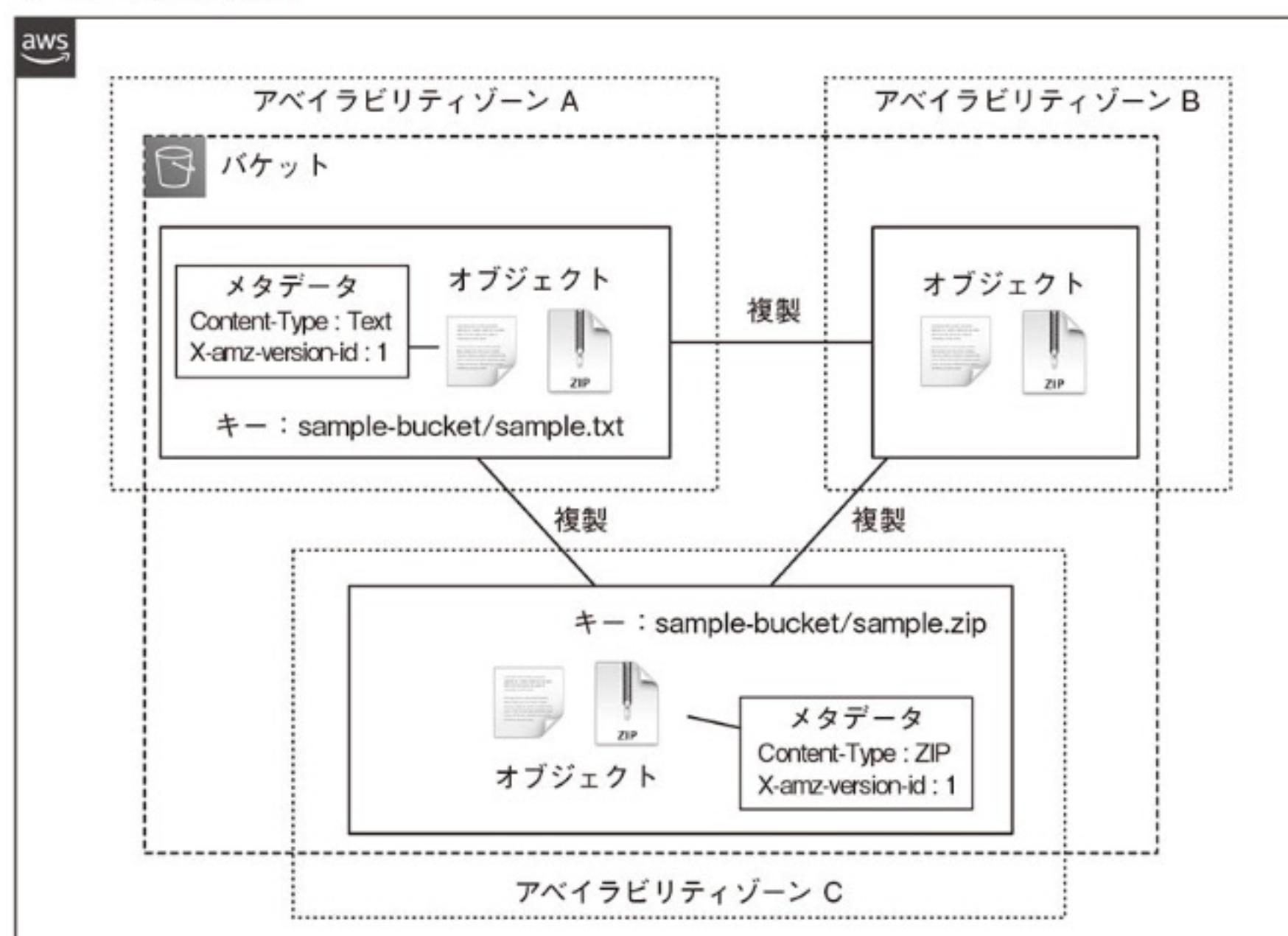
●オブジェクトストレージ

ストレージには、サーバにマウントしてブロック単位でデータを読み書きするブロックストレージやディレクトリ・ファイルを階層構造として扱うファイルストレージなどが代表的な分類としてありますが、S3はオブジェクトストレージと呼ばれるカテゴリに位置付けられます。オブジェクトストレージはデータをオブジェクトと呼ばれる単位で管理します。オブジェクトには固有のURLやメタデータが付与され、HTTP/HTTPS通信でファイルを読み書きできます。

●S3 の構成要素

オブジェクトストレージであるS3を利用するにあたり、以下の基本的な構成要素・用語を押さえておく必要があります。

【S3の構成要素】



●オブジェクト

PCなどで扱われるファイルと同じ単位で保存する実体データを「オブジェクト」といいます。オブジェクトには固有のキーとメタデータが付与され、1オブジェクトあたり最大5TBまでのデータサイズが許容されます。

●バケット

オブジェクトを格納するルートとなる保存場所です。バケットのデータは異なるアベイラビリティゾーンの3箇所に複製して保存することで高可用性を実現しています。その反面、データを上書き更新・削除する際は、わずかな時間ですが、各データ間で差分が発生します（結果整合性モデル）。すべてのリージョンの中でユニークな文字列にする必要があり、デフォルトでは各アカウントあたり100バケットまで作成可能です。

●キー

オブジェクトを指示するパスです。バケット内でユニークな文字列にする必要があります。キー自体にスラッシュなどの区切り記号を使用することで論理的な階層構造を構成できます。「バケット + キー + バージョン」が必ずユニークな値となります。なお、キーとして使用可能な文字には特殊文字などの制限があります。詳細は公式ガイド^{※46}を参考にしてください。

●メタデータ

オブジェクトに付与される、名前と値のペアセットとなる属性情報です。AWSが用意している固有のファイルシステム定義メタデータとユーザが任意に設定できるものの2種類あります。

●バージョン

作成・更新時にS3によって、オブジェクトに付与するIDです。

●S3 の使用方法

S3を利用するには、以下のような方法があります。

●マネジメントコンソール

マネジメントコンソールのサービスメニューからS3を選択し、バケットの作成やオブジェクトのアップロード・更新・削除などを行います。ローカル端末のファイルをアップロードしたり、バケット内のオブジェクトの状況確認をするなど、GUIベースで簡単に使用できます。

●AWS CLI

コマンドラインインターフェースを使って、バケットやオブジェクトを操作し

※46 https://docs.aws.amazon.com/ja_jp/AmazonS3/latest/userguide/object-keys.html

ます。シェルスクリプトによる環境作成など、開発時や運用の際に、何度もS3を利用する場合などによく使われる方法です。

● AWS SDK

JavaやPython、.NET (C#)、PHP、Rubyなどさまざまなプログラミング言語から提供されているプログラム向けのライブラリです。主にアプリケーションなどでS3バケットやオブジェクトにアクセスしたい場合に利用されます。

● REST API / SOAP API

オブジェクトストレージであるS3の特徴で、REST API / SOAP APIをHTTPS通信で呼び出してダイレクトアクセスが可能です。後述する静的Webサイトなどのホスティングや、ブラウザからのリダイレクトによるアクセスなどを実現します。なお、上述のAWS CLIやSDKでは、内部処理でこのREST APIを呼び出しています。

上述のいずれかの手段を用いて、特定のリージョンにバケットを作成し、メタデータやアクセス権限などを設定してファイルをアップロードするなどして利用します。

ストレージクラス	概要	コスト順位 (高コスト順)
S3 1ゾーン - 低頻度アクセス (infrequent access : IA)	単一のアベイラビリティゾーンに保存されるため可用性が低く、取り出しに利用料金が発生するが、容量あたりの利用料金が最も低いストレージ。S3の可用性低下を受容できるシステムで、アクセス頻度の低いデータの格納に適している。	4
S3 Glacier Instant Retrieval	データのアーカイブに適したストレージ。取り出しに利用料金が発生するが、長期保存を目的としたGlacierストレージの中で、即時に取り出すことができるオプション。四半期に1度程度のアクセスだとS3 Standard IAに比べて安価。	5
S3 Glacier Flexible Retrieval	データのアーカイブに適したストレージ。取り出しに利用料金が発生することに加え、取り出すために時間がかかる。年に1~2度程度のアクセスだと最もコスト最適化されるオプション。	6
S3 Glacier Deep Archive	最も低成本のストレージで、年に1~2度程度しかアクセスされないようなデータの保存に適している。7~10年間保存されるようなデータの長期保存を想定した設計が行われている。取り出しには最大12時間をする。	7

2 ストレージクラス

S3ではアクセス頻度や用途に応じて、以下のようなストレージクラスを定義しています。オブジェクトには保存時にこのストレージクラスを関連付けて保存しますが、ストレージクラスに応じて、オブジェクトの保存にかかるコストは変わってきます。

【ストレージクラスとコスト】

ストレージクラス	概要	コスト順位 (高コスト順)
S3 標準	高耐久、高可用性な一般的なストレージ。デフォルトではこのストレージクラスが適用される。頻繁にアクセスされるデータに適している。	1
S3 Intelligent-Tiering	アクセスの頻度に応じて自動的にストレージタイプを変更するストレージ。最初はアクセス頻度が高いが、のちにほとんどアクセスされなくなるなど、アクセスパターンが変化する場合に適している。	2
S3 標準 - 低頻度アクセス (infrequent access : IA)	取り出しに料金が発生するストレージ。アクセス頻度の低いデータの格納に適している。	3



S3には低冗長化ストレージ (Reduced Redundancy Storage : RRS) もオプションとしては用意されていますが、低い冗長性でかつコストもS3標準よりかかるため、ここでは記載していません。



re:Invent 2021の発表で、アーカイブストレージとしてこれまで利用されてきたS3 Glacierの中で、データの即時取り出しを可能にするオプションであるInstant Retrievalが発表されました。なお、従来のS3 GlacierはFlexible Retrievalとしてリネームされています。

ストレージクラスによって耐久性や可用性、保存されるアベイラビリティゾーンの数などの仕様が異なります。また、少しでも利用すると、最低限その期間分の費用が発生する最小ストレージ期間という条件が設定されているストレージクラスもあります。次の表では、各ストレージクラスの可用性や耐久性、最小ストレージ期間、保存アベイラビリティゾーン数を比較しています。

【ストレージクラスの仕様を比較】

ストレージ クラス	耐久性	可用性	アベイラビリ ティゾーン数	最小スト レージ期間
S3 標準	99.999999999% (イレブンナイン)	99.99%	3箇所以上	なし
S3 Intelligent-Tiering	99.999999999%	99.9%	3箇所以上	30日間
S3 標準 - 低頻度アクセス	99.999999999%	99.9%	3箇所以上	30日間
S3 1ゾーン - 低頻度アクセス	99.999999999%	99.5%	1箇所	30日間
S3 Glacier Instant Retrieval	99.999999999%	99.99%	3箇所以上	90日間
S3 Glacier Flexible Retrieval	99.999999999%	99.99%	3箇所以上	90日間
S3 Glacier Deep Archive	99.999999999%	99.99%	3箇所以上	180日間
RRS	99.99%	99.99%	3箇所以上	なし

3 セキュリティ保護

S3はインターネットから直接接続されるオブジェクトストレージのため、セキュリティ対策を徹底する必要があります。本節では、S3が提供するセキュリティ保護機能を解説します。

●アクセスコントロール

S3に保存したオブジェクトはさまざまなユーザからアクセスされることを想定して、きめ細やかなアクセス権を設定できます。ここで紹介する「ユーザポリシー」「バケットポリシー」「ACL (Access Control List)」を任意に設定することで、オブジェクトへのアクセスコントロールを制御します。

●ユーザポリシー

AWSアカウントで作成されるIAMユーザに対して、アクセス権限を設定する場合に利用されます。実体はIAMポリシーのアクセス権限設定と同一です。以下はユーザポリシーの一例です。Effectに「許可」もしくは「拒否」を、Principalに制御対象とするユーザを、Actionに許可もしくは拒否対象となるオペレーションを、Resourceにバケットおよびアクセス対象のオブジェクトのキーを指定します。

```
{
  "Version": "2012-10-17",
  "Id": "ExamplePolicy01",
  "Statement": [
    {
      "Sid": "ExampleStatement01",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::awsexamplebucket1/*"
    }
  ]
}
```

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::123456789012:user/Dave"
  },
  "Action": [
    "s3:GetObject",
    "s3:GetBucketLocation",
    "s3>ListBucket"
  ],
  "Resource": [
    "arn:aws:s3:::awsexamplebucket1/*",
    "arn:aws:s3:::awsexamplebucket1"
  ]
}
```

●バケットポリシー

バケット単位に設定するアクセス権限です。クロスアカウントでのアクセス権限を設定したい場合などに利用されます。バケットの設定で付与されるものですが、実体はユーザポリシーと同じく、IAMポリシーのアクセス権限設定と同様です。IAMポリシーのCondition要素を使って、IPアドレスやHTTPヘッダのReferrer、MFAを使ったアクセス制御も可能です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AddCannedAcl",
      "Effect": "Allow",
      "Principal": {
        "AWS": ["arn:aws:iam::111122223333:root", "arn:aws:iam::444455556666:root"]
      },
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
    }
  ]
}
```

```

    "Condition": {
        "StringEquals": {"s3:x-amz-acl": ["public-read"]}
    }
}
]
}
}

```



ユーザポリシーとバケットポリシーの違い

上記の説明で、ユーザポリシーとバケットポリシーの違いが理解できない読者も多いかと思います。基本的にどちらもIAMポリシーをベースとしたアクセス制御であり、前者はIAMユーザに対して付与し、後者はバケットに設定するものです。上記の例では単一のユーザに対し、複数のバケットへのアクセスを許可しているか、単一のバケットに対し、複数のユーザのアクセスを許可しているかの違いでしかありません（同じIAMポリシーという意味で違いはありません）。

アクセスするユーザとバケットの所有者がいずれも同一のアカウントであれば、いずれかが付与されればアクセス可能です。反対に、アクセスユーザとバケットの所有者が異なるクロスアカウントの場合、アクセスするユーザが所属するAWSアカウントにユーザポリシーを、バケットの所有者のアカウントにバケットポリシーをそれぞれ設定する必要があります。そのため、実体は同じですが、2種類のポリシーに分類し、それぞれ設定方法を区別しています。

なお、バケットポリシーでは、以下のサンプルのようにCondition要素に「aws:SourceVpce」条件を指定することで、特定のVPCエンドポイントからのアクセスを制御したり、「aws:SecureTransport」条件でHTTPSを経由したアクセスを強制したりすることができます^{※47}。こうした設定は、ユーザを指定することなくアクセス条件を規定するバケットポリシーの特徴が際立っています。

```

{
    "Version": "2012-10-17",
    "Id": "Policy1415115909152",
    "Statement": [
        {
            "Sid": "Access-to-specific-VPCE-only",
            "Principal": "*",
            "Action": "s3:*",
            "Effect": "Deny",
            "Resource": [ "arn:aws:s3:::awsexamplebucket1",

```

```

                "arn:aws:s3:::awsexamplebucket1/*" ],
                "Condition": {
                    "StringNotEquals": {
                        "aws:SourceVpce": "vpce-1a2b3c4d"
                    }
                }
            ]
        }
    ]
}

```

●ACL (Access Control List)

バケットおよびオブジェクトに設定するアクセス権限です。AWSアカウントに所属しない不特定多数のユーザへのアクセス権限を設定したい場合などに利用されます。ACLには、「被付与者」という、個別にアクセス権を付与できる4つの対象があります。

1. バケット・オブジェクト所有者
バケットやそこにアップロードされるオブジェクトを所有するユーザです。
2. All Usersグループ
AWSアカウントを所持しているかを問わず、AWSへアクセス可能な不特定多数を含むすべてのユーザです。
3. Authenticated Usersグループ
認証された任意のAWSアカウントのユーザを指します。
4. Log Deliveryグループ
サーバアクセスログを記録するためのグループです。

ACLは、被付与者ごとに、以下のようなアクセス許可を組み合わせて、バケット・オブジェクトへ設定できます。許可されるとバケット・オブジェクトそれぞれで以下のようなAPIの実行が許可されるようになります。

※47 https://docs.aws.amazon.com/ja_jp/AmazonS3/latest/userguide/example-bucket-policies-vpc-endpoint.html

【ACLのアクセス許可】

アクセス許可	バケットに付与された場合に許可されるAPI	オブジェクトに付与された場合に許可されるAPI
READ	被付与者に対し、バケット内のオブジェクトリストを参照できる以下のAPIを実行可能にします。 ・ s3>ListBucket ・ s3>ListBucketVersions ・ s3>ListBucketMultipartUploads	被付与者に対し、オブジェクトのデータ本体とメタデータを読み込める以下のAPIを実行可能にします。 ・ s3:GetObject ・ s3:GetObjectVersion ・ s3:GetObjectTorrent
WRITE	被付与者に対し、バケット内に新しいオブジェクトを作成できる以下のAPIを実行可能にします。 ・ s3:PutObject	該当APIなし
READ_ACP	被付与者に対し、バケットに設定されているACLを読み込める以下のAPIを実行可能にします。 ・ s3:GetBucketAcl	被付与者に対し、オブジェクトに設定されているACLを読み込める以下のAPIを実行可能にします。 ・ s3:GetObjectAcl ・ s3:GetObjectVersionAcl
WRITE_ACP	被付与者に対し、バケットに設定されているACLを書き込める以下のAPIを実行可能にします。 ・ s3:PutBucketAcl	被付与者に対し、オブジェクトに設定されているACLを書き込める以下のAPIを実行可能にします。 ・ s3:PutObjectAcl ・ s3:PutObjectVersionAcl
FULL_CONTROL	被付与者に対し、バケットへREAD、WRITE、READ_ACP、WRITE_ACPすべてのアクセス許可を付与します。	被付与者に対し、オブジェクトへREAD、READ_ACP、WRITE_ACPすべてのアクセス許可を付与します。



AWS公式のドキュメントにも記載されているとおり^{※48}、All Users グループには、WRITE、WRITE_ACP、または FULL_CONTROL アクセス許可を一切付与しないことが強く推奨されています。書き込みを許可することは、不正な攻撃スクリプトが仕掛けられるようなものと考えましょう。

バケットへファイルアップロードする際に、アクセス権限を個別に設定していくことも可能ですが、実際の運用ではAWSで用意されている「事前定義されたACL」を選んで設定することもできます。代表的な「事前定義されたACL」は以下のようなものがあります。

※48 https://docs.aws.amazon.com/ja_jp/AmazonS3/latest/userguide/acl-overview.html

1. プライベート

被付与者「オブジェクトの所有者」のみに対し、FULL_CONTROLを与えるオプションです。デフォルトでは、こちらのオプションでアップロードされます。

2. パブリックアクセス

被付与者「オブジェクトの所有者」へFULL_CONTROLを与えるのに加えて、被付与者「All Usersグループ」に対し、READを与えるオプションです。この設定は、AWSへアクセス可能な不特定多数のユーザにオブジェクトへの読み取りアクセス権限を与えることを意味します。

これらのオプションはアップロード後も変更が可能であり、「パブリックアクセス」により、意図しない機密情報や個人情報の漏洩につながる例が数多く報告されています。AWSとしても、S3へのオブジェクトのアクセス権限は、基本的にはバケットポリシーを利用することを推奨しているので^{※49}、特別な事情がなければ、アクセス制御はバケットポリシーを利用るようにしてください。また、ACLの設定は後述する「ロックパブリックアクセス」設定により、明示的にブロックできます。



バケットポリシーでパブリックアクセスを行う設定

パブリックアクセスは上述のACLで設定できますが、バケットポリシーで設定することも可能です。バケットポリシーで匿名アクセスと呼ばれるアクセス許可を全員に付与するには、Principalの値として、以下のとおり、ワイルドカード ("*") を設定します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SamplePublicRead",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [ "s3:GetObject", "s3:GetObjectVersion" ],
      "Resource": [ "arn:aws:s3:::sample-bucket/*" ]
    }
  ]
}
```

なお、ACL、バケットポリシーのどちらを使うにせよ、S3バケットへの匿名書き込みアクセスは一切付与しないことをAWSでは強く推奨しています^{※50}。

※49 https://docs.aws.amazon.com/ja_jp/AmazonS3/latest/userguide/managing-acls.html

※50 https://docs.aws.amazon.com/ja_jp/AmazonS3/latest/userguide/s3-bucket-user-policy-specifying-principal-intro.html



上述のアクセス制御設定で相反する設定が行われた場合、以下のルールにもとづいた挙動になります。

- ・バケットに設定されたACLよりもオブジェクトに設定されたACLが優先されます。
- ・ACLよりもユーザポリシーが優先されます。
- ・ユーザポリシーとバケットポリシー間では、拒否の設定が優先されます。

● ブロックパブリックアクセス

ブロックパブリックアクセスは、アカウントもしくはバケットレベルで意図しない「パブリックアクセス」を抑止するための機能です。バケットを作成する際にコンソール上で選択できますが、2023年12月時点で表示されている日本語だと非常にわかりづらくなっています。以下の表を参考に、各オプションを有効化した際の挙動をしっかりと理解しておいたほうがよいでしょう。

【ブロックパブリックアクセスのオプション】

オプション名	コンソール上の表示	有効化した場合の挙動
BlockPublicAcls	新しいアクセスコントロールリスト (ACL) を介して付与されたバケットとオブジェクトへのパブリックアクセスをブロックする	パブリックアクセスを許可するACL設定の更新や、パブリックアクセス設定されたオブジェクトのアップロードを許可しない
IgnorePublicAcls	任意のアクセスコントロールリスト (ACL) を介して付与されたバケットとオブジェクトへのパブリックアクセスをブロックする	パブリックアクセスを許可するACLが設定されたバケットやオブジェクトが存在したとしても、パブリックアクセスを無視する
BlockPublicPolicy	新しいパブリックバケットポリシーまたはアクセスポイントポリシーを介して付与されたバケットとオブジェクトへのパブリックアクセスをブロックする	パブリックアクセスを許可するバケットポリシーの設定・更新を許可しない
RestrictPublicBuckets	任意のパブリックバケットポリシーまたはアクセスポイントポリシーを介したバケットとオブジェクトへのパブリックアクセスとクロスアカウントアクセスをブロックする	パブリックアクセスを許可するバケットポリシー設定が存在したとしても、パブリックアクセスおよびクロスアカウントアクセスを無視する

これらの設定は、既存のバケットポリシー或はACLの設定を変更するものではなく、パブリックアクセスができないよう、バケットに蓋をするイメージです。そのため、このブロックパブリックアクセスを無効化すると、元の状態に戻ってしまうことに留意が必要です。



パブリックアクセスを抑止するために、AWS Configを使ってバケットの状態をチェックしたり、設定変更イベントを捕捉して設定をLambda関数などで通知したり設定変更する方法をとることもできます。設定や作り込みが必要ですが、パブリックアクセスの無効化を担保したい場合には有効なオプションです。

● VPC エンドポイント

通常、S3にはインターネットを通じてアクセスします。VPCのプライベートサブネットなどからS3にアクセスしたい場合は、NAT Gatewayを設置して一度インターネットに出てS3へアクセスする、もしくはGateway型のVPCエンドポイントを作成して、直接プライベートサブネットからS3へアクセスするルートを作ります。後者のインターネットにアクセスさせない手順としては、VPCの管理画面からS3エンドポイントを作成し、プライベートサブネットのルートテーブルにアタッチします^{※51}。その際、以下のように、VPCエンドポイントにアクセスポリシーを定義して、アクセス対象のバケットのみを指定することができます。

```
{
  "Effect": "Allow",
  "Action": ["s3>ListBucket", "s3:GetObject", "s3:PutObject"],
  "Resource": ["arn:aws:s3:::example-bucket", "arn:aws:s3:::example-bucket/*"]
}
```

● アクセスポイント

アクセスポイントはバケットにアタッチする名前付きのネットワークエンドポイントです。GetObjectやPutObjectなどのオブジェクトに対するオペレーションに対し、IAMポリシーを設定してアクセスを制御できます。バケットポリシーと同様、特定のVPCからのアクセスを許可するよう設定したり、アクセスポイントごとに、上述したブロックパブリックアクセスをカスタム設定したりすることも可能です。バケットへアクセスするユーザが増えてくると管理が煩雑になるため、オブジェクトに対するアクセスコントロールを簡素化したい場合に利用を検討するといいでしょう。

※51 https://docs.aws.amazon.com/ja_jp/vpc/latest/privatelink/vpc-endpoints-s3.html#vpc-endpoints-policies-s3

●S3 Object Ownership

S3 Object Ownershipをバケットに対し有効化すると、そのバケットに格納されたすべてのオブジェクトがバケットの所有者となります。これによりアップロード時にACLを設定する必要がなくなり、アクセス権を一元的に管理可能です。

●署名付き URL (Pre-signed Object URL)

署名付きURLはAWS SDKもしくはCLIを使って、一時的にS3へアクセス可能なURLを発行する機能です^{※52}。下記は、指定されたオブジェクトキーのデータへ15分間だけアクセス可能な署名付きURLをJava言語を使って実装する例です。下記の実装例では、オブジェクトキーでアクセスするオブジェクトのARNや、IAMポリシーを文字列として生成しています。そのポリシーに対しアクセス許可を設定したロールを引き受けるS3クライアントを作成し、クライアントからアクセスのためのURLを生成できます。

```
// オブジェクトキーを指定して一時的にS3へアクセス可能なURLを発行するメソッドの例
public URL getPresignedUrl(String objectKey){
    // 指定したオブジェクトキーのデータへアクセスするためのS3クライアントを取得する
    AmazonS3 amazonS3 = getS3ClientWithDownloadPolicy(objectKey);
    // 有効期限を15分間に設定する。
    Date expiration = Date.from(ZonedDateTime.now().plusSeconds(900).
        toInstant());
    return amazonS3.generatePresignedUrl("sample-bucket", objectKey,
        expiration);
}

private AmazonS3 getS3ClientWithDownloadPolicy(String objectKey) {
    // アクセスするオブジェクトのARNを作成する
    String resourceArn = new StringBuilder()
        .append(arn:aws:s3:::sample-bucket/)
        .append(objectKey)
        .toString();
    // ARNに対し、GetObjectを実行するためのIAMポリシーをJSON表現の文字列で作成する
    Statement statement = new Statement(Statement.Effect.Allow)
        .withActions(S3Actions.GetObject)
        .withResources(new com.amazonaws.auth.policy.
            Resource(resourceArn));
    String iamPolicy = new Policy().withStatements(statement).
        toJson();
}
```

```
// ロール名やIAMポリシーを指定して、GetObjectを実行するためのSDKクライアント
// を生成する
GetRoleRequest getRoleRequest = new GetRoleRequest()
    .withRoleName("sample-s3-download-role");
roleArn = AmazonIdentityManagementClientBuilder.standard().
    build()
    .getRole(getRoleRequest).getRole().getArn();
return AmazonS3ClientBuilder.standard()
    .withCredentials(new
    STSSAssumeRoleSessionCredentialsProvider
        .Builder(roleArn, "roleSessionName")
        .withRoleSessionDurationSeconds(
            (int) TimeUnit.MINUTES.toSeconds(15))
        .withScopeDownPolicy(iamPolicy)
        .build())
    .build();
}
```

署名付きURLは、前述したオブジェクトのアクセスコントロールがプライベートの場合でもアクセスできます。また、上記の実装例では、有効期限内であれば、署名付きURLを取得したユーザは誰でもアクセスできてしまいます。引き受けるロールに、ユーザIDなど独自のメタデータを加えたポリシーを渡してURLを生成すれば、指定されたパラメータを含まないリクエストはアクセス不能にするなどの制御を加えることもできます。S3に保存してあるデータサイズの大きいコンテンツに対し、特定のユーザのみが直接アクセスできるアプリケーションを構築したい場合などに特に有用な機能です。



上記ではGetObjectを使った例を取り上げていますが、PutObjectで直接S3へダイレクトアップロードする方法もあります。詳細は公式ガイド^{※53}や著者の連載記事「AWSで作るクラウドネイティブアプリケーションの応用^{※54}」も参考にしてください。

●暗号化

S3では次の2種類のサーバサイド暗号化をサポートしています（3章3節の「AWS KMS」もあわせて参照）。

1. SSE-S3

S3へファイルアップロードする際に、S3が管理するキーを使った、S3上で実行される暗号化です。キーのローテーションを行いたくない場合や煩雑なポリシー管理を避けたい場合に利用したい方法です。

※52 https://docs.aws.amazon.com/ja_jp/AmazonS3/latest/userguide/using-presigned-url.html

※53 https://docs.aws.amazon.com/ja_jp/AmazonS3/latest/userguide/PresignedUrlUploadObject.html
 ※54 <https://news.mynavi.jp/itsearch/article/devsoft/4979>（閲覧に会員登録が必要）

2. SSE-KMS

KMSが管理するキーを使った、S3上で実行される暗号化です。暗号化自体はSSE-S3と同等ですが、CloudTrailを用いて、暗号化したファイルの監査ログを残したい場合や、バケットポリシーを定義してクロスアカウントアクセスする場合などは、こちらの方法を使用します。

バケットの作成時や設定変更で「デフォルトの暗号化」として、上記2つのうち1つを選択できます。なお、完全な暗号化を求められる場合、ユーザ側で用意したキーを使って、S3へ送信する前にデータを暗号化して保存することができます（クライアントサイド暗号化：CSE-C）。3章3節の「AWS KMS」でも解説する、Encryption SDKやCLI、REST APIを使ってオブジェクトをアップロードするようにしてください。



バケット内のオブジェクトが暗号化されているかは、バケットポリシーのConditionで「x-amz-server-side-encryption」が含まれていないリクエストを拒否する設定により担保できます。

4 その他の機能

● 静的 Web サイトホスティング

Webサイトを運用するための環境をユーザに提供することをWebサイトホスティングといいます。S3の静的Webサイトホスティングは、バケットにWebサーバの役割を付与する機能です。バケットの設定でこの機能を有効化すると、以下の形式でWebサイト用のURLが発行されます。

`http://<bucket-name>.s3-website-<region>.amazon.com`

ユーザはバケットをルートとして、HTMLやCSS、JavaScriptといった静的コンテンツを配置するだけで、バケットをWebサーバとして利用できます。独自のドメインを利用したい場合は、Route 53を利用して、ドメイン名とS3のバケット名を紐付けてレコード登録することで簡易的に対応できます。商用環境などで低レイテンシな静的Webサイトホスティングが求められる場合は、5章でも解説するCloudFrontと併用するとよいでしょう。エッジロケーションを活用して、アクセスするユーザのロケーションに応じて、高速にアクセスが可能となります。上述したバケットポリシーを利用して、CloudFrontからのHTTP/HTTPSリクエストのみを許可することもできます。

● イベント通知機能

バケットに対し、データの追加や更新、削除などが行われた際に、AWS Lambda関数やSQS、SNSに対して通知を行う機能です。イベントを受け取るサービスは、イベント発生対象のバケットやオブジェクトキーの情報のほか、メタデータを受け取ることもできます。S3へデータを保存した後に関連情報をデータベースへ保存したい場合や、ファイルに対する後処理、通知などを実行したい場合に有効な機能です。



S3でイベント通知設定を行い、SQSやLambda関数を組み合わせて、ファイルのメタデータをデータベースに保存するデザインパターンを押さえておきましょう。

● CORS (Cross-Origin Resource Sharing) 機能

S3はCORSをサポートしており、静的Webサイトホスティングと同様、バケット単位にCORSを有効化できます。

通常Webサイトなどにアクセスする場合、ブラウザなどの代表的なクライアントは、セキュリティ上の理由から 基本的にアクセスしたURLと同一ドメインの静的コンテンツリソースのみを取得してレンダリングする仕様になっています。具体的には スクリプトなどのブラウザのバックエンドで動作する処理で、異なるオリジン（ドメイン+プロトコル+ポート番号の組み合わせ）の静的コンテンツリソースの取得を防止しています（これは同一生成元ポリシー：Same Origin Policyと呼ばれます。なお、HTTPドキュメントのimageタグなどはこの制約にはあてはまりません）。

Ajaxなどの非同期通信を利用して、ブラウザで実行されるスクリプトからオリジンをまたいで静的リソースを取得したい場合には、アクセス先となるオリジンで、CORSの設定を有効化します。ブラウザからのリクエストで異なるドメインへアクセスする際は、HTTPリクエストにCORSヘッダ（Originフィールド）が含まれます。WebサーバとなるS3側で、以下のように、CORSを許可するオリジンやメソッドを指定しておけば、CORSヘッダを含むリクエストに対して正常応答が可能になります。サンプルとして、CORS設定の例を紹介します。

```
<corsConfiguration>
  <corsRule>
    <allowedOrigin>http://www.example1.com</allowedOrigin>
    <allowedMethod>GET</allowedMethod>
    <allowedMethod>POST</allowedMethod>
    <allowedMethod>OPTION</allowedMethod>
  </corsRule>
</corsConfiguration>
```

CORSが設定されたS3では、リクエストに含まれるOriginフィールドパラメータがAllowedOrigin要素で指定されたものと一致するか、リクエストメソッドがAllowedMethod要素で指定されたものかどうかを検証します。問題なれば、要求されたリクエストに関する応答を返します。この設定は、前項で紹介した署名付きURLなどを使って、ブラウザからスクリプトでダイレクトアクセスする場合も同様に必要です。シングルページアプリケーションやS3へのダイレクトアップロードなどでAjaxを使用して、クロスドメインでS3にアクセスする場合は、設定を忘れないようにしましょう。

●クロスリージョン/同一リージョンレプリケーション

S3はリージョンサービスであり、同一リージョン内の3箇所のアベイラビリティゾーンにデータが複製されて保存されます。クロスリージョンレプリケーションは、非同期でデータを別のリージョンへ転送する機能です。災害対策や静的Webサイトのバックアップなどでよく用いられる機能です。同一リージョン内でレプリケーションを構成し、レプリケーション先のバケットを別のストレージクラスに変更することもできます。なお、クロス・同一両方とも異なるAWSアカウントのバケットを指定することも可能です。

●バージョン管理機能

S3はバージョン管理機能をサポートしており、ユーザやアプリケーションが誤った処理を行った場合、データが失われることがないよう、オブジェクトにバージョンを付与して管理することができます。もちろん保存したバージョンのデータサイズ分だけ課金されますが、バージョンを指定して参照できるため、履歴や監査が必要なデータに対して利用を検討したい機能です。

●ライフサイクルルール設定

ライフサイクルルール設定は、バケット内のオブジェクトに対して、作成後の日数を指定してストレージクラスを変更したり、不要データの削除などのハウスキーピング処理を自動実行したりする機能です。対象となるオブジェクトにはプレフィックスやタグなどを設定し、以下のようなライフサイクルアクションを設定できます。

1. よりアクセス頻度が低い低価格なストレージクラスへ変更
2. 上述したバージョン管理機能で最新のバージョンでないオブジェクトのストレージクラスを変更
3. 上述したバージョン管理機能で有効期限切れのバージョンや古いバージョンを削除
4. 期限切れの削除マーカーが付与されていたり、未完了のマルチアップロードファイルを削除

これらの処理は毎日0:00に自動実行されます。上記に該当するような不要データが大量に発生することが想定されるときに利用を検討したい機能です。

●オブジェクトロック機能

オブジェクトロックはバケット単位に、読み取り専用の期間を設定する機能です。以下の2種類のモードがあり、用途に応じて選択できます。

1. コンプライアンスマード

Rootユーザであってもオブジェクトの削除が期間中不可になるモードです。データの改竄や隠蔽などを防止したい場合に利用します。

2. ガバナンスマード

コンプライアンスマードと同様に、オブジェクトの削除が期間中不可になるモードですが、Lock解除オペレーションにより削除は可能です。

見落とされがちですが、設定する読み取り専用期間も当然イミュータブルであり、変更することはできません。コンプライアンスマードで誤って長期間大量のデータを保存すると、利用料金がその期間分発生してしまいますので、厳格に保存期間や将来的なデータサイズ、料金を見積もった上で、利用を検討するようにしてください。

●S3 Storage Lens

S3 Storage Lensは、アカウント内のすべてのバケットのトータルデータ量や、バケットごとの使用量など、リージョンをまたいで一元的に可視化する機能です。データ量の変化やトレンドを時系列的にトレースすることが可能です。

●S3 Transfer Acceleration

S3へのファイルアップロードを高速化したい場合に、AWSのエッジネットワークを利用してファイル転送する機能です。バケットのプロパティで有効化設定するだけで、利用者がS3へアクセスする際、自動的に世界各所に配置されている最短のエッジネットワークに誘導されるようになります。転送したデータサイズに応じて料金が上乗せされますが、AWSのマネージドバックボーンネットワークを活用して、安定・高速にS3へアクセスしたい場合に利用を検討したい機能です。

●リクエスタ支払い機能

リクエスタ支払い機能とは、バケットに対してアクセス可能なユーザを特定し、データ転送にかかる費用をユーザ側が負担する機能です。バケット単位に設定されるものですが、この機能を有効化すると、ユーザを特定するためのリクエストヘッダを含まない匿名のリクエストなどのアクセスは拒否されます。バケットにあるオブジェクトに多数アクセスする特定のサードパーティ側へ費用を負担してもらいたい場合に、利用を検討したい機能です。

●S3 アナリティクス

S3に保存されたオブジェクトに対するアクセス状況や読み込まれたデータ量を可視化するサービスです。ストレージクラスの変更を検討する場合に利用を検討したい機能です。なお、QuickSightを使って、メトリクスを時系列で参照することも可能です。

●S3 インベントリ

S3に保存されたオブジェクトのリストをCSVもしくはORCファイルで出力する機能です。スケジュール実行機能も備えており、定期的にバケット内にあるデータのリストを出力する要件がある場合などに利用を検討したい機能です。

●サーバログアクセス機能

S3はCloudTrailを用いたログ監査機能を利用できますが、バケットに対するアクセスログを記録する機能も提供しています。ただし、ログの取得はベストエフォートであり、記録に時間がかかったり、配信されない場合もあるなど完全性や適時性が保証されません。そのため、アナリティクスと同様、バケットに対するアクセス状況を検討する際などに利用を検討したい機能です。

2-10 Amazon Kinesis

Kinesisは、位置情報やセンサーデータ、動画・音声データなど、連続性に重要な意味を持つストリームデータを扱うサービスです。対象のデータや用途によっていくつかのサービスがあるので、各サービスの違いや役割、扱い方を押さえておくことが重要です。

1 ストリームデータ

Kinesisの具体的な説明に入る前に、まずKinesisで取り扱う対象となるストリームデータとそのユースケースについて説明します。データには、それ自体で意味を持つものと、データ自体の連続的な変化が重要な意味を持つものがあります。例えば、特定のエリアでどれくらいの人が出入りしたのかを知りたい場合、個々の人の位置情報データ自体より、それが特定の時間にどのくらい変化したかのほうが重要になります。異常なものを検知されたときに通知を行いたいときは、連続的な観測データで大きな変化が検出された際に、通知に必要なアクションをとることが必要になります。

これらのユースケースの処理方式を考える際、一般的にすべてのデータを恒久的に保存するという手段はありません。データ量を抑えるためにも、データを抽出(Extract)して、変換・加工(Transform)を施し、必要なデータを書き出す(Load)といった処理を行います。これらの一連の処理は各操作の英語の頭文字をとって「ETL」と呼ばれ、サポートするサービスやツール群が広く普及しています。

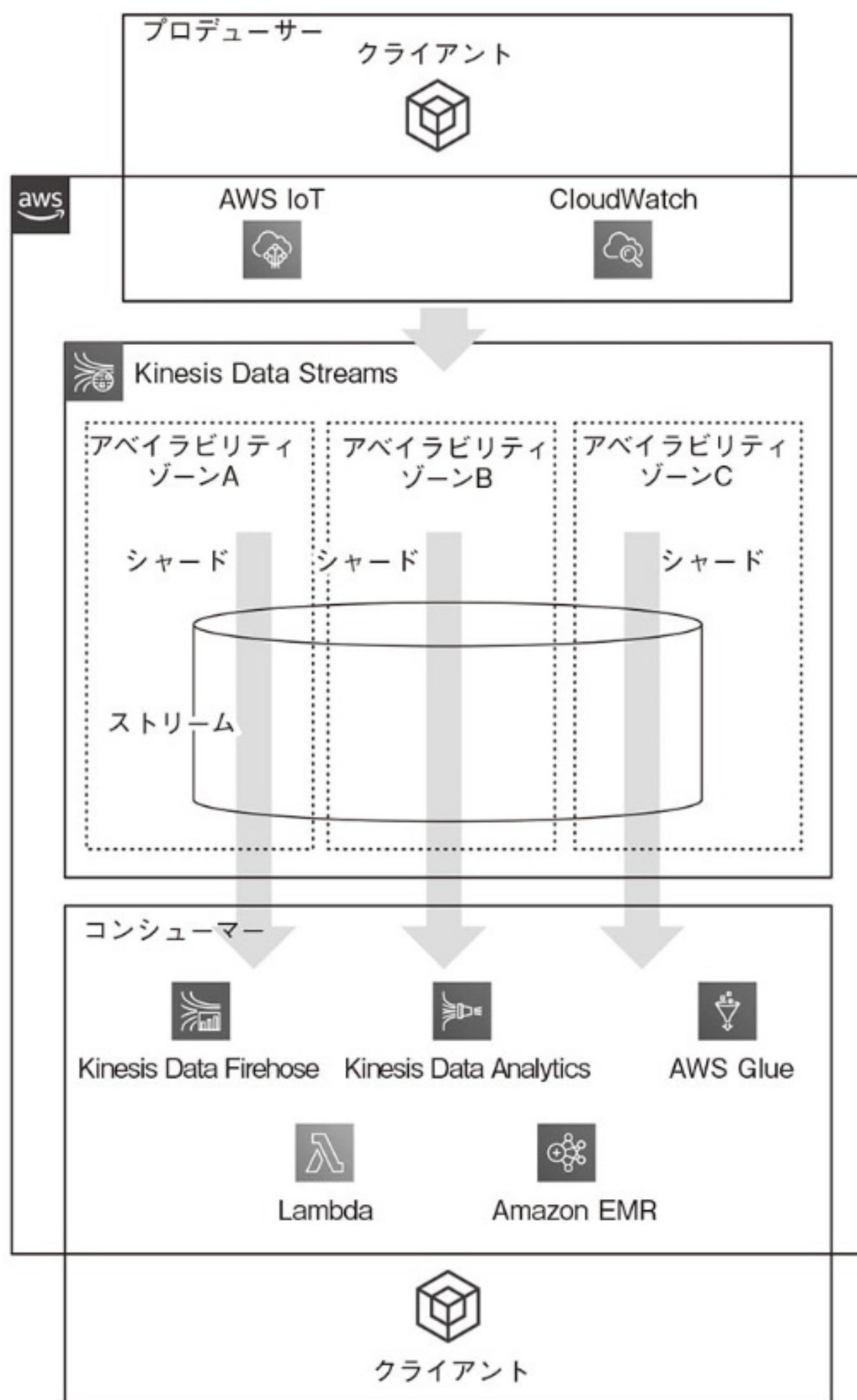
また、動画・音声データのように、時間の経過に伴い、定点の状態変化を連続的に処理して意味を持つ場合もあります。このように連続的・断続的に発生する時系列データをストリームデータと呼びます。KinesisはAWSクラウド上でストリームデータを扱うマネージドサービスで、以下のサービス群で構成されています。本節では、各サービスの詳細を解説していきます。

- Kinesis Data Stream
- Kinesis Data Firehose
- Kinesis Data Analytics
- Kinesis Video Streams

2 Kinesis Data Streams

Kinesis Data Streamsは、データ発生源となるコンポーネント（プロデューサー）から発生するストリームデータを高可用性・低コストで処理し、データを受信するコンポーネント（コンシューマー）へメッセージとして中継するためのマネージドサービスです。以下にKinesis Data Streamsを使った処理アーキテクチャの概念図を示します。

【Kinesis Data Streamsの処理アーキテクチャのイメージ図】



●プロデューサーとコンシューマー

データを送信するプロデューサーやデータを受信するコンシューマーには、次の表のようにさまざまな種類があり、ユースケースに応じて選択できます。プロデューサーやクライアントは、設定次第でAWSネットワーク内外どちらでも実行できます。

【Kinesis Data Streamsにおけるプロデューサーのオプション】

プロデューサーコンポーネント	説明
プロデューサー AWS SDK・CLIを使用したアプリケーション	AWS SDK (Software Development Kit) やCLI (Command Line Interface) を組み込んだアプリケーション。アプリケーション内の任意の処理として、Kinesis Data Streamsへデータ送信用APIを呼び出す。
Kinesis Producer Library (KPL) を使用したアプリケーション	AWSが提供するKinesis Data Streamsに特化した送信処理用ライブラリ。バッファリングや非同期実行、CloudWatchを使ったモニタリングなどがサポートされている。KPL自体はC++で記述されているが、Java Wrapperライブラリによる外部プロセスとして並列実行されるため、JavaランタイムがインストールされたOSが必要となる。
Kinesis Agent	データを収集してKinesis Data Streamsへデータを簡易的に送信できる、Javaを用いたエージェントミドルウェア。Amazon Linux AMIまたはRed Hat Enterprise Linuxがサポート対象であり、Javaランタイムのインストールが必要となる。
サーバーパーティソフトウェアが提供するクライアントライブラリ	ログ出力を行うプロダクトや、オープンソースソフトウェアなどの提供元が実装したKinesis Data Streamsデータ送信処理用のライブラリ。
Kinesis Data Generator (KDG)	AWSが提供するKinesis Data StreamsおよびFirehoseにテストデータを簡易的に送信できるUIテストツール。
AWS IoT	AWS IoTでは設定したルールにもとづいて、MQTTプロトコルから発生したデータをKinesis Data Streamsへメッセージとして送信できる。
Amazon CloudWatch	CloudWatch Logsのサブスクリプションフィルタを用いて、収集したログデータをKinesis Data Streamsへ送信できる。また、CloudWatch Eventsで発生したイベントでKinesis Data Streamsをターゲットとして呼び出すこともできる。

【Kinesis Data Streamsにおけるコンシューマーのオプション】

コンシューマーコンポーネント	説明
クライアント	AWS SDK・CLIを使用したアプリケーション AWS SDK (Software Development Kit) やCLI (Command Line Interface) を組み込んだアプリケーションから、Kinesis Data Streamsの取得用APIを呼び出す。
	Kinesis Client Libraryを使用したアプリケーション AWSが提供するKinesis Data Streamsに特化した受信処理用ライブラリ。ストリームへの接続、各シャードとワーカーの関連付け（後述）、データの取得などを実行する。さまざまなプログラミング言語のライブラリが提供されているが、処理の実態はJavaライブラリで行われるため、JavaランタイムがインストールされたOSが必要となる。
	サードパーティソフトウェアが提供するクライアントライブラリ オープンソースソフトウェアなどの提供元が実装したKinesis Data Streamsデータ受信処理用のライブラリ。
Kinesis Data Firehose	Data Streamsから取得したデータをS3やAmazon Redshift、Amazon OpenSearch Serviceなどへ配信するサービス（次項で詳述）。
Kinesis Data Analytics	Data Streamsから取得したデータをSQLクエリでリアルタイムに分析するサービス（次節で詳述）。
Lambda	Data Streamsから取得したデータをサーバレスで処理する。
Amazon EMR	Kinesis用コネクタを使って、Hadoopエコシステムツール向けに、Data Streamsから直接データの読み取りとクエリを実行できる ^{※55} 。対応するHadoopエコシステムツールは、EMRでサポートされるHive、Pig、MapReduce、Hadoop Streaming、Cascadingなどが挙げられる。

●データストリームとデータレコード

Kinesis Data Streamsでは、収集したストリームデータのことを「データストリーム」と呼んでいます。Kinesis Data Streamsで「データストリーム」としてデータを扱うと、以下のようなメリットが得られます。

1. ストリームデータの一時的な保存

Kinesis Data Streamsでは取り込んだストリームデータをデフォルトで24時間、最大8,760時間（365日分）保存できます。実装次第ではありますが、発生したデータを保存しておく機能がないプロデューサーも多く存在します。何らかの理由でデータが失われてしまった場合に再取得できるように、一時的に保存しておく目的で利用します。また、データストリームは複数のアベイラビリティゾーンに分散して保存されるため、高い可用性を保ちながらストリーム処理を行えます。

タを保存しておく機能がないプロデューサーも多く存在します。何らかの理由でデータが失われてしまった場合に再取得できるように、一時的に保存しておく目的で利用します。また、データストリームは複数のアベイラビリティゾーンに分散して保存されるため、高い可用性を保ちながらストリーム処理を行えます。

2. データの時系列化やシーケンスの付与

プロデューサーが大量のデータを連続的・断続的に送信する場合、個々のデータの通信トラフィック状況により順番が前後したり、一部欠損してしまったりする可能性があります。そこで、送信元のプロデューサーはKinesisが提供するAPIやライブラリを使用して、シーケンス番号を指定して時系列化・順序性を担保します。

3. 大量のデータ受信処理

単位時間に発生するデータ量が膨大になる場合があり、当然プロデューサーから受け取るデータの処理プロセスが1つではまかないきれないケースも発生します。そこで、ストリームを1つ以上の「シャード」と呼ばれる単位に分割して、プロセスを分けて並行処理を行い、膨大なトラフィック処理に対応します。

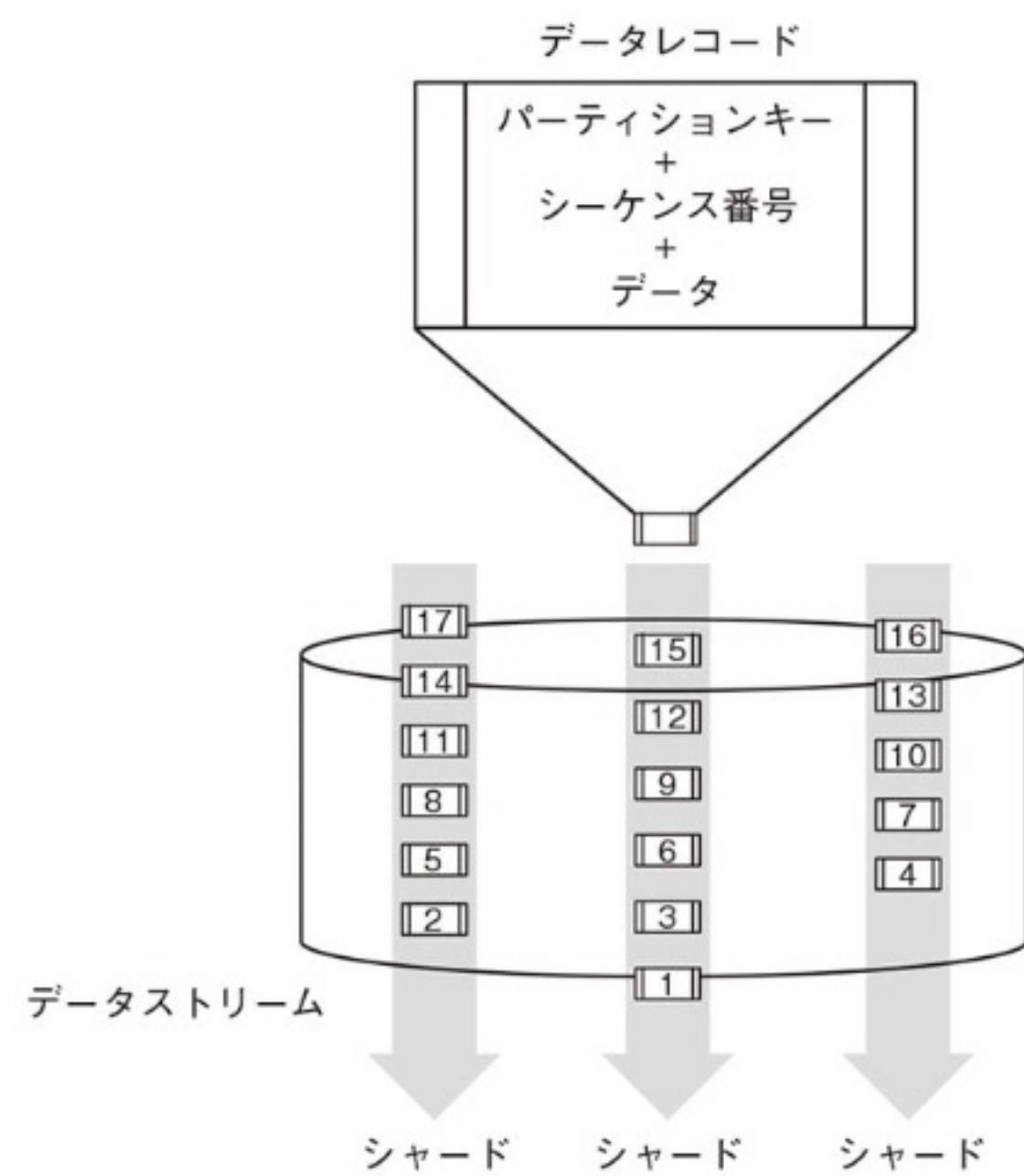
なお、Kinesis Data Streamsではデータストリームで扱われる個々のデータを「データレコード」という単位で定義しています。データレコードの最大サイズは1MBです。

●シャード

シャードはデータストリームを構成する要素で、各シャードには複数のデータレコードが保存されます。各データレコードは「パーティションキー」と呼ばれる、レコードを一意に識別する最大256文字のUnicode文字列を持ち、パーティションキーをハッシュ化した値にもとづいて、保存するシャードを決定します。また、各レコードには順序性を定義するために、パーティションキーに付随して、すべてのシャード間でユニークとなるシーケンス番号が割り振られます。シーケンス番号はデータ発生した順に単調増加する数字です。そのため、複数のシャードから断続的に受け取るデータでも、コンシューマー側では順序を指定して処理することができます。

※55 <https://aws.amazon.com/jp/emr/faqs/#kinesis-connector>

【Kinesis Data Streamsのシャードの構成イメージ】



Kinesis Data Streamsでは、プロデューサーから送信されるデータキャパシティの上限として、1シャードあたり秒間1MBもしくは、1,000レコードの送信、データレコード受信側のキャパシティ上限として、1シャードあたり秒間2MBもしくは5回の読み取りトランザクションを保証しています。全体のリクエスト処理量を決めるスループットは、このシャード数を増減させることで調整します。シャードを増減させるには、データ処理量を増加させる「シャード分割」と、データ処理量を減少させる「シャード結合」の2つのオペレーションがあります。こうした作成済みのシャードを調整することを「リシャーディング」と呼びます。

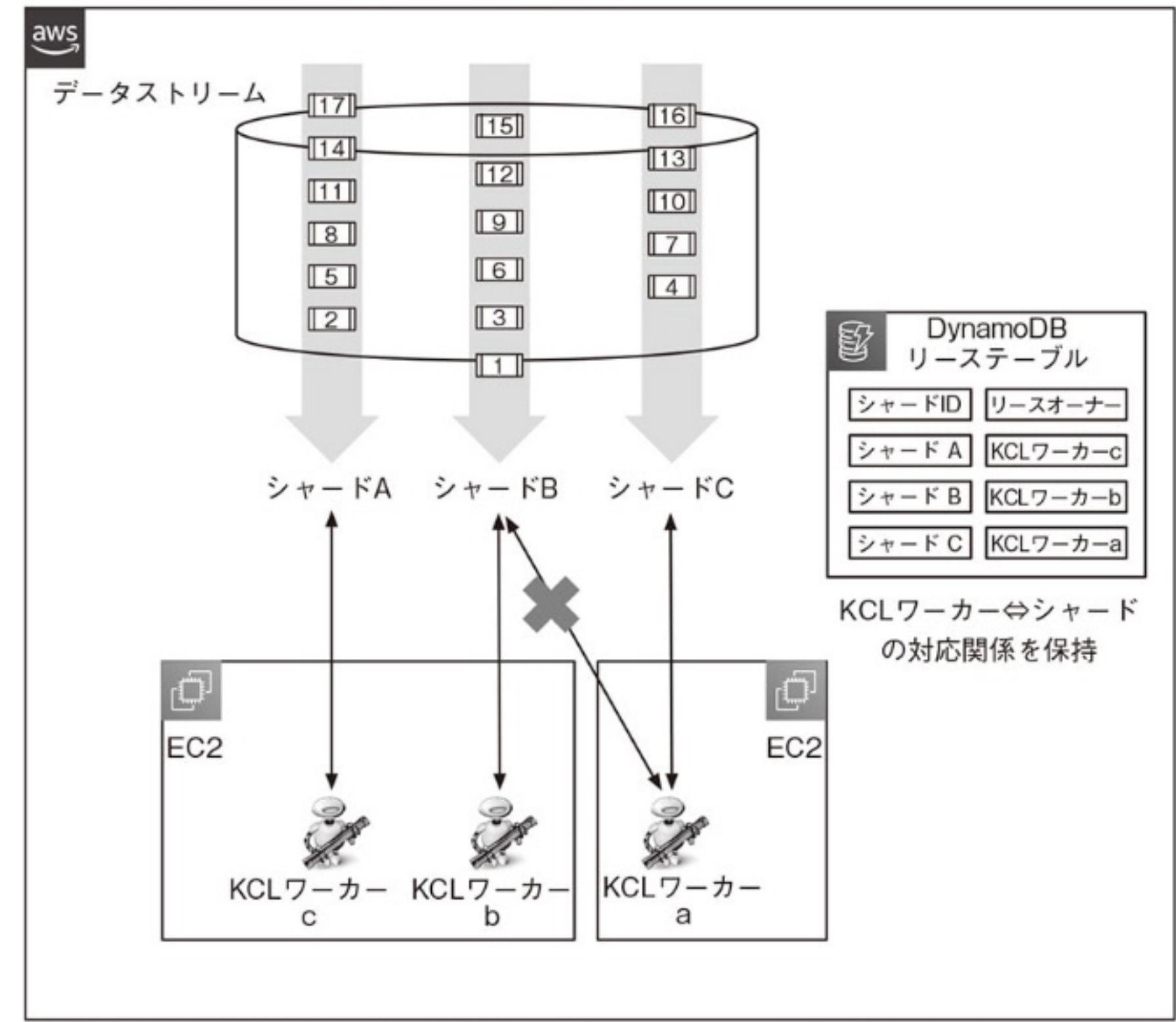
●KCL ワーカーの最適化

シャードに保存されているデータレコードはコンシューマーによって処理されます。コンシューマーとなるサービスはいくつかありますが、中でも代表的な存在といえるのがKinesis Client Library (KCL) です。KCLを使ったクライアントアプリケーションでは、シャードからデータレコードを取得し、任意の処理を実装できます。このとき、適切なスループットで処理を行うには、主にデータレコード取得処理を担当するKCLワーカーの扱いに注意が必要があります。

KCLワーカーは「リース」と呼ばれる、データを処理するワーカーと受け取るシャードを対応付ける情報を持ち、どのシャードのデータを処理しているかを監視・追跡します。この情報はAmazon DynamoDBに作られる「リーステーブル^{※56}」内に保存されています。

KCLワーカーはEC2インスタンス上で実行され、1つのEC2インスタンス上で複数のKCLワーカーを実行することもできます。一方、KCLワーカーはシャードの数だけ必要です。通常は1つのシャードに対し、1つのKCLワーカーが対応するよう構成します（1つのワーカーが複数のシャードのデータレコードを処理することはできません）。

【Kinesis Client LibraryのKCLワーカー】



言い換えると、シャードの数を超えるEC2インスタンスを用意しても、複数KCLワーカーを配置して分散処理することはできません。リシャーディングする際に調整するコンシューマー用のインスタンスの数として、スループットを最大化するのであれば、シャードと同数となるようにEC2インスタンスおよびKCLワーカーを1:1で構成します。処理に余裕ができるシャードを結合させる場合は、1つのインスタンスに複数のシャードおよびKCLワーカーを1:Nとなるように構成して実行させるとよいでしょう。

※56 https://docs.aws.amazon.com/ja_jpstreams/latest/dev/shared-throughput-kcl-consumers.html#shared-throughput-kcl-consumers-leasetable

3 Kinesis Data Firehose

Kinesis Data Firehoseは、ストリームデータを配信するサービスです。配信先となる対象は、S3、Amazon Redshift、Amazon OpenSearch ServiceといったAWSのマネージドサービスのほか、Splunk、Datadog、New Relicといったサードパーティサービスが含まれます。

Kinesis Data Firehoseを用いることで、コンシューマーアプリケーションの実装やその実行基盤となるインフラストラクチャの管理を一切行わずに、ターゲットに配信できます。また、ストリームデータのバッチ化・データ圧縮・暗号化、エラーログの記録、バッファリング機能も付属しています。Lambdaやオープンソースソフトウェアを利用して、ストリームデータの整形処理を行うことも可能です。Data Firehoseでは、Data Streamsとは違い、配信先となる「配信ストリーム」を作成するだけで環境構築が完了します。シャードの作成やパーティションキーの指定は必要ありません。データレコードのサイズはData Streamsと同様、1MBまでとなっています。

配信データのプロデューサーとして選択できる代表的なサービスは次のとおりです。

1. Kinesis Data Streams

Kinesis Data Streamsで出力したストリームデータをプロデューサーとして選択できます。S3へそのままバックアップとして保存する場合や、RedshiftでETL解析に用いる場合にFirehoseを経由するとコンシューマーを実装せずに済みます。なお、Redshiftへ保存した場合は、一度S3に保存されてからRedshiftにロードされます。

2. AWS SDK・CLIを使用したアプリケーション

AWS SDK (Software Development Kit) やCLI (Command Line Interface) を組み込んだアプリケーションです。アプリケーション内の任意の処理として、Kinesis Data Firehoseへデータ送信用APIを呼び出します。

3. Kinesis Agent

データを収集してKinesis Data Firehoseへデータを簡易的に送信できる、Javaを用いたエージェントミドルウェアです。Amazon Linux AMIまたはRed Hat Enterprise Linuxがサポート対象であり、Javaランタイムのインストールが必要となります。送信対象となるエンドポイントが異なるだけで、Data Streamsのエージェントと同一です。

4. AWS IoT

AWS IoTでは設定したルールにもとづいて、MQTTプロトコルから発生したデータをKinesis Data Firehoseへメッセージとして送信できます。

5. Amazon CloudWatch

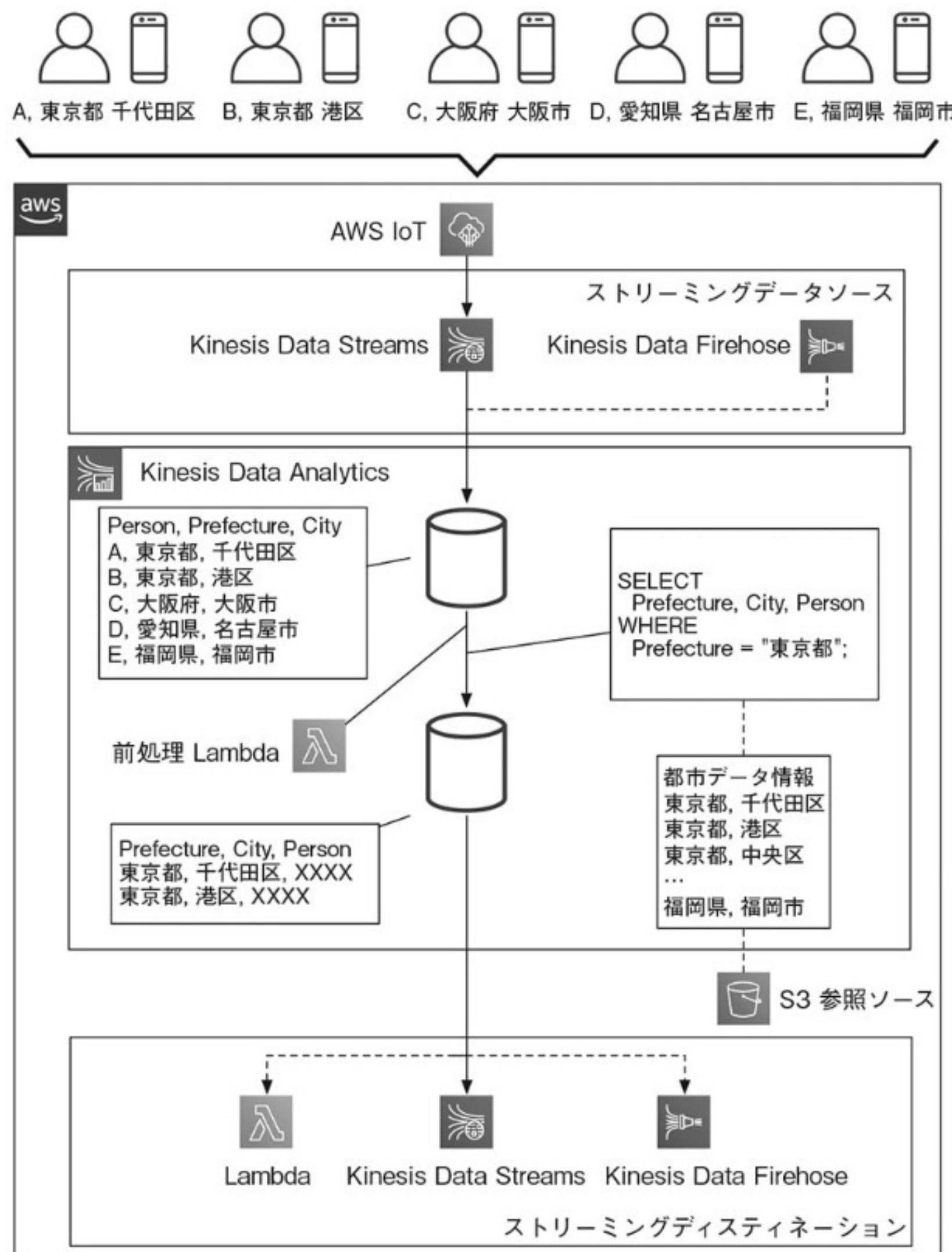
CloudWatch Logsのサブスクリプションフィルタを用いて、収集したログデータをKinesis Data Firehoseへ送信できます。また、CloudWatch Eventsで発生したイベントでKinesis Data Firehoseをターゲットとして呼び出すこともできます。

4 Kinesis Data Analytics

Kinesis Data Analyticsは、Kinesis Data Streams、Kinesis Data Firehoseに格納されたデータのリアルタイム分析を提供するサービスです。Kinesis Data AnalyticsはSQLと、分散処理のオープンソースソフトウェア「Apache Flink」を利用するオプションがあります。

次の図は、SQLオプションを使ってリアルタイム分析を行う例を示したものです。例えば、位置情報発信が可能なスマートデバイスを持った人から、ロケーションデータをAWS IoTを通じて収集し、Kinesis Data Streamsでデータを一時保存した上で、特定のロケーションにどれくらいの人が存在するか解析したいとします。

Kinesis Data Analyticsの例



Kinesis Data Analyticsでは、Kinesis Data Streams（ストリーミングデータソース）から収集したデータを入力用ストリームとして取り込みます。これは継続的に更新されるデータベースのテーブルのようなものです。次にリアルタイムで分析したい内容に応じて、出力データを生成するSQLステートメントを記述します。この例では、入力ストリームから、ロケーションの都道府県が「東京都」であるデータを抽出しています。結果、該当するデータが再びストリームデータとして出力されます。

Kinesis Data Analyticsでは入力となるストリーミングデータソースとして、Kinesis Data Streams、Kinesis Data Firehoseを選択することができます。必要に応じて、S3内に格納したデータをリアルタイム解析するSQLの参照用のソースデータとして利用することもできます。図中に記載している例のように、抽出条件となる都市データの一覧など、SQLの実行をサポートする参照用テーブルとして活用するとよいでしょう。

また、SQLクエリを実行する前に前処理としてLambda関数を挟めます。図中の例では、人物の個人情報をマスク化するLambda関数を図示しています。

ストリームデータを出力する対象を「ストリーミングディスティネーション」と呼びますが、こちらもストリーミングデータソースと同様、Kinesis Streams、Kinesis Firehoseに加えて、Lambda関数を選択可能です。ストリーミングディスティネーションには、SQLの実行中エラーとなったストリームデータもオプションで出力できます。エラーの原因となるのは、不正な形式やデータサイズが一因です。なお、入力データソースとなる1行あたりのデータサイズは最大50KB、参照ソースの最大サイズは1GBという制限があるので注意が必要です。

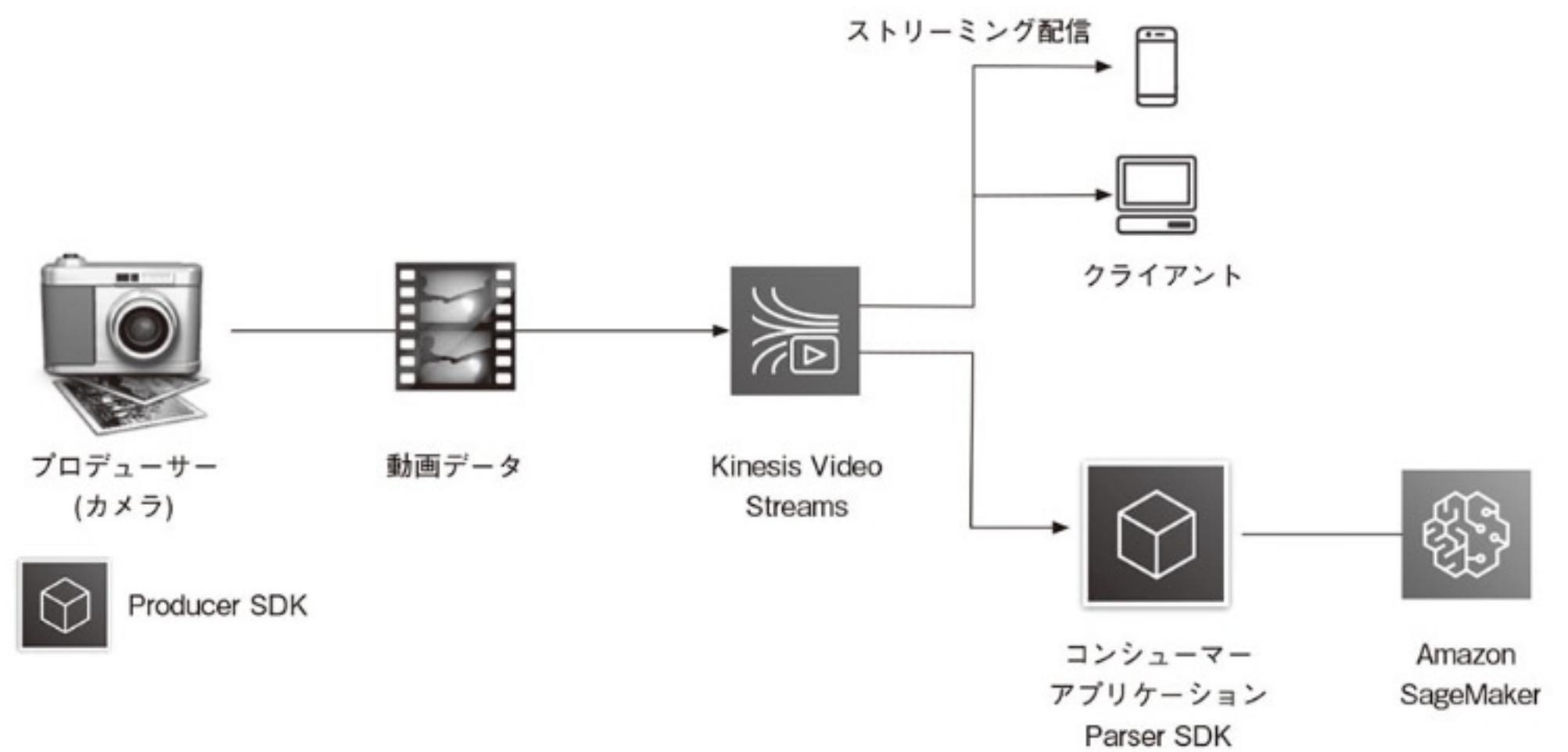
5 Kinesis Video Streams

Kinesis Video Streamsは、動画・音声ストリームデータの収集やストリーミング、ライブ配信、オンデマンド再生を行うためのサービスです。MP4/AVI/MOVなどのメディア形式のストリームデータを扱う方式と、双方向通信が可能なWebRTC形式による2種類の方式をサポートします。

● メディア形式による取り込み

C、C++、Javaといったプログラミング言語に対応したKinesis Video Streams Producer SDKを使って、メディア形式のストリームデータを収集できるほか、ストリーミングによるライブ配信、オンデマンド再生が行えます。

Kinesis Video Streamsを使ってメディア形式のストリームデータを扱う例



ストリームデータを暗号化して保存でき、収集したデータを、Amazon Rekognition VideoやSageMakerをはじめとした機械学習系のサービスへ連携することもできます。動画データを解析するためのKinesis Video Streams Parser Libraryも提供されており、動画内の特定の画像を抽出して、機械学習サービスと組み合わせることにより、特定のイメージを検出・認識するといったことも可能です。

●WebRTCによる取り込み

WebRTC (Real-Time Communication) とは、ブラウザやモバイルアプリケーションとピア・ツー・ピア通信でリアルタイム通信を行う技術です。Kinesis Video StreamsではこのWebRTCを使用して、オンラインビデオ会議のような双方向動画通信を、低遅延で100万台規模のデバイス間で実行できます。双方向ビデオアプリケーションを開発するには、JavaScriptや、C、Android、iOSといった開発環境向けに提供されているWebRTC用のSDKを使用します。

2-11 その他の開発関連サービス

AWSでは、そのほかにも開発をサポートするさまざまなサービスがあり、認定試験でも内容を問われる場合があります。似たような機能を提供する開発サービスが多くありますので、しっかり内容を押さえておくようにしましょう。

1 AWS AppSync

AWS AppSyncは、GraphQL APIの完全マネージド型サービスです。GraphQLはオープンソースソフトウェアのクエリ言語で、簡単な実装でサーバからJSONデータを取得することができます。

AppSyncでは、Cognitoの認証・認可やDynamoDB、Lambdaといったバックエンドマネージドサービスと連携し、GraphQL APIを容易に構築することができます。

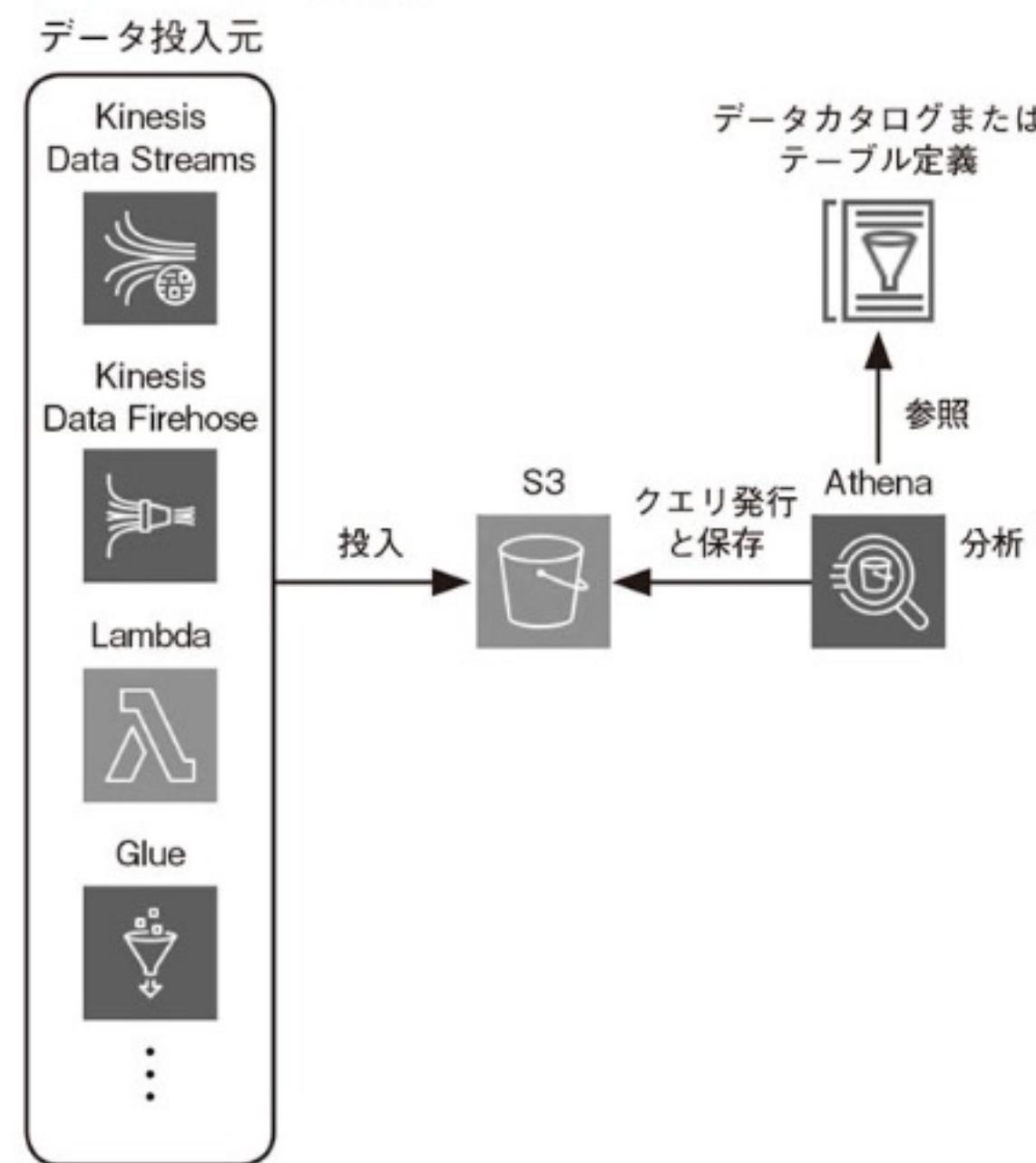
2 AWS Amplify

AWS Amplifyは、モバイルアプリケーションおよびWebアプリケーション開発のためのプラットフォームです。AWSがオープンソースソフトウェアとして提供しています。JavaScript (React、Angular、Vue、Next.js) や iOS、Android、React Native、Ionic、Flutterなどのモバイル向けソフトウェアのクライアントライブラリやUIコンポーネント、CLIが、Amplify Frameworkとしてサポートされています。AmplifyではCLIを使って、簡単にバックエンドサービスを構築することができます。Cognitoの認証・認可やDynamoDB、Lambdaといったバックエンドマネージドサービスと連携したり、前述のAppSyncをベースとしたGraphQL APIを容易に構築することも可能です。

3 Amazon Athena

Athena（アテナ）は、S3に格納されているさまざまなフォーマットのデータに対して直接SQLを発行して分析を行えるマネージドサービスです。SQLを処理するサーバはAWSによって管理されるため、ユーザは分析に集中できます。

【Athenaの全体像】



クエリを実行する際は、AWS Glueのデータカタログかテーブル定義を設定し、事前に分析する対象の構造を明らかにする必要があります。

4 AWS App Runner

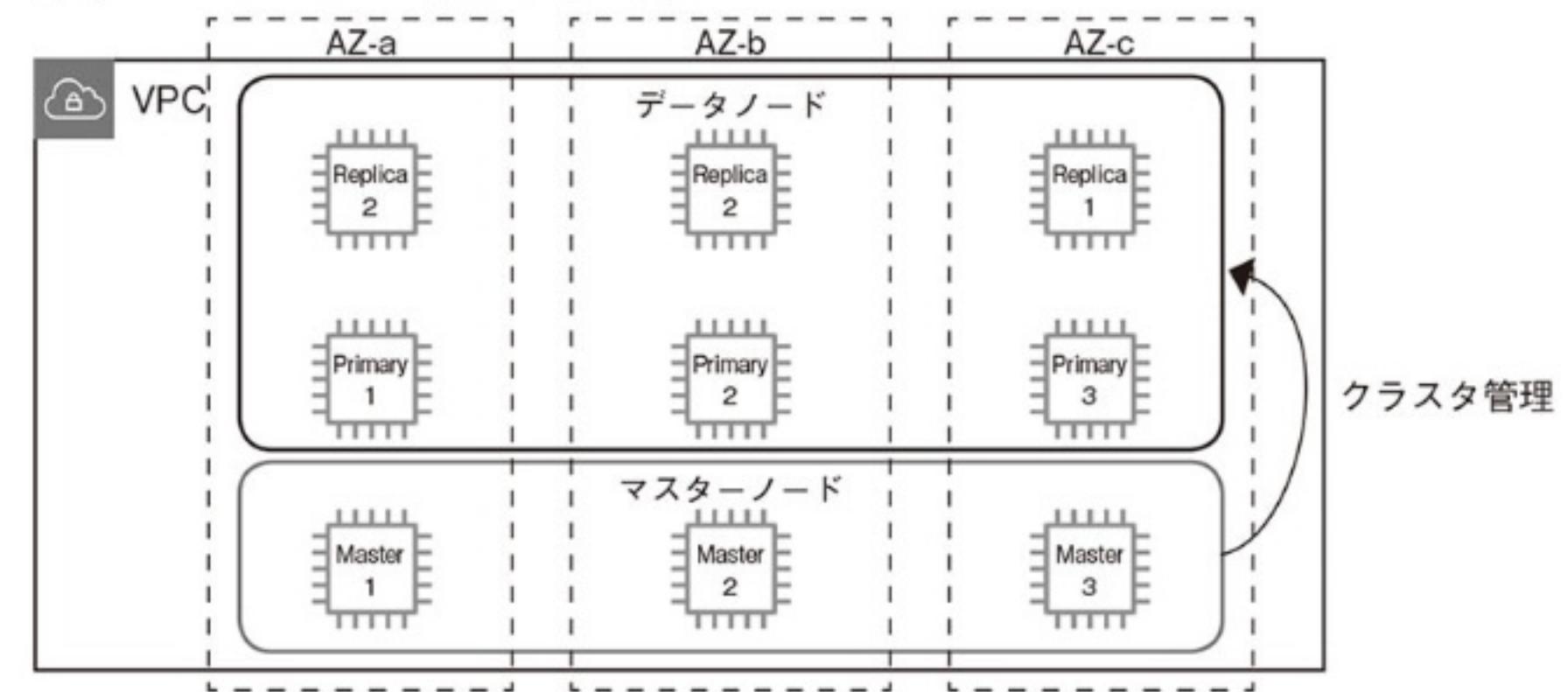
AWS App Runnerは2021年5月に発表された、コンテナアプリケーションを簡単にデプロイするサービスです。マネジメントコンソールなどから、簡単にロードバランサーやTLS、カスタムドメインの設定を行い、ECRやGitHubへプッシュされたコンテナイメージをデプロイできます。

5 Amazon OpenSearch Service

Amazon OpenSearch Serviceは、分析機能のOpenSearchと可視化機能のKibanaを導入したクラスタを提供するマネージドサービスです。EMRと同様に、データ処理量に応じて簡単にスケールアウトできます。

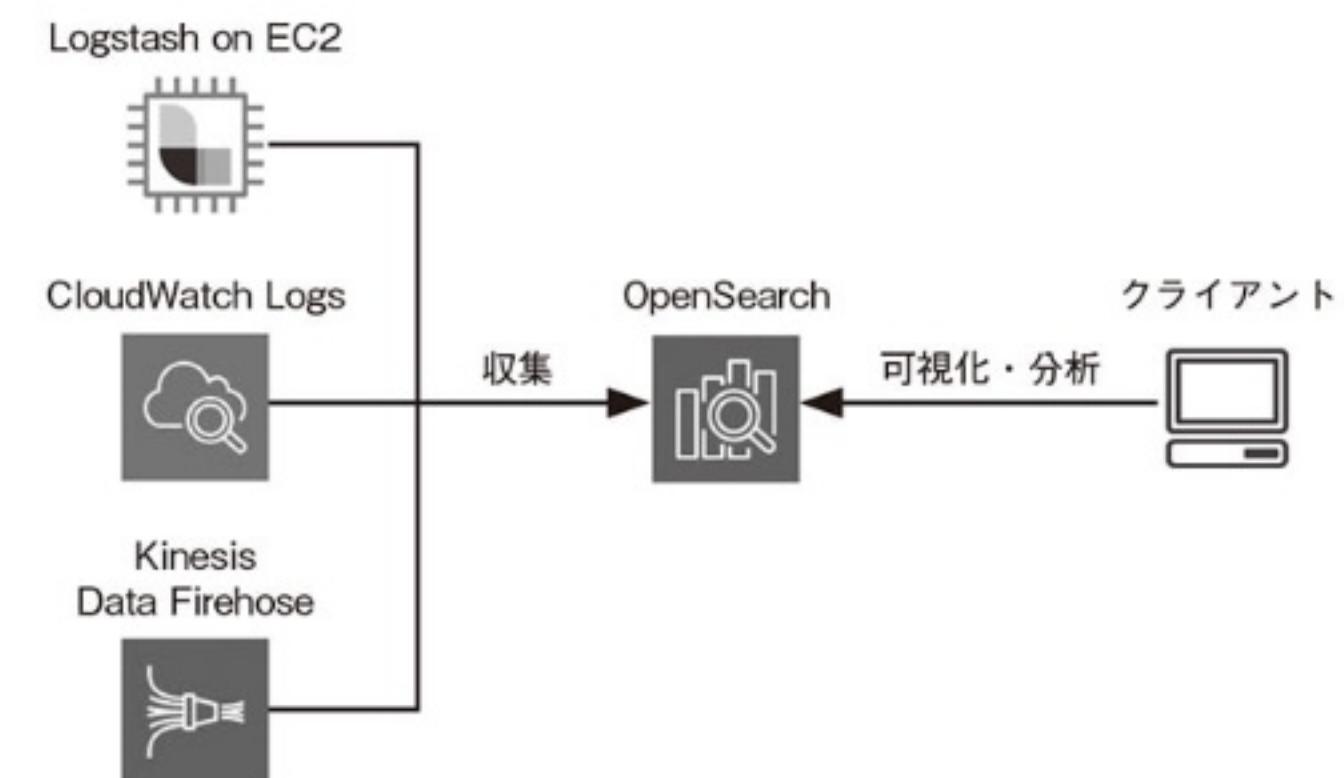
Amazon OpenSearch Serviceのクラスタは、クラスタ全体の管理を行うマスターノードと、データを保持して分析結果を返却するデータノードで構成されます。これらのノードはレプリカを持つことができ、異なるAZに配置することで可用性の高いクラスタを構築できます。

【OpenSearchアーキテクチャ】

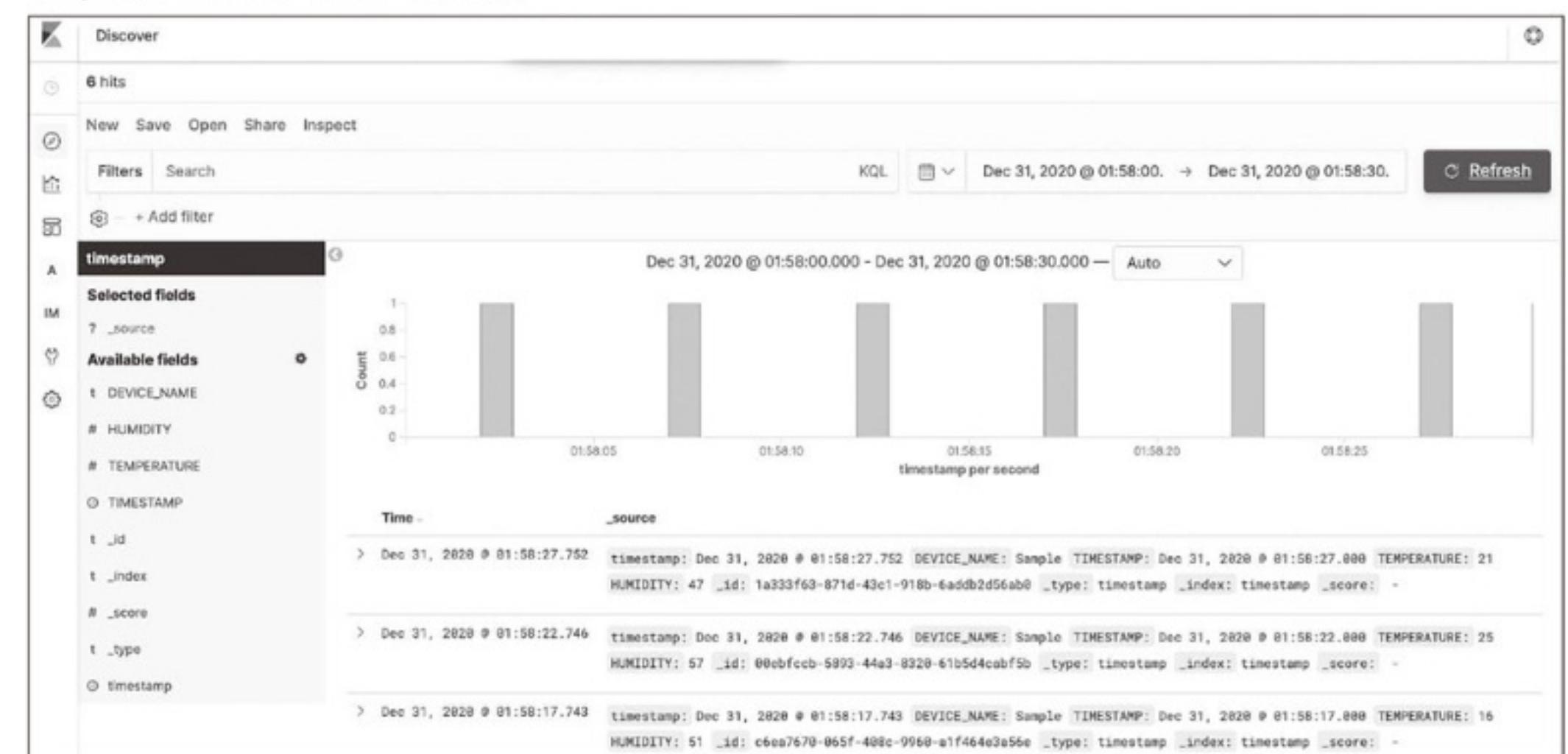


OpenSearchへのデータ投入は、データ収集エンジンのLogstashやAWSサービス間の連携機能などさまざまな方法が利用できます。収集したデータはOpenSearchで可視化・分析することができます。

【OpenSearchへのデータ投入】



【OpenSearchによる可視化】



演習問題

1 ある企業では新しく提供するWebサービスのプラットフォームとして、ソーシャルログイン可能なモバイルアプリケーションのバックエンド構成をAPI Gatewayを使って構築することを検討しています。次のうち、構成可能なものを2つ選択してください。

- A. API Gateway REST APIを構成し、JWTオーソライザを利用してリクエストを認証します。バックエンド処理でDynamoDBへアクセスするLambda関数を実装します。
- B. API Gateway REST APIを構成し、JWTオーソライザを利用してリクエストを認証します。バックエンド処理としてECS上で実行するコンテナアプリケーションにDynamoDBへアクセスする処理を実装し、ALBのターゲットグループにルーティングするよう構成します。
- C. API Gateway REST APIを構成し、ソーシャルIDプロバイダを追加したCognitoオーソライザを利用してリクエストを認証します。バックエンド処理としてECS上で実行するコンテナアプリケーションにDynamoDBへアクセスする処理を実装し、ALBのターゲットグループにルーティングするよう構成します。
- D. API Gateway HTTP APIを構成し、ソーシャルIDプロバイダを追加したCognitoオーソライザを利用してリクエストを認証します。バックエンド処理でDynamoDBへアクセスするLambda関数を実装します。
- E. API Gateway HTTP APIを構成し、ソーシャルIDプロバイダを追加したCognitoオーソライザを利用してリクエストを認証します。バックエンド処理としてECS上で実行するコンテナアプリケーションにDynamoDBへアクセスする処理を実装し、ALBのターゲットグループにルーティングするよう構成します。

2 ある企業では、オンプレミス環境にある既存システムから、AWSクラウドにあるWebサービスへのアクセスをAPI Gatewayを使って構築することを検討しています。このWebサービスはセキュリティ上、プライベートなネットワークからのアクセスのみ許可し、インターネットからのパブリックアクセスは許可されません。要件を満たすことができる適切なものを選択してください。

- A. オンプレミス内のDNSサーバにRoute 53 Resolverのフォワーダを設定し、プライベートAPIエンドポイントでAPI Gateway REST APIを構成します。オンプレミスネットワークからのみのアクセスを許可するようリソースポリシーを設定して、API Gatewayにアタッチします。
- B. オンプレミス内のDNSサーバにRoute 53 Resolverのフォワーダを設定し、プライベートAPIエンドポイントでAPI Gateway REST APIを構成します。オンプレミスネットワークからのみのアクセスを許可するよう、API Gatewayへ接続するVPCリンクのセキュリティグループを設定します。
- C. プライベートホストゾーンをRoute 53に設定し、リージョンAPIエンドポイントでAPI Gateway REST APIを構成します。オンプレミスネットワークからのみのアクセスを許可するよう、API Gatewayへ接続するVPCリンクのセキュリティグループを設定します。
- D. プライベートホストゾーンをRoute 53に設定し、リージョンAPIエンドポイントでAPI Gateway REST APIを構成します。オンプレミスネットワークからのみのアクセスを許可するようリソースポリシーを設定して、API Gatewayにアタッチします。

3 ある開発者がモバイルアプリケーションで必要なデータを、レガシーなWebサービス連携を含むバックエンドサービスから取得するためにAPI Gateway REST APIの設計をしています。開発者が進める設計の中で、誤った記述を1つ選択してください。

- A. OpenAPI (Swagger) v2.0/3.0に準拠した定義ファイルからAPIを作成したり、既存APIをクローンすることにより作成できます。また、逆にJSON/YAML形式でのエクスポートもサポートしています。
- B. ステージごとにスロットリングやキャッシュ、Web ACL、Canary、変数、ログ/トレースの設定ができます。
- C. APIキーはクライアントと使用プランの紐付けを行うために利用します。
- D. マッピングテンプレートはメソッドリクエスト・レスポンス設定でデータモデルフォーマットを変更するために使用されます。

4 あるモバイルアプリケーション開発者は、API Gatewayを使ったリクエスト認証・認可で必要なセキュリティ方式を検討しています。対策として費用対効果が高く、適切なものを選択してください。

- A. Cognitoで認証されたユーザに対して、アクセスキーから生成したAWS署名バージョン4のハッシュ値をHTTPヘッダに設定してAPI Gatewayで認証します。
- B. クライアントから、API Gatewayに対してBearerトークンもしくはHTTPリクエストヘッダのパラメータに認証情報を付与したリクエストを送信してLambda関数で検証し、検証が問題なければアクセストークンを返却してAWSリソースへアクセスさせます。
- C. Cognitoでユーザの認証を行った後、取得したJWTアクセストークンをHTTPリクエストヘッダに設定してAPI Gatewayに送信します。API GatewayはCognitoに対してトークン検証を行い、検証が問題なければ、AWSリソースへアクセスさせます。
- D. サードパーティのOIDCプロバイダでユーザの認証を行った後、取得了JWTアクセストークンをHTTPリクエストヘッダに設定してAPI Gatewayに送信します。API GatewayはLambdaオーソライザでトークン検証を行い、検証が問題なければ、AWSリソースへアクセスさせます。

5 ある開発者がすでにカットオーバーした、API GatewayおよびLambdaで構成されるアプリケーションへの新しいバージョンのリリース方法を検討しています。次のうち、ダウン時間やリスクを最小化しながら最も低コストで対応できる方法を選択してください。

- A. すでに作成しているプロダクションステージに「Canary」を定義し、リクエストの10%を割り当て、10%で特に問題が発生しなければ、「Canary」を昇格させ、アプリケーションを置き換えます。
- B. 新たに「Canary」が有効化されたプロダクションステージを定義し、リクエストの10%を割り当て、10%で特に問題が発生しなければ、「Canary」を昇格させ、アプリケーションを置き換えます。
- C. 新たにAPIを作成し直し、「Canary」ステージが有効化されたプロダクションステージを定義し、リクエストの10%を割り当て、10%で特に問題が発生しなければ、「Canary」を昇格させ、アプリケーションを置き換えます。
- D. 新たに「Canary」が有効化された開発用ステージを定義し、リクエストの10%を割り当て、10%で特に問題が発生しなければ、「Canary」を昇格させ、アプリケーションを置き換えます。

6 ある企業では、新たなモバイル向けアプリケーションのバックエンドサービス処理をLambda関数で実装しようとしています。バックエンドサービスでは、RDSへアクセスし、またサードパーティサービスへインターネット経由でアクセスする必要があります。Lambdaの設定構成の中で誤っているものを選択してください。なお、RDSへのアクセスはRDS Proxyを用いるものとし、RDS Proxyへの接続はIAM認証で実行するものとします。

- A. RDS Proxyを構成し、プロキシの識別子とアクセスするRDSインスタンス、データベースのユーザ名とパスワードを保存するSecrets Manager、およびプロキシがシークレットにアクセスするためのIAMロールの設定を行います。
- B. NAT Gatewayを作成し、RDSが配置されているVPCのプライベートサブネットのルートテーブルルーティング情報を追加します。
- C. Lambda実行のIAMロールに「AWSLambdaBasicExecutionRole」のほか、「AWSLambdaVPCAccessExecutionRole」ポリシーをアタッチします。
- D. アプリケーションからデータベースへアクセスするためのRDS ProxyのエンドポイントURLのための環境変数を設定します。

7 ある企業では、定期的にS3バケットの不要データをクリアリングするためのバッチ処理をLambda関数で実行しようとしています。最も簡易的に実装できる手法を選択してください。

- A. Lambdaのカスタムランタイム機能を用いて、Linuxベースのカスタムコンテナイメージを用意して、/etc/crontabに定期的にLambda関数を実行する設定を追加してデプロイします。
- B. EventBridgeを設定して、ターゲットイベントにLambda関数を設定します。
- C. PERIOD_EXECUTION環境変数に、実行したい間隔(秒単位)で値を設定し、Lambda関数をデプロイします。
- D. /etc/crontabに定期的にLambda関数を呼び出す設定を追加したEC2インスタンスを常時起動します。

8 あるモバイルECサイトでは、購入処理のバックエンドサービスをAPI GatewayとLambda関数を使って構築しています。商品の在庫不足のエラーが発生した場合に、オーナーへ通知したいと考えています。最も効率的に実装できる手法を選択してください。

- A. 在庫不足エラーが発生した際にデッドレターキューを発行し、SNSでオーナーへ通知します。
- B. 在庫不足エラーが発生した際にデッドレターキューを発行し、SQSでキューイングし、SNSを経由してオーナーへ通知します。
- C. 在庫不足エラーが発生した際にLambda関数のログを出力しているCloudWatch Logsのサブスクリプションフィルタで検知し、SNSでオーナーへ通知します。
- D. 在庫不足エラーが発生した際にLambda関数のログを出力しているCloudWatch Logsのサブスクリプションフィルタで検知し、オーナーへ通知する処理を新たなLambda関数で実装します。

9 あるECサイトでは、取り扱っている商品の画像をS3へアップロードして顧客向けサイトに表示させています。商品画像がアップロードされたのち、サムネイル画像を生成する処理をLambda関数で実装したいと考えています。画像のアップロードは一度に大量に行われるため、すべてのサムネイル画像処理が正常に行われたかチェックする必要があります。これを実現するための最も簡易的な方法を選択してください。

- A. S3へアップロードされたイベントをイベントソースマッピングとしてLambda関数に設定します。サムネイル画像生成処理でエラーが発生した場合、デッドレターキューを管理者側へ発行するよう設定します。
- B. S3へアップロードされたイベントをイベントソースマッピングとしてLambda関数に設定します。サムネイル画像生成処理が完了した後、送信先の設定で、「On failure」、「On success」条件で、処理が正常に完了したかを振り分けて管理者へ通知します。
- C. S3へアップロードされたイベントをトリガーとしてLambda関数に設定します。サムネイル画像生成処理でエラーが発生した場合、デッドレターキューを管理者側へ発行するよう設定します。
- D. S3へアップロードされたイベントをトリガーとしてLambda関数に設定します。サムネイル画像生成処理が完了した後、送信先の設定で、「On failure」、「On success」条件で、処理が正常に完了したかを振り分けて管理者へ通知します。

10 東京リージョンで展開されている大規模ECサイトでは、購入処理のバックエンドサービスをLambda関数を使って構築しています。リクエストは1処理完了するのに2秒を必要とし、ピーク時には平均2,000リクエストが見込まれます。ピーク時のエラーを抑制するために実施すべき内容を選択してください。

- A. アクセスが集中した場合に備えて、Lambda関数がAWS APIをSDKを使って呼び出す処理に対してエクスプロンシャルバックオフを実装します。
- B. AWSサポートにLambda関数の同時実行の制限を引き上げるようリクエストします。
- C. Lambda関数のCPUおよびタイムアウト時間を拡張します。
- D. アクセスが集中した場合に備えて、Lambda関数のウォームスタートを行うよう、AWSサポートにリクエストします。

11 ある移動体通信事業者は、利用者の位置情報を保存するデータベースとして DynamoDBを利用することを検討しています。事業者が開発するアプリケーションには指定された時間帯で、特定のロケーションに位置したユーザをトレースする機能要件があります。要件を満たすために最も適切なテーブル・インデックス構成を選択してください。

- A. パーティションキーとしてユーザID、ソートキーとしてタイムスタンプとすることで、位置情報を含むテーブルを構成します。特定のロケーションを一意に識別するコードをグローバルセカンダリインデックスのパーティションキー、タイムスタンプをソートキーとして指定します。
- B. パーティションキーとしてユーザID、ソートキーとしてタイムスタンプとすることで、位置情報を含むテーブルを構成します。特定のロケーションを一意に識別するコードをローカルセカンダリインデックスのパーティションキー、タイムスタンプをソートキーとして指定します。
- C. パーティションキーとして特定のロケーションを一意に識別するコード、ソートキーとしてユーザIDとすることで、位置情報を含むテーブルを構成します。ユーザIDをグローバルセカンダリインデックスのパーティションキー、タイムスタンプをソートキーとして指定します。
- D. パーティションキーとして特定のロケーションを一意に識別するコード、ソートキーとしてユーザIDとすることで、位置情報を含むテーブルを構成します。ユーザIDをローカルセカンダリインデックスのパーティションキー、タイムスタンプをソートキーとして指定します。

12 ある開発者がDynamoDBテーブルにアクセスするサーバレスアプリケーションを構築しています。データの平均サイズは7KBであり、1秒間に平均30回の強い一貫性が必要な読み込みと、平均10回のトランザクション書き込みが発生することが予想されています。プロビジョンドスループットとして最低限設定すべきキャパシティユニットを選択してください。

- A. 読み込み120RCUsと書き込み140WCUs
- B. 読み込み120RCUsと書き込み70WCUs
- C. 読み込み60RCUsと書き込み140WCUs
- D. 読み込み60RCUsと書き込み70WCUs

13 ある企業では、多数の書き込みトランザクションが発生し、非常に低いレイテンシでの読み込み・書き込み応答が要求される市場取引アプリケーションを開発しています。性能要求を満たすために開発者が選択すべき、一般に最も費用対効果が高く、適切なオプションを選択してください。

- A. ElastiCacheを使用します。
- B. グローバルセカンダリインデックスを使用します。
- C. DAXを利用します。
- D. 読み取り・書き込みキャパシティユニットを最適なレベルに調整します。

14 コンビニエンスストアを運営している、ある企業は、1日のトランザクションデータを集約し、売上データを計算する必要があります。集計のために一時的にDynamoDBに計算対象となるトランザクションデータを保存するテーブルを作成し、計算処理が完了した後はコスト削減のためにこのテーブルのデータを消去します。設計者が採用すべき最も費用対効果が高い手法を選択してください。

- A. データを消去する処理は、条件付き書き込みを使用して、計算処理済みのデータをDeleteItem APIを呼び出して削除します。
- B. データを消去する処理は、TTLを利用して特定の時間が経過した後、削除されるように設定します。
- C. データを消去する処理は、一括でデータ削除されるようBatchWriteItem APIを使用して削除します。
- D. データを消去する処理は、計算処理完了後にテーブルごと削除します。

15 ある金融機関の担当者は、DynamoDBで管理されている、取引に使用される重要なマスターデータが更新された場合に通知を受け取り、変更内容を把握したいと考えています。要件を実現するために開発者が実装する最もシンプルな方法を選択してください。

- A. CloudWatch Eventを使用して、DynamoDBの項目が変更されたたびに、通知を送信するように設定します。
- B. CloudWatch Logsを使用して、項目の更新処理のログをサブスクリプションフィルタで検出して、通知を送信するように設定します。
- C. DynamoDB Streamsを有効化して、変更前後のデータをLambda関数を使用して加工し、通知を送信するように実装します。
- D. DynamoDB Streamsを有効化して、DynamoDBの項目が変更されたたびに、通知を送信するように設定します。

- 16 ある企業のアプリケーションはECSコンテナ上で実行されており、ALBによりロードバランシングされていて、ALBにはRoute 53によって固有のドメイン(www.sample.com)が割り当てられています。今、この環境と同等構成のステージング環境のドメイン(www.staging.sample.com)で、試験中に新人エンジニアが設定オペレーションを誤ってしまい、ドメイン名を使ってALBのバックエンドで実行されているECSアプリケーションにアクセスできなくなってしまいました。新人エンジニアはドメインのAレコードに新しく構築したALBのプライベートアドレスを設定しようとしたと説明しており、アクセス不可となっているRoute 53のパブリックホストゾーンは以下のとおり設定されています。このアクセス不可の原因と復旧対処について最も適切な方法を選択してください。

```
staging.sample.com 3600 IN NS      ns-xxxx.awsdns-xx.org
                    ,ns-xxxx.awsdns-xx.com.
                    ,ns-xxxx.awsdns-xx.net.
                    ,ns-xxxx.awsdns-xx.co.uk.

staging.sample.com 3600 IN SOA     ns-xxxx.awsdns-xx.com awsdns-
hostmaster.amazon.com. 1 7200 900 1209600 86400

www.staging.sample.com 3600 IN A      XXX.XXX.XXX.XXX
```

- A. 設定レコードが誤っているため接続不能になっています。プライベートIPアドレスを設定するのはAレコードではなく、AAAAレコードです。Aレコードを削除し、AAAAレコードにロードバランサーのARNを指定します。
- B. 設定レコードが誤っているため接続不能になっています。プライベートIPアドレスを設定するのはAレコードではなく、CNAMEレコードです。CNAMEレコードにエイリアスとして新しく作成したロードバランサーを指定します
- C. レコードで認識できないプライベートIPアドレスを設定したため、接続不能になっています。Aレコードにエイリアスとして新しく作成したロードバランサーを指定します。
- D. レコードで認識できないプライベートIPアドレスを設定したため、接続不能になっています。CNAMEレコードにロードバランサーのARNを指定します。

- 17 現在設計中のEC2上で実行されるアプリケーションは、オンプレミスネットワーク内にあるサーバへファイルをSFTPで転送する要件があります。オンプレミスネットワークのサーバへ接続するために必要な設定で正しいものを選択してください。

- A. プライベートホストゾーンを設定します。ゾーンレコードにオンプレミスネットワークで使われるドメインを設定し、Aレコードに転送対象のサーバのIPアドレスを設定します。
- B. プライベートホストゾーンを設定します。ゾーンレコードにオンプレミスネットワークで使われるドメインを設定し、CNAMEレコードに転送対象のサーバのIPアドレスを設定します。
- C. Route 53 Resolverを使ってアウトバウンドエンドポイントを定義します。EC2のVPCおよびセキュリティグループを設定し、オンプレミスネットワークで使われるドメイン名および転送対象サーバのIPアドレスとポートを設定します。
- D. Route 53 Resolverを使ってアウトバウンドエンドポイントを定義します。EC2のVPCおよびセキュリティグループを設定し、オンプレミスネットワークで使われるドメイン名およびオンプレミスネットワークにあるDNSサーバのIPアドレスとポートを設定します。

- 18 さまざまなリージョンでALBによりロードバランスされているEC2上のアプリケーションへのルーティングを、ある開発者がRoute 53のトラフィックフローを用いて設定しようとしています。ルーティングの要件として、リージョン障害発生時に異なるリージョンのS3に配置した静的リソースsorryページに転送し、それ以外はクライアントの位置情報にもとづいて、最も近接するAWSリージョンのエンドポイントヘルーティングしなければなりません。トラフィックフローで設定すべき最も適切なものを選択してください。

- A. 最初に地理的近接性ルーティングの設定を行います。次に各エンドポイントの設定でフェイルオーバールーティングを設定し、プライマリでALBのドメインを、セカンダリで異なるリージョンのS3バケットのURLを設定します。
 - B. 最初にフェイルオーバールーティングを設定し、プライマリで、位置情報ルーティングの設定を行います。セカンダリに、sorryページが配置されている異なるリージョンのS3バケットのURLを設定します。
 - C. 最初にフェイルオーバールーティングを設定し、プライマリで、地理的近接性ルーティングの設定を行います。セカンダリに、sorryページが配置されている異なるリージョンのS3バケットのURLを設定します。
 - D. 最初に位置情報ルーティングの設定を行います。次に各ロケーションの設定でフェイルオーバールーティングを設定し、プライマリでALBのドメインを、セカンダリで異なるリージョンのS3バケットのURLを設定します。

- 19 ある掲示板機能を提供するWebアプリケーションでは、メッセージがポストされると該当機能を使用しているユーザにプッシュする機能がWebSocket(Socket.IO)を使って実装されています。このアプリケーションでALBを介して通信する場合に必要な設定のうち、正しいものを選択してください。

- A. ALBのWebSocket設定を有効化します。
 - B. ALBのスティックィセッションを有効化します。
 - C. ALBでSSL/TLSターミネーションを行うよう構成し、アプリケーションとの暗号化通信を確立します。
 - D. ALBのHTTPヘッダベースルーティングを使って、WebSocketプロトコルヘッダがある通信のルーティング設定を行います。

- 20 あるアプリケーション開発プロジェクトの機能要件として、クライアント証明書によるユーザ認証が求められています。ELBを使用してこの要件を満たす場合の構成で正しいものを選択してください。

- A. ALBでSSLターミネーションを構成します。クライアント証明書のルート証明書をALBに設定し、リスナーポート443からターゲットグループへポート80でフォワードします。
 - B. ALBでSSLターミネーションを構成します。クライアント証明書のルート証明書をALBに設定し、リスナーポート443からターゲットグループへポート443でフォワードします。
 - C. ALBでSSLパススルー通信を構成します。リスナーポート443からターゲットグループへポート443でフォワードします。
 - D. NLBでSSLパススルー通信を構成します。リスナーポート443からターゲットグループへポート443でフォワードします。

- 21 ある開発者がECSを使って、8080番ポートを使用するアプリケーションサーバ用のコンテナを実行しようとしています。ECSサービスへポート番号を動的に割り当てるときに、ECSタスク定義で設定するポートマッピング定義で正しいものを選択してください。

- A. ホストポート: ブランクで設定、コンテナポート: 8080
 - B. ホストポート: 0、コンテナポート: 8080
 - C. ホストポート: 8080、コンテナポート: ブランクで設定
 - D. ホストポート: 8080、コンテナポート: 0

22 ある開発者がECSを使ってコンテナアプリケーションをタスク定義してサービス実行しました。サービスの状態を確認したところ、定期的にサービスが再起動しており、アプリケーションへアクセスできません。ローカル環境でコンテナイメージを実行したところ正常にアプリケーションが起動できることを確認しています。考えられる原因として正しいものを2つ選択してください。

- A. タスク定義でイメージのURL設定が誤っており、ECSエージェントがレジストリからのコンテナイメージの取得に失敗しています。
- B. タスク定義でヘルスチェックパスの設定が誤っており、サービス起動した後ヘルスチェックに失敗しています。
- C. タスク定義で割り当てたメモリが小さいため、サービス起動に時間がかかりヘルスチェックに失敗しています。
- D. タスクロールとタスク実行ロール設定のうち、タスク実行ロールにAmazonECSTaskExecutionRolePolicyが設定されていません。
- E. タスクロールとタスク実行ロール設定のうち、タスクロールにAmazonECSTaskExecutionRolePolicyが設定されていません。

23 ある企業では、ECSで実行されるコンテナアプリケーションをステージング環境で試験したのち、プロダクション環境へリリースしようとしています。最も効率的に実現可能なECS定義やアプリケーション構成で誤っているものを選択してください。

- A. アプリケーションをDockerコンテナイメージにビルドする際、設定した環境変数に応じてパラメータを入れ替えるようにアプリケーションを構成します。
- B. ステージング環境でテストが完了したコンテナを、プロダクション環境で実行可能なようにビルドします。
- C. Systems Manager Parameter Storeを使用して、タスク定義にパラメータを設定します。
- D. プロダクション環境にリリースする際に、Systems Manager Parameter Store設定を更新して、プロダクション向けのパラメータを設定します。

24 ある若手エンジニアから、下記のタスク配置戦略の意味を確認されました。正しい説明を選択してください。

```
"placementStrategy": [ { "field": "attribute:ecs.availability-zone", "type": "spread" }, { "field": "instanceId", "type": "spread" } ]
```

- A. アベイラビリティゾーン全体にタスクを均等に分散し、各アベイラビリティゾーン内のインスタンス全体にタスクを均等に分散します。
- B. アベイラビリティゾーン全体にタスクを均等に分散し、各アベイラビリティゾーン内の固有のインスタンスに集中してタスク配置します。
- C. アベイラビリティゾーン内でタスクを均等に分散し、各アベイラビリティゾーン内の個別のインスタンス間でタスクを均等に分散します。
- D. アベイラビリティゾーン内でタスクを均等に分散し、各アベイラビリティゾーン内のインスタンス全体でタスクを均等に分散します。

25 ある開発者は、ECSでバッチ処理を並列実行したいと考えています。このバッチ処理では、複数のコンテナが共有ストレージでデータを共有し、できるだけレイテンシを抑えて実行したいと考えています。開発者が検討すべきECSタスク定義やサービス実行のうち、高速化に寄与しないと考えられるものを選択してください。

- A. タスク定義のボリューム・ストレージ設定で、ボリュームタイプとしてEFSを使用します。
- B. タスク定義のボリューム・ストレージ設定で、ボリュームタイプとしてDockerボリュームを使用し、単一のタスク定義で実行する複数のコンテナ定義を行います。
- C. タスク配置戦略として、binpackを採用します。
- D. タスク配置制約として、memberOf条件を使用し、特定のアベイラビリティゾーンでインスタンスをグループ化します。

26 あるチケット販売サービスを提供するWebサイトはRDSを使用しており、急激に増加するトラフィックの対処に悩まされています。そこでピークの発生が予想される時間帯にリクエストをオフロードする方法を検討しています。最も低成本で対応可能な方法を選択してください。

- A. RDSのインスタンスタイプをハイエンドなCPU/RAMに変更します。
- B. RDSのマルチアベイラビリティゾーン配置とし、スタンバイDBにリクエストをオフロードします。
- C. ElastiCacheクラスタを作成し、マスターデータの読み込みトラフィックをキャッシュへオフロードします。
- D. RDSのリードレプリカ機能を使用して、リクエストをオフロードします。

27 欧州、北米、アフリカ、アジア、日本といったリージョンで、さまざまな商品を取り扱うグローバルなリテールサービス企業は1日の販売に関するトランザクションデータをそれぞれのリージョンのRDSに保存しています。各リージョンの販売データを、本社があるリージョンのETLツールで集計しようと考えています。このワークフローを最も効率的に構築できる方法を選択してください。

- A. S3に保存されている、RDSのバックアップスナップショットをクロスリージョンコピーし、本社のリージョンのRDSへロードします。
- B. 本社のあるリージョンから、VPN接続で各リージョンのRDSのデータをロードします。
- C. RDSのクロスリージョンレプリカを、本社のあるリージョンに作成し、各々ETLツールで参照します。
- D. 各リージョンのRDSにあるデータを一元的に集約できるクロスリージョンデータアグリゲーション機能を使って、本社リージョンのRDSにデータを集め、ETLツールで参照します。

28 ある企業はマイクロサービスアーキテクチャを採用し、スケーラビリティに優れたアプリケーションを構成しようとしています。次のデータベース構成のうち、最も効率的に多数のリクエストを処理できると考えられる構成を選択してください。

- A. 読み取りトラフィックが中心のサービスと書き込みトラフィックが中心のサービスに分け、前者をRDSのリードレプリカエンドポイント、後者をDBエンドポイントを参照するよう構成します。
- B. 読み取りトラフィックが中心のサービスと書き込みトラフィックが中心のサービスに分け、前者をDynamoDBのリードレプリカエンドポイント、後者をDynamoDBエンドポイントを参照するよう構成します。
- C. 読み取りトラフィックが中心のサービスと書き込みトラフィックが中心のサービスに分け、前者をAuroraのクラスタエンドポイント、後者をカスタムエンドポイントを参照するよう構成します。
- D. 読み取りトラフィックが中心のサービスと書き込みトラフィックが中心のサービスに分け、前者をElastiCache(Redis)の読み取りエンドポイント、後者を設定エンドポイントを参照するよう構成します。

29 あるアプリケーションで、S3に保存したオブジェクトにアクセスする処理があります。このオブジェクトは、作成された直後は高頻度でアクセスされますが、その後時間の経過とともにアクセス頻度は少なくなります。コスト最適化の観点からオブジェクトへ適用するストレージクラス設定のうち最も適切なものを選択してください。

- A. オブジェクトは「S3 標準」で保存します。バケットに対して「ライフサイクルルール」で「S3 標準IA」へストレージクラスを移行するように設定を行います。
- B. オブジェクトは「S3 標準」で保存します。バケットに対して「ライフサイクルルール」で「S3 1ゾーンIA」へストレージクラスを移行するように設定を行います。
- C. オブジェクトは「S3 標準」で保存します。バケットに対して「ライフサイクルルール」で「S3 RRS」へストレージクラスを移行するように設定を行います。
- D. オブジェクトを「S3 Intelligent-Tiering」で保存します。

30 ある組織では、開発リソースの共有のために、アカウントAのS3バケットに、アカウントBからアクセスしようとしています。アカウントに必要な設定のうち適切なものを選択してください。

- A. アカウントAのS3バケットに対し、アカウントBのユーザのアクセスを許可するバケットポリシーを設定します。
- B. アカウントBでIAMグループを作成し、アカウントAのS3バケットへアクセスできるポリシーを設定して、このグループにユーザを追加します。
- C. アカウントAのS3バケットに対し、アカウントBのユーザのアクセスを許可するバケットポリシーを設定します。アカウントAでクロスアカウントIAMロールを作成します。
- D. アカウントAのS3バケットに対し、アカウントBのユーザのアクセスを許可するバケットポリシーを設定します。アカウントBでIAMグループを作成し、アカウントAのS3バケットへアクセスできるポリシーを設定して、このグループにユーザを追加します。

31 ある開発者が静的WebサイトをS3を使って構築しようとしています。index.htmlやCSSなどのリソースをバケットのルートに配置し、パブリックアクセスを許可しようとしています。次の選択肢のうち、設定が正しいものを選択してください。

- A. バケットポリシーで以下のポリシー設定を付与します。

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "SamplePublicRead",  
      "Effect": "Allow",  
      "Principal": "*",  
      "Action": ["s3:*"],  
      "Resource": ["arn:aws:s3:::sample-static-website-bucket/*"]  
    }  
  ]  
}
```

2

- B. ACLで「オブジェクトの所有者」に対し「FULL_CONTROL」アクセス許可を付与し、「Authenticated Usersグループ」と「All Usersグループ」に対し「READ」アクセス許可を付与します。
- C. ACLで「オブジェクトの所有者」と「Authenticated Usersグループ」に対し「FULL_CONTROL」アクセス許可を付与し、「All Usersグループ」に対し「READ」アクセス許可を付与します。
- D. アクセスポイントポリシーで以下の設定を行います。

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "SamplePublicRead",  
      "Effect": "Allow",  
      "Principal": "*",  
      "Action": ["s3:GetObject"],  
      "Resource": ["arn:aws:s3:::sample-static-website-bucket/*"]  
    }  
  ]  
}
```

32 あるマッチングアプリケーションの開発元企業は、ユーザの本人確認で利用した機密データをS3に暗号化して保存するようにしたいと考えています。すべてのデータが暗号化されていることを担保するために必要な設定で正しいものを選択してください。

- A. S3バケットの暗号化設定でSSE-S3を選択します。このオプションではオブジェクトを保存する際に暗号化が強制されます。
- B. S3バケットの暗号化設定でSSE-KMSを選択します。このオプションではオブジェクトを保存する際に暗号化が強制されます。
- C. S3バケットの暗号化設定でSSE-KMSを選択します。データを保存するPutObjectリクエストに対し、「x-amz-acl」が「public-read」、「public-read-write」、「authenticated-read」であった場合、Denyとする条件を持つバケットポリシーを設定します。
- D. S3バケットの暗号化設定でSSE-S3を選択します。データを保存するPutObjectリクエストに対し、「x-amz-server-side-encryption」が含まれていない場合、Denyとする条件を持つバケットポリシーを設定します。

33 あるモバイル向けWebアプリケーションを開発する企業は静的コンテンツをS3に保存してホストしています。アプリケーションの一機能である写真・動画共有では限られたユーザにのみコンテンツにアクセスさせたいと考えています。この要求を実現するために必要な設定のうち、正しいものを選択してください。

- A. S3に保存するコンテンツはプライベートACLで保存しておきます。S3のバケットに対してCORSの設定を行い、一時的にアクセス可能な署名付きURLをバックエンドサーバから取得し、HTML要素のimageタグに設定します。
- B. S3に保存するコンテンツはプライベートACLで保存しておきます。写真や動画を表示させる際に、一時的にアクセス可能な署名付きURLをバックエンドサーバで生成するため、S3で署名付きURLの設定を有効化します。
- C. S3に保存するコンテンツはプライベートACLで保存しておきます。写真や動画を表示させる際に、一時的にアクセス可能な署名付きURLをバックエンドサーバから取得するため、バックエンドサーバに対してCORSの設定を行います。
- D. S3に保存するコンテンツはプライベートACLで保存しておきます。一時的にアクセス可能な署名付きURLをバックエンドサーバから取得し、HTML要素のimageタグに設定します。

34 ある移動体通信事業者は、利用者の位置情報を収集するためにKinesis Data Streamsを利用することを検討しています。Kinesis Data Streamsを使って構成する場合に、誤っている記述を含むものを選択してください。

- A. プロデューサーとしてAWS IoTを用いて、MQTTプロトコルで収集した位置情報データをKinesis Data Streamsへ送信します。
- B. モバイル端末にインストールされたアプリケーション内でSDKを使って、Kinesis Data Streamsへデータ送信用のAPIを直接呼び出します。
- C. Kinesis Data Streamsでストリームデータを一時的に蓄積する際に、Lambda関数を使って個人情報をマスク化する処理を加えます。
- D. Kinesis Data Streamsで収集した位置情報を処理するために、Kinesis Client Libraryを使ってアプリケーションを作成する場合は、Javaランタイムがインストールされた環境が必要です。

35 ある製造業企業ではKinesis Data Streamsを使って、工場からのセンサーデータを10シャード・5台のECインスタンスで処理しています。設備増強に伴い、センサーからのトラフィックが増加するため、開発者はデータストリームのシャードを10から30に増加させたいと考えています。併せてコンシューマー側の処理スループットが最大化するようにEC2インスタンスの台数を最適化してリシャーディングする場合、追加するインスタンス数は何台でしょうか。

- A. 10台
- B. 15台
- C. 25台
- D. 30台

36 ある製造業企業ではKinesisを使って、工場から収集したセンサーデータを、ETL処理するためにRedshiftに保存しようと考えています。最も低コストに実装できるオプションを選択してください。

- A. Kinesis Agentを用いてKinesis Data Firehoseにデータを送信し、S3に一度保存してRedshiftへデータをロードします。
- B. Kinesis Agentを用いてKinesis Data Streamsにデータを送信し、Kinesis Data Firehoseを経由してRedshiftへ直接データ保存します。
- C. Kinesis Agentを用いてKinesis Data Streamsにデータを送信し、Lambda関数を使ってRedshiftへデータ保存します。
- D. Kinesis Agentを用いてKinesis Data Firehoseにデータを送信し、Lambda関数を使ってRedshiftへデータ保存します。

37 ある移動体通信事業者は、Kinesis Data Streamsを使って収集した利用者の位置情報を解析して、特定エリアにいる利用者数を統計情報として算出しようとを考えています。最も低コストに実装できるオプションを選択してください。

- A. Kinesis Data Analyticsを使って、位置情報を入力ストリームとして取り込み、統計情報を抽出するSQLを設定します。
- B. Kinesis Data Analyticsを使って、位置情報を入力ストリームとして取り込み、統計情報を抽出するロジックをApache Flinkで実装して設定します。
- C. Kinesis Data Analyticsを使って、位置情報を入力ストリームとして取り込み、統計情報を抽出するロジックをLambda関数で実装してストリーミングディスティネーションに設定します。
- D. Kinesis Data Analyticsを使って、位置情報を入力ストリームとして取り込み、統計情報を抽出するロジックをLambda関数で実装して前処理として設定します。

38 ショッピングセンターを展開する企業は、店舗に入店した顧客の顔画像をカメラで検出し、あらかじめ顧客が欲しいものリストとして設定したデータに合致する商品情報があれば、通知したいと考えています。要件を実現するために開発者が実装する方法を選択してください。

- A. Kinesis Video Streamsを用いて、メディア形式で収集した顧客の顔画像を含む動画データから、Producer LibraryおよびRekognition Videoを活用して検出、顧客のIDを特定し、欲しいものリストに合致する商品データをその店舗で取り扱っているならば、SNSで通知するようコンシューマーアプリケーションを実装します。
- B. Kinesis Video Streamsを用いて、WebRTC形式で収集した顧客の顔画像を含む動画データから、Producer LibraryおよびRekognition Videoを活用して検出、顧客のIDを特定し、欲しいものリストに合致する商品データをその店舗で取り扱っているならば、SNSで通知するようコンシューマーアプリケーションを実装します。
- C. Kinesis Video Streamsを用いて、メディア形式で収集した顧客の顔画像を含む動画データから、Parser LibraryおよびRekognition Videoを活用して検出、顧客のIDを特定し、欲しいものリストに合致する商品データをその店舗で取り扱っているならば、SNSで通知するようコンシューマーアプリケーションを実装します。
- D. Kinesis Video Streamsを用いて、WebRTC形式で収集した顧客の顔画像を含む動画データから、Parser LibraryおよびRekognition Videoを活用して検出、顧客のIDを特定し、欲しいものリストに合致する商品データをその店舗で取り扱っているならば、SNSで通知するようコンシューマーアプリケーションを実装します。

解 答**1 D、E**

→ 問題：168 ページ、本文説明：51-52 ページ

- 選択肢A：REST APIはJWTオーソライザを構成できません。CognitoにソーシャルIDプロバイダを追加してCognitoオーソライザで構成するか、HTTP APIでJWTオーソライザを構成するオプションがあります。
- 選択肢B：REST APIはJWTオーソライザを構成できません。CognitoにソーシャルIDプロバイダを追加してCognitoオーソライザで構成するか、HTTP APIでJWTオーソライザを構成するオプションがあります。また、REST APIではALBに直接中継することはできません。NLBを配置します。
- 選択肢C：Cognitoオーソライザを使用する構成はとることができますが、REST APIではALBに直接中継することはできません。NLBを配置します。
- 選択肢D：HTTP APIはCognitoオーソライザを構成することができます。またバックエンドにLambdaを配置する構成をとることも可能です。
- 選択肢E：HTTP APIはCognitoオーソライザを構成することができます。またバックエンドにALBを配置する構成をとることも可能です。

2 A

→ 問題：169 ページ、本文説明：46-47 ページ

- 選択肢A：オンプレミスからプライベートAPIエンドポイントへ接続するには、オンプレミス環境内のDNSサーバにRoute 53 Resolverのフォワーダを追加するか、Route 53エイリアスレコードを定義して、VPNやDirect Connectで接続されたVPC内にあるインターフェースVPCエンドポイントを経由してアクセスできます。エンドポイントのアクセス制御はリソースポリシーを使用して、アカウントやIPアドレスレベルでアクセス制御することができます。
- 選択肢B：オンプレミスからAPI Gatewayへアクセスすることはできますが、プライベートなネットワークからのアクセスのみを許可するには、API Gatewayのリソースポリシーを使用する必要があります。この選択肢の構成では、他のVPCからのAPI Gatewayへのアクセスを制御することはできません。
- 選択肢C：プライベートホストゾーンを設定しても、オンプレミスからAPI Gatewayヘルーティングすることはできません。
- 選択肢D：プライベートホストゾーンを設定しても、オンプレミスからAPI Gatewayヘルーティングすることはできません。また、この選択肢の構成では、他のVPCからのAPI Gatewayへのアクセスを制御することはできません。

3 D

→ 問題：170 ページ、本文説明：48 ページ

- 選択肢A：API GatewayはOpenAPI (Swagger) v2.0/3.0に準拠した定義ファイルのインポートや既存APIのクローン、そして、構築したAPI定義をエクスポートすることもできます。
- 選択肢B：API Gatewayではステージごとにスロットリングやキャッシュ、Web ACL、Canary、ステージ変数、ログ/トレースを設定することができます。
- 選択肢C：APIキーはリクエストと使用プランを紐付けるために使用します。
- 選択肢D：マッピングテンプレートは統合リクエスト・レスポンス設定で使用します。

4 C

→ 問題：170 ページ、本文説明：54 ページ

- 選択肢A：アクセキーはIAMユーザに対して使用されるもので、Cognitoで認証されたユーザに対して使用されるものではありません。
- 選択肢B：API Gatewayでは、Lambdaオーソライザを使用して、BearerトークンもしくはHTTPリクエストヘッダのパラメータに認証情報を付与したリクエストを送信して検証できますが、検証で返却すべきものはアクセストークンではなくアクセスポリシーのJSON表現です。
- 選択肢C：CognitoのHostedUIなどで認証を行った後、発行されるJWT形式のアクセストークンをHTTPリクエストヘッダ(デフォルトではAuthorizationヘッダ)に付与して送信します。API GatewayはCognitoに対し、アクセストークンの検証を行った後、検証が問題なければ、AWSリソースへアクセスさせます。なお、その際、転送されるリクエストのAuthorizationヘッダにアクセストークンがセットされます。
- 選択肢D：サードパーティのOIDCプロバイダでユーザが認証を行った後、発行されるJWT形式のアクセストークンをHTTPリクエストヘッダ(デフォルトではAuthorizationヘッダ)に付与して送信します。API GatewayはLambdaオーソライザではなく、JWTオーソライザでトークン検証を行うほうがより省力的な実現方法です。

2