

5 A

➡ 問題：171 ページ、本文説明：57 ページ

- 選択肢A：任意のステージに「Canary」を定義することができます。ここでは、すでにカットオーバーしたプロダクションステージに定義し、リクエストを配分して、状況を確認しながら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。
- 選択肢B：新たにステージを作成してしまうと、既存のものからリクエストを振り分けて構成することができません。
- 選択肢C：新たにAPIを作成してしまうと、既存のものからリクエストを振り分けて構成することができません。
- 選択肢D：選択肢Bと同様に、新たにステージを作成すると、既存のものからリクエストを振り分けて構成できません。

6 C

➡ 問題：171 ページ、本文説明：61 ページ

この設問のポイントはRDS Proxyを使ったRDSへの接続に必要な設定と、LambdaがVPCに接続した際の、インターネットを通じたサードパーティサービスへのアクセス設定方法を理解しているかです。RDS Proxyの接続は、プロキシの識別子、アクセス対象のインスタンスといった設定情報に加え、プロキシがRDSにアクセスするための認証情報をSecrets Managerに保存しておく必要があります。プロキシを通じたRDSへのアクセスはIAM認証および、RDSデータベース認証情報を使った2種類の方法がサポートされています。前者を使ってアクセスする場合、実行するLambda関数の権限に「rds-db:connect」パーミッションも必要です。また、VPCにLambda関数が接続する場合、インターネットアクセスするにはNAT Gatewayの設置が必要になります。誤った選択肢であるCでは「rds-db:connect」パーミッションが足りておらず、他の選択肢はすべて必要な設定です。

7 B

➡ 問題：172 ページ、本文説明：70 ページ

Lambda関数をスケジュール実行する最も簡易的な方法はEventBridge(CloudWatch Events)を使用して、cron式もしくはrate式で設定する方法です。他の選択肢の誤っている理由は以下のとおりです。

- 選択肢A：通常カスタムコンテナイメージの/etc/crontabを設定してもプロセスが実行されないため、この方法ではLambda関数を起動できません。
- 選択肢C：スケジュール起動するための環境変数はサポートされていません。
- 選択肢D：これを実現することはできますが、選択肢Bの方法を使ったほうがより簡易的に実装することができます。

8 D

➡ 問題：172 ページ、本文説明：67-69 ページ

API Gatewayを使ってLambda関数を同期的に呼び出した場合、管理者へ通知するための手法を問う問題です。この場合、CloudWatch Logsのサブスクリプションで在庫不足のエラーを検知し、Lambda関数で通知を処理するのが最も効率的に実装できる手法です。他の選択肢が誤っている理由は以下のとおりです。

- 選択肢A、B：デッドレターキューはLambda関数が非同期的に呼び出された場合に利用できる機能です。
- 選択肢C：サブスクリプションフィルタで検知したログは、一度Lambda関数で処理する必要があります。直接SNSで通知することはできません。

9 D

➡ 問題：173 ページ、本文説明：67、70 ページ

イベントソースマッピングは、Lambda関数がイベントをポーリングして検知する場合に設定するものであり、S3のようにイベントドリブンでLambda関数を起動するサービスはトリガーとしてLambda関数に設定します。S3から発生したイベントをすべてLambdaが実行しているか、およびサムネイル画像作成処理が正常終了したかの判断処理は、デッドレターキューによる検知だけでは、すべての処理が正常完了したことの担保にはなりません。よって、「送信先」の設定で、「On failure」、「On success」条件両方で、サムネイル画像作成処理が正常終了したかを含め、通知する必要があります。

10 B

➡ 問題：173 ページ、本文説明：60-61 ページ

Lambda関数では、各リージョンごとに同時実行数の上限が設定されています。東京リージョンでは最大1,000であり、この設問では、ピーク時に平均2,000リクエスト、1リクエストあたり2秒を要するので、最大4,000の同時実行数が求められます。よって、まずAWSのサポートに緩和申請を行う必要があります。その他の選択肢の不正解理由は以下のとおりです。

- 選択肢A：SDKを使ったAPI呼び出しでは、SDKの中でエクスプロンシャルバックオフが実装されているので個別に実装する必要はありません。
- 選択肢C：Lambda関数のCPUだけを拡張することはできません。メモリの拡張に伴ってCPUも拡張されます。
- 選択肢D：ウォームスタートをAWSサポートへリクエストすることはできません。

2

11 A

→ 問題：174 ページ、本文説明：74-75 ページ

- 選択肢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、ユーザの位置情報を時系列で保存することができます。特定のロケーションからユーザをトレースできるようにグローバルセカンダリインデックスを構成しますが、ロケーションを一意に識別するコードでパーティションキーとして、タイムスタンプをソートキーとすることで、場所を指定し、時間帯でフィルタしてデータを抽出することができます。
- 選択肢B：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は、テーブルのパーティションキーと同じものになるため誤りです。
- 選択肢C：このテーブルのキー構成では、ロケーションのコードとユーザIDがプライマリキーになるので、ユーザが特定のロケーションにいるときのデータを1件しか保存することができません。
- 選択肢D：このテーブルのキー構成では、プライマリキーとしてロケーションのコードとユーザIDになるので、ユーザが特定のロケーションにいるときのデータを1件しか保存することができません。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も、テーブルのパーティションキーと同じものになるため誤りです。

12 C

→ 問題：174 ページ、本文説明：77 ページ

読み込みに必要なキャパシティユニットは、平均7KBのデータサイズのため、1件あたり2RCUs、1秒間に平均30回強い一貫性読み込みが発生するため60RCUsが必要です(結果整合性読み込みだと半分の30RCUsで済みます)。また、書き込みは、1件あたり7WCUs、1秒間に平均10回トランザクション書き込みが発生するため、70WCUsの倍の140WCUsが必要になります。その他の選択肢の不正解理由は以下のとおりです。

- 選択肢A：RCUsの計算が誤っています。
- 選択肢B：RCUsとWCUs双方の計算が誤っています。WCUsは標準書き込みであれば70WCUsです。
- 選択肢D：RCUsの計算は正しいですが、WCUsの計算が誤っています。WCUsは標準書き込みであれば70WCUsです。

13 C

→ 問題：175 ページ、本文説明：82 ページ

- 選択肢A：ElastiCacheはデータベースがRDBの場合に有効なキャッシングリューションです。ここでは、多数の書き込みトランザクションが発生して、低レイテンシな状況を実現するために、RDBを使用する構成ではハイスペックな環境が必要になります。
- 選択肢B：グローバルセカンダリインデックスはDynamoDBで、プライマリキー以外の属性で検索するために使用されるオプションであり、高速化のためのものではありません。
- 選択肢C：コスト面を踏まえ、多数の書き込みトランザクションに対応し、キャッシングを用いて低レイテンシな要求に対応するアーキテクチャを持つのはDAXです。
- 選択肢D：キャパシティユニットを最適化しても、低レイテンシな要求を満たせるとは限りません。キャパシティユニットの最適化はリクエスト数に応じたパフォーマンスを担保するために実施するものです。

14 D

→ 問題：175 ページ、本文説明：78、81 ページ

- 選択肢A：DeleteItem APIは書き込みキャパシティを消費するため、テーブルごと削除する場合に比べてコストが必要になります。
- 選択肢B：TTLを有効化することで、指定した時間後にコストを要せずデータを削除することは可能ですが、データ削除する指定時間を計算して実装するコストが発生します。
- 選択肢C：BatchWriteItem APIは書き込みキャパシティを消費するため、テーブルごと削除する場合に比べてコストが必要になります。
- 選択肢D：計算処理後にテーブルごと削除するのが最も費用対効果が高くシンプルな方法です。

15 C

→ 問題：175 ページ、本文説明：83 ページ

- 選択肢A：CloudWatch Eventでは項目の変更がイベントとしてサポートされていません。
- 選択肢B：この方法を実装することはできますが、更新処理のログ出力やサブスクリプションの設定など実装コストが追加で発生します。
- 選択肢C：DynamoDB Streamsを有効化し、DynamoDB Trigger機能でLambda関数を実行して通知を送信する方法が最もシンプルな手法です。
- 選択肢D：DynamoDB TriggerはLambda関数で変更内容をポーリングし、イベント起動を行うことができます。Streamsを有効化して、変更データを記録するだけでなく、通知するにはLambda関数で実装する必要があります。

16**C**

➡ 問題：176 ページ、本文説明：96-97 ページ

パブリックホストゾーンにはゾーンの起点となるSOAレコードと、親ゾーンに含まれるものと同じNSレコード、およびルーティング対象となるAWSサービスを示したAレコード(IPv4アドレス)もしくはAAAAレコード(IPv6アドレス)を含める必要があります。この設定では、パブリックホストゾーンで認識できないプライベートネットワークのIPアドレスが指定されたことからルーティングされなくなったものと想定されます。したがって、Aレコードに新しく作成されたロードバランサーのドメインを指定すればよいですが、Aレコードには、エイリアスコードというRoute 53の拡張機能レコードで、ELBやCloudFront、Amazon S3バケットといったAWSリソースにトラフィックをルーティングするレコードを設定できます。

17**D**

➡ 問題：177 ページ、本文説明：102 ページ

VPCからオンプレミスネットワークへのルーティングを設定する場合は、Route 53でアウトバウンドエンドポイントを設定し、アクセス元のVPCおよびセキュリティグループ、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指定します。プライベートホストゾーンはAWSネットワーク内の異なるドメインへの名前解決に利用されるものであり、オンプレミスネットワーク内のリソース情報を設定してもルーティングすることはできません。

18**A**

➡ 問題：178 ページ、本文説明：98 ページ

トラフィックフローを用いて、さまざまなルーティングポリシーを組み合わせて設定できます。問題の要件では、特定のリージョンへ位置情報にもとづいてルーティングし、かつそのリージョンに障害が発生した場合、異なるリージョンのS3バケットにあるsorryページヘルーティングしなければならないため、リージョン単位でエンドポイントを設定する「地理的近接性」ルーティングを設定し、各リージョンごとにフェイルオーバールーティングで異なるリージョンのS3バケットへのルーティングを設定するのが正しい設定です。なお、位置情報ルーティングは、クライアントの位置情報にもとづいてルーティングする国や地域を指定するルールのため、リージョンを詳細に指定する設定ではありません。

19**B**

➡ 問題：178 ページ、本文説明：107 ページ

Socket.IOを使った通信はサーバとクライアントのWebSocket通信を確立して、サーバからクライアントへプッシュ送信します。したがって、クライアントから常に同じサーバと通信する必要があり、スティッキーセッションを有効化して、同じクライアントからのリクエストを常に同じサーバへ振り分けるよう構成する必要があります。その他の選択肢の不正解理由は、以下のとおりです。

- ・選択肢A：WebSocketを有効化するような設定はありません。ALBではデフォルトでWebSocket通信がサポートされています。
- ・選択肢C：暗号化通信の設定とWebSocket通信の設定は関連性がありません。
- ・選択肢D：WebSocket通信には固有のHTTPヘッダが含まれていますが、これを使ってルーティングしても、同一のサーバとのリクエスト通信が設定されるとは限りません。スティッキーセッションを有効化するのが正しい選択肢です。

20**D**

➡ 問題：179 ページ、本文説明：113 ページ

ELBではクライアント証明書のルート証明書をインストールして、クライアント認証を行う方法はサポートされていません。バックエンドのターゲットとなるサーバでクライアント証明書による認証確認処理を行う必要があります。ALBでは、アプリケーション層レベル(L7)でロードバランスするために、一度ロードバランサー内でTCPコネクションを終端して振り分けします。そのため、ALBでTCPトランスポートでそのままリクエストをフォワードすることはサポートされていません(HTTP/HTTPSのみです)。よって、NLBを使ってSSLパススルー通信を構成する方法が正しい選択肢です。

21**B**

➡ 問題：179 ページ、本文説明：121 ページ

ECSタスク定義でポートマッピングする場合、コンテナポートにはコンテナが使用するポート番号を、ホストポートにはホストOSがコンテナへ割り当てるポート番号を指定します。動的に割り当てる場合には0を設定します。

22 B、C

➡ 問題：180 ページ、本文説明：120 ページ

ECSで、サービスを実行した際に再起動が繰り返される場合のトラブルシューティングに関する設問です。ローカル環境でコンテナイメージを実行したところ正しく実行できているので、イメージ自体には問題ありません。サービスが再起動を繰り返しているということは、イメージの取得とコンテナイメージ実行までは正常にできています。したがって、原因として考えられるのはヘルスチェックなどでサービスが異常状態とみなされ、再起動を繰り返している可能性です。なお、タスク実行ロールにはAmazonECSTaskExecutionRolePolicyがアタッチされていることが必要ですが、これが設定されていない場合、そもそもコンテナの実行時にエラーが発生するため解答としては誤りです。

23 B

➡ 問題：180 ページ、本文説明：121 ページ

ステージング環境でテスト完了したコンテナをわざわざ、プロダクション環境に向けてビルドし直す必要はありません。そもそもステージング環境はプロダクション環境を想定したコンテナイメージでテストするためのものです。ステージング環境とプロダクション環境のパラメータの差異は、環境変数やSystems Manager Parameter Storeで吸収します。後者の設定では、タスク定義時にパラメータを設定しますが、実際はサービス実行時に、実行コンテナへ設定しているパラメータが注入されます。

24 A

➡ 問題：181 ページ、本文説明：123 ページ

設問のタスク配置戦略は、デフォルトで使用される「アベイラビリティゾーンバランススプレッド」です。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。

25 A

➡ 問題：181 ページ、本文説明：121、125 ページ

ECSで共有ストレージを使ってデータを共有しながら、複数のコンテナで処理を並列実行する場合、できるだけ近いロケーションにあるほうが、レイテンシを低く抑えながら実行できると考えられます。また、ボリュームを複数のコンテナで共有するにはボリュームタイプとして、Dockerボリュームで単一のタスク定義で複数のコンテナ定義を行うと、共通のストレージがマウントされるので高速化に寄与するはずです。したがって、タスク配置戦略としてはbinpackを採用したり、タスク配置制約で共通のアベイラビリティゾーンで実行し、共通のストレージをマウントするのが高速化に寄与すると考えられる選択肢です。EFSを使用しても高速化に寄与するとは考えられません。むしろネットワークIOでボトルネックが発生する可能性があります。

26 D

➡ 問題：182 ページ、本文説明：129 ページ

RDSでリクエストをオフロードする場合に有効な方法はリードレプリカを使い、読み取りリクエストを逃すことです。他の選択肢でもトラフィックの急増に対し、機能する方法もありますが、不正解理由は以下のとおりです。

- ・選択肢A：これはスケールアップによる方法であり、性能限界や高いコストが発生するため、水平スケールできるリードレプリカのほうがよりよい選択肢です。
- ・選択肢B：これは障害発生時にフェイルオーバーが可能なよう、異なるアベイラビリティゾーンにデータベースのスタンバイコピーを作成する方法です。パフォーマンス向上を目的に行われる方法ではありません。
- ・選択肢C：これは機能する方法ですが、アプリケーションがキャッシュを参照するよう改修が必要になります。エンドポイントを変更するだけで対処できるリードレプリカを使用するほうがよりよい選択肢です。

27 C

➡ 問題：182 ページ、本文説明：130 ページ

RDSではクロスリージョンレプリカ機能がサポートされており、災害対策やデータベースの参照目的で利用されます。その他の選択肢の不正解理由は、以下のとおりです。

- ・選択肢A：この方法は実現できることはありませんが、クロスリージョンレプリカよりロードする手間などがかかります。また、暗号化されたデータベースだと、リージョンをまたいで復号を行うことはできないため、現実的な選択肢ではありません。
- ・選択肢B：この方法は実現可能ですが、クロスリージョンレプリカを使用するほうがより効率的に実現できます。
- ・選択肢D：クロスリージョンデータアグリゲーション機能はRDSにはありません。

28 A

➡ 問題：183 ページ、本文説明：129 ページ

多数のリクエストを処理する場合のデータベース構成で最適なものを問う問題です。マイクロサービスアーキテクチャにおける有名なデザインパターンとして、サービスを読み取りと書き込みのトラフィックに分けるCQRS(Command Query Responsibility Separation: コマンド・クエリ責務分離)が知られています。分離された各サービスが適切なデータベースへアクセスするためのエンドポイントとして適切な選択肢は、Aのみです。その他の選択肢の不正解理由は、以下のとおりです。

- ・選択肢B: DynamoDBではリードレプリカがそもそも機能としてありません。キャッシュとしてDAXの利用は選択肢としてあり得ますが、DynamoDB自体に読み取りリクエストをオフロードする機能はありません。
- ・選択肢C: Auroraでは、リードレプリカ用のエンドポイントとして「読み取りエンドポイント」、書き込み用のエンドポイントとして「クラスタエンドポイント」が正しい組み合わせです。
- ・選択肢D: ElastiCache(Redis)は、クラスタモードが無効の場合、リードレプリカ用のエンドポイントとして「読み取りエンドポイント」、書き込み用のエンドポイントとして「プライマリエンドポイント」が提供されています。クラスタモードが有効な場合、「設定エンドポイント」のみが提供されています。選択肢の組み合わせはクラスタモードが混在しており、正しくありません。

29 D

➡ 問題：183 ページ、本文説明：136 ページ

S3 Intelligent-Tieringはアクセス頻度に応じて自動的にストレージタイプを変更するものです。このユースケースのようなアクセス頻度は高いが、後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ます。その他の選択肢の不正解理由は、以下のとおりです。

- ・選択肢A、B: S3標準IAや、S3 1ゾーンIAは低頻度アクセスストレージですが、取り出しに料金がかかります。ライフサイクルルールは、オブジェクト作成後の日数を指定してストレージクラスを変更するものであり、変更したのちアクセスが発生して追加料金が継続的に発生する可能性があるため、このユースケースではS3 Intelligent-Tieringがよりよい選択肢です。
- ・選択肢D: 低冗長化ストレージ(Reduced Redundancy Storage: RRS)もオプションとしてはありますが、低い冗長性でかつコストもS3標準より発生するため正しい選択肢ではありません。

30 D

➡ 問題：184 ページ、本文説明：140 ページ

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。

31 D

➡ 問題：185 ページ、本文説明：138-145 ページ

パブリックアクセスを許可する方法はいくつかあります。バケットポリシーでPrincipalをワイルドカードで指定し、匿名アクセスを許可する方法と、ACLでAll UsersグループにREADアクセス許可を与える方法、そしてアクセスポイントポリシーでパブリックアクセスをバケットポリシーと同様に許可する方法です。各選択肢ではそれぞれの方法を説明していますが、正解となるD以外は明確に誤りを含んでいます。

- ・選択肢A: アクションがs3:GetObject以外にもすべてのアクションを含んでいるため設定として正しくありません。
- ・選択肢B、C: ACLでパブリックアクセスを設定する場合は、「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、「All Usersグループ」に対し「READ」アクセス許可を付与すればよいです。

32 D ➔ 問題：186 ページ、本文説明：147-148 ページ

S3にはデフォルトバケット暗号化のオプションがありますが、SSE-S3、SSE-KMSいずれのオプションで有効化しても、すべてのオブジェクトが暗号化されているという保証はされません。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する必要があります^{※57}。なお、選択肢Cのオプションはパブリックアクセスが許可される設定のリクエストのため、誤りです。

33 D ➔ 問題：186 ページ、本文説明：146、149 ページ

S3に保存する写真・動画コンテンツはプライベートACLとして保存します。このオブジェクトには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内で実行されるSDKで生成してクライアントに渡すことで、アクセス制御を実現できます。静的コンテンツはS3にホストされているため、Webアプリケーションと写真・動画コンテンツのオリジンは同一となります。したがってCORSの設定も必要ありません。署名付きURLはSDKで生成でき、特にS3で設定を有効化する必要はなく、その設定もS3には存在しません。

34 C ➔ 問題：187 ページ、本文説明：156 ページ

Kinesis Data Streamsでは収集して一時保存したデータのコンシューマーとしてLambda関数を実行できます。一時蓄積する際にマスク化はできません。

35 C ➔ 問題：187 ページ、本文説明：158-159 ページ

ストリームのシャード内データを処理するためのKCLワーカーがEC2インスタンス上で実行されますが、最もスループットが最大化するのは1インスタンスにつき、1シャードを処理するワーカーが1つの場合です(1インスタンスで複数のワーカーを実行することもできますが、ワーカーとインスタンスが1:1となるのが最適です)。シャード数は30に増加したため、最適なインスタンス数は30台となり、元々あった5台の差分である25台が追加する必要があるインスタンス数です。

36 A ➔ 問題：188 ページ、本文説明：160 ページ

収集したストリームデータをRedshiftへ保存するためには、Kinesis Agentを使ってKinesis Data Firehoseへデータ送信するのが最も低成本で実装できるオプションです。なお、Redshiftに保存する場合は、一度S3に保存されてからその後、Redshiftにロードされます。その他の選択肢の不正解の理由は以下のとおりです。

- ・選択肢B：Data Streamsを経由せずともData FirehoseでRedshiftへ直接送信するようにAgentを構成したほうが低成本です。
- ・選択肢C：Lambda関数を実装するのはコストが発生します。
- ・選択肢D：Lambda関数を実装するのはコストが発生します。なお、前処理に設定することはできません。

37 A ➔ 問題：188 ページ、本文説明：161 ページ

Kinesis Data AnalyticsはSQLやApache Flinkなどを使った解析処理をData StreamsやFirehoseからの入力ストリームへ適用することができます。この問題のユースケースでは利用者の位置情報のレコード数をカウントすればよいと考えられるので、SQLを使って実装するのが最も簡易なオプションです。

38 C ➔ 問題：189 ページ、本文説明：163 ページ

Kinesis Video Streamsを使って、カメラなどのプロデューサーから動画ストリームデータを収集し、Parser LibraryとRekognition Videoを組み合わせて、顧客の顔画像を検出できます。検出した画像データから顧客のIDを特定し、欲しいもののリストと店舗の商品データを照らし合わせて、合致するものがいればSNSで通知するよう実装できます。

※57 https://docs.aws.amazon.com/ja_jp/AmazonS3/latest/userguide/default-bucket-encryption.html

第3章

セキュリティ関連サービス

- 3-1. AWS IAM (Identity and Access Management)
- 3-2. AWS STS (Security Token Service)
- 3-3. AWS KMS (Key Management System)
- 3-4. Amazon Cognito
- 3-5. その他のセキュリティサービス

3-1 AWS IAM (Identity and Access Management)

IAMはAWSのサービスやリソースへのアクセスを管理する認証・認可サービスです。ほぼすべてのAWSサービスのアクセス権限の管理は、IAMで行われます。

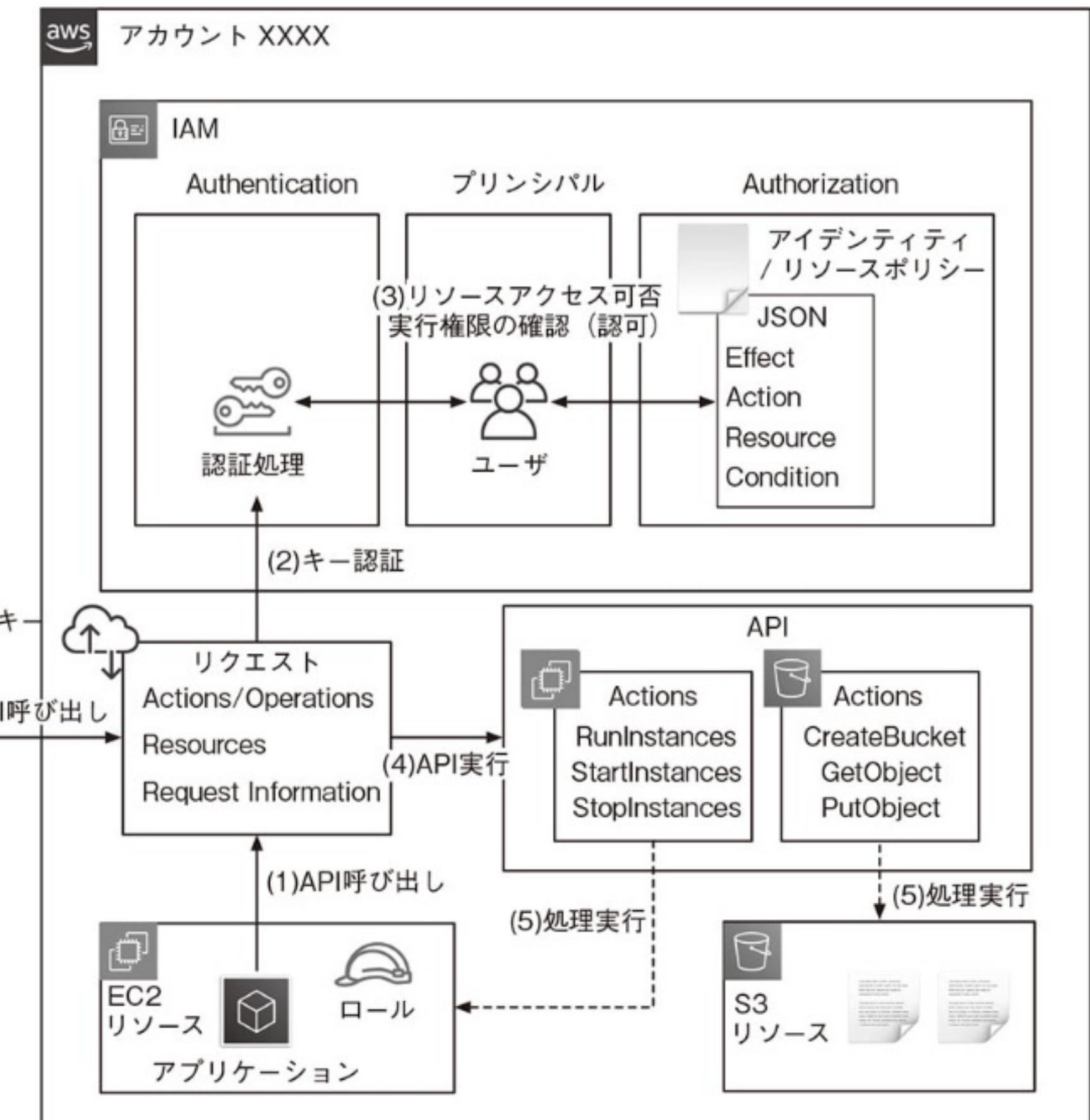
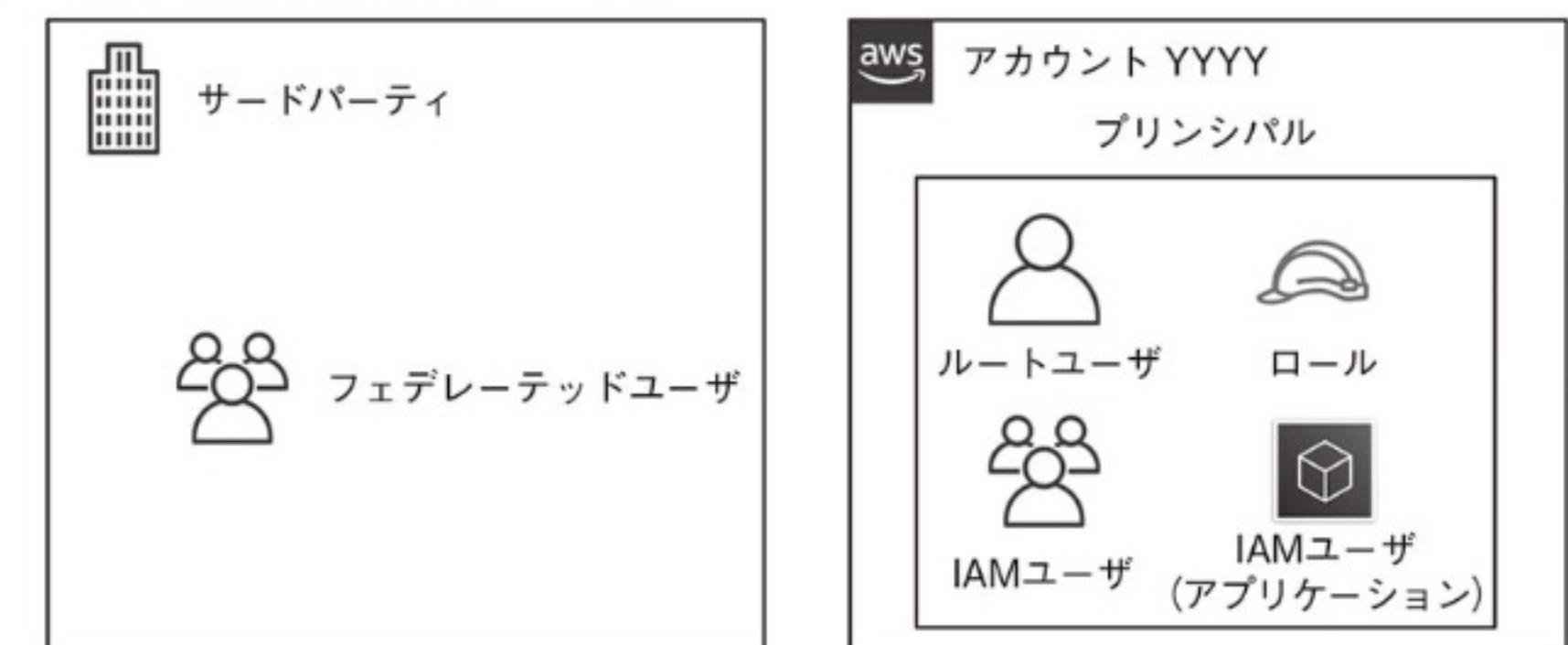
1 IAMの概念と重要な用語

AWSサービスを利用する場合、1章でも解説したとおり、マネジメントコンソールやSDK、CLIを通じて、サービスのAPIを呼び出すリクエストを送信します。

方法	概要
AWSマネジメントコンソール	AWSのサイトからログインできるマネジメントコンソールというGUIでアクセスして、各サービスを実行・利用します。幅広いユーザーに一般的によく利用される方法で、視覚的にわかりやすいため、サービス運用状況の確認などでも利用されます。
AWS CLI (Command Line Interface)	自身のパソコン端末にあるターミナルなどのコマンドラインを実行し、各サービスを実行・利用する方法です。AWS上に展開するアプリケーションを開発する場合など、高速に、繰り返しサービスを起動・実行・停止する用途に適しています。
AWS SDK (Software Development Kit)	AWSが提供するSDKと呼ばれるライブラリを使ったアクセス方法です。主にアプリケーションから利用されることを想定しています。2023年12月時点では、C++、Go、Java、JavaScript、Kotlin、.NET、Node.js、PHP、Python、Ruby、Rust、Swift、SAP ABAPといった13種類のプログラミング言語にそれぞれSDKが提供され、開発したアプリケーションの処理の中でSDKのライブラリをコールしてAWSサービスへアクセスできるようになっています。

リクエストは以下のイメージのように、IAMを通じて検証され、APIの実行可否が判断されます。

【IAMのリクエスト検証処理のイメージ】



IAMがどのように認証（Authentication）・認可（Authorization）を行うかを理解するために必要な概念、用語を解説します。

●アカウント

AWSを利用する際はまずアカウントを作成する必要があります。個人や組織単位で作成されるものであり、最初はアカウント単位で1つのルートユーザーのみが作成されています。

●ユーザ

AWSリソースの利用やAPIの実行を行うユーザです。ユーザには以下の種類があります。

●ルートユーザ

アカウントと同時に作成される、すべてのAWSサービスとリソースに対し、完全なアクセス権を持つユーザです。AWSでは、後述するIAMユーザを最初に作成するとき以外、極力使用を控えることが推奨されています。

●IAMユーザ

マネジメントコンソールのIAMメニューなどから作成されるユーザです。通常、AWSを利用する個人またはアプリケーション単位で作成します。

●フェデレーテッドユーザ

企業内の別システムのディレクトリサービスや、ソーシャルメディアなどで使用されているユーザです。外部のユーザをフェデレート（連合）させてIAMユーザの代わりとして利用できます。ソーシャルメディアのユーザは通常、アプリケーションのWeb IDフェデレーション（OpenID ConnectのIDプロバイダ）として利用されるケースがほとんどで、IAMユーザの代わりとして利用されることはありません^{※1}。次節で解説するCognito「ユーザープールと外部フェデレーション」でも解説しているので、詳細はそちらも参照してください。

●ロール

ロールはユーザと同様、特定のアクセス権限を付与可能な「役割」に相当する概念です。特定の個人を一意に関連付けるわけではなく、それを引き受ける人に對し、一時的な認証情報が提供されます。わかりやすくいえば、「管理者」や「窓口担当」のような個人を特定しないユーザに近い概念と考えるとよいでしょう。ロールもマネジメントコンソールなどから作成でき、アクセス権限を付与できます。

●プリンシバル

プリンシバルは、AWSリソースに対するアクション・オペレーションをリクエストできるユーザおよびロールを指します。

●リクエスト

プリンシバルがAWSサービスへのアクションやオペレーションを実行したり、AWSリソースへアクセスしたりするために、マネジメントコンソールやAWS SDK、CLIなどを経由してAWSのAPIへ送信するリクエストです。リクエストには以下のような情報が含まれます。

※1 https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/id_roles_providers_oidc.html

●アクションまたはオペレーション

プリンシバルが実行するアクションです。マネジメントコンソールから実行する操作はアクション、SDKやCLIから実行する操作はオペレーションと呼んで区別しています。

●リソース

アクションまたはオペレーションの実行対象となるAWSリソースです。

●環境データ

IPアドレスや、ユーザエージェント、SSL有効化ステータス、時刻などのクライアントに関する情報です。

●リソースデータ

アクションやオペレーションの実行に必要なリソースに関するデータです。

●認証情報

リクエストを発出したプリンシバルやその正当性を証明する署名などが含まれます。

●ポリシー

ポリシーはアクセス許可を定義します。以下のように許可・拒否するアクションと対象となるリソースをJSON表現で定義できます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:*",
      "Resource": "arn:aws:dynamodb:us-east-2:123456789012:table/Books"
    }
  ]
}
```

ポリシーに定義できる主な要素は以下のとおりです。

●Version

ポリシー言語のバージョンです。“2012-10-17”が最新バージョンです。Version要素がなければポリシー変数（\${aws:username}など、ポリシーを記述するときに正確な値が特定できない場合などに使われる）は文字列として扱われるため注意してください。

● Statement

アクセス許可の対象を定義します。複数の要素を記述することも可能です。

● Sid

Statementの子要素で、複数のStatementを区別するためのIDを指定します。この要素はオプションです。

● Effect

Statementの子要素で、対象のアクションを「Allow (許可)」するか「Deny (拒否)」するかを定義します。

● Principal

Statementの子要素で、アクションを実行するプリンシパルをARN形式で定義します。AWSアカウントを指定するときは、アカウントARN (arn:aws:iam::AWS-account-ID:root)、または「AWS:」プレフィックスの後にIDを付けた短縮形を使用できます。

● Action

Statementの子要素で、Effectで指定した許可または拒否対象のアクションを記述します。大文字小文字の区別はありません。

● Resource

Statementの子要素で、Actionの対象となるリソースをARNで定義します。「*(ワイルドカード)」も利用できます。

● Condition

Statementの子要素で、ポリシーを適用する条件を設定できます。条件演算子やポリシー変数、条件キーなどを記述できます。Conditionに記述した要素はブロック単位ではAND条件で、配列などで複数記述した場合はOR条件として扱われます。

ポリシーは用途や管理主体に応じて、いくつか種類があり、以下のとおり分類されます。

【ポリシーの種類】

ポリシーの種類	説明
アイデンティティベースポリシー	ユーザやグループ、ロールに直接設定するポリシー。メンテナンス、再利用性を踏まえ、AWSではインラインポリシーではなく管理ポリシーの使用が推奨されている。
管理ポリシー	AWSアカウント内の複数のユーザ、グループ、ロールにアタッチする独立したポリシー。アタッチ可能なポリシーはそれぞれ最大10個までなので注意が必要。
AWS管理ポリシー	AWSが作成および管理する編集不可のポリシー。「AmazonEC2FullAccess」や「AmazonS3ReadOnlyAccess」のように各リソースアクセスのユースケースにもとづいて定義されている。
カスタマーマネジメントポリシー	ユーザがカスタマイズ可能な管理ポリシー。ビジュアルエディタでポリシーを作成、編集および検証することも、JSON形式のポリシードキュメントを直接作成することもできる。
リソースベースポリシー	S3やSQSなどAWSリソースに対して設定するポリシー。プリンシパルを指定して、実行可能なアクションや条件を制御する。クロスアカウントでリソースアクセスする場合、アクセスされる側のリソースベースポリシーを作成し、アクセスさせるアカウントのプリンシパルを指定する。



ユーザに設定するアイデンティティベースポリシーとリソースベースポリシーについては、2章9節「S3 アクセスコントロール」で具体例を踏まえて解説しているので、そちらも参照してください。

2 IAMでの認証・認可

AWSサービスやリソースにアクセスしたいユーザやロールは、前項の図の(1)～(5)のフローで、AWS APIに対しリクエストを送信します。AWS APIを呼び出す際に、まずリクエストの正当性をキーを使って検証し、リクエストを発行したプリンシパルが持つ権限を確認します。呼び出し対象のAPIのアクションやリソースが、定義された権限から許可されていれば、APIが実行される流れになります。以降、この流れに関して補足します。

●リクエスト認証

AWS APIに送信するリクエストには、認証情報（クレデンシャル）を付与する必要があります。付与する方法は、以下の3つのパターンに分けることができます。

1. IDとパスワードによる認証

主にマネジメントコンソールを使用する場合の方法です。マネジメントコンソールへログインする際に、IDとパスワードを入力します。ログイン後は任意のオペレーションを実行できるようになります。

2. アクセスキーIDとシークレットアクセスキーによる認証

主にSDKやCLIを使用する場合の方法です。アクセスキーIDとシークレットアクセスキーという2つのキーをSDKやCLIを実行するクライアントに設定します。これらのキーは、通常、「~/.aws/credentials」（Windowsの場合は%UserProfile%配下）に保存します。EC2などにデプロイした商用アプリケーションでもこれらのキーを使用できます。ただしAWSでは、インスタンスに設定したIAMロールから一時的な認証を利用する方法（下記参照）が推奨されています^{*2}。アクセスキーIDとシークレットキーによる認証方法は、AWS外の開発端末などの利用に留めておきましょう。

3. STSを使った一時的な認証

AWS Security Token Service (AWS STS) を使用して、信頼されたユーザを作成および提供する方法です。このユーザは、AWSリソースへのアクセスをコントロールできる一時のセキュリティ認証情報を持ち、期間内はアクセスキーIDとシークレットアクセスキーと同等の機能を利用できます。IDフェデレーションやロールをプリンシパルとして指定して、AssumeRole（ロールを引き受ける）アクションをSTS API経由で呼び出すことにより、一時的な認証情報を取得できます。



AWSコンソール上でIAMでユーザを作成する際は、「パスワード」もしくは「アクセスキーとシークレットキー」を作成するかを選択します。

【IAMユーザの作成】

ユーザーを追加

ユーザー詳細の設定

同じアクセスの種類とアクセス権限を使用して複数のユーザーを一度に追加できます。 詳細はこちら

ユーザー名:

別のユーザーの追加

AWS アクセスの種類を選択

これらのユーザーから AWS にアクセスする方法を選択します。アクセスキーと自動生成パスワードは前のステップで提供されています。 詳細はこちら

アクセスの種類: プログラムによるアクセス
AWS API, CLI, SDK などの開発ツールの アクセスキー ID と シークレットアクセスキー を有効にします。

AWS マネジメントコンソールへのアクセス
ユーザーに AWS マネジメントコンソールへのサインインを許可するための パスワード を有効にします。



IAM Roles Anywhere の発表

2022年7月に、IAM Roles Anywhereが発表されました^{*3}。これは従来、AWS以外の環境下にあるオンプレミスマシンや開発端末で、AWSアクセスキーIDとシークレットアクセスキーを使って、AWSのリソースアクセスのための認証情報として使用していたところ、代わりに設定したプライベートな認証局（Private Certificate Authority）を通じて、証明書をやりとりし、一時認証情報を発行する仕組みです。

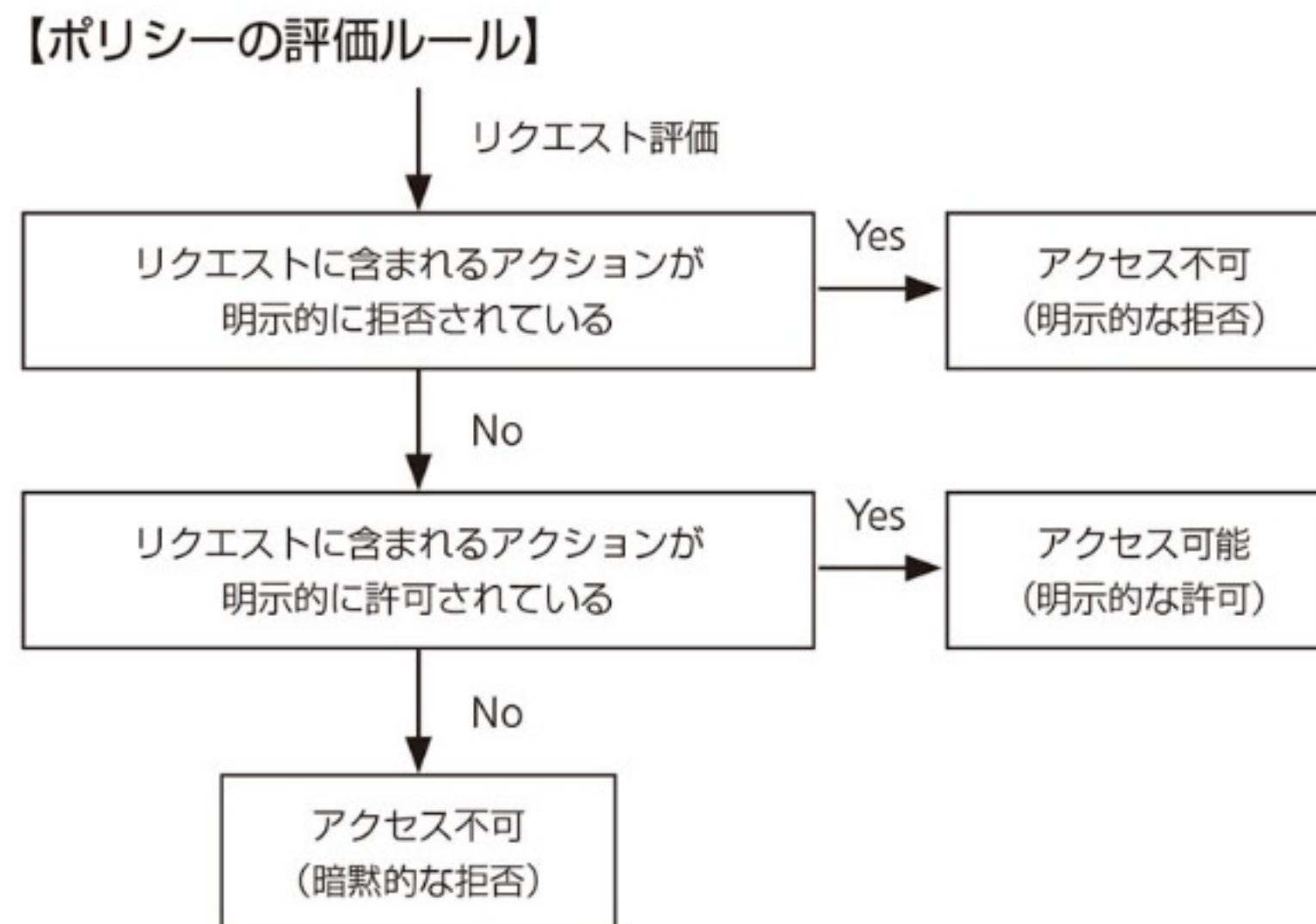
この仕組みにより、AWS外の環境においてもSAMLなど外部フェデレーション認証なしで、一時的な認証を利用することができます。

●認可の評価ロジック

認証が正常に確認されたリクエストは、リクエスト送信元となるプリンシパルの権限を確認し、APIの実行可否を評価します。プリンシパルに設定されているポリシーの評価は、基本的に以下のルールに則って判定されます。

*2 https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/id_roles_use_switch-role-ec2.html

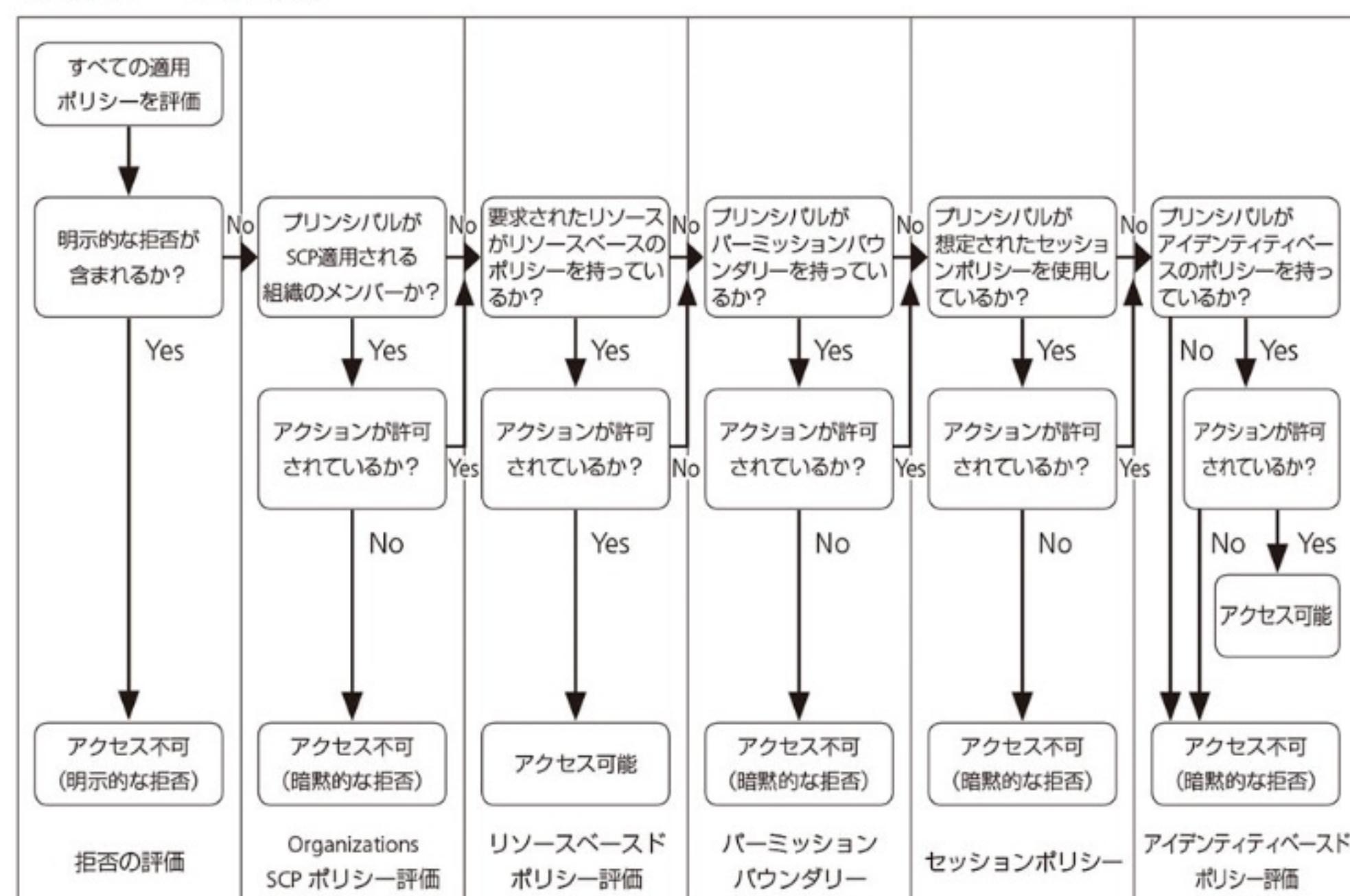
*3 <https://aws.amazon.com/jp/blogs/news/extend-aws-iam-roles-to-workloads-outside-of-aws-with-iam-roles-anywhere/>



1. 明示的にDenyが含まれている場合は拒否（明示的な拒否）
2. 明示的にDenyが含まれておらず、Allowが含まれる場合は許可
3. 何も定義されていない場合はデフォルトで拒否（暗黙的な拒否）

上記を踏まえて、実際の評価ロジックを解説します。AWSでは前項で解説したアイデンティティベースのポリシーや、リソースベースのポリシーだけでなく、パーミッションバウンダリーやAWS Organizationsのサービスコントロールポリシー(SCP)、セッションポリシーを通してアクションやオペレーションのAPI実行可否が決定されます。異なる種類のポリシーが複数設定されていた場合や評価される順序は以下のとおりです。

【ポリシーの評価】



AWS OrganizationsのSCPは本書の範囲外としていますので説明は割愛しますが、パーミッションバウンダリーやセッションポリシーは後述します。ここでは、API実行にはさまざまなポリシーが評価されて実行可否が判断されることを意識してください。

また、この図からわかるとおり、リソースベースのポリシーが許可されていれば、アイデンティティベースのポリシーで拒否されていたとしてもアクセスは可能になります。つまりリソースベースのポリシーとアイデンティティベースのポリシーはOR条件です。一方、パーミッションバウンダリーとアイデンティティベースのポリシーはそれぞれが許可されて初めてアクセス可能になります(AND条件)。

●パーミッションバウンダリー

パーミッションバウンダリーは、ユーザやロールなどのプリンシパルに追加で管理ポリシーを設定する機能です。パーミッションバウンダリーを利用することで、ロールを引き受けたりするユーザに対し、特定の権限をさらに制限するといったことが可能になります。ただし、グループに対してはパーミッションバウンダリーの設定はできません。

●セッションポリシー

セッションポリシーは、STSで一時的な認証情報をアプリケーションで生成する場合に、任意のパラメータを条件に加えるなどカスタマイズしたポリシーを評価条件に付け加える機能です^{※4}。詳細は3章2節「AWS STS」の「AssumeRole API」を参照してください。

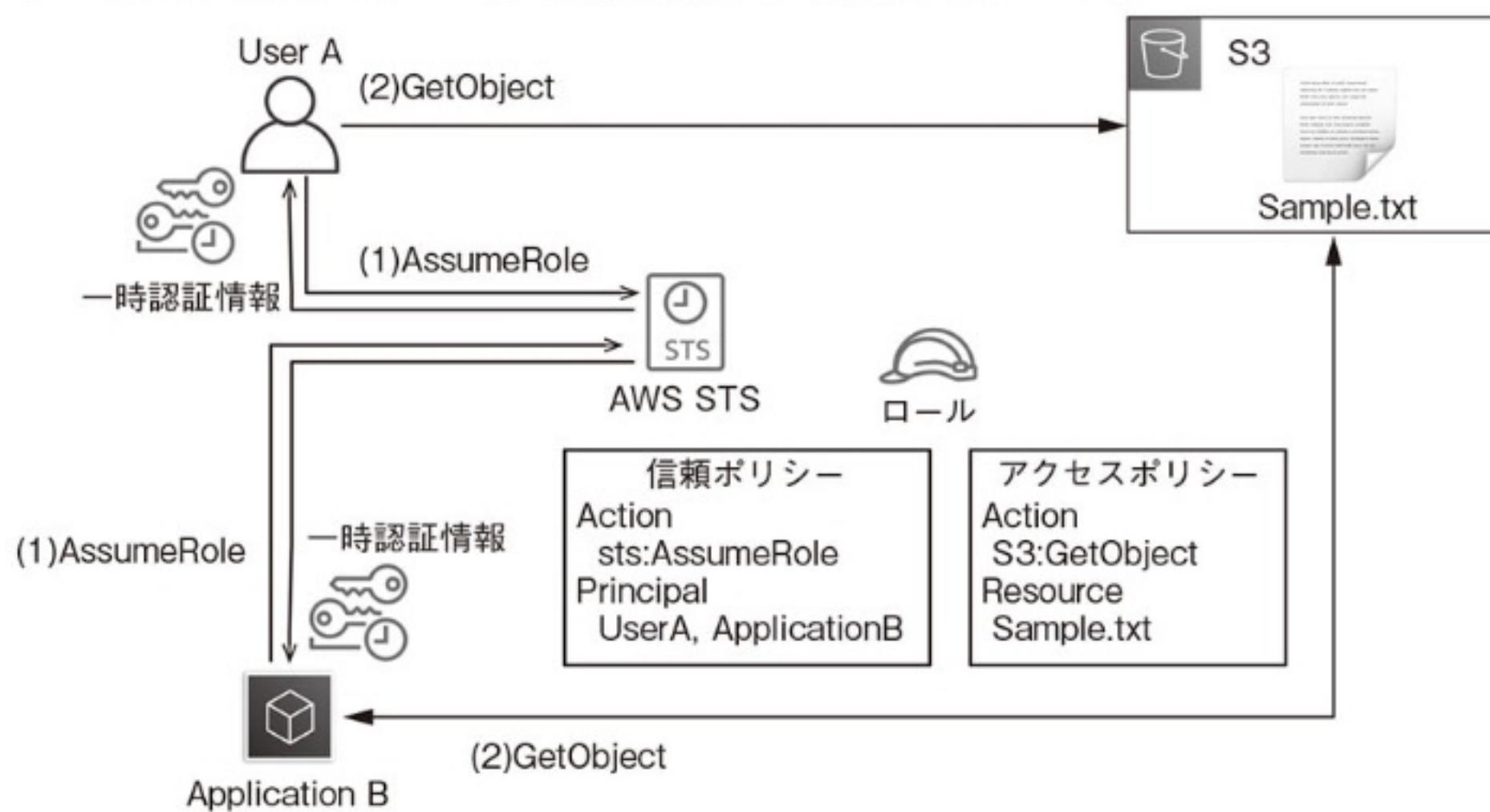
3 権限の委任

●ロールの引き受け (AssumeRole)

前項で、リクエスト認証の1つとして、STSで一時的な認証があることを述べました。次節「AWS STS」でも詳述しますが、ここでは、IAMロールを使って、アクセス権限が付与された一時的な認証情報を作成し、AWSリソースにアクセスする仕組みを解説します。下記は、一時認証情報を用いて、S3にあるテキストドキュメントにアクセスする例を示したものです。

※4 https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/id_credentials_temp_request.html

S3へアクセスするロールを引き受けける場合のイメージ



ユーザAやアプリケーションBは、S3にあるテキストドキュメントにアクセスする権限は持っていません。ここで、IAMロールを作成し、信頼されたエンティティとアクセスポリシーを設定します。信頼されたエンティティとは、そのロールの使用を許可するターゲット（プリンシパル）のことです。IAMロールにリソースベースのポリシーを付与することで設定します。このポリシーのことを信頼ポリシーと呼びます。以下は信頼されたエンティティとして、UserAとApplicationBを信頼ポリシーで指定した例です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SampleTrustedPolicy",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "AWS": [
          "arn:aws:iam::XXXXXXXX:user/UserA",
          "arn:aws:iam::XXXXXXXX:user/ApplicationB"
        ]
      }
    }
  ]
}
```

また、IAMロールにアクセスポリシーを設定するには、以下のようなカスタマーマネジメントポリシーをアタッチします。これで、目的のテキストドキュメントへのアクセス権限が付与されます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::example_bucket/Sample.txt"
    }
  ]
}
```

このロール名を指定して、信頼されたエンティティがSTSに対して上図の(1) AssumeRole APIを実行することにより、時限的にS3にアクセス可能な認証情報が作成されます。この操作を「ロールの引き受け（AssumeRole）」と呼びます。取得した一時認証情報をリクエストに付与し、(2) S3へアクセスするAPIを実行すると、リソースにアクセスできるようになります。

なお、信頼されたエンティティとしてロールを指定することで、複数のロールを引き受けることも可能です（ロールの連鎖^{※5}）。

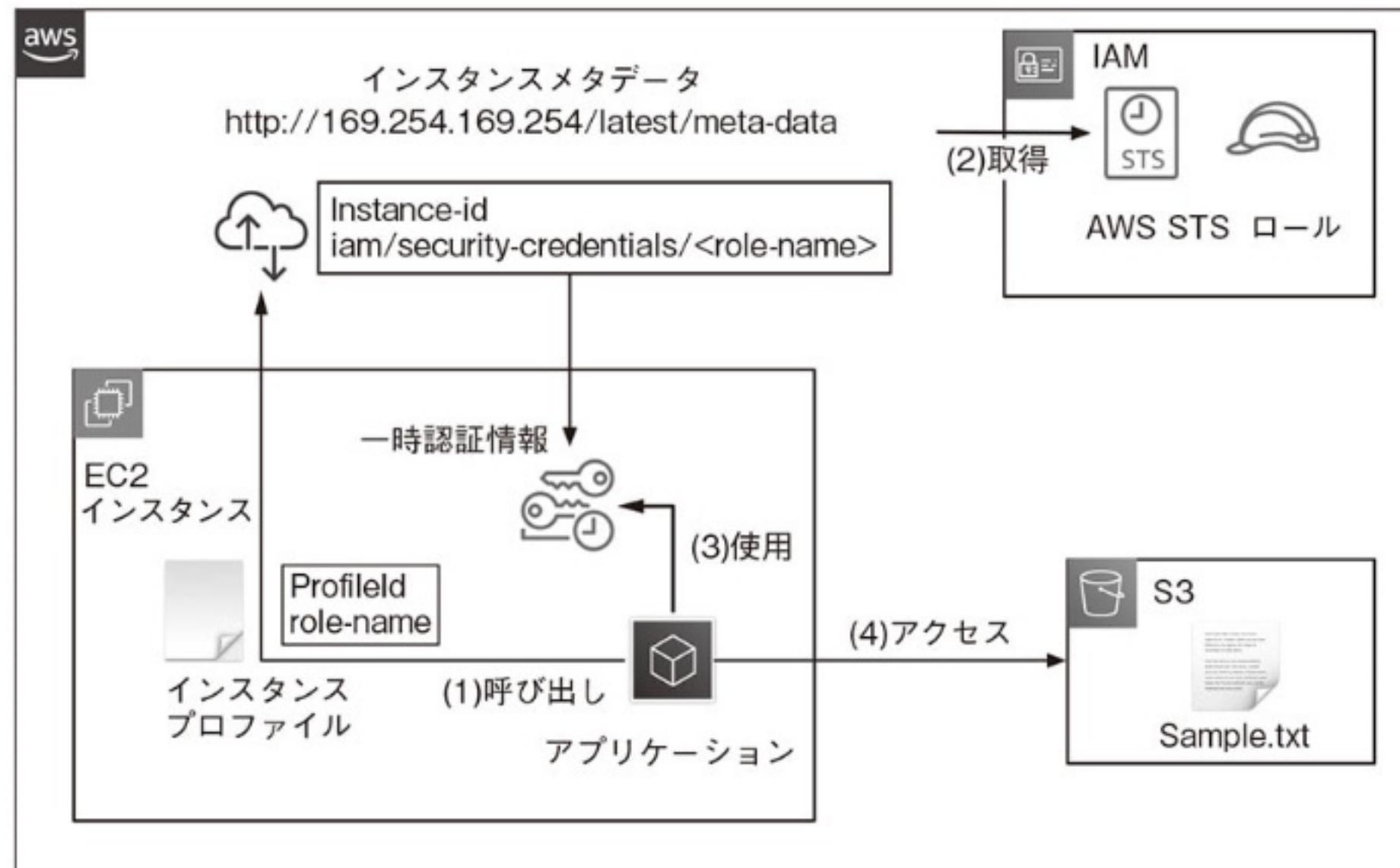
●EC2インスタンスのIAMロール

アプリケーションは通常EC2インスタンス上などで実行されます。EC2ではインスタンスを起動する際、インスタンスプロファイルを使用して割り当てるIAMロール名を取得し、インスタンスマタデータを経由して一時的な認証情報を透過的（IAMロールの存在を意識することなく）に取得することができます^{※6}。

※5 https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/id_roles_terms-and-concepts.html#iam-term-role-chaining

※6 https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/id_roles_use_switch-role-ec2.html

【S3へアクセスするロールを引き受ける場合のイメージ】



インスタンスマタデータ^{※7}は、リンクローカルアドレス (<http://169.254.169.254/>) 経由でインスタンスのさまざまな情報を取得できる機能です。「iam/security-credentials/<role-name>」カテゴリからロール名を指定することで、一時的な認証情報を取得できます。

アプリケーション内で使用されるSDKやCLIがEC2インスタンス上で実行される場合は、このメタデータを参照して一時的な認証情報を取得し、APIを呼び出します。そのため開発者はインスタンス起動時にIAMロールを割り当てるだけで、アクセスキーなどを設定することなく、透過的に一時認証情報を使ってアクセスできます。

なおこのとき、ロールの信頼ポリシーに設定するプリンシパルは「ec2.amazonaws.com」になります。



デベロッパー - アソシエイト試験に限りませんが、AWS資格試験には、EC2インスタンスにロールを割り当てたかたちでAWSリソースへのアクセスを問う問題が頻出します。確実に問題を解けるように理解しておきましょう。



PassRoleでEC2インスタンス起動時にIAMロールを指定する

IAMロールを使ったEC2インスタンスの起動は、起動を実行したユーザに対し、不要な権限の一時認証情報を与えてしまします。ユーザ自身が保持していない権限を持つIAMロールを設定して、インスタンスを起動した後ログインしてしまえば、そこに一時認証情報が利用可能な状態でCLIなどを実行できてしまうからです。

これを防止するために、PassRoleというポリシーが用意されています。このポリシーを設定しておけば、インスタンス起動時に設定するロールを制限することができます。

下記のようなポリシーを、インスタンス起動を実行するユーザやグループにアタッチしておくと、インスタンスの起動時にユーザが異なるロールを指定した場合、アクションは失敗します。

【PassRoleを使って、インスタンスを起動する際に設定するロールを制限する例】

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:RunInstances",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::XXXXX:role/specify-role"
    }
  ]
}
  
```

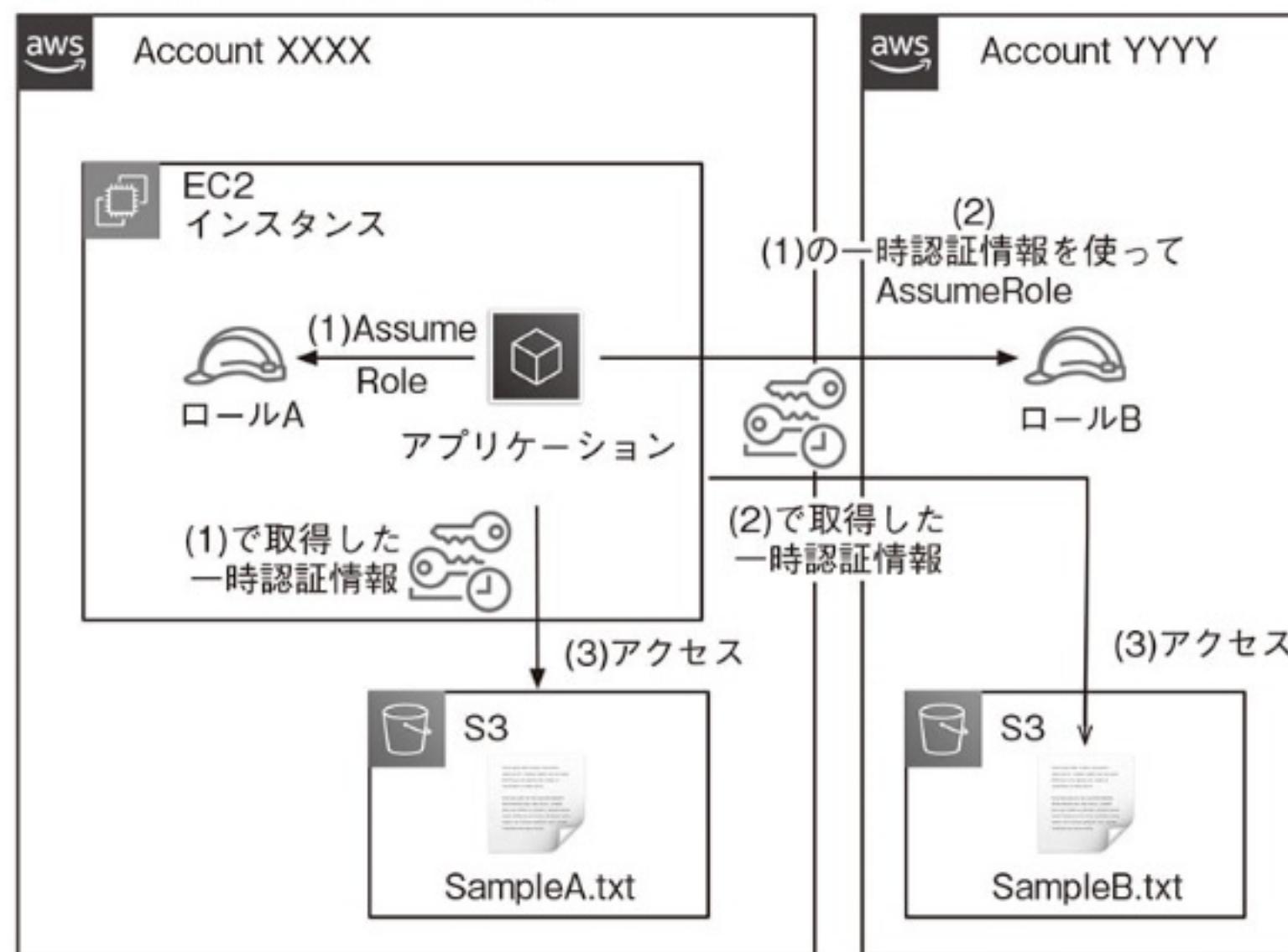
3

●クロスアカウントロール

続いて、ロールを使ってクロスアカウントアクセスを行う例について紹介します。以下の図は、あるアカウントXXXXにあるEC2インスタンスのアプリケーションが、自アカウントXXXXおよび別のアカウントYYYYのS3のファイルへアクセスしたい場合に、クロスアカウントでロールを連鎖させる例です。

※7 https://docs.aws.amazon.com/ja_jp/AWSEC2/latest/UserGuide/instancedata-data-retrieval.html

【クロスアカウントロール】



EC2インスタンスで実行されているアプリケーションは、インスタンスプロファイルから取得したロール名を使って、インスタンスマタデータからIAMロールを引き受けて一時認証情報を取得します。この認証情報では、自アカウントXXXXがS3へアクセスする権限を持っていればよく、また、アカウントYYYYのロールBを引き受ける権限が必要になります。そのため、ロールAの信頼ポリシーおよびアクセスポリシーは、以下のとおり設定されている必要があります。

【ロールAの信頼ポリシー】

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SampleRoleATrustPolicy",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {"Service": "ec2.amazonaws.com"}
    }
  ]
}
```

【ロールAのアクセスポリシー】

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccountXXXBucketAccess",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::SampleBucketA/SampleA.txt"
    },
    {
      "Sid": "AllowAssumeCrossAccountRole",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::YYYY:role/SampleRoleB"
    }
  ]
}
```

続いて、ロールBでは、アカウントYYYYのS3へアクセスする権限を持っており、信頼されたエンティティとして、ロールAを指定する必要があります。ロールBに設定しておく信頼ポリシーおよびアクセスポリシーは以下のとおりです。

【ロールBの信頼ポリシー】

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SampleRoleBTrustPolicy",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {"AWS": "arn:aws:iam::XXXX:role/RoleA"}
    }
  ]
}
```

【ロールBのアクセスポリシー】

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccountYYYYBucketAccess",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::SampleBucketB/SampleB.txt"
    }
  ]
}
```

アプリケーションではSDKやCLIがインスタンスマタデータから一時認証情報を取得するので、自アカウントXXXXのバケットにはそのままアクセスできます。一方、アカウントYYYYのバケットへのアクセスは別途SDKやCLIから、STSのAPIを呼び出して新たな一時認証情報を取得し、S3バケットへアクセスするAPIを呼び出してアクセスします。

詳細な実装の例は次節「AWS STS」を参照してください。



クロスアカウントでのS3のバケットアクセスは、ロールの代わりにリソースベースポリシーを設定することでも可能です。詳細は2章9節「Amazon S3」の「セキュリティ保護」を参照してください。

●外部ID フェデレーションユーザのロール引き受け

社内ディレクトリサービスや、別のシステム、サービス、IDプロバイダ（OpenID Connect:OIDCプロバイダ）で使用されているユーザがAWSリソースを利用する必要がある場合も、ロールの引き受けが利用できます。これらのユーザをフェデレート（連合）し、ロールを引き受けることで、一時認証情報が提供されるようになります。

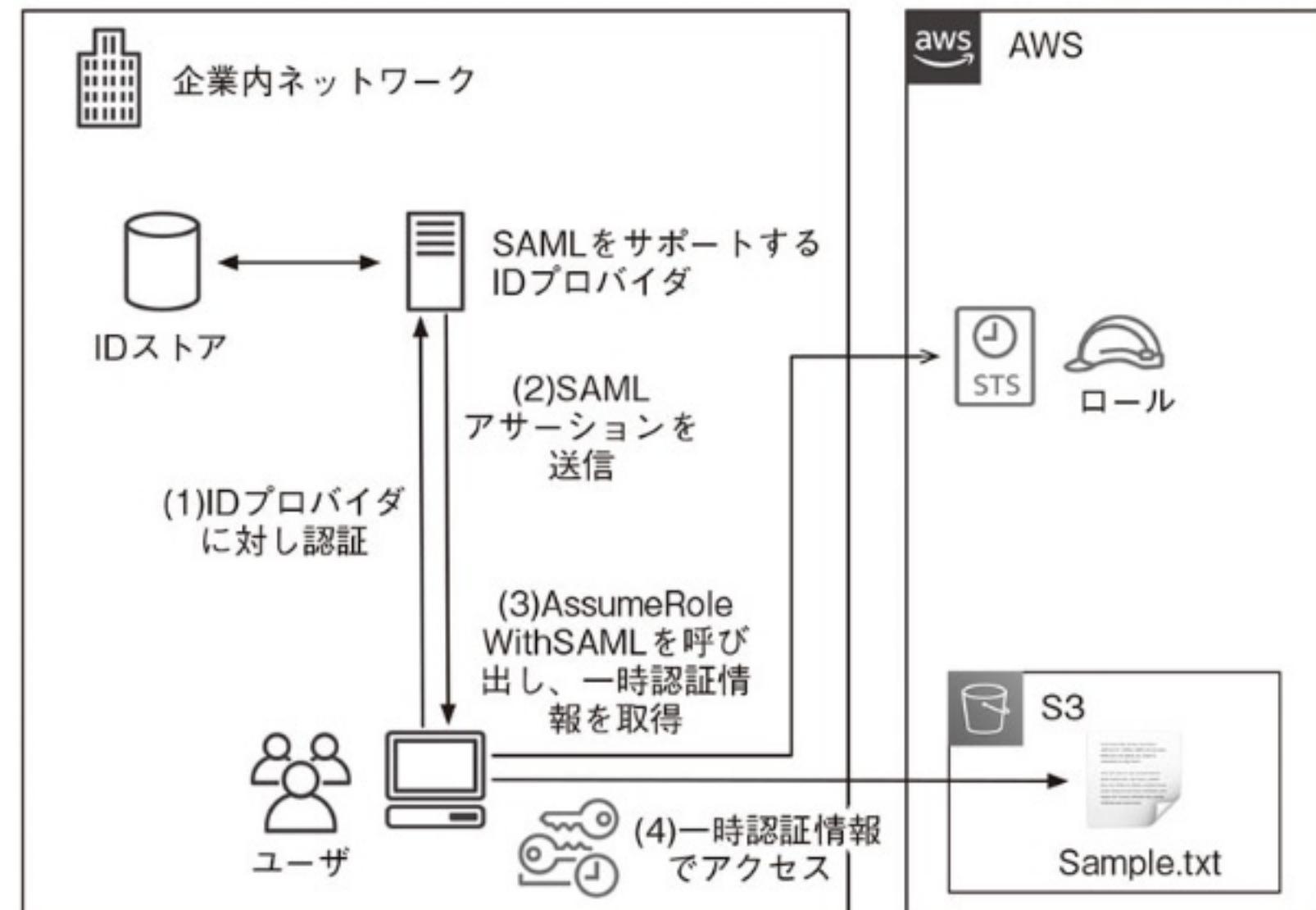
IAMでは、以下のようなユースケースのフェデレーションを想定して、ロールの引き受けを設定する仕組みが提供されています。

●SAML 2.0に準拠したIDプロバイダのフェデレーション^{*8}

組織内にMicrosoft Active Directoryなどのディレクトリサービスがあり、この認証情報を用いてAWS内のリソースへアクセスしたい場合は、以下のステップでロールを引き受けます。

*8 https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/id_roles_providers_saml.html

【SAML 2.0に準拠したIDプロバイダのフェデレーション】



1. 企業内ネットワークにあるIDプロバイダに対して、認証を行う
2. 結果としてSAMLアサーションを取得する
3. AssumeRoleWithSAMLを呼び出し、ロールを引き受け一時認証情報を取得する
4. AWSリソースへアクセスする

IAMロールに設定する信頼ポリシーのプリンシパルには、事前にIAMメニューから、SAMLをサポートするIDプロバイダの定義を作成し、以下のようにARNを指定します。また、アサーションを提供するエンドポイントや、発行元、ユーザの属性など条件を設定してアクセス制御することも可能です^{*9}。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {"Federated": "arn:aws:iam::XXXX:saml-provider/SampleProvider"},
      "Action": "sts:AssumeRoleWithSAML",
      "Condition": {
        "StringEquals": {
          "saml:aud": "https://signin.aws.amazon.com/saml",
          "saml:iss": "https://openidp.feide.no"
        },
        "ForAllValues:StringLike": {"saml:edupersonaffiliation": ["staff"]}
      }
    }
  ]
}
```

*9 https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/reference_policies_iam-condition-keys.html#condition-keys-saml

```

    }
  }
}

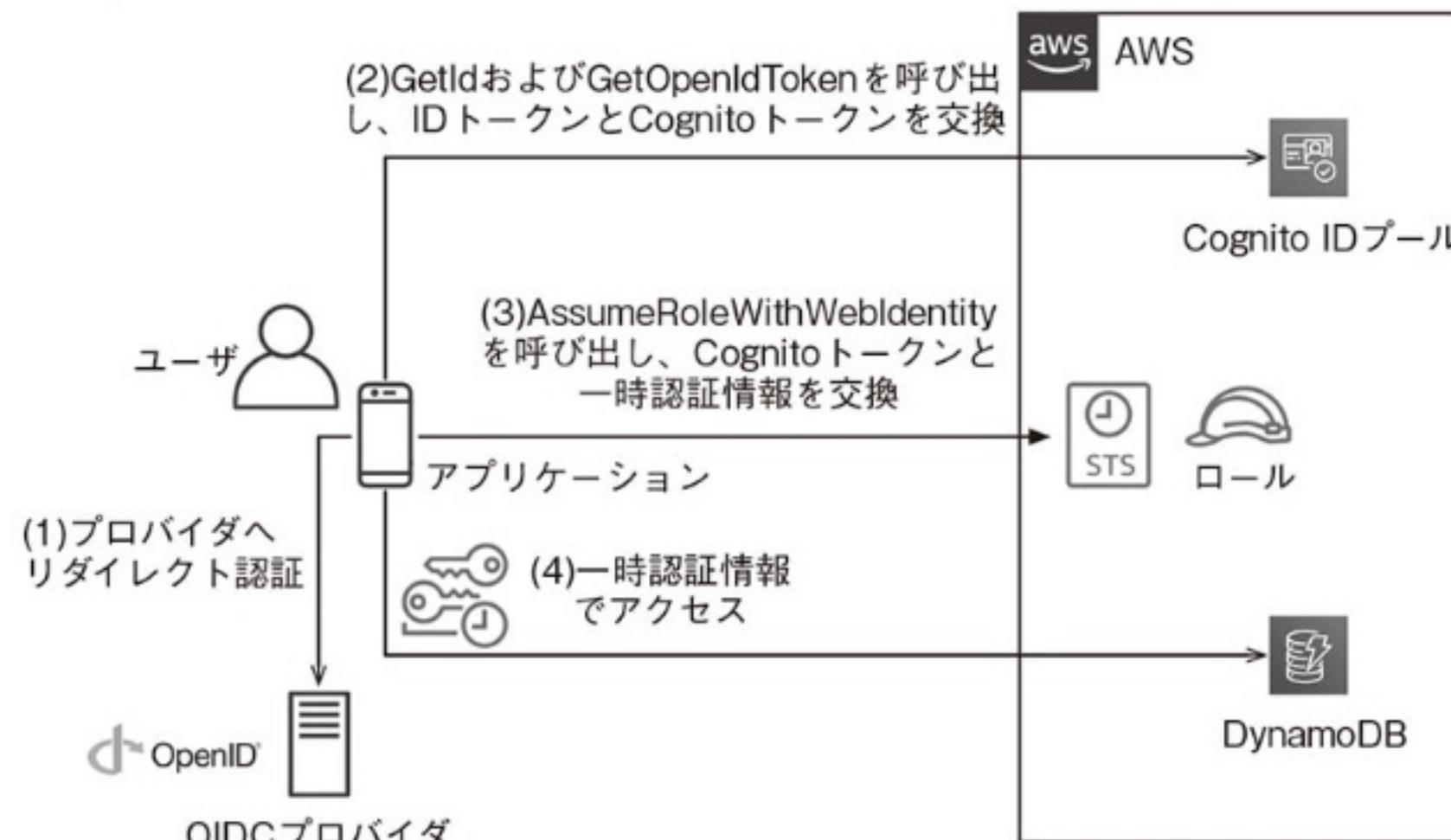
```

●Amazon Cognitoを使用したモバイル・Webアプリケーションやソーシャルメディア、OIDCに準拠したIDプロバイダのフェデレーション^{*10}

Amazon Cognitoユーザプールや、FacebookやGoogleといったソーシャルメディアなどOIDCに準拠したオープンIDプロバイダのユーザ情報を使用して、AWSリソースへアクセスしたい場合もIAMロールによる引き受けが可能です。ロールを引き受け、一時認証情報を取得することでAWSリソースにアクセスできます。

次の図は、モバイルアプリケーションでOIDCプロバイダからアクセスするステップです。

【Cognitoに準拠したIDプロバイダのフェデレーション^{*11}】



- モバイルアプリケーションは、ソーシャルメディアなどのオープンなOIDCプロバイダのユーザ情報を使ってアクセスするために、プロバイダへリダイレクトする
- プロバイダで認証後、IDトークンもしくはアクセストークンを取得し（IDプロバイダで異なる）、Cognitoに関連付けられたIDトークンを交換する
- AssumeRoleWithWebIdentityを呼び出し、ロールを引き受け一時認証情報を取得する
- AWSリソースへアクセスする

*10 https://docs.aws.amazon.com/ja_jp/cognito/latest/developerguide/authentication-flow.html

*11 https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/id_roles_providers_oidc_cognito.html



この例では、STSに対し、一時認証情報をAssumeRoleWithWebIdentityで取得していますが、Cognitoに対し、GetCredentialsForIdentity APIを実行することにより、Cognitoが内部的にSTSへ一時認証情報を要求する認証フローも選択できます。詳細は「Cognito IDプールの認証フロー」を参考にしてください^{*12}。また、Cognitoの詳細については3章4節「Amazon Cognito」を参照してください。

ロールに設定する信頼ポリシーのプリンシパルには、「Federated」プロパティに、IDプロバイダのURLを指定します。次のコードは、CognitoをIDプロバイダとして設定する場合の例です^{*13}。

```

{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "RoleForCognito",
    "Effect": "Allow",
    "Principal": {"Federated": "cognito-identity.amazonaws.com"},
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {"StringEquals": {"cognito-identity.amazonaws.com:aud": "us-east-1:12345678-ffff-ffff-ffff-123456"}}
  }
}

```

こちらのロールを設定する前にも、事前にIAMメニューからOIDC IDプロバイダを作成し、設定対象の、実際のIDプロバイダのJWK_URIから取得可能な証明書をハッシュ化したサムプリントを設定します。なお、GoogleやFacebook、CognitoなどはすでにAWSに組み込まれているため、サムプリントを作成する必要はありません。

●SAMLやOIDCと互換性がないIDストアを中継するカスタムIDブローカーアプリケーションのフェデレーション^{*14}

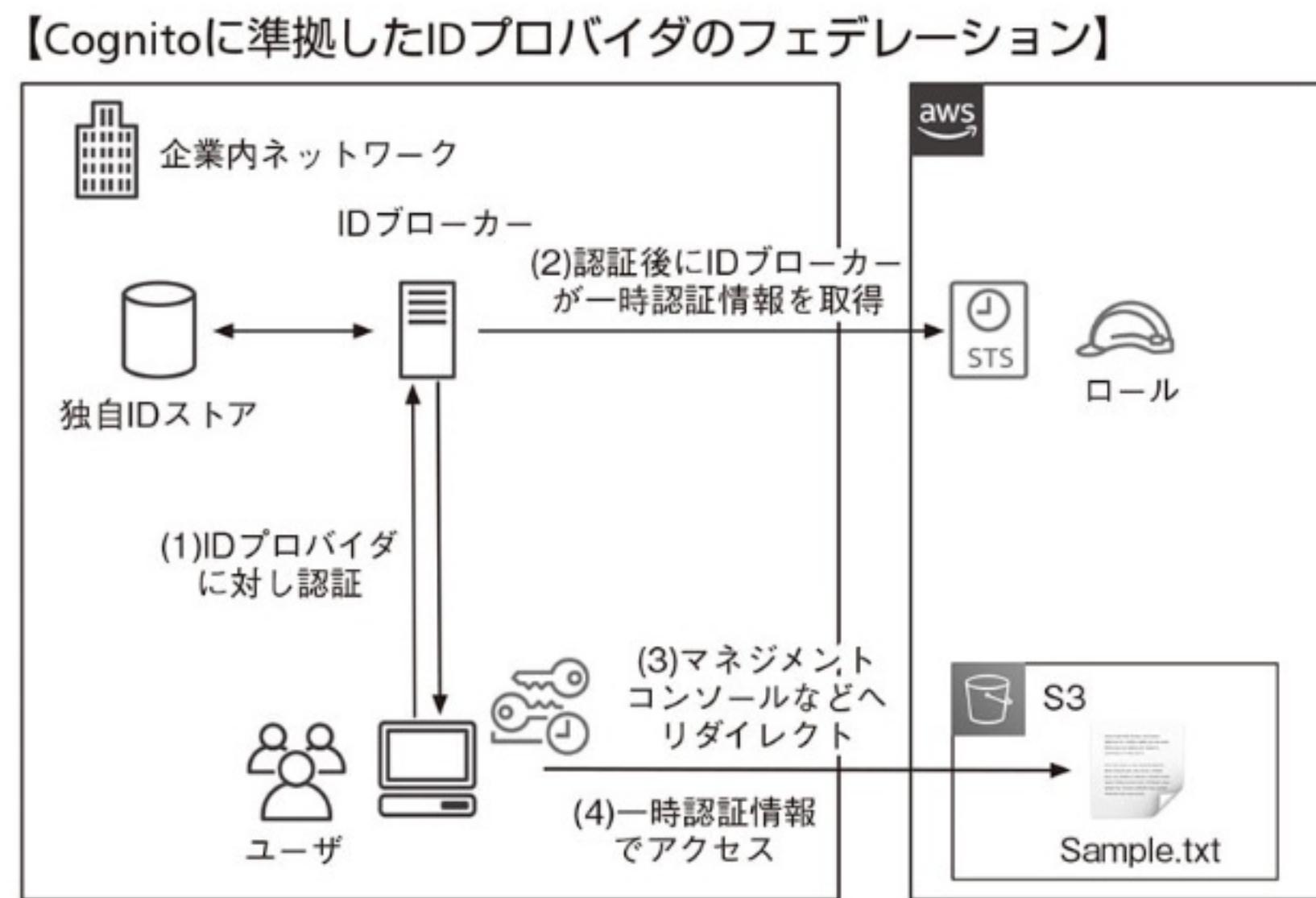
この方式はSAML準拠のIDプロバイダとも似ていますが、SAMLアサーションをAWSに送信することなく、IDブローカーがSTSから一時認証情報を取得します^{*15}。処理のフローは以下のようになります。

*12 https://docs.aws.amazon.com/ja_jp/cognito/latest/developerguide/authentication-flow.html

*13 https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/id_roles_create_for-idp_oidc.html

*14 https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/id_roles_common-scenarios_federated-users.html

*15 https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/id_roles_providers_enable-console-custom-url.html



1. 企業内ネットワークにあるIDプロバイダ（認証処理を行うアプリケーション兼IDプローカー）に対して、認証を行う
2. IDプローカーがAssumeRole APIまたはGetFederationToken APIオペレーションを呼び出して、ロールを引き受け、一時認証情報を取得する
3. 認証情報を使って、リダイレクトなどで、AWSリソースへアクセスする

IDプローカーは、AWS外にあるアプリケーションのため、アクセスキーIDとシークレットアクセスキーによりアクセスするユーザとなります。そのため、ロールの信頼ポリシーに設定するプリンシパルは次のようにになります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/onpremis-idbroker-app"
            },
            "Action": "sts:AssumeRole",
            "Condition": {}
        }
    ]
}
```

IDプローカーアプリケーションでAssumeRole APIを呼び出す実装は次節「AWS STS」を参照してください。

3-2 AWS STS(Security Token Service)

STSは一時的な認証情報を生成できる、IAMと表裏一体となるサービスです。難解な分野でもあるため、注目されることは少ないですが、AWSでクラウドネイティブなアプリケーションを構築する場合に避けては通れない重要なサービスです。

1 STSの概要

STSは難解なサービスですが、提供している内容はシンプルです。指定された（短い）期間でAWSリソースにアクセス可能な認証情報を生成するだけです。STSはマネジメントコンソールで提供されないサービスであり、SDKやCLIで、APIを呼び出すかたちで利用されることを前提としています。用途のほとんどは、前節「AWS IAM」の「権限の委任」で解説したとおり、IAMロールを引き受けて、一時的な認証情報を提供することです。

STSでは、以下のAPIが提供されています。

- AssumeRole
- AssumeRoleWithSAML
- AssumeRoleWithWebIdentity
- DecodeAuthorizationMessage
- GetAccessKeyId
- GetCallerIdentity
- GetFederationToken
- GetSessionToken

ここでは、特に重要なAssumeRole APIについて解説を進めていきます。

2 AssumeRole API

AssumeRole APIは、前節「AWS IAM」の「権限の委任」の中で、繰り返し登場した一時認証情報を生成するためのAPIです。このAPIは、アプリケーション内で使用されているSDKやCLIで利用されます。

一例として、次の信頼ポリシーで作成されているロール「sample-role」を引き受ける際の実装を解説します。このロールは、アプリケーション「sample-app」ユーザのリクエストからのみ、ロールを引き受けることを想定します。

【sample-roleの信頼ポリシー】

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/sample-app"
      },
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

このロールには、S3バケット「sample-bucket」にあるファイルへのアクセス権限を付与する、以下のようなアクセスポリシーがアタッチされているとします。

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

SDKを使ってAPIをコールするJava言語での実装例は以下のとおりです。S3にあるファイルに対し、一時的にアクセス可能なURLを生成しています。

```

import com.amazonaws.HttpMethod;
import com.amazonaws.auth.STSShareSessionCredentialsProvider;
import com.amazonaws.auth.policy.Policy;
import com.amazonaws.auth.policy.Statement;
import com.amazonaws.auth.policy.actions.S3Actions;
import com.amazonaws.services.identitymanagement.*;
AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.GetRoleRequest;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import com.amazonaws.services.s3.model.ResponseHeaderOverrides;

// omit

public URL getPresignedUrl(String filePath){
  // 一時認証情報が設定されたS3クライアントを生成
  AmazonS3 amazonS3 = getAmazonS3ClientWithDownloadPolicy(filePath);
  Date expiration = Date.from(ZonedDateTime.now().plusSeconds(300).toInstant());
  // S3に300秒アクセス可能な署名付きURLを生成
  return amazonS3.generatePresignedUrl("sample-bucket", filePath, expiration);
}

private AmazonS3 getAmazonS3ClientWithDownloadPolicy(String objectKey){
  // 対象のオブジェクトキーのARN文字列を生成
  String resourceArn = "arn:aws:s3:::sample-bucket/" + objectKey;
  // セッションポリシーのステートメント文字列を作成
  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();

  // return セッションポリシーを加えたロールの一時認証情報を設定したS3クライアントを生成
  return AmazonS3ClientBuilder.standard()
    .withCredentials(new STSShareSessionCredentialsProvider
      .Builder(getRoleArn(), "sample-role-session-name")
      .withRoleSessionDurationSeconds(
        (int)TimeUnit.MINUTES.toSeconds(300))
      .withScopeDownPolicy(iamPolicy)
      .build())
    .build();
}

```

```
private String getRoleArn(){
    // AssumeRoleを実行するために必要なロールのARN文字列を生成
    GetRoleRequest getRoleRequest = new GetRoleRequest().withRoleName("sample-role");
    roleArn = AmazonIdentityManagementClientBuilder.standard().build()
        .getRole(getRoleRequest).getRole().getArn();
}
```

このコードの中で、STSへAssumeRoleする処理を実装している箇所は以下になります。

```
return AmazonS3ClientBuilder.standard()
    .withCredentials(new STSShareSessionCredentialsProvider
        .Builder(getRoleArn(), "sample-role-session-name")
        .withRoleSessionDurationSeconds(
            (int)TimeUnit.MILLISECONDS.toSeconds(300))
        .withScopeDownPolicy(iamPolicy)
        .build())
    .build();
```

S3へアクセスするクライアントを組み立てる際に一時認証情報を利用しています。一時認証情報はSTSShareSessionCredentialsProviderを使って、引き受ける対象となるロールのARNとロールセッション名、有効期限を設定しています。

コード的には単にnewされているようにしか見えないのでわかりづらいですが、内部的にはこの処理により、STSに以下のようなリクエストが送信されることになります。

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=AssumeRole
&RoleSessionName=sample-role-session-name
&RoleArn=arn:aws::iam::123456789012:role/sample-role
&Policy=%7B%22Version%22%3A%222012-10-
17%22%2C%22Statement%22%3A%5B%7B%22Sid%22%3A%20
%22Stmt1%22%2C%22Effect%22%3A%20%22Allow%22%2C%22Action%22%3A%20%-
22s3%3A*%22%2C%22Resource%22%3A%20%22*%22%7D%5D%7D
&DurationSeconds=300
```

S3へのアクセスは、このリクエストのレスポンスで得られた一時認証情報を使って行います。またこの処理では、セッションポリシーとして、以下のポリシーを文字列で生成して、リクエストに含めています。

```
// Create ARN for download S3 Object.
String resourceArn = "arn:aws:s3:::sample-bucket/" + objectKey;
// Create IAM Policy provided temporary security credentials(Session
Policy).
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();
```

セッションポリシーは、前節「AWS IAM」の「IAMでの認証・認可」でも説明したとおり、リソースベースポリシーやアイデンティティベースポリシーと一緒に評価されるポリシーです。このコードでは、次のように実装することにより、セキュリティを確保した状態でファイルにアクセスさせています。

- ・セッションポリシーを動的に作成し、対象オブジェクトにのみアクセスできるGetObject権限を付与する
- ・セッションポリシーを含めてAssumeRole APIをコール

またこのコードでは、一時認証情報でアクセスするS3クライアントを生成したのち、最終的に300秒だけS3へアクセス可能な署名付きURLを生成しています(generatePresignedUrlメソッドを実行)。アプリケーションは一時認証情報が設定されたS3クライアントを使って、直接リソースへアクセスできます。また、アプリケーションのクライアントへこのURLを返却することで、クライアントからダイレクトにS3から対象のオブジェクトを取得することも可能になります。

このURLは、一時認証情報が付与されたHTTPリクエストであり、一般に以下のようない形式で生成されます。

```
https://sample-bucket.s3.ap-northeast-1.amazonaws.com/Sample.txt?
X-Amz-Security-Token=XXXXXXX
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Date=20210717T182529Z
&X-Amz-SignedHeaders=host
&X-Amz-Expires=300
&X-Amz-Credential=ASIAJMKZL34VT64RL4I2%2F20210717%2Fap-northeast-
1%2Fs3%2Faws4_request
&X-Amz-Signature=94e21fc6f36e41a9eb0f4b8875d70af530122fb7e61940a690bcf
3bac037cb1e
```

X-Amz-Credentialパラメータが一時認証情報であり、X-Amz-Signatureが「署名バージョン4を使用したAWSリクエストへの署名^{※16}」になります。詳細な説明は割愛します

※16 https://docs.aws.amazon.com/ja_jp/general/latest/gr/sigv4_signing.html

が、認証情報はリクエストパラメータに指定するほか、HTTP Authorizationヘッダに設定することもできます。認証情報には次に示す情報が含まれます。

- ・署名に使用したアルゴリズム (AWS4-HMAC-SHA256)
- ・認証情報スコープ (アクセスキー ID を含む)
- ・署名付きヘッダの一覧
- ・計算された署名。この署名はAWSシークレットアクセスキーを使用して署名を生成する。この署名により、プリンシバルの正当性が保証される



認証パラメータはリクエストパラメータもしくは、Authorizationヘッダいずれかに設定する方法が選択できますが、セキュリティを考慮して、パラメータ値が記録されないAuthorizationヘッダへの設定が推奨されます。上記では、解説のため、リクエストパラメータに付与するリクエストを例示しています。



シークレットアクセスキーで署名を生成するロジックは、例えば、Java言語では、以下のように実装されます^{※17}。ミスが起こりやすく、またテストもしやすい内容のため、アプリケーション実装時は注意してください。

```

/**
 * X-Amz-Credentialに設定する文字列を作成する
 * 書式: https://docs.aws.amazon.com/AmazonS3/latest/API/sigv4-query-string-
auth.html
 * @param accessKey
 * @param date
 * @param region
 * @param serviceName
 * @return
 */
private String createCredentialString(String accessKey, String date,
String region, String serviceName){
    return new StringBuilder()
        .append(accessKey).append("/")
        .append(date).append("/")
        .append(region).append("/")
        .append(serviceName).append("/aws4_request")
        .toString();
}

/**
 * シークレットキーから署名を作成するメソッド
 * @param key
 * @param dateStamp
 * @param region
 * @param serviceName
 * @return
 */
private byte[] getSignatureKey(String key, String dateStamp, String region,
String serviceName){
    byte[] kSecret = new StringBuilder().append("AWS4").append(key).
        toString()
        .getBytes(StandardCharsets.UTF_8);
    byte[] kDate = calculateHmacSHA256(dateStamp, kSecret);
    byte[] kRegion = calculateHmacSHA256(region, kDate);
    byte[] kService = calculateHmacSHA256(serviceName, kRegion);
    byte[] kSigning = calculateHmacSHA256("aws4_request", kService);
    return kSigning;
}

/**
 * 署名キーを使って、指定したアルゴリズム(HMAC)でハッシュ操作するメソッド
 * @param stringToSign
 * @param signingKey
 * @return
 */
private byte[] calculateHmacSHA256(String stringToSign, byte[] signingKey){
    Mac mac = null;
    try{
        mac = Mac.getInstance("HmacSHA256");
        mac.init(new SecretKeySpec(signingKey, "HmacSHA256"));
    }catch (NoSuchAlgorithmException | InvalidKeyException e){
        e.printStackTrace();
    }
    return mac.doFinal(stringToSign.getBytes(StandardCharsets.UTF_8));
}

```

3-3 AWS KMS (Key Management Service)

KMSはデータを暗号化するための鍵を管理するサービスです。暗号鍵（以降、キー）はAWSの責任で管理され、可用性やセキュリティが保証されています。KMSはEBSやRDS、S3などと統合されており、それぞれのサービスで保存されているデータを暗号化できます。

1 KMSの概念と重要な用語

そもそも暗号化自体難解で複雑なものであり、さまざまなユースケースで似通った方があって開発者を混乱させます。まず、KMSで登場する重要な用語とその概念を整理しておきましょう。

●暗号化

暗号化とは、意図しない第三者からデータが読み取られることを防ぐ技術ですが、主に以下の2つが対象となります。KMSでは「保存データの暗号化」を対象としています。

【暗号化の対象】

通信の暗号化	保存データの暗号化
通信する経路を暗号化してデータを保護する。SSL/TLSやIPsecなどを使った技術が代表的。	保管されたデータを暗号化してデータを保護する。共通鍵暗号方式と、公開鍵暗号方式、双方を組み合わせたハイブリッド暗号化方式がある。キーを使って、ファイルシステムやデータベースのデータをAESやRSAなどの暗号化アルゴリズムにより暗号化処理を行う。

●暗号化キー

KMSは、「保存データを暗号化」するためのキーを扱うサービスです。KMSで扱われるキーも複数種類があり、管理元や用途によって以下のように分類できます。

【キーの種類】

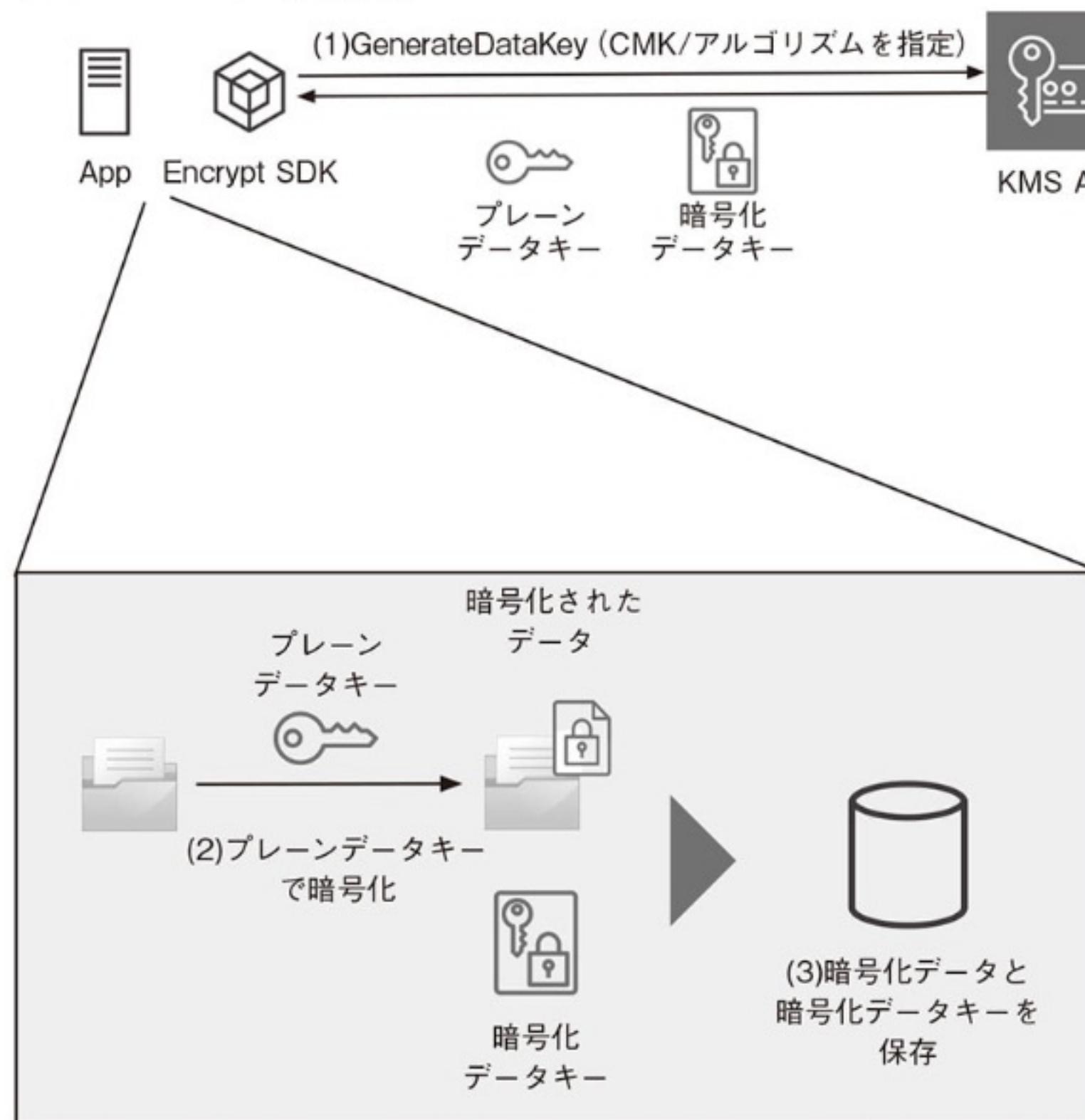
キー	説明
カスタマー マスターキー (Customer Master Key : CMK)	KMSで管理する、ユーザがKMSおよびAWSの外で作成したキー。KMSでキー権限やIAMポリシーを付与し、アクセス制御を行う。キーの有効化・無効化することもできる。KMSで生成したキーは1年間で自動ローテーションする。
	AWSマネージドCMK
	AWS所有CMK
カスタマー データキー (Customer Data Key : CDK)	後述するエンベロープ暗号化で用いられるAES共通鍵。暗号化する際は平文（ブレーンテキストな）形式で使用し、復号のために暗号化した形式で保存しておく。
	データキーペア (非対称キー)
	後述するエンベロープ暗号化で用いられる公開鍵・秘密鍵のペア。デジタル証明書で一般的に使われるRSAキーペアと、デジタル署名や暗号通貨などで使用される省データ高強度の椭円曲線（Elliptic Curve Cryptography : ECC）キーペアなどをサポート。

●エンベロープ暗号化

KMSで取り扱うカスタマー管理CMKでは、KMS APIを呼び出して直接暗号化・復号することもできます。ただし、暗号化可能なデータサイズが4KBまでに制限され、そもそもアプリケーションなど一般に扱うデータを暗号化するように設計されていません。そのため、基本的には、エンベロープ暗号化という方式によって、データの暗号化処理を実装します。

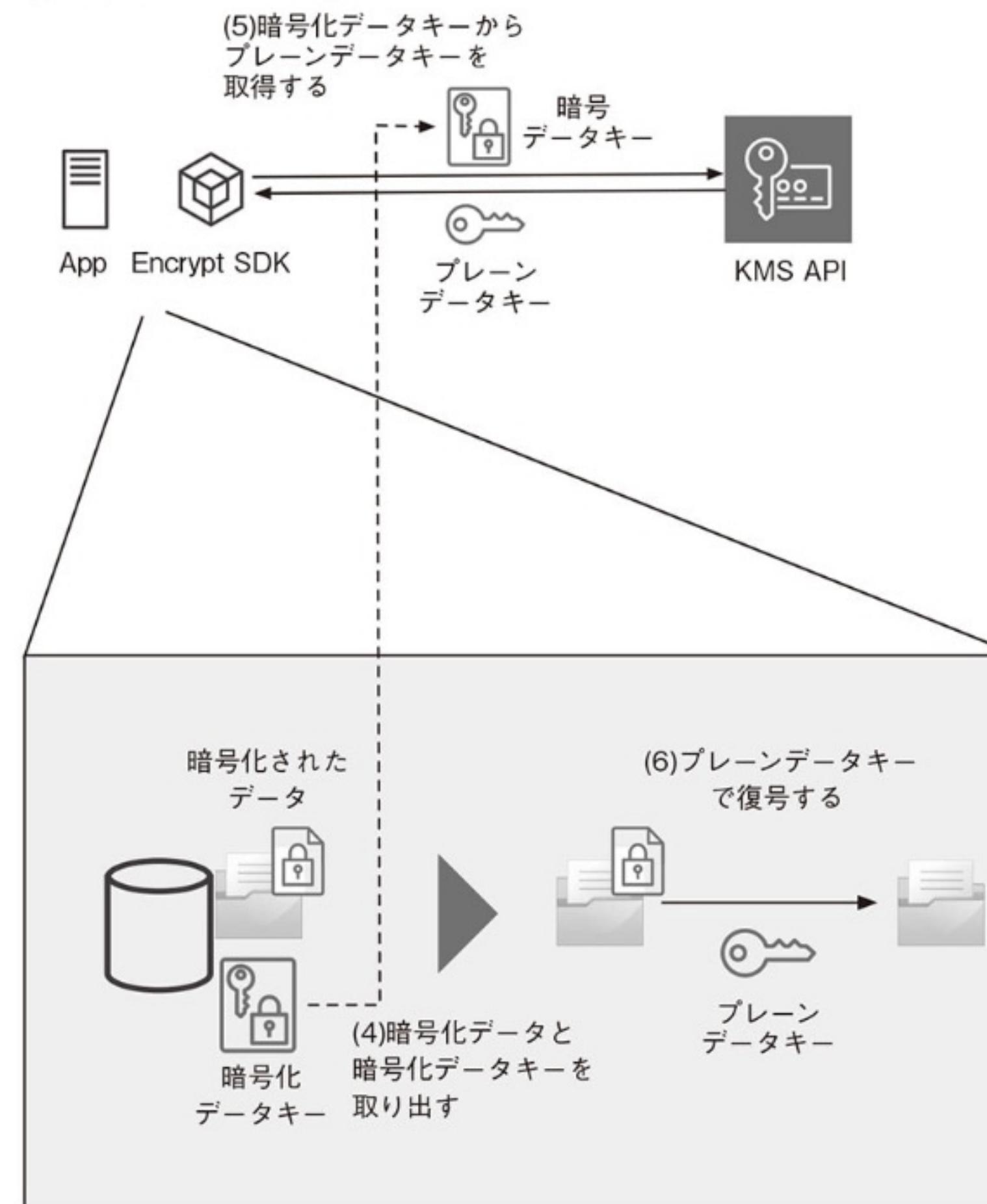
エンベロープ暗号化では、カスタマー管理CMKを使って、指定された暗号化アルゴリズムをもとにデータキーを生成します。具体的には次のようなフローで暗号化・復号を行います。

【エンベロープ暗号化】



復号する場合のプロセスは以下のとおりです。

【エンベロープ復号】



1. CLIやSDKから、KMS GenerateDataKey APIに対し、カスタマー管理CMKを指定して呼び出すと、plainなデータキーおよび暗号化されたデータキーが返されます。
2. 受け取ったplainデータキーを使って、アプリケーションはデータを暗号化します。
3. 暗号化したデータは、復号のために暗号化データキーとともに保存します。なお、暗号化に使用したplainデータキーはすぐに破棄することが推奨されています。

4. アプリケーションは保存した暗号化データキーを取り出します。
5. KMS Decrypt APIに渡して呼び出すと、破棄したものと同じplainデータキーを再び入手できます。
6. plainデータキーでデータを復号します。

2 KMSの機能

上述した暗号化のプロセスの中で、KMSが提供する役割や機能を改めて整理すると以下のようになります。一連の機能はKMS APIを呼び出すことにより実行されます。

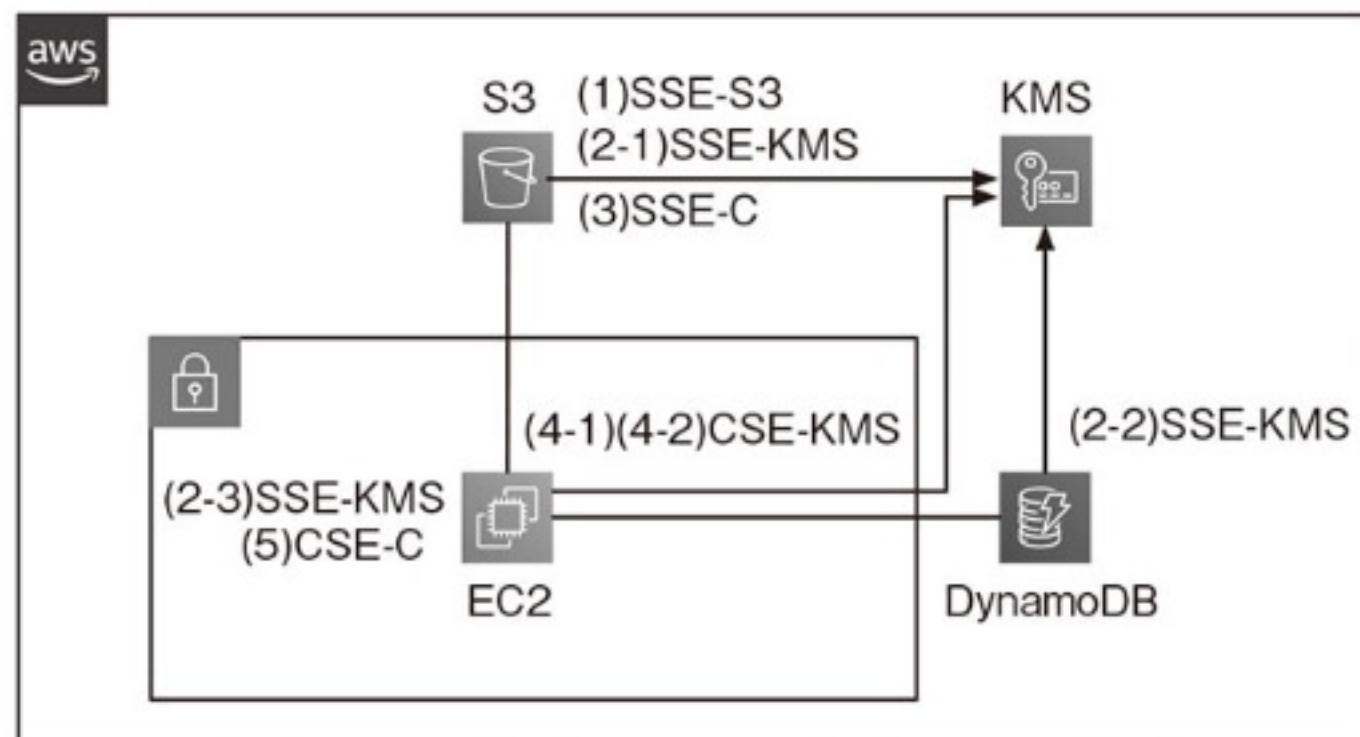
【KMSが提供する機能】

機能	説明
鍵の生成	単一のエイリアスで識別されるカスタマーマスターキー (CMK) の作成
鍵のインポート	AWS外部で作成したCMKのインポート (Bring Your Own Key : BYOK)。なお、キー自体のデータをキーマテリアルと呼び、CMKはキーマテリアル、有効期限やオリジンといったメタデータ、識別子から構成される。キーマテリアルは主に共通鍵として用いられる対称キーのみがサポートされる。なお、re:Invent 2022の発表より、外部の鍵管理システムと統合するExternal Key Store (XKS) 機能も提供されるようになった
鍵へのアクセス管理	<ul style="list-style-type: none"> ・鍵を管理するIAMユーザおよびロールの定義 ・鍵を使用するIAMユーザおよびロールの定義 ・キー policyによるアクセス制御
鍵管理	CMKの無効化・有効化・削除・ローテーション

3 AWSで実行される保存データ暗号化の主なユースケース

KMSの使用有無にかかわらず、AWSでは以下の図のようなユースケースで暗号化が実行されます。

【保存データの暗号化の主な例】



各ユースケースと暗号化の方法を整理すると次のようにになります。

【AWSでの主な保存データ暗号化の方法とユースケース】

暗号化の方法	説明・ユースケース
サーバサイド暗号化 (Server Side Encryption : SSE)	<ul style="list-style-type: none"> (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。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ケースで利用する。 (2) SSE-KMS (注1) <ul style="list-style-type: none"> (2-1) S3 S3へファイルアップロードする際に、KMSが管理するキーを使った、S3上で実行される暗号化。暗号化自体は(1)と機能的に同等だが、暗号化したファイルへの監査ログを残したり(注2)、アクセスポリシーを定義してクロスアカウントアクセスするケースで利用する。 (2-2) DynamoDB DynamoDBへのデータの書き込み時に、KMSが管理するキーを使った、DynamoDB上で実行される暗号化。なお、暗号化に使用するキーはローテーション期間が異なるカスタマー管理CMKとAWSマネージドCMK、いずれかをオプションで選択できる。 (2-3) EBS EBSボリュームに対して実行する暗号化。EC2インスタンス起動時の暗号化EBSボリュームのアタッチ・デタッチや、スナップショットを暗号化する際、EC2からKMSの暗号化・復号APIが実行される。なお、KMSはリージョン間でキーの共有はできないため、別のリージョンに暗号化されたデータを単純に複製・復号できない点に注意したい。また、1つのEC2インスタンスに対して、同一のCMKで暗号化されたEBSボリュームをアタッチできる数には制限がある。 (3) SSE-C S3へファイルアップロードする際に、ユーザ側で用意したキーを使った、S3上で実行される暗号化^{※18}。AWSによる管理キーが認められないケースで利用する。なお、ここでユーザ側が用意するキーとは暗号化処理を実行する際に指定する任意のキーを指す(カスタマー管理CMKとして登録したBYOKキーのことではない)。



サーバサイド暗号化では、ユーザが意識することなく、透過的に暗号化・復号処理が実行されます。

※18 https://docs.aws.amazon.com/ja_jp/AmazonS3/latest/userguide/specifying-s3-c-encryption.html

暗号化の方法	説明・ユースケース
クライアントサイド暗号化 (Client Side Encryption : CSE)	<p>(4) CSE-KMS</p> <p>(4-1) エンベロープ暗号化 KMSで管理されているキーを使用して、前項で解説したエンベロープ暗号化方式による、クライアントサイド（EC2などサーバがクライアントとなる場合もある）で実行する暗号化。主にアプリケーションで特定のデータに暗号化処理を施したいケースで利用する。AWS Encryption SDKなどを用いて実装する。235ページでも解説したとおり、エンベロープ暗号化でのデータキーとして、対称キーと非対称キーを選択できる。</p> <p>(4-2) ダイレクト暗号化 カスタマー管理CMKを使用して、KMS APIでダイレクトに暗号化したいケースで利用する。暗号化可能なファイルサイズは4KBまでとなるが、この方式は、エンベロープ暗号化と異なり、暗号化されたデータキーを外部に保存しなくてもよいので、少量のデータの暗号化・保存するには有用。</p> <p>(5) CSE-C ユーザ側で用意したキーを使った、クライアント側で実行する暗号化。AWS Encryption SDKなどを用いて実装する。例えば、S3へアップロード前に暗号化処理を施したいケース^{*19}で利用する。なお、ここでユーザ側が用意するキーとは暗号化処理を実行する際に指定する任意のキーを指す（カスタマー管理CMKとして登録したBYOKキーのことではない）。</p>

(注1) SSE-KMSの代表的な例として、S3、DynamoDB、EBSなどを（2-1～2-3で）列挙していますが、AWSの多数のサービスがこのSSE-KMSによる暗号化を利用可能です。

(注2) KMSで管理される暗号化の全操作はCloudTrailで記録されます。



暗号化・復号処理が短期的に集中して行われた場合、KMS APIのスロットルエラー（ThrottlingException）が発生することがあります。KMS APIのリクエスト上限を引き上げたり、タイミングをずらして再度実行したりするとエラーが解消する場合があります。AWS SDKを利用したサービス呼び出しでは、エラーが発生するたびに呼び出し間隔を指数関数的に延ばしてリトライする「エクスponentialバックオフ」が標準で組み込まれています。



ユースケースや目的に応じて、最もふさわしい暗号化の方法を問われる場合があります。KMSを用いた場合のメリットや制約を理解した上で、適切な手法を選択できるようにしておきましょう。

*19 https://docs.aws.amazon.com/ja_jp/AmazonS3/latest/userguide/UsingClientSideEncryption.html

3-4 Amazon Cognito

Amazon Cognitoは、Webアプリケーション、モバイルアプリケーションに認証機能を提供するAPIベースで実装されたサービスです。Cognitoの主要な機能は、ユーザディレクトリサービスである「ユーザープール」と、認証されたユーザに対し権限を付与する機能を持つ「IDプール」があります。

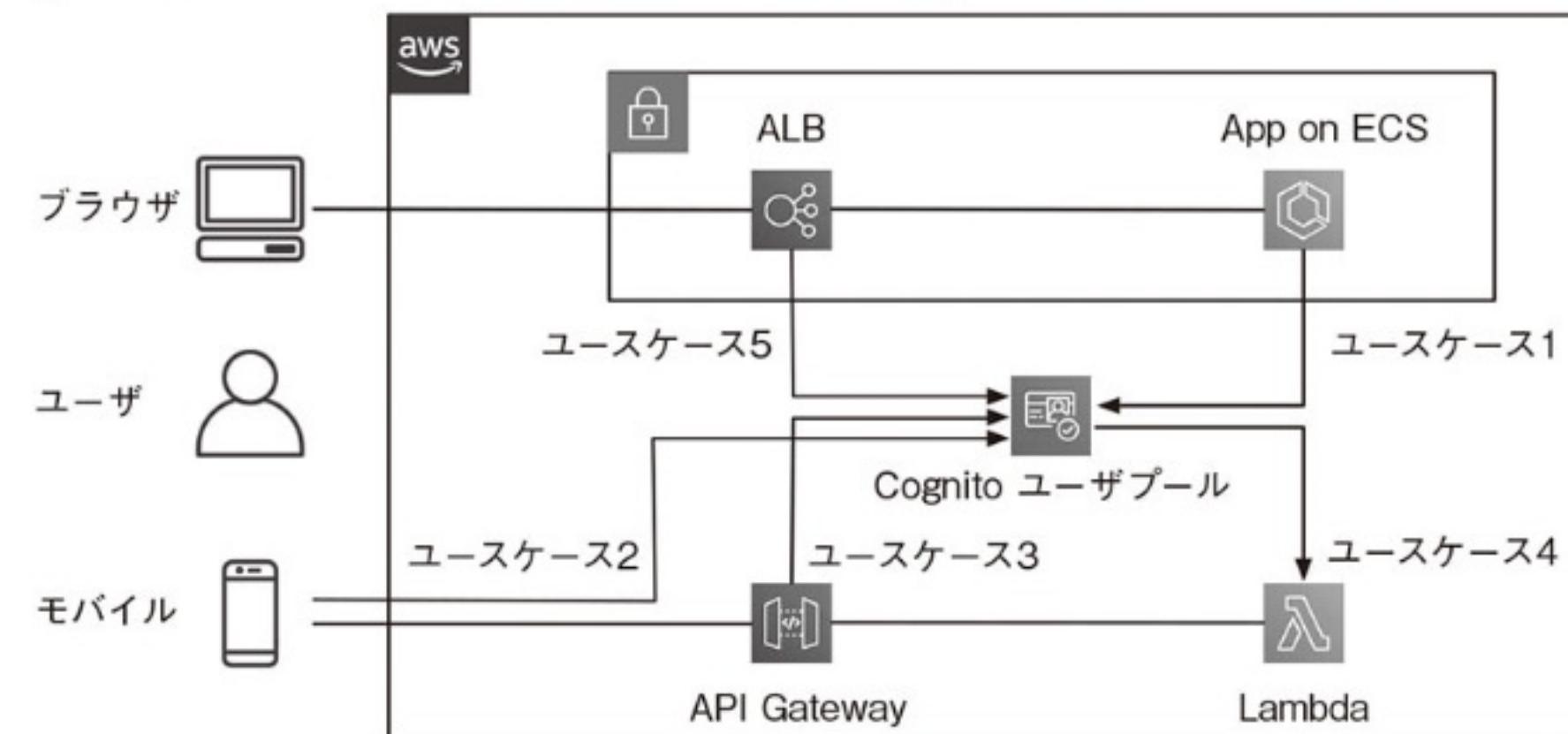
1 ユーザープールの概要とユースケース

ユーザープールは、ユーザデータベースに相当するディレクトリサービスです。AWSマネージドであるため、サーバを立てて構築する必要がありません。Cognitoでは開発者が作成したアプリケーションをはじめ、AWSのさまざまなサービスへ認証機能を提供します。Cognitoが提供するユーザープールの主要機能は次のとおりです。

- ・ユーザディレクトリとユーザプロファイルの管理
- ・サインアップ、サインイン（メールアドレスも可）
- ・外部IDプロバイダとの連携
- ・パスワードポリシーの設定
- ・多要素認証（Multi Factor Authentication : MFA）
- ・電話番号やEメールアドレスの有効性の検証
- ・Lambdaトリガーによるサインインなどのワークフローのカスタマイズ

Cognitoユーザープールを使用する、5つの代表的なユースケースは次のとおりです。

【ユーザープールの代表的なユースケース】



1. EC2 や ECS 上に配置した Web アプリケーションの認証

開発者が実装したWebアプリケーションへのユーザのサインアップ・サインインでユーザープールを利用します。

2. モバイルアプリケーションの認証

開発者が実装したモバイルアプリケーションへのユーザのサインアップ・サインインでユーザープールを利用します。認証後は払い出されるトークンを使用し、API Gateway経由でAWS内のリソースにアクセスします。

3. API Gateway からの Cognito オーソライザ

API Gatewayが受けたリクエストの認証で利用します。リクエストはモバイルからのみに限らず、サードパーティサービスなどもあり得ます。ユーザープールはAPI Gatewayからトークンを検証し、後述するIDプールと連携してLambdaやECSといったAWSリソースへアクセスします。

4. AWS Lambda を使ったカスタムオーソライザ

CognitoはLambdaを使って、サインインのワークフローをカスタマイズすることもできます^{※20}。矢印が逆であることに注意してください。Cognitoからイベント記録などLambda関数をトリガーします。

5. ALB からの Cognito オーソライザ

ALBに送信されてくるリクエストの認証で利用します。ユースケース3と同様、ユーザープールは送信されてきたトークンをもとに、後述するIDプールと連携してLambdaやEC2、ECSといったAWSリソースへアクセスします。

2 ユーザープールの利用方法

Cognitoの具体的な利用方法を見ていきましょう。開発者はまず、ユーザディレクトリを定義し、サインアップの方法やパスワードポリシー、MFA（多要素認証）などを設定します。

【ユーザープールの定義】

ユーザープール | フェデレーション・アイデンティティ
ユーザープールを作成する

名前
属性
ポリシー
MFAとして確認
メッセージのカスタマイズ
タグ
デバイス
アプリケーション
トリガー
確認

エンドユーザーをどのようにサインインさせますか?

Eメールアドレス、電話番号、ユーザー名、または任意のユーザー名とパスワードを使用してユーザーがサインインする選択ができます。詳細はこちら。
 ユーザー名 - ユーザーは、ユーザー名を使用するか、オプションで複数の選択肢を使用してサインアップおよびサインインできます。
 検証済みのEメールアドレスでのサインインも許可
 検証済みの電話番号でのサインインも許可
 任意のユーザー名(ユーザーが変更できるユーザー名)でのサインインも許可

Eメールアドレスおよび電話番号 - ユーザーは、Eメールアドレスまたは電話番号を「ユーザー名」として使用してサインアップおよびサインインできます。
 Eメールアドレスを許可
 電話番号を許可
 Eメールアドレスと電話番号の両方を許可(ユーザーは1つを選択できます)

選択したサインインオプションのユーザー名入力で大文字と小文字を区別しないよう選択できます。たとえば、このオプションを選択すると、ユーザー名は「username」または「Username」のいずれでもサインインできます。

(推奨) ユーザー名入力で大文字と小文字を区別しないことを有効に

どの標準属性が必要ですか?

すべての標準属性をユーザープロファイルで使用できますが、選択した属性はサインアップのために必要です。プールの作成後に、これらの要件を変更することはできません。属性を選択してエイリアスにすると、ユーザーはその番号またはユーザー名を使用してサインインできます。属性の詳細をご覗ください。

必須	属性	必須	属性
<input type="checkbox"/>	address	<input type="checkbox"/>	nickname
<input type="checkbox"/>	birthdate	<input type="checkbox"/>	phone number
<input checked="" type="checkbox"/>	email	<input type="checkbox"/>	picture
<input type="checkbox"/>	family name	<input type="checkbox"/>	preferred username
<input type="checkbox"/>	gender	<input type="checkbox"/>	profile
<input type="checkbox"/>	given name	<input type="checkbox"/>	timezone

3

重要な設定の1つが「アプリケーション」の項目です。「アプリケーション」とは、Cognitoに認証を求めてくるアプリケーションクライアントのことを指しており、サーバサイドのWebアプリケーションか、モバイルで実行されるアプリケーションかで、推奨される認証フローの設定内容が異なります。

【アプリケーションの定義】

ユーザープール | フェデレーション・アイデンティティ
ユーザープールを作成する

名前
属性
ポリシー
MFAとして確認
メッセージのカスタマイズ
タグ
デバイス
アプリケーション
トリガー
確認

このユーザープールへのアクセス権限があるアプリケーションはどれですか?

以下の追加したアプリケーションには、このユーザープールにアクセスするための固有のIDとオプションのシークレットキーが付与されます。

アプリケーション名

トークンの有効期限を更新
 日と 分
60分～3650日の間に必ずあります。

アクセストークンの有効期限
 日と 分
5分～1日の間に必ずあります。更新トークンの有効期限を超えることはできません。

IDトークンの有効期限
 日と 分
5分～1日の間に必ずあります。更新トークンの有効期限を超えることはできません。

クライアントシークレットを生成

認証フローの設定

認証用の管理APIのユーザー名/パスワード認証を有効にする(ALLOW_ADMIN_USER_PASSWORD_AUTH) 詳細はこちら。

Lambdaトリガーベースのカスタム認証を有効にする(ALLOW_CUSTOM_AUTH) 詳細はこちら。

ユーザー名/パスワードベースの認証を有効にする(ALLOW_USER_PASSWORD_AUTH) 詳細はこちら。

SRP(セキュアリモート/パスワード)プロトコルベースの認証を有効にする(ALLOW_USER_SRPP_AUTH) 詳細はこちら。

更新トークンベースの認証を有効にする(ALLOW_REFRESH_TOKEN_AUTH) 詳細はこちら。

セキュリティ設定

ユーザー存在エラーを防ぐ 詳細はこちら。

レガシー
 有効(推奨)

※20 https://docs.aws.amazon.com/ja_jp/cognito/latest/developerguide/cognito-user-identity-pools-working-with-aws-lambda-triggers.html

認証フローの設定内容は以下のとおりです。

●認証用の管理 API のユーザ名パスワード認証を有効にする (ALLOW_ADMIN_USER_PASSWORD_AUTH)

CLIやSDKを使って管理者ユーザとして認証を処理する場合のオプションです。リクエストにパスワードパラメータが含まれるため、安全なサーバサイドのWebアプリケーションがCognitoを使って認証する場合に利用します。

●Lambda トリガーベースのカスタム認証を有効にする (ALLOW_CUSTOM_AUTH)

Lambdaでカスタム認証を行う場合に有効化します。

●ユーザ名パスワードベースの認証を有効にする (ALLOW_USER_PASSWORD_AUTH)

CLIやSDKを使って認証の処理を行う場合のオプションです。リクエストにパスワードパラメータが含まれるため、モバイルアプリケーションで使用する場合は、より安全性の高いALLOW_USER_SRPPASSWORD_AUTHの使用が推奨されます。

●SRP (セキュアリモートパスワード) プロトコルベースの認証を有効にする (ALLOW_USER_SRPPASSWORD_AUTH)

Secret Saltやチャレンジレスポンスなどでよりパスワード交換の安全性を高めたかたちで認証を行います。

●更新トークンベースの認証を有効にする (ALLOW_REFRESH_TOKEN_AUTH)

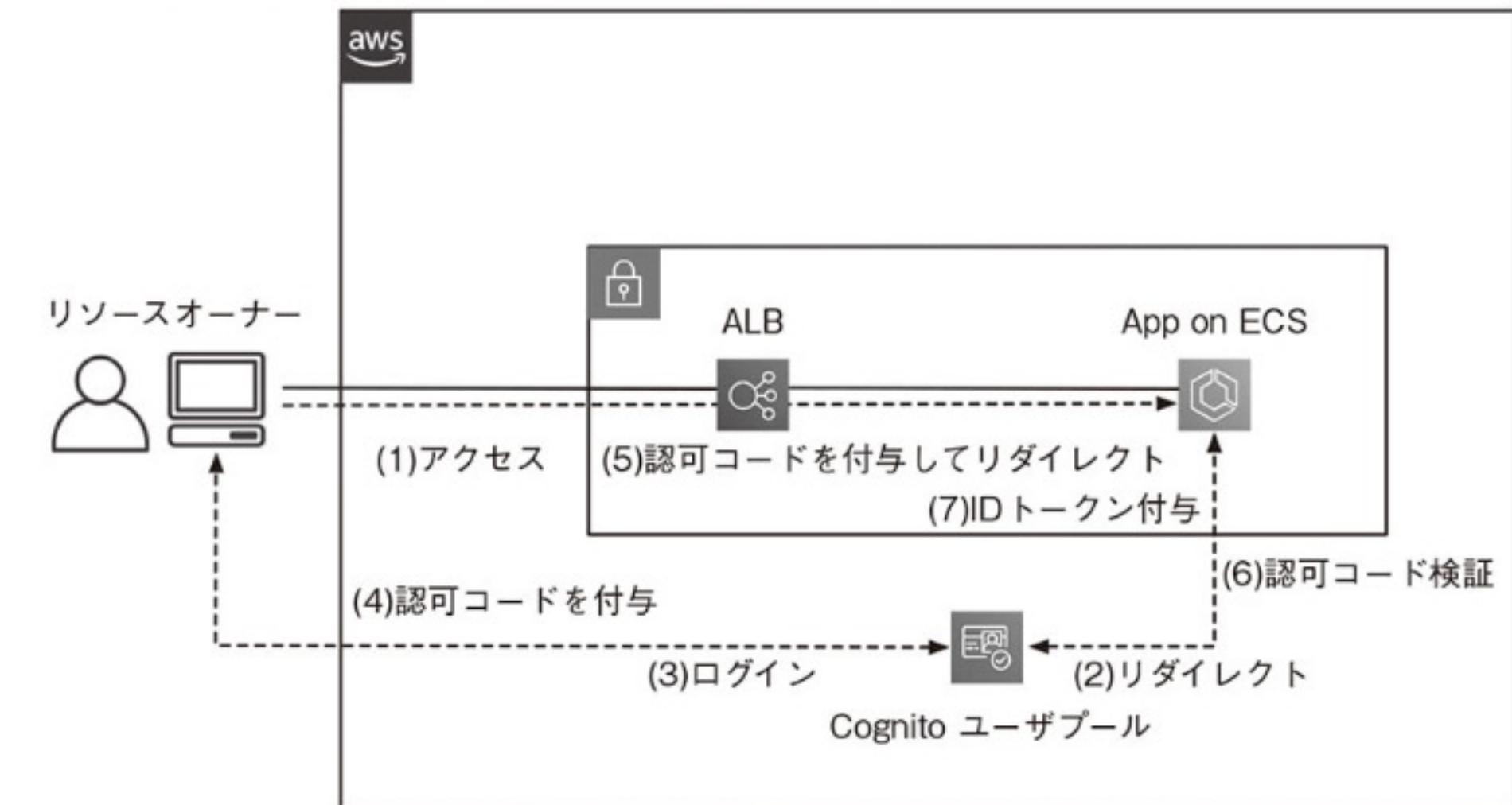
リフレッシュトークンを使って、認証を行う場合のオプションです。コンソール上からは必須選択されます。

また、先の設定画面では「クライアントシークレットを生成」というオプションが用意されています。クライアントシークレットはアプリクライアントの正当性を検証する場合のパスワードに相当するパラメータです。デコンパイルなどにより、不特定多数の攻撃者に参照される可能性があるモバイルアプリケーションでの使用は推奨されません。

3 ユーザープールとアプリケーションの統合

Webアプリケーションやモバイルアプリケーションでは、上述したSDKを使ったAPI認証に加えて、ユーザープールを認証サービスとして利用することができます。

【ユーザープールの代表的なユースケース】



Cognitoでは、OIDC (OpenID Connect) の仕様が定めるいくつかのエンドポイントが提供されており^{※21}、OIDC/OAuth 2.0のフローに則ったIDトークンおよびアクセストークンを発行することができます。

本書の対象から外れるので、ここではOIDC/OAuth 2.0についての説明は省略しますが、上記の図では、OIDCの認可コードフローを使って、アプリケーションから認証をCognitoに委譲するかたちでアプリケーションとCognitoを統合できます。OIDC準拠のIDプロバイダから発行されたトークンを使って、Cognitoをプロバイダとするさまざまなアプリケーションと一元的なアクセス（シングルサインオン）が可能となります。

※21 <https://wwwopenid.or.jp/document/#op-doc-openid-connect>

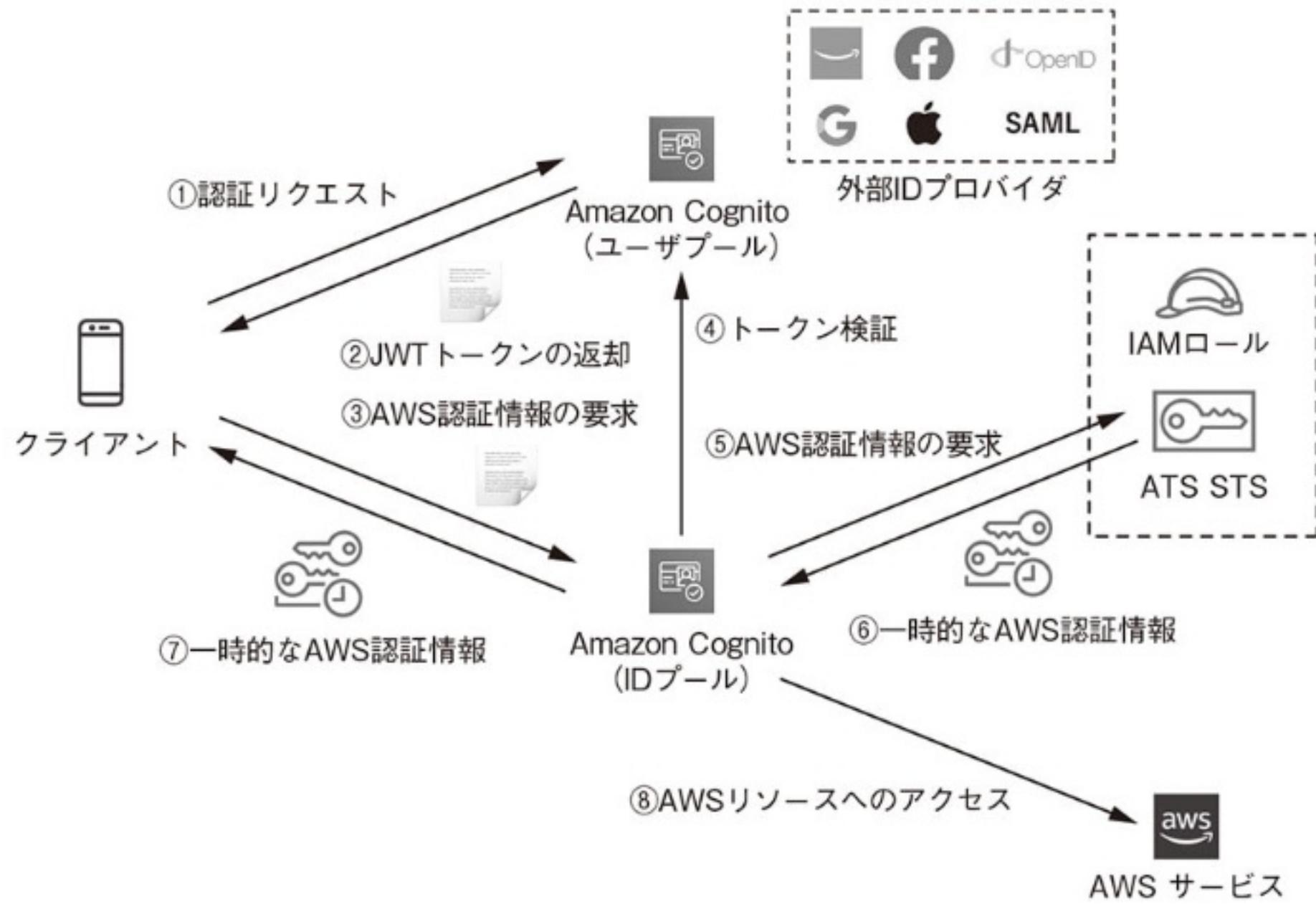
4 ユーザープールと外部フェデレーション

Cognitoでは、外部フェデレーテッドIDプロバイダ（さまざまなサービスでユーザ情報を利用できるように構成しているIDプロバイダ）と統合できます。主なプロバイダとして、FacebookやGoogle、Amazon、Apple、SAML、OpenID Connectプロバイダなどがあり、これらのユーザ情報を利用するには、Cognito認証プロバイダに情報を渡す必要があります。そのため、ユーザープールで定義した属性とマッピング設定をする必要があります。

5 IDプールの概要

IDプールでは、ユーザープールや外部IDフェデレーションで認証したユーザに対して、AWSサービスへのアクセスを認可する機能を提供します。

【ユーザープールの代表的なユースケース】



Cognitoユーザープールおよび外部IDプロバイダへの認証に成功するとJWT (JSON Web Token) ベースのトークンが発行されます。アプリケーションクライアントは、IDプールに対して認証情報を渡して、IDプールはトークンの有効性を確認します。問題がなければ、IDプールに紐付いたIAMロールの権限にもとづく一時的なAWS認証情報をAWS STSが払い出します。これで、クライアントはAWSリソースへのアクセスが可能となります。



JWTについて

JWTはRFC7519で定められた署名付きの認証情報を含むトークンです。JWTは以下の例のとおり、ヘッダ・ペイロード・署名をそれぞれBase64エンコードしてピリオドでつなぎ合わせたかたちで構成されます。

【ヘッダ文字列】: {"kid":"sample", "alg":"RS256"}

【ペイロード文字列】: {"sub":"sample-sub", "event_id":"xxxx", "token_use":"access", "scope":"openid profile", "auth_time":1629986340, "iss":"https://cognito-idp.ap-northeast-1.amazonaws.com/ap-northeast-1_xxxxxxx", "exp":1629989940, "iat":1629986340, "version":2, "jti":"xxxxxx", "client_id":"xxxxxxxx", "username":"taro.sample"}

【署名】: RS256(base64urlEncoding(上記のヘッダ文字列) + '.' + base64urlEncoding(上記のペイロード文字列), secret)

JWT

eyJraWQiOijzYW1wbGUiLCJhbGciOiJSUzI1NiJ9.eyJdWliOijzYW1wbGUtc3ViliwiZXZlbnRfaWQiOij4eHh4liwidG9rZW5fdXNlljoiYWNjZXNzliwic2NvcGUiOijvcGVuaWQgcHJvZmlsZSlsmF1dGhfdGltZSI6MTYyOTk4NjM0MCwiaXNzIjoiaHR0cHM6XC9cL2NvZ25pdG8taWRwLmFwLW5vcnRoZWfzdC0xLmFtYXpvbmF3cy5jb21cL2FwLW5vcnRoZWfzdC0xX3h4eHh4eHgiLCJleHAiOjE2Mjk5ODk5NDAsImhdCI6MTYyOTk4NjM0MCwidmVyc2lvbi6MiwanRpIjoieHh4eHh4liwiY2xpZW50X2lkjioieHh4eHh4eCIsInVzZXJuYW1lljoidGFyby5zYW1wbGUifQ0K.X_J0h2HQ1NbVyhjqXlkDF7AMP0fmJkVsCvJMv-xzClSQdUsJMGtIGSyX86bqfUZORUpZO1NO1bxF0gUJYTTWAVRhUfN7Pt24Ogp7M20ps9PE9RTU2PuyeGgXARQVXqZuuAWE06_O8Xm0onuipti5vp7wxl2tagSpun5e2kKXqkSOZnCKwMlwqjRA1jU9yoB_ljwzzoz3l9NstqN9mlyrUWWHux8vJNqZJQwEcrxgbUvFybd1bZDdj3Q2Z1ICyzRN34xfX-DUE1BKW-8MXWDudezv0VpN1K0ku6gH4hfJrhSncZ5bKla8wuTaK4o9xLIXGTF3oHuqpn-ASNjKyeh5w

JWTが有用な点は、「署名がトークン生成元の秘密鍵でエンコードされており、ヘッダやペイロードから情報が正しいか、対となる公開鍵で複合して、署名を検証できること」にあります。発行元にトークンが正しいか問い合わせをしなくても、公開鍵でトークンが正しい発行元から生成されたものか検証することができます。ただし、JWTは有効期限付きの合鍵のようなもので、それを持っていれば、期限内であれば誰でもアクセスできてしまいます。悪意のある攻撃者はJWT形式で構成されている、OIDCのIDトークンや、OAuth 2.0のアクセストークンの奪取を基本的な目的とするケースが多いです。トークンは十分注意して取り扱うようにしてください。

6 IDプールの作成

IDプールでは、権限を付与するユーザープールのアプリクライアントおよび、認証されていないユーザのアクセス可否、認証・非認証ユーザにそれぞれ割り当てる権限を設定して構成します。

【IDプールの作成】

使用開始ウィザード*

新しい ID プールの作成

ID プールはエンドユーザー ID を保存するために使用されます。新しい ID プールを宣言するには、一意の名前を入力します。

ID プール名*
例: My App Name

認証されていない ID の
Amazon Cognito は、ID プロバイダーを使って認証を行わないユーザー用に、一意の識別子や AWS 認証情報を提供して、認証されていない ID をサポートできます。アプリケーションで、ログインすることなくユーザーにアプリケーションの使用を許可している場合、認証されていない ID に対してアクセスを有効にできます。認証されていない ID の詳細を参照してください。

認証されていない ID に対してアクセスを有効にする

このオプションを有効にすると、インターネットアクセスが可能なユーザー全員に AWS 認証情報を付与することができます。認証されていないアイデンティティとは通常、お客様のアプリケーションにログインしていないユーザーのことです。通常、認証されていないアイデンティティに割り当てられるアクセス許可のほうが、認証されたアイデンティティに割り当てられるアクセス許可よりも制限が厳しくなります。

認証フローの設定
Amazon Cognito を使用したユーザーの認証は、認証情報をブートストラップする複数のステップからなるプロセスを通じて行われます。Amazon Cognito には、パブリックプロバイダーとの認証のための拡張と基本という 2 つの異なるフローがあります。Cognito では、拡張された認証フローの使用を推奨しています。ただし、基本的なフローを使用する場合は、ここで有効にすることができます。認証フローの詳細をご確認ください。

基本(クラシック)フローを許可する

認証プロバイダー
Amazon Cognito は、Amazon Cognito サインインまたはすべてのパブリックプロバイダーで、以下の認証方法をサポートしています。これらのパブリック ID プロバイダーを使用した認証をユーザーに許可している場合、ここでアプリケーション ID を指定できます。警告: ID プールがリンクされたアプリケーション ID を変更すると、既存のユーザーは Amazon Cognito を使用して認証できなくなります。パブリック ID プロバイダーの詳細を参照してください。

Cognito Amazon Apple Facebook Google+ Twitter / Digits OpenID SAML カスタム

ユーザー プール ID およびアプリクライアント ID を指定して、Cognito ユーザープールとフェデレーションされたユーザーを許可するよう Cognito ID プールを設定します。

ユーザー プール ID 例: us-east-1_Ab123faBb
アプリクライアント ID 例: 7890kfbfb4q5kpp90urffao

別のプロバイダーの追加
キャンセル プールの作成

*必須

3-5 その他のセキュリティ関連サービス

AWSをはじめとしたパブリッククラウドでは、インターネットからダイレクトにアクセスできる利便性を持つ一方で、さまざまなセキュリティ対策を施す必要があります。開発の中で意識しておくべきセキュリティ対策やAWSが提供するセキュリティサービスには何があるか押さえておきましょう。

1 セキュリティ脅威とクラウドにおけるリスク

情報システムにおけるセキュリティ脆弱性を突いた攻撃とその対応策はイタチごっこの世界でもあります。ウイルスを含んだ添付ファイルを送信するメールや標的型攻撃といった特定のクライアントをターゲットにしたものから、DDoS攻撃、バッファオーバーフロー、クロスサイトスクリプティングなどアプリケーションの不備を突くものなど、セキュリティに関する脅威は多岐にわたります。

代表的なセキュリティの脅威は以下のとおりです。

●情報システムへの進入・情報奪取・破壊

バックドアなどの手法を用いてシステムへアクセスし、機密情報を不正に取得したり、システムを破壊したりする脅威があります。

●情報システム自体への攻撃

DDoS攻撃やSYNフラッド攻撃、ボットなどの手法を用いて、システム自体のアクセスを困難にしたり、使用不能にしたりする脅威があります。

●マルウェア・ウイルス感染

標的型攻撃や感染したサーバ・クライアントが属するネットワークへ攻撃を伝播させて、機密情報を不正に取得したり、システムを破壊したりする脅威があります。

●ソフトウェアの脆弱性を突く攻撃

バッファオーバーフロー、クロスサイトスクリプティング、SQLインジェクションなどOSやミドルウェア、アプリケーションの脆弱性を突いて、バックドアなどを仕掛けたり、機密情報を不正に取得したりする脅威があります。

●通信傍受・改竄

システムの通信を監視し、機密・認証情報などを奪取することで、改竄・なりすましを行う脅威があります。

特にパブリッククラウドはインターネットからダイレクトにアクセスできるため、これらの脅威に晒されるリスクが高まります。AWSに限らないものもありますが、パブリッククラウド特有のセキュリティリスクを理解し、対策を施す必要があります。

よく挙げられるのが、以下のような問題点です。

●認証情報の漏洩

アクセスキーやシークレットキーなどが何らかの理由で漏洩すると、AWS APIが不正実行されたり、既存のAWSネットワークへ進入され機密・個人情報が漏洩したり、仮想通貨のマイニング処理にハイスペックなインスタンスを使用されたりして、高額な使用料金を請求される被害が発生します。

●過剰な権限割り当て

ルートユーザが使用されていたり、EC2インスタンスへ設定したIAMロールの権限や、実行されているアプリケーションのIAMユーザの権限が過剰に割り当てられていたりすると、不正なオペレーションを実行される恐れがあります。

●オブジェクトストレージの不正アクセス

オブジェクトストレージの公開範囲や対象の設定の誤りにより、保存していた機密・個人情報が漏洩する被害が発生します。

●暗号化が未設定

漏洩した個人情報データでパスワードやクレジットカード番号・暗証番号などが暗号化されておらず、二次被害が発生します。

●通信アクセス制御不備

セキュリティグループやVPCルートテーブル設定の不備により、不正アクセスが発生する恐れがあります。

●インスタンスへの不正アクセス

キーペアの情報が漏洩したり、SSHポートのアクセス制御設定の不備により、不正アクセスが発生する恐れがあります。

●インスタンスマタデータの参照

アプリケーションやミドルウェア、サードパーティプロダクトやオープンソースソフトウェアの脆弱性を突かれることで、ソフトウェアが実行されているEC2インスタンスのマタデータから機密情報が漏洩する被害が発生します。これはSSRF(Server Side Request Forgery)攻撃とも呼ばれます。

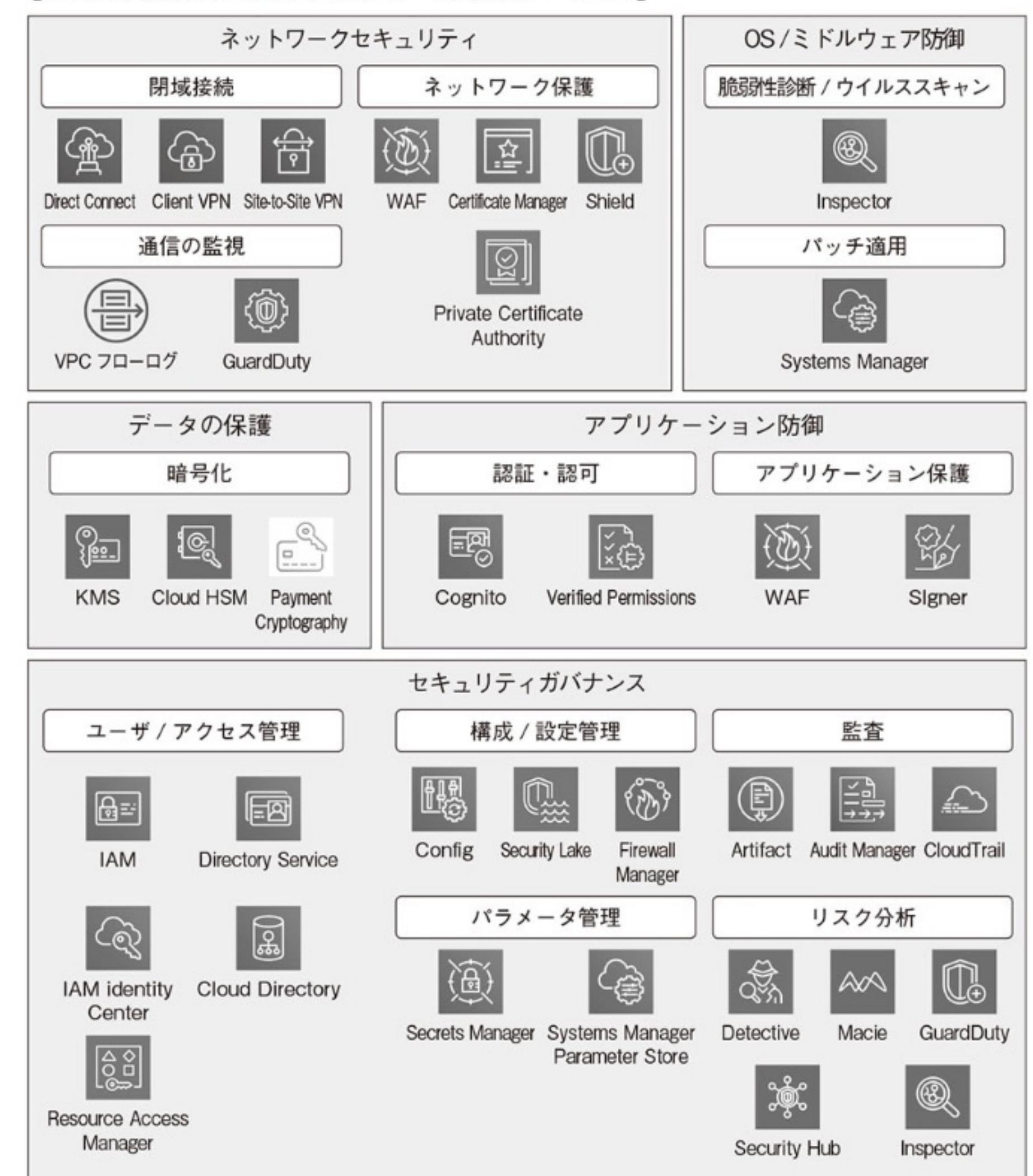
実際に発生したセキュリティインシデントの詳細を探っていくと、上記の問題が複合的に発生している事例が多く見受けられます。

セキュリティ事故として認知されるのは一部にすぎません。問題として表面化していないだけで、現状運用されているAWS環境でも個々の対策が十分に行われていない可能性が考えられます。開発の中でも、上記の対策が徹底されているか定期的に監視・点検する環境整備が必要になります。

2 AWSが提供するセキュリティサービス

前項で解説したセキュリティ脅威への対策として、AWSでは数多くのセキュリティに関するサービスを提供しています。

【AWSが提供するセキュリティ関連サービス】



サービスの一覧は前回のとおりですが、本書ですでに解説の章を設けているものを除き、開発を行う上で押さえておくべきAWSのセキュリティサービスは以下のようないのが挙げられます。

●WAF

AWS WAFは、AWSが提供するWebアプリケーションファイアウォール(WAF)です。WAFとは、Webアプリケーションの脆弱性を悪用した攻撃からWebアプリケーションを守る仕組みです。

AWS WAFは以下の要素で構成されます。

【AWS WAFの構成要素】

コンポーネント	概要
Webアクセスコントロールリスト (Web ACL)	ルールの集合体
ルール	ルールステートメントとルールアクションの含む条件と処理の定義
ルールステートメント	リクエストの検査条件
ルールアクション	ルールステートメントの条件を満たした際の処理
IPセット	ルールステートメントで参照できるIPアドレスとIPレンジの集合体
正規表現パターンセット	ルールステートメントで条件として利用できる正規表現パターンの集合体
ルールグループ	Web ACLで参照（利用）できるルールの集合体。AWSまたはセキュリティベンダーが提供するマネージドルールを利用可能

ルールステートメントにより、IPアドレスやアクセス元の地域、リクエストのサイズなど、さまざまな条件でリクエストの制御を行えます。

●Amazon Certificate Manager / Private Certificate Authority

Amazon Certificate Manager (ACM) は、SSL/TLS (Secure Sockets Layer/Transport Layer Security) サーバ証明書の管理機能を提供するマネージドサービスです。

SSL/TLSとは、データを暗号化して送受信する仕組みです。一般的にはSSLまたはTLSと記述されます（TLSはSSLを置き換える目的で開発された技術ですが、混乱を避けるためSSLの名称が今でも使われています）。SSLを有効化するには、通信を行うサーバに対してSSL/TLSサーバ証明書を発行・配備する必要があります。SSLが有効化されているWebサイトでは、HTTPを暗号化したHTTPS通信を利用します。

SSL/TLSサーバ証明書は発行元によって用途が異なります。公に認められて

る第三者機関（パブリック認証局）から発行される証明書をパブリックSSL/TLS証明書、自前の認証局（プライベート認証局）を構築し、発行する証明書をプライベートSSL/TLS証明書と呼びます。パブリックSSL/TLS証明書は基本的には有償で発行され、不特定多数の人が利用するようなサイトで、接続の安全性を証明するために利用されます。プライベートSSL/TLS証明書は自身で無償で発行できるため、社内ネットワークなど、アクセス元、通信先が明確な場合に使用されます。

近年、多くのシステムでは、WebページのすべてをSSL化し、通信を保護することが当たり前となりつつあります。オンプレミスのシステムでは、サーバやドメインの数だけSSL/TLSサーバ証明書を用意したり、それらが期限切れを起こしたりしないよう管理する必要があります。

ACMでは、AWS上でTLSサーバ証明書の管理を容易にするため、以下の機能を提供しています。

【ACMの提供機能】

項目	設定内容
証明書発行機能	ドメイン検証済みパブリック証明書/プライベート証明書を発行し、AWSで利用可能にする機能
証明書のインポート機能	オンプレミスなどで使われている既存の証明書をインポートし、AWSで利用可能にする機能
証明書のデプロイメント機能	AWSサービスに証明書を配備する機能

ACMは各種AWSサービスと連携しており、ACMで管理している証明書をELB、EC2、後述のCloudFront、2章1節で解説したAPI Gatewayで利用可能です。

なお、Private Certificate Authority (Private CA) は、自前の認証局（プライベート認証局）をAWS側で提供するマネージドサービスです。

●AWS Shield

AWS ShieldはL3、L4、L7レイヤーへのDistributed Denial-of-Service（分散型サービス妨害、DDoS）攻撃からシステムを保護する仕組みです。AWS ShieldにはStandardとAdvancedがあり、それぞれ保護するレイヤーと提供機能に違いがあります。

AWS Shield StandardはL3、L4を対象としたDDoS攻撃からシステムを保護する機能です。AWS Shield Standardはデフォルトで有効化されており、無料で利用できます。

AWS Shield AdvancedはStandardの機能に加えて、L3、L4を対象とした攻撃の通知やレポーティング機能を提供します。また、AWS WAFとも連携することでStandardでは対応できないL7への攻撃にも対応しています。L7を対象とした攻撃への防御のほか、通知、レポーティング機能を提供します。

AWS Shield Advancedを利用することで、AWSが保有するDDoSの専用サポートチームのサポートを24/365（24時間365日）で受けられます。機能、サポート共

に充実していますが、AWS Shield Advancedの料金は年間3,000USDに加え、通信量に応じた金額が請求されるため、相応のコストがかかります。必ずしも有効化するのではなく、システムが取り扱うデータの重要度や保存先に応じて使い分けましょう。

●Amazon GuardDuty

Amazon GuardDutyは、ユーザの動作や通信をモニタリング・分析し、脅威を識別する脅威検出サービスです。GuardDutyは脅威インテリジェンス（脅威検知のインプットとなる情報）と機械学習モデルを用いており、継続的に進化しながら脅威検出の精度を高めています。AWSのセキュリティのベストプラクティスでは、GuardDutyは有効化することが推奨されています。

Amazon GuardDutyはVPCフローログ、CloudTrail、Route 53のDNSログをインプットして解析を行います。このとき、これらのログの設定をユーザ側で有効化する必要はありません。各サービスから取得した情報をもとに、Amazon GuardDutyは既知の脅威と未知の脅威を検出します。既知の脅威とは、脅威インテリジェンスに含まれた、既知の不正な操作や通信として登録されているものを指します。未知の脅威とは機械学習によって検出される異常な操作、通信パターンを指します。

●Amazon Inspector

RDSやLambdaなどのPaaSまたはサーバレスのサービスは、責任共有モデルにもとづきAWSの責任で脆弱性に対する対策が行われます。一方、EC2はOSの設定やソフトウェアの脆弱性管理はユーザの責任で実施する必要があります。

こうした脆弱性管理をサポートするサービスが、EC2の脆弱性を診断するAmazon Inspectorです。Amazon Inspectorでは、診断の基準となるルールパッケージを提供しており、このルールパッケージにもとづきEC2を評価します。ユーザはルールパッケージを複数組み合わせることで、より高度な脆弱性管理を実現できます。

Amazon Inspectorは大きく分けて2つの診断機能を有しています。外部ネットワーク型診断とホスト型診断です。

●ホスト型診断

ホスト型診断はサーバ内部から脆弱性を確認する診断方法です。Amazon Inspectorでは、EC2にAmazon Inspectorエージェントをインストールすることで診断を行います。エージェントはEC2からAmazon Inspectorにテレメトリ（インストール済みのパッケージ情報やソフトウェアの設定）を送信し、Amazon Inspectorは受け取った情報と選択されたルールパッケージにもとづいてEC2を診断します。診断結果はS3に保存できるほか、SNSでの通知にも対応しています。

●外部ネットワーク型診断

外部ネットワーク型診断は外部ネットワークから脆弱性を確認する診断方法です。最も一般的なネットワーク脆弱性診断の方法です。ホスト型診断と異なり、こちらの診断方法ではAmazon Inspectorエージェントの導入は不要です。

●AWS Systems Manager (Parameter Store)

Systems Managerは、AWSにおけるシステム運用で、ソフトウェアインベントリの収集やOSのパッチ適用、運用コマンド実行、環境変数の管理、セキュアなサーバアクセス、メンテナンス作業の自動化などを実行するサービスです。

ここでは、その機能の1つであるParameter Storeを紹介します。

「パラメータストア」は、AWSのサービスやアプリケーションで利用するパラメータを管理する機能です。KMSと統合されており、パスワードやDBの接続文字列などのセキュリティに関連するパラメータを暗号化してセキュアに管理することもできます。

パラメータストアはEC2やSystems Managerのほか、下記のサービスからもパラメータを参照できます。これらのサービスで利用するパラメータや環境変数をパラメータストアで一元管理することが可能となります※22。

- EC2
- ECS
- Lambda
- CloudFormation
- CodeBuild
- CodeDeploy
- CodePipeline

●AWS Secrets Manager

AWS Secrets ManagerはデータベースのパスワードやAPIキーなどデータ流出の危険性がある認証情報（シークレット）を集約し、管理するサービスです。アプリケーションはAWS Secrets Managerにアクセスすることでシークレットを取得できるため、シークレットをローカルに保持する必要がなくなります。また、シークレットはKMSで暗号化して保存することで、セキュアに管理できます。

前項で解説したAWS Systems Managerのパラメータストアでも、AWS Secrets Managerと同様に認証情報を一元管理できます。両者の違いとして挙げられるのが、AWS Secrets Managerによるシークレットの自動ローテーション機能です。

Secrets Managerは、シークレットの更新間隔を指定すると自動でパスワード変更を行います。この機能はRDS、Redshiftなどのデータベースサービスと統合されており、Secrets Manager内のDBのパスワードがローテーションされると自動的

※22 連携可能なサービスの詳細および最新情報は開発者ガイドを参照してください。https://docs.aws.amazon.com/ja_jp/systems-manager/latest/userguide/systems-manager-parameter-store.html

にデータベースサービス側のパスワードを変更します。パスワードローテーションの仕組みはSecrets Managerによって自動的に作成されるLambdaによって実現されます。このLambda関数を変更することで、APIキー以外のシークレットのローテーションも実機と連動させることができます。

●Amazon Detective

Amazon Detectiveは、各種AWSサービスから収集できるデータを分析、可視化し、インシデントの原因を特定するサービスです。Amazon DetectiveはAWS Security HubとAmazon GuardDutyと統合されており、ロールやAPI、インスタンス、ユーザを条件として、収集された情報をドリルダウン（収集された情報を掘り下げて情報を詳細化すること）できます。

●Amazon Macie

Amazon Macieは、S3バケットとS3バケット内のオブジェクトを分析し、機械学習とパターンマッチングを用いて、脅威の検出やデータ分類を行うサービスです。暗号化無効化や公開検知といったバケットの脅威を検出するだけでなく、オブジェクトを分類してクレジットカードや電話番号など個人情報や機密データを検出できます。Amazon GuardDutyと同様にAmazon Macieも機械学習を用いており、継続的な改善が行われています。

- ・2章9節「Amazon S3 - セキュリティ保護」にて、S3のセキュリティ対策全般について解説。
- ・2章9節「Amazon S3 - その他の機能」にて、CORSやサーバログアクセス機能について解説。
- ・4章1節「AWS CodeCommit - CodeCommitへアクセスする端末のセットアップ方法」にて、アクセス設定について解説。
- ・4章2節「Amazon ECR - ECRの接続」にて、アクセス設定について解説。
- ・5章2節「AWS CloudTrail - CloudTrailの機能」にて、イベントログの暗号化について解説。
- ・5章5節「Amazon SQS - SQSの利用」にて、メッセージの暗号化やアクセス制御について解説。
- ・5章7節「Amazon CloudFront - セキュリティとその他の機能」にて、通信の暗号化やアクセスポリシー等について解説。

3

各マネージドサービスのセキュリティに関するサポート

AWSが提供するマネージドサービスのセキュリティに関しては、本節で記述しているもの以外にも、別章のサービスの中でも解説しています（読みやすさの観点から、各サービスの解説は章ごとに1箇所にまとめています）。認定試験の対策で、AWSのセキュリティに関する知識を振り返って学習したい場合は、以下の章のセキュリティに関する記述も参考にしてください。

- ・1章3節「AWS開発の基本となるサービス - VPCのアクセス制御」にて、セキュリティグループやACLについて解説。
- ・2章1節「Amazon API Gateway - API Gatewayの重要な機能と特徴」にて、API Gatewayの認証・認可について解説。
- ・2章6節「ELB - ELBで共通する特徴」にて、SSL/TLSターミネーションについて解説。
- ・2章6節「ELB - ALB」にて、OIDCプロバイダ認証、Certificate Manager/WAFなどとの連携について解説。
- ・2章6節「ELB - NLB」にて、セキュリティグループ設定の考慮点やSSLパススルーパスについて解説。

演習問題

- 1** あるアプリケーションはSQSを使用して、キュー「MyQueue」にメッセージを送信する処理が実装されています。グループ「developers」に所属する、あるエンジニアが開発中にテストとして、マネジメントコンソール上から、キューにメッセージを送信しようとしたところ、メッセージが送信できませんでした。「MyQueue」には、以下のようなリソースベースポリシーが設定されています。メッセージが送信できない原因として考えられるものを選択してください。

```
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__owner_statement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "SQS:*",
      "Resource": "arn:aws:sqs:ap-northeast-1:123456789012:MyQueue"
    }
  ]
}
```

- A. このリソースポリシーはルートユーザしかSQSオペレーションが許可されていません。
- B. グループ「developer」にSQSオペレーションを許可する、アイデンティティベースポリシーが付与されていない可能性があります。
- C. パーミッションバウンダリーで、グループ「developer」にSQSオペレーションを禁止するポリシーが記述されている可能性があります。
- D. グループ「developer」にこのキューに対するオペレーションを禁止する、アイデンティティベースポリシーがアタッチされている可能性があります。

- 2** 開発チームは商用環境のEC2インスタンスにアプリケーションをデプロイしようとしています。このアプリケーションはS3にアクセスする必要があり、インスタンスにはIAMロール「application-role」を設定して起動します。アプリケーションはインスタンスプロファイルで「application-role」を使って、インスタンスマタデータから一時認証情報を取得し、S3へアクセスすることを想定します。ロールに設定するポリシーとして正しいものを選択してください。

- A. ロールの信頼ポリシーとして以下を設定します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SampleTrustedPolicy",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "arn:aws:iam::123456789012:role/application-role"
      }
    }
  ]
}
```

3

- B. ロールの信頼ポリシーとして以下を設定します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SampleTrustedPolicy",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      }
    }
  ]
}
```

C. ロールのアクセスポリシーとして以下を設定します。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Effect": "Allow",  
         "Action": "s3:GetObject",  
         "Principal": {"Service": "ec2.amazonaws.com"},  
         "Resource": "arn:aws:s3:::example_bucket/*"}  
    ]  
}
```

D. ロールのアクセスポリシーとして以下を設定します。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Effect": "Allow",  
         "Action": "s3:GetObject",  
         "Principal": {"Service": "arn:aws:iam::123456789012:role/application-role"},  
         "Resource": "arn:aws:s3:::example_bucket/*"}  
    ]  
}
```

3 開発チームは商用、ステージング、開発といった用途に沿ってアカウントを分けており、ステージング環境アカウントXXXXで使用する、テスト資材・データの取得のために、開発環境アカウントYYYYのS3のTestResourceBucketへ、ロールを使ってクロスアカウントアクセスしたいと考えています。各ロールに設定するポリシーのうち、誤っているものを選択してください。

A. ステージング環境のロールの信頼ポリシーとして以下を設定します。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Sid": "StagingRoleTrustPolicy",  
         "Effect": "Allow",  
         "Action": "sts:AssumeRole",  
         "Principal": {"AWS": "arn:aws:iam::YYYY:role/DevelopmentRole"}  
    ]  
}
```

B. 開発環境のロールの信頼ポリシーとして以下を設定します。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Sid": "DevelopmentRoleTrustPolicy",  
         "Effect": "Allow",  
         "Action": "sts:AssumeRole",  
         "Principal": {"AWS": "arn:aws:iam::XXXX:role/StagingRole"}  
    ]  
}
```

- C. ステージング環境のロールのアクセスポリシーとして以下を設定します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAssumeCrossAccountRole",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::YYYY:role/DevelopmentRole"
    }
  ]
}
```

- D. 開発環境のロールのアクセスポリシーとして以下を設定します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccountYYYYBucketAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::TestResourceBucket/*"
    }
  ]
}
```

- 4** ある企業では、オンプレミス環境にある Microsoft Active Directory で従業員のユーザデータが管理されており、この認証情報を使って、AWSリソースへアクセスさせたいと考えています。実現可能な構成のうち、最も適切なものを選択してください。

- A. SAML 2.0準拠のIDフェデレーションで実現します。IDプロバイダに対して認証を行った後、SAMLアサーションをユーザが受け取り、それをを使って、AssumeRoleWithSAML APIにより一時認証情報を取得して、AWSリソースへアクセスします。
- B. CognitoのIDプールで、SAML IDプロバイダを設定することで実現します。IDプロバイダに対して認証を行った後、GetIdでSAMLアサーションを取得して、AssumeRoleWithSAML APIにより一時認証情報を取得して、AWSリソースへアクセスします。
- C. CognitoのIDプールで、OIDCプロバイダを設定することで実現します。IDプロバイダに対して認証を行った後、GetIdおよびGetOpenIdToken APIでIDトークンを取得して、AssumeRoleWithWebIdentity APIにより一時認証情報を取得して、AWSリソースへアクセスします。
- D. カスタムIDブローカーアプリケーションで実現します。IDプロバイダに対して認証を行った後、ブローカーがAssumeRole APIにより一時認証情報を取得して、AWSリソースへアクセスします。

- 5** あるアプリケーションでは、S3に格納された画像や動画などのメディアコンテンツをクライアントへ提供しています。画像や動画は、クライアント固有のもので秘匿性が高く、またサイズが大きいものも含まれるため、クライアントからダイレクトにS3へアクセスする方式で実装したいと考えています。次の記述のうち正しいものを選択してください。

- A. STSでFederationToken APIを呼び出し、このトークンを使って、S3へアクセス可能な、有効期限がある署名付きURLを生成してクライアントへ返却します。
- B. STSでGetSessionToken APIを呼び出し、このトークンを使って、S3へアクセス可能な、有効期限がある署名付きURLを生成してクライアントへ返却します。
- C. STSでAssumeRole APIから、セッションポリシーを設定した一時認証情報を作成し、有効期限がある署名付きURLを生成してクライアントへ返却します。
- D. リソースベースポリシーとポリシー変数を利用して、ユーザごとに引き受けるロールを切り替えて、STSでAssumeRole APIから、一時認証情報を作成し、有効期限がある署名付きURLを生成してクライアントへ返却します。

6 ある企業で、新しいサービスを実現するためのアプリケーションを開発しており、画像を含む個人情報データをS3に保存することを検討しています。個人情報の保護には高いレベルのセキュリティが求められ、データを暗号化することはもちろん、暗号化・復号処理に対する監査ログを残すことも法律上求められています。このセキュリティ要件を満たす最も簡易な方法を選択してください。

- A. Amazon S3でバケットを作成する際に、SSE-S3による暗号化を有効化します。
- B. Amazon S3でバケットを作成する際に、SSE-KMSによる暗号化を有効化します。
- C. KMSカスタマーマスターkeyを指定して、クライアントサイドでファイルを暗号化してファイルをアップロードするSDK APIを使って、S3へファイルを送信します。
- D. KMS Encrypt APIで個人情報データを暗号化してから、S3へファイルアップロードします。

7 EC2で実行されるアプリケーションは、KMSを利用して、共有ストレージに出力するデータをエンベロープ暗号化により保護することを検討しています。正しい実装方法を選択してください。

- A. KMS GenerateDataKey APIでカスタマーマスターkeyを指定して、暗号化されたデータキーとブレーンなデータキーを取得します。暗号化されたデータキーで、出力するデータを暗号化し、この暗号化に使用したデータキーはその後ただちに破棄します。
- B. KMS GenerateDataKey APIでカスタマーマスターkeyを指定して、暗号化されたデータキーとブレーンなデータキーを取得します。ブレーンなデータキーで出力するデータを暗号化し、この暗号化に使用したデータキーはその後ただちに破棄します。
- C. KMS Encrypt APIでエンベロープ暗号化オプションを指定して、暗号化されたデータキーとブレーンなデータキーを取得します。暗号化されたデータキーで、出力するデータを暗号化し、この暗号化に使用したデータキーはその後ただちに破棄します。
- D. KMS Encrypt APIでエンベロープ暗号化オプションを指定して、暗号化されたデータキーとブレーンなデータキーを取得します。ブレーンなデータキーで出力するデータを暗号化し、この暗号化に使用したデータキーはその後ただちに破棄します。

8 仮想通貨取引に関する処理で使用する非対称キーをKMSで利用したいと考えています。EC2上でこれらのキーを使った処理を実現するために最も効率的な手法を選択してください。

- A. 中間CA証明書を発行するサードパーティから入手した非対称キーを、KMSでキーマテリアルインポートします。
- B. AWS SSE-KMS(サーバサイド暗号化)でKMS API経由で非対称キーを取得します。
- C. AWS CSE-KMS(クライアントサイド暗号化)でKMS API経由で非対称キーを取得します。
- D. AWS KMSでは非対称キーをサポートしていません。EC2上で OpenSSLなどのライブラリを使用して非対称キーを取り扱います。

9 商用環境でKMSのカスタマーマスターkeyを使用しているアプリケーションがあります。運用担当者が意識しておくべきキーの運用で誤っているものを選択してください。

- A. 使用されているキーのうちカスタマー管理CMKで、AWS KMSを使って作成したキーは、3年後にキーが自動ローテーションされます。
- B. 使用されているキーのうちカスタマー管理CMKで、キーマテリアルインポートしたキーは、1年後に手動でローテーションする必要があります。
- C. 使用されているキーのうちAWSマネージドCMKで、AWSマネージドサービスが使用しているキーは、3年後に自動ローテーションされます。
- D. 使用されているキーのうちAWS所有CMKは、AWSが所有および管理しているので、キーのローテーションを意識する必要はありません。

10 EC2上で実行されているアプリケーションで、CloudWatch LogsからKMS ThrottlingExceptionが発生したことを検知しました。原因として考えられるものを2つ選択してください。

- A. EC2上で実行しているアプリケーションの中で、SSE-KMSを使った暗号化を有効化しているマネージドサービスへのアクセス処理に問題がある。
- B. EC2上で実行している暗号化したEBSローカルストレージへの大量のアクセスが発生している。
- C. EC2上で実行されているアプリケーションの中で、SDKを使ったマネージドサービスへのアクセスでエクスプロンシャルバックオフが実装されていない可能性がある。
- D. KMS APIのリクエスト上限を超えるアクセスが瞬間的に発生した可能性がある。
- E. EC2上で実行されているアプリケーションの中で、KMS Encrypt APIの上限を超えたファイル数が取り扱われた可能性がある。

11 ある開発者が、何千人のユーザが利用するWebアプリケーションを設計しています。ユーザは自身のメールアドレスを使用して、サインアップしアプリケーションは各ユーザの属性を保存する必要があります。ユーザがWebアプリケーションのサインアップができるようにするにはどのサービスを利用するのが適切ですか。1つ選択してください。

- A. Amazon Cognito Sync
- B. Amazon Cognito ユーザプール
- C. Amazon Cognito ID プール
- D. Amazon AppSync

12 ある開発者が、金融のトランザクションと取引統計を表示するバンキング向けアプリケーションを作成しています。アプリケーションはログイン時のプロトコルとして多要素認証(MFA)を追加する必要があります。この要件を満たすためにどのサービスを利用すべきでしょうか。1つ選択してください。

- A. MFAが設定されたAWS IAMを利用します。
- B. MFAが設定されたAmazon Cognitoユーザプールを利用します。
- C. AWS Directory Serviceを利用します。
- D. MFAが設定されたAmazon CognitoID プールを利用します。

13 ある開発者が、エンターテインメント業界向けのモバイルアプリケーションを作成しています。アプリケーションは個人情報を含んでいるため、できるだけ安全にやりとりする必要があります。開発者が選択すべき認証フローの設定のうち、適切なものを2つ選択してください。

- A. Cognito管理者APIを使った認証(ALLOW_ADMIN_USER_PASSWORD_AUTH)で安全にサーバとのやりとりを行います。
- B. Lambdaトリガーによるカスタム認証(ALLOW_CUSTOM_AUTH)を有効にし、追加で検証処理を行います。
- C. クライアントシークレットを発行し、検証処理を追加します。
- D. クライアントシークレットを発行せず、認証フローを構成します。
- E. SRPプロトコルベースの認証(ALLOW_USER_SRP_AUTH)を有効にします。

14 ある組織がAmazon API Gatewayを通してAPIを提供するアプリケーションを開発しています。APIはAmazonやGoogle、Facebook、AppleなどのOpenIDプロバイダをベースとして認証され、APIはカスタムの認可モデルにもとづいてアクセス許可設定されなければなりません。APIの認証・認可モデルを構築するために最も簡単でセキュリティ性の高い設計方法はどれでしょうか。1つ選択してください。

- A. 認証と認可のためにAPIリクエストへ付与するクレデンシャルをAmazon ElastiCacheに保存します。
- B. Amazon DynamoDBを使ってクレデンシャルを保存し、AWS STSからの一時認証情報をアプリケーションに取得させます。認証と認可のためにクレデンシャルを設定してAPIをコールします。
- C. Amazon CognitoユーザプールとJWT(JSON Web Token)をベースとしたユーザ認証・認可のためのオーソライザを使用します。
- D. Amazon、FacebookおよびAppleのOpenIDトークンプロバイダーを構築します。ユーザはこのIDプロバイダを認証し、それぞれのAPIコールの認証にJWTを渡します。

15 あるWebサイトで、Amazon S3バケットに保存された画像を配信しています。このサイトはAmazon Cognitoの利用を可能にしており、ログインしていないゲストユーザがS3バケットから画像を表示できるようにする必要があります。開発者はどのようにしてゲストユーザがAWSリソースにアクセスできるようにすることができるか、正しいものを1つ選択してください。

- A. ユーザプールにブランクユーザIDを作成し、ユーザグループに追加して、AWSリソースへのアクセスを許可します。
- B. 新しいIDプールを作成し、認証されていないIDへのアクセスを有効にして、AWSリソースへのアクセスを許可します。
- C. 新しいユーザプールを作成し、認証されていないIDへのアクセスを有効にして、AWSリソースへのアクセスを許可します。
- D. ユーザプールにアノニマスユーザを作成し、アクセスを制限して、AWSリソースへのアクセスを許可します。

16 あるWebサイトはEC2インスタンス上にホスティングされています。月額の使用料金が、前月と比べて10倍以上上昇しています。考えられる問題と対応すべき対処のうち、誤っているものを選択してください。

- A. アクセスキーやシークレットキーが漏洩し、悪意のある攻撃者に不正利用されている可能性があります。キーをただちに無効化して再発行します。
- B. Webサイトで使用されているサーバミドルウェアの脆弱性を突いて、インスタンスマタデータから認証情報が漏洩し、悪意のある攻撃者に不正利用されている可能性があります。サーバミドルウェアを再点検し最新バージョンにアップデートします。
- C. Webサイトで使用されているサーバミドルウェアの脆弱性を突いて、インスタンスマタデータから認証情報が漏洩し、悪意のある攻撃者に不正利用されている可能性があります。EC2に割り当てられているIAMロールの権限を再点検し、必要最小限のものにアップデートします。
- D. キーペアが漏洩し、悪意のある攻撃者に不正利用されている可能性があります。キーペアをただちに無効化して再発行します。

17 あるアプリケーションはAWS LambdaからAmazon RDSデータベースを使用しています。セキュリティ要件として、RDSデータベースへの接続するユーザのパスワードは1か月でローテーションすることが求められています。要件を達成するために最も簡単な方法を選択してください。

- A. Secrets Managerを利用します。パスワード用のパラメータに対し、1か月ごとに自動ローテーションするよう設定します。
- B. Systems Manager Parameter Storeを利用します。パスワード用のパラメータに対し、Secure Stringオプションを有効にし、1か月ごとに自動ローテーションするよう設定します。
- C. AWS KMSを利用します。パスワード用のパラメータに対し、1か月ごとに自動ローテーションするよう設定します。
- D. AWS CloudHSMを利用します。パスワード用のパラメータに対し、1か月ごとに自動ローテーションするよう設定します。

18 ある開発者は開発中のシステムで採用するAWSセキュリティサービスを検討しています。提供されているサービスと機能の説明のうち正しいものを選択してください。

- A. DDoS攻撃を抑止するため、AWS WAFの利用を検討します。
- B. EC2の脆弱性を診断するため、Amazon Detectiveの利用を検討します。
- C. ユーザの動作や通信をモニタリング・分析し、脅威を識別するため、AWS GuardDutyの利用を検討します。
- D. 特定のIPアドレスからアクセスを抑止するため、AWS Shieldの利用を検討します。

解 答**1 D** ➡ 問題：258 ページ、本文解説：213-215 ページ

このポリシーは、SQSに設定されているリソースベースポリシーです。アクションの実行可否はリソースベースポリシーやアイデンティティベースのポリシーの組み合わせ方で決まります。特に明示的な拒否が1つでも含まれていると、その操作は実行不可となる点がポイントです。各選択肢の不正解理由は以下のとおりです。

- ・選択肢A：このプリンシパルの指定はルートユーザを指定しているわけではなくアカウント全体を設定するときに記載される方法です。アカウントARN(arn:aws:iam::123456789012:root)、または「AWS:」プレフィックスの後にIDを付けた短縮形を使用できます。
- ・選択肢B：リソースベースポリシーとアイデンティティベースポリシーはどちらかが許可されていれば、アクセス可能になります(OR条件)。問題のリソースベースポリシーはアカウント全体で「MyQueue」に関するSQSオペレーションがすべて許可されているので、考えられるのは明示的な拒否が存在する可能性です。
- ・選択肢C：パーミッションバウンダリーはグループに設定することはできません。ロールかユーザが対象です。

2 B ➡ 問題：259 ページ、本文解説：218 ページ

EC2インスタンスにIAMロールを設定して、インスタンスマタデータから一時認証情報を取得する場合、プリンシパルを「ec2.amazonaws.com」とする信頼ポリシーをロールに設定しておく必要があります。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：プリンシパルはロールを引き受ける、信頼されたエンティティを指定する必要があります。ここで、プリンシパルに自身のロールを設定するのは正しくありません。
- ・選択肢C、D：アクセスポリシーとしてプリンシパルを指定する必要はありません。また、ロールをプリンシパルに設定する場合の属性は「Service」ではなく、「AWS」です。

3 A ➡ 問題：261 ページ、本文解説：219-222 ページ

クロスアカウントでのロールの引き受けは、引き受ける対象のロール(この問題では開発アカウント)の信頼ポリシーとS3へのアクセスポリシーを、AssumeRole APIを呼び出すロール(この問題ではステージングアカウント)のアクセスポリシーにAssumeRole API実行権限を設定しておく必要があります。各選択肢の解説は以下のとおりです。

- ・選択肢A：これは不要な設定です。ステージング環境のロールは引き受けの対象ではありません。
- ・選択肢B：開発環境のロールはステージング環境のロールから引き受けられるため、信頼されたエンティティとして、プリンシパルにステージングのロールの設定が必要になります。
- ・選択肢C：開発環境のロールに対し、AssumeRole APIの実行権限が必要なため、このアクセスポリシーは必要です。
- ・選択肢D：開発環境のロールには、S3へのアクセス権限が必要になるため、このアクセスポリシーは必要です。

4 A ➡ 問題：263 ページ、本文解説：222-223 ページ

Microsoft Active Directoryは、SAML 2.0がサポートされたユーザディレクトリサービスのため、SAML 2.0準拠のIDフェデレーションで実現します。IDプロバイダに対して認証を行った後、SAMLアサーションをユーザが受け取り、それを使って、AssumeRoleWithSAML APIにより一時認証情報を取得してAWSリソースにアクセスします。他の選択肢の不正解理由は以下のとおりです。

- ・選択肢B：CognitoのIDプールでSAMLプロバイダを指定することはできますが、GetIdでSAMLアサーションを取得することはありません。このAPIはOIDCトークンをCognitoトークンに関連付ける場合に呼ばれるCognitoのAPIです。
- ・選択肢C：これはOIDCに準拠したIDプロバイダで有効な一時認証情報の取得方法です。
- ・選択肢D：これはSAMLやOIDCに準拠していない独自のIDストアを持つシステムで有効な一時認証情報の取得方法です。

5 C ➔ 問題：263 ページ、本文解説：228-232 ページ

STSを用いて、セッションポリシーを指定してアクセス対象のコンテンツのアクセス制御を詳細に設定することができます。コンテンツはクライアント固有のもので秘匿性が高いので、オブジェクト単位でアクセス制御を設定し、期限付きのURLでクライアントからアクセスさせるのがよい実装方式です。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：FederationToken APIで取得したトークンを使って、S3へアクセス可能な有効期限付き署名URLを生成することはできません。SDKで有効期限がある署名付きURLは生成できます。
- ・選択肢B：GetSessionToken APIで取得したトークンを使って、S3へアクセス可能な有効期限付き署名URLを生成することはできません。SDKで有効期限がある署名付きURLは生成できます。
- ・選択肢D：リソースベースポリシー変数を使って、アクセス可能な対象のバケットやオブジェクトキーをユーザごとに切り替えることはできますが、ユーザごとに引き受けるロールを切り替えることはできません。

6 B ➔ 問題：264 ページ、本文解説：238-240 ページ

SSE-KMSはS3バケット作成時に暗号化のデフォルトオプションとして選択できます。アップロード時にS3上で暗号化され、CloudTrailに暗号化の証跡がログとして蓄積されます。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：SSE-S3はS3バケット作成時に暗号化のデフォルトオプションとして選択できますが、デフォルトで暗号化に対する監査ログを残すことはありません。
- ・選択肢C：クライアントサイドでエンベロープ暗号化してファイルをS3へアップロードするオプションです。実装も容易で、データキーを生成するときKMS APIがコールされ、監査ログも証跡として記録されますが、アプリケーションがすべてこの方法で実装されていることを確認する必要があり、また、アプリケーションを介さずS3にアップロードされた場合の暗号化が考慮されていません。
- ・選択肢D：KMS Encrypt APIでデータを暗号化することはできます。実装も容易で、KMS APIがコールされる際に監査ログも証跡として記録されます。しかし、このAPIの暗号化するデータサイズ上限は4KB以下に制限されます。問題では画像を含むデータであり、大きめのファイルサイズとなることが想定されるので、この方法では実現が困難です。

7 B ➔ 問題：264 ページ、本文解説：235-237 ページ

KMS GenerateDataKey APIで生成した暗号化キーとプレーンデータキーを取得し、暗号化をプレーンデータキーで行います。また、暗号化に使用したプレーンデータキーはただちに破棄し、暗号化されたデータキーは復元のときに必要なため破棄してはいけません。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：KMS GenerateDataKey APIで生成した暗号化キーとプレーンデータキーを取得することは正しいですが、暗号化はプレーンデータキーで行う必要があります。また、暗号化されたデータキーは復元のときに必要なため破棄してはいけません。
- ・選択肢C、D：KMS Encrypt APIは暗号化を行うAPIであり、エンベロープ暗号化のためのデータキーを生成するオプションはありません。

8 C ➔ 問題：265 ページ、本文解説：234-235 ページ

CSE-KMSを使って、エンベロープ暗号化でのデータキーとしては対称キーと非対称キーを選択できます。非対称キーはKMS GenerateDataKeyPair APIを利用して、指定されたCMKから取得することができます。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：KMSのキーマテリアルインポートは非対称キーをサポートしません。対称キーのみです。
- ・選択肢B：サーバサイド暗号化はAWSマネジドサービスがKMSのキーを使用して暗号化する方式であり、非対称キーをユーザが選択して利用することはできません。
- ・選択肢D：2019年のアップデートでKMSは非対称キーをサポートしています^{※23}。

9 A ➔ 問題：265 ページ、本文解説：235 ページ

カスタマー管理CMKで、AWS KMSを使って作成したキーの自動ローテーション期間は1年です。

※23 <https://aws.amazon.com/jp/about-aws/whats-new/2019/11/aws-key-management-service-supports-asymmetric-keys/>

10 B、D

→ 問題：266 ページ、本文解説：240 ページ

暗号化されたEBSボリュームは、暗号化・復号が必要なタイミングでKMS APIにアクセスします。その過程でスロットルエラーが発生した可能性があります。もう1つ原因として考えられるものは、東京リージョンのKMS APIのリクエスト上限はデフォルトで10,000に設定されています。偶発的に処理が集中しスロットルエラーが発生した可能性があります。その他の選択肢の不正解理由は以下のとおりです。

- 選択肢A：例えばS3などのSSE-KMSを使った暗号化を有効化しているマネージドサービスでスロットル超過エラーが発生する可能性はありますが、EC2のログに出力されることはありません。また、発生したとしても、マネージドサービスのアクセスが発生しないとスロットルエラーは起こりえません。
- 選択肢C：AWS SDKのマネージドサービスへのアクセスはエクスプロンシャルバックオフが組み込まれた状態で実装されています。
- 選択肢E：Encrypt APIは暗号化可能なファイルサイズが4KBに制限されていますが、暗号化するファイル数に上限はありません。ファイル数だけAPIを呼び出すだけです。なお、通常、暗号化は、エンベロープ暗号化で実装されるため、Encrypt APIの使用頻度はそんなに高くありません。

11 B

→ 問題：266 ページ、本文解説：241-242 ページ

何千ものユーザが利用するアプリケーションのユーザディレクトリサービスとしてCognitoユーザプールが適しています。ユーザプールでメールアドレスを利用したサインアップが可能です。その他の選択肢の不正解理由は以下のとおりです。

- 選択肢A：Cognito Syncはさまざまなデバイスでデータを同期するのに使われるサービスです。なお、現在は、AppSyncの利用が推奨されています。
- 選択肢C：IDプールは認証されたユーザに対し、AWSリソースへの認可を与えるサービスです。
- 選択肢D：AppSyncは複数のデバイスでデータを同期するために使われるサービスです。

12 B

→ 問題：266 ページ、本文解説：241 ページ

多くのユーザが利用するアプリケーションのユーザディレクトリサービスとしてCognitoユーザプールが適しています。ユーザプールではMFAを追加することができます。その他の選択肢の不正解理由は以下のとおりです。

- 選択肢A：IAMユーザにMFAを設定することはできますが、IAMユーザはユーザプールではなく、AWSアカウントに存在するユーザです。
- 選択肢C：AWS Directory ServiceはActive Directoryを管理するためのサービスです。
- 選択肢D：IDプールは認証されたユーザに対し、AWSリソースへの認可を与えるサービスです。

13 D、E

→ 問題：267 ページ、本文解説：243-244 ページ

モバイルアプリケーションで使用する場合は、より安全性の高いALLOW_USER_SR_P_AUTHの使用が推奨されます。また、クライアントシークレットはアプリクライアントの正当性を検証する場合のパスワードに相当するパラメータです。デバッグやデコンパイルなどにより、不特定多数の攻撃者に参照される可能性があるモバイルアプリケーションでの使用は推奨されません。その他の選択肢の不正解理由は以下のとおりです。

- 選択肢A：これは安全なサーバサイドのアプリクライアントで実装する方式です。
- 選択肢B：カスタム認証を追加して検証を行ってもよいですが、モバイルアプリケーションでの安全を保証するものではありません。
- 選択肢C：デバッグやデコンパイルなどを使用して、不特定多数に見られる可能性があるモバイルアプリケーションでクライアントシークレットを使用するべきではありません。

14**C**

➡ 問題：267 ページ、本文解説：242、245 ページ

API Gatewayでは、Cognitoを使用して、サードパーティのOpenIDプロバイダと連携し、JWTを使用したオーソライザを構築できます。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：API Gatewayは、デフォルトでElastiCacheを使用してOpenIDプロバイダのトークンを保存するオプションはありません。カスタムオーソライザで利用することはできますが、そのままでは安全な保存方法ではありません。
- ・選択肢B：API Gatewayは、デフォルトでDynamoDBを使用してOpenIDプロバイダのトークンを保存するオプションはありません。カスタムオーソライザで利用することはできますが、そのままでは安全な保存方法ではありません。
- ・選択肢D：OpenIDプロバイダのトークンブローカーとなる環境を構築することはできますが、自らAuthorization Serverを構築する必要があります。煩雑になります。

15**B**

➡ 問題：268 ページ、本文解説：248 ページ

Cognito IDプールには認証されていないゲストユーザーのアクセス権限を設定でき、これを使ってAWSリソースへのアクセスを許可します。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：CognitoユーザプールでブランクのユーザIDを作成することはできません。
- ・選択肢C：ユーザプールは認証のためのユーザディレクトリサービスであり、認可の仕組みは有していません。IDプールと連携する必要があります。
- ・選択肢D：アノニマス向けのユーザを作成することはできますが、不特定のリクエストをすべてアノニマスユーザー1つに結びつけることはCognitoにはできません。

16**D**

➡ 問題：268 ページ、本文解説：250 ページ

キーペアが漏洩すると、そのキーペアを設定しているEC2インスタンスが不正にアクセスされて、EC2メタデータなどから認証情報などがすでに漏洩している可能性があります。したがって、キーペアをただちに無効化して再発行するだけでは不十分です。認証情報を一新して環境を構築し直す必要があります。なお、本設問には盛り込んでいませんが、操作履歴を記録しておくためにCloudTrailを有効化しておくのもインシデント対応の基本手順の1つです。

17**A**

➡ 問題：269 ページ、本文解説：255 ページ

AWS Systems Manager Parameter Storeと、Secrets Managerは認証情報を一元管理できます。両者の機能的な違いとして挙げられるのが、AWS Secrets Managerではシークレットの自動ローテーションがサポートされていることです。Secrets Managerは、シークレットの更新間隔を指定すると自動でパスワード変更を行います。

18**C**

➡ 問題：269 ページ、本文解説：254 ページ

AWS GuardDutyはユーザの動作や通信をモニタリング・分析し、脅威を識別する脅威検出サービスです。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：これはAWS Shieldの説明です。
- ・選択肢B：これはAmazon Inspectorの説明です。
- ・選択肢D：これはAWS WAFの説明です。