IT資格道場

# AWS SAP-C03 学習用テキスト

予備知識ゼロから合格ラインへ ―― 読んで理解する1冊目

無料サンプル（第1章まで）

AWS 公式 Exam Guide（SAP-C03・2026年11月17日開始）準拠
2026年11月版

SAMPLE

## ご利用にあたって

- このテキストは「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルも著作物です。転載・改変しての再配布・二次販売はできません。
- AI モデルの学習データとしてのご利用(学習目的の入力・データセット化を含む)はできません。
- 製品版(学習用テキスト・頻出度別ノート・暗記用ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

SAMPLE

# 目次

**この教材の使い方**

試験の基本情報

試験の設計図：5 ドメインと配点

**第1章 クラウド向けのアーキテクチャの設計と実装 [1.1.1/1.1.2/1.1.3/1.1.4/1.1.5/1.1.6/1.1.7/1.1.8/1.1.9/1.1.10/1.1.11/1.1.12/1.1.13/1.2.1/1.2.2/1.2.3/1.2.4/1.3**

イベント駆動の設計 [1.1.1]

状態を外に出したステートレスな設計 [1.1.2]

サーバーレスのデータ処理・分析パイプライン [1.1.3]

永続的な実行の型を持つ多段のワークフロー [1.1.4]

アカウント・リージョンをまたぐイベントのルーティング [1.1.5]

マネージドのコンテナオーケストレーション [1.1.6]

サービスメッシュ [1.1.7]

コンテナイメージの管理 [1.1.8]

サーバーレス・コンテナのオートスケーリング [1.1.9]

入口の型と資源の最適化 [1.1.10]

レイテンシ要件のためのエッジ [1.1.11]

WebSocket と出版・購読によるリアルタイム処理 [1.1.12]

マルチテナントとテナントの分離 [1.1.13]

生成 AI の組み込み [1.2.1]

ツールと連携する AI エージェント [1.2.2]

ML の推論のアーキテクチャ [1.2.3]

RAG（取り込みと検索の戦略） [1.2.4]

冗長とフェイルオーバーを備えたハイブリッド接続 [1.3.1]

ハイブリッドのデータ同期 [1.3.2]

データ変換によるレガシーの近代化 [1.3.3]

ハイブリッドの DNS [1.3.4]

ハイブリッドのコンピュート [1.3.5]

複数 VPC・マルチアカウントの接続 [1.4.1]

VPC の設計 [1.4.2]

ネットワークのセキュリティ [1.4.3]

IPv6 の導入とデュアルスタックへの移行 [1.4.4]

マルチリージョンのグローバルな経路制御 [1.4.5]

データレイク・レイクハウス [1.5.1]

リアルタイムとバッチの分析パイプライン [1.5.2]

要件に合うデータベースの選定 [1.5.3]

高可用のためのデータベースのスケーリング [1.5.4]

アクセスの型と性能要件に合うストレージ [1.5.5]

データのキャッシュ戦略 [1.5.6]

データカタログとアカウントをまたぐデータ共有 [1.5.7]

SAMPLE

**サンプルはここまでです**

SAMPLE

3 冊の使い方は、ダウンロード欄の案内を参照してください。

SAMPLE

## この教材の使い方

このテキストの目的は、**AWS Certified Solutions Architect - Professional（SAP-C03）**（実施は Amazon Web Services（AWS））の合格です。

Professional 級が問うのは「サービスを知っているか」ではなく、**長い状況設定と複数の要件・制約を読んで、条件に合う設計を選べるか**です。公式の試験ガイドは、次の 5 つのドメインから成ります。

- クラウド向けのアーキテクチャの設計と実装
- セキュリティ、コンプライアンス、ガバナンス
- コストを最適化したアーキテクチャの設計
- 回復力、移行、事業継続
- 運用の卓越性と自動化

第 1 章〜第 5 章は、その順番どおりに並べています。各節の見出しの角かっこは、試験ガイドのスキル番号です。

### 試験の基本情報

| 項目 | 内容 |
|---|---|
| 試験名 | AWS Certified Solutions Architect - Professional（**SAP-C03**） |
| 運営 | Amazon Web Services |
| 問題数 | **75 問**（採点対象 65 問＋採点されない 10 問） |
| 出題形式 | **択一（4 択から 1 つ）と複数選択（5 つ以上から 2 つ以上）** |
| 試験時間 | **180 分** |
| 合格ライン | **100〜1,000 のスケールスコアで 720 点** |
| 受験料 | 300 USD |

### 試験の設計図：5 ドメインと配点

| ドメイン | 配点 | 対応する章 |
|---|---|---|
| ① クラウド向けのアーキテクチャの設計と実装 | **25%** | 第 1 章 |
| ② セキュリティ、コンプライアンス、ガバナンス | **25%** | 第 2 章 |
| ③ コストを最適化したアーキテクチャの設計 | **15%** | 第 3 章 |
| ④ 回復力、移行、事業継続 | **15%** | 第 4 章 |
| ⑤ 運用の卓越性と自動化 | **20%** | 第 5 章 |

配点に大きな偏りはなく、どれか 1 つのドメインを捨てる作戦は成立しません。

SAMPLE

# 第1章 クラウド向けのアーキテクチャの設計と実装 [1.1.1/1.1.2/1.1.3/1.1.4/1.1.5/1.1.6/1.1.7/1.1.8/1.1.9/1.1.10/1.1.11/1.1.12

この章は、分野 1 の 5 つのタスクを順に扱います。5 つとは、アプリの作り方、生成 AI、ハイブリッド接続、ネットワーク、データです（この順に並べます）。出題は「要件を読み、条件に合うサービスと設定を選ぶ」形が中心なので、各節では選び分けの軸と数値を先に押さえます。

### イベント駆動の設計 [1.1.1]

イベント駆動は、送り手と受け手をメッセージで切り離す設計です。ここで問われるのは、順序・重複・配信の保証を、どのサービスのどの設定で実現するかです。

**SQS** には標準キューと FIFO キューがあります。標準キューは順序を保証しません。配信は少なくとも 1 回なので、重複が起こり得ます。FIFO キューは MessageGroupId（必須・最大 128 文字）ごとに順序どおりに処理し、重複排除を備えます。スループットは非高スループットモードでパーティションあたり API アクションごとに 300 TPS、バッチ利用で 3,000 メッセージ/秒です。[^c1-1]

FIFO の重複排除は、5 分の重複排除間隔の中で同じ送信を再試行しても重複を入れない仕組みです。コンテンツベース重複排除は本文（属性は含まない）の SHA-256 で ID を作ります。無効にしたキューでは MessageDeduplicationId を明示します。[^c1-2]

SQS の数値は次のとおりです。

| 項目 | 既定 | 範囲 |
|---|---|---|
| メッセージ保持期間 | 4 日 | 60 秒〜1,209,600 秒（14 日） |
| 可視性タイムアウト | 30 秒 | 0 秒〜12 時間 |
| メッセージタイマー（遅延） | 0 秒 | 最大 15 分 |
| メッセージの最大サイズ | 1,048,576 バイト（1 MiB） | 1,024 バイト〜1,048,576 バイト（1 MiB） |

1 MiB を超える本文は S3 に保存し、キューには S3 の場所だけを送ります（Extended Client Library の型）。

可視性タイムアウトは、受け取ったメッセージを他のコンシューマーから隠す時間です。処理が間に合わないと同じメッセージが再び見え、二重処理になります。処理時間より短く設定してはいけません。繰り返し失敗するメッセージは、受信回数の上限を決めたうえで **DLQ**（デッドレターキュー）へ送ります。DLQ は元のキューと同じ種類（標準か FIFO）にします。[^c1-1]

**SNS** は出版・購読のトピックです。1 つのトピックから複数の SQS キューや Lambda へ配るのが **ファンアウト**で、受け手ごとにキューを持たせると、受け手の障害が他へ波及しません。SNS FIFO トピックは、購読先に FIFO の SQS キューを選ぶと、順序と重複排除をファンアウトに持ち込めます。購読者ごとに **メッセージフィルタ**（フィルタポリシー）を付けると、属性や本文の条件に合うものだけが届き、受け手側で捨てる処理が要りません。[^c1-3][^c1-4]

**EventBridge** のルールは、イベントパターンに一致したイベントを最大 5 つのターゲット（Classic のカスタムイベントバスで変更不可）へ送ります。既定のイベントバスと Classic のカスタムイベントバスへの PutEvents は 1 リクエスト最大 10 エントリで、新しいカスタムイベントバスは、PutEvents が 1 リクエスト最大 100 エントリ（重複排除ウィン

SAMPLE

ドウは 300 秒）です。[^c1-5] 5 つを超える受け手に配るなら、ターゲットの 1 つを SNS トピックにしてファンアウトします。配信に失敗したターゲットへは、既定で 24 時間・最大 185 回、指数バックオフとジッターつきで再試行します。尽きるとイベントは捨てられ、再び配信されません。失われると困るなら、ターゲットに DLQ を付けます。[^c1-6] [^c1-7]

**Kinesis Data Streams** の順序は、パーティションキーが決めるシャード内で保たれます。同じキーのレコードは同じシャードに入ります。シャードをまたぐ全体の順序は保証されません。キーが偏ると特定シャードに負荷が集中するため、キーは値の種類が多く均等なものにします。プロビジョンドの 1 シャードは書き込み 1 MB/秒または 1,000 レコード/秒、読み取り 2 MB/秒（GetRecords は 5 回/秒）です。拡張ファンアウトの登録コンシューマーは 1 ストリームに 20 までで、各コンシューマーが専用の読み取り帯域を得ます。[^c1-8][^c1-9]

必要なシャード数は、書き込みの MB/秒とレコード数/秒の大きいほうで決まり、共有コンシューマーの本数が増えれば読み取り側でも足りなくなります。たとえば書き込み 5 MB/秒・4,500 レコード/秒（1 KB）なら書き込みで 5 シャードです。共有の読み取りが 3 本で 15 MB/秒なら、読み取り側は 8 シャード必要です（15÷2 の切り上げ）。読み取りで詰まる場合は、シャードを増やすか拡張ファンアウトにします。

**冪等な受け手**は、同じメッセージを 2 回受けても結果が変わらない処理です。標準キュー、SNS、EventBridge、Kinesis、Firehose はいずれも重複を起こし得るので、受け手側で備えます。メッセージ ID や業務キーを冪等キーとして DynamoDB の条件付き書き込みで記録し、処理済みなら何もしません。FIFO の重複排除は 5 分間だけなので、それより長い間隔の重複は冪等キーで防ぎます。

選び分けは次のとおりです。順序が要り、グループ単位で並列にしたいなら SQS FIFO（MessageGroupId）。同じイベントを複数の受け手へ配るなら SNS のファンアウト。ルールで振り分け、SaaS や他アカウントとつなぐなら EventBridge。順序つきの大量ストリームを複数の受け手で再読み取りするなら Kinesis です。

たとえば、注文を店舗ごとに順序どおり、かつ重複なしで処理したいとします。店舗 ID を MessageGroupId にした SQS FIFO キューを使い、Lambda のコンシューマーは冪等キーで二重処理を防ぎます。失敗が続く注文は FIFO の DLQ へ送ります。標準キューは順序を保証せず、EventBridge も順序を保証しません。この構成はグループ内が直列になるので、グループを細かく分けるほど並列度が上がります（店舗ごとのグループ）。

### 状態を外に出したステートレスな設計 [1.1.2]

ステートレスなアプリは、セッションや状態を自分のメモリに持たず、外のサービスに置きます。インスタンスをいつでも増減・置換でき、障害時も状態が失われません。

セッション情報は **ElastiCache**（低遅延）か **DynamoDB**（耐久性・サーバーレス）に置きます。DynamoDB の **TTL** 属性は Number 型の Unix エポック時刻（秒単位）で、期限切れの項目は期限後「通常は数日以内」にシステムが削除します。削除が済むまでは期限切れの項目を更新でき、TTL を未来にすれば残せます。読み取り時に期限を自分で確認する設計にします。[^c1-10]

ElastiCache のキャッシュ戦略は 2 つです。**遅延読み込み**（lazy loading）はキャッシュミスの時だけ DB から読み、キャッシュに書きます（読まれたデータだけが載る）。ミスが 3 往復（キャッシュ確認・DB 読み取り・キャッシュ書き込み）になり、古いデータが残り得ますが、ノード障害に強い方式です。**write-through** は DB へ書くたびにキャッシュも更新するので常に最新です。ただし新ノードではデータが欠け、読まれないデータでキャッシュが無駄になります。どちらも TTL を併用して、古さと無駄を抑えます。[^c1-11]

SAMPLE

大きなファイルや生成物は S3 に置きます。S3 は全リージョンで PUT・DELETE に強い read-after-write 整合性を持ちます。同じキーへの同時書き込みは最後の書き込みが勝ち、オブジェクトのロックはありません。バケット設定は結果整合で、バージョニングを有効化した後は 15 分待ってから書き込むよう推奨されています。[^c1-12]

**スティッキーセッション**は、同じクライアントを同じターゲットに固定する機能です。ALB は期間ベース（AWSALB Cookie）とアプリケーションベースの両方に対応します。固定されたターゲットが落ちるとセッションが消え、負荷も偏り、スケールインも妨げます。状態を外に出せば固定が要りません。加重ランダムのアルゴリズムはスティッキーセッションと併用できず、クロスゾーン負荷分散がオフだとターゲットのスティッキーは使えません。[^c1-13]

### サーバーレスのデータ処理・分析パイプライン [1.1.3]

**Lambda** の上限が設計の出発点です。1 回の呼び出しは最大 900 秒（15 分）、メモリは 128〜10,240 MB（1 MB 刻み）、同期呼び出しのリクエストとレスポンスは各最大 6 MB です。15 分を超える処理や大きなペイロードは、Step Functions・Fargate・S3 経由の受け渡しに逃がします。リージョンの同時実行の既定は 1,000 です。1 関数は 10 秒ごとに 1,000 実行環境ずつ増やせます（スケール速度）。新しい AWS アカウントは同時実行・メモリの上限が下げられ、利用実績に応じて自動で引き上げられます。[^c1-14]

**ストリーム処理**は、Kinesis Data Streams を Lambda のイベントソースマッピングで読みます。Lambda がバッチ（レコードの束）を受け取り、バッチ全体が成功するまで同じバッチを再試行するので、1 件の不良レコードがシャードを止めます。失敗時の扱いとして、FunctionResponseTypes に ReportBatchItemFailures を入れ、失敗したレコードだけを返すと、成功分の再処理を減らせます。再試行の上限と、失敗時の送り先（SQS や SNS）も設定します。[^c1-15][^c1-16]

**Glue** は ETL のマネージドサービスです。1 DPU は 4 vCPU・16 GB のメモリです。G.1X は 1 DPU、G.2X は 2 DPU、G.4X は 4 DPU、G.8X は 8 DPU、G.025X は 0.25 DPU に相当します。**クローラー**は S3 などをスキャンして Data Catalog にテーブルとスキーマを登録します。[^c1-17][^c1-18]

**Athena** は S3 のデータを SQL で直接クエリし、スキャン量で課金されます。コストと速度は、データを **パーティション**で区切り、**列指向形式**（Parquet・ORC）にして圧縮すると下がります。DML クエリの実行時間にはタイムアウトがあり、引き上げを申請すると最大 240 分まで延ばせます。同時実行は Active DML・DDL クエリの上限を実行中と待機中の合計で数え、超えると TooManyRequestsException です。[^c1-19][^c1-20]

**Amazon Data Firehose** は、ストリームをバッファリングしてまとめて S3・Redshift・OpenSearch などへ配る、サーバー管理なしのサービスです。配信は少なくとも 1 回で、重複が出ることがあります。下流の処理を冪等にして、二重の処理を防ぎます。バッファリング設定に基づく配信で、頻度は配信先ごとに違い、リアルタイムではなくニアリアルタイムです。[^c1-21]

サーバーレス分析の構成の選び分けは、要件で決まります。

| 要件 | 選ぶもの |
|---|---|
| 取り込みと配信だけ | Firehose |
| 独自の処理を挟む（レコード単位の独自判定・低遅延） | Kinesis Data Streams＋Lambda（Firehose も Lambda で軽い変換は挟める） |
| 大規模な変換や結合 | Glue |
| 保存済みデータの対話的な集計 | Athena |

SAMPLE

たとえば IoT のログを Firehose で Parquet に変換して S3 に置き、Glue クローラーでカタログ化し、Athena で集計する流れは、サーバーを一切持たない典型の構成です。

### 永続的な実行の型を持つ多段のワークフロー [1.1.4]

**Step Functions** はステートマシンで複数のサービスをつなぎ、ワークフローの型は 2 つあります。

| 項目 | Standard | Express |
|---|---|---|
| 実行の保証 | exactly-once | at-least-once |
| 最長 | 1 年 | 5 分 |
| 向く処理 | 支払いなど非冪等な処理 | 冪等な処理・大量の短い処理 |
| 課金 | 状態遷移 | 実行回数・時間・メモリ |
| 実行開始（StartExecution）のスロットル | バケット 1,300・毎秒 300 を補充（バージニア北部・オレゴン・アイルランド。他のリージョンは 800・150） | バケット 6,000・毎秒 6,000 を補充 |

Standard の 1 実行の履歴は最大 25,000 イベントで、超えると実行がエラーで終わり、続けられません。長いループは子ワークフローに分けます。[^c1-22][^c1-23]

**エラー処理**は、Retry（エラー名・間隔・最大回数・バックオフ）と Catch（失敗時の遷移先）で書きます。複数のサービスにまたがる更新を取り消すには、**補償トランザクション**（Saga）を使います。各ステップに打ち消しの処理を用意し、途中で失敗したら、完了済みのステップの打ち消しだけを Catch から逆順に実行します（元に戻す処理。失敗したステップ自身は打ち消しません）。**コールバックの待ち**は、タスクトークンを渡して外部の処理の完了を待つ方式（`.waitForTaskToken`）です。人の承認や外部システムの応答を待つ時に使い、Standard の長い実行時間が効きます。Express はこの待ちに対応しません。[^c1-24][^c1-25][^c1-26]

**durable な Lambda 関数**（永続実行）は、チェックポイントとリプレイの仕組みで最長 1 年の多段ワークフローを関数コードの中で書けます。待機中は関数が一時停止し、その間は計算料金がかかりません。SDK でステップと待機（DurableContext）を使います。再開時は関数コードが最初から再実行され、完了済みのチェックポイントは保存済みの結果を使って飛ばされます。したがって、ステップの外にある副作用（時刻や乱数の取得、外部への書き込み）は再実行されて結果が変わるため、ステップの中に入れます。[^c1-27]

選び分けです。Lambda だけで完結し、コードで流れを書きたいなら durable な Lambda。複数サービスの組み合わせを図で見える化し、組み込みのリトライと多数のサービス統合に任せたいなら Step Functions です。

### アカウント・リージョンをまたぐイベントのルーティング [1.1.5]

クロスアカウントでは、送信側のルールが受信側アカウントのイベントバスをターゲットにし、送信側には送信を許す IAM ロールが要ります。受信側は、イベントバスの **リソースポリシー**で他アカウントからのイベントを許可します。AWS Organizations を使っていれば、組織を指定してその組織の全アカウントにバスへのアクセスを許可でき、特定のルールや送信元を許可・拒否することもできます。[^c1-28]

クロスリージョンは、別リージョンのイベントバスをルールのターゲット（バスの ARN を指定）に選んで転送します。送信側アカウントに IAM ロールを付け、特定のリージョンやルールを許可・拒否もできます。[^c1-29]

SAMPLE

**グローバルエンドポイント**は、リージョン障害に備える仕組みです。Route 53 のヘルスチェックを割り当て、異常になるとフェイルオーバーが始まります。「数分以内」に全カスタムイベントが副リージョンのイベントバスへ回り、追加料金はありません（ヘルスチェックや複製されたイベントには通常の料金がかかります）。送り手は 1 つのエンドポイントを使い続けるだけでよく、副リージョンにも同じバスとルールを用意しておきます。[^c1-30]

### マネージドのコンテナオーケストレーション [1.1.6]

**ECS** は AWS 独自のオーケストレーターで、運用が軽く、AWS との統合が深い選択肢です。**EKS** はマネージドの Kubernetes で、既存の Kubernetes の資産・ツール・移植性が要件にある時に選びます（コントロールプレーンは AWS が運用）。[^c1-31]

起動タイプは 2 つです。**Fargate** はサーバー管理が要らず、タスクや Pod ごとに専用の計算境界（カーネルや ENI を共有しない）を持ちます。**EC2** はインスタンスを自分で管理する代わりに、GPU や特権コンテナ、大きなメモリ、コスト最適化（Spot・Savings Plans）の自由度を得ます（運用の負担は増えます）。Fargate も Fargate Spot と Compute Savings Plans で割り引けますが、EC2 はインスタンスの種類や購入方法を細かく選べます。ECS の **キャパシティプロバイダ**は、Fargate・Fargate Spot・EC2 の Auto Scaling グループを組み合わせて、ベースと重みで配分します。[^c1-32]

ECS on Fargate のタスク定義は、CPU が 256（0.25 vCPU）から 32768（32 vCPU）までです。メモリは CPU に応じた組み合わせで決まり、16 vCPU は 32〜120 GB（8 GB 刻み）、32 vCPU は 60・120・244 GB のどれかで、最大は 244 GB です（8 vCPU 以上は Linux のみ）。Windows のタスクは 4 vCPU までです。ネットワークモードは awsvpc です。プラットフォームバージョン 1.4.0 以降の Linux タスクは、最低 20 GiB のエフェメラルストレージを受け取ります。ephemeralStorage パラメータで最大 200 GiB まで増やせます（1.4.0 以降は AES-256 で暗号化）。[^c1-33][^c1-34]

EKS on Fargate には制約があります。DaemonSet・特権コンテナ・GPU が使えません（デーモンはサイドカーにします）。Pod は Fargate プロファイルに一致しないと Pending のままです。Pod はプライベートサブネットだけで動き、AWS サービスへは NAT ゲートウェイ経由で出ます。NLB・ALB は IP ターゲットだけが使えます。[^c1-35]

EKS のノードは 3 通りです。**マネージドノードグループ**は EC2 のノード（Auto Scaling グループ）の作成・更新を AWS が支援します。**Karpenter** は未配置の Pod を見て、必要な大きさのインスタンスを直接起動する自動スケーラーで、自分のクラスターで動かし、設定や更新の方針を自分で決めます。**EKS Auto Mode** はノードの管理（インスタンスの選択・スケール・更新）まで AWS に任せます。[^c1-36][^c1-37]

### サービスメッシュ [1.1.7]

マイクロサービス間の通信と可観測性を整える手段が 2 つあります。

**ECS Service Connect** は、同じタスク内のプロキシ（サイドカー）がサービス間通信を仲介し、**AWS Cloud Map** の名前空間の短い名前でサービスを呼べます。接続先の選択はラウンドロビンで、過去の失敗情報を外れ値検出に使います。接続性能のメトリクスが ECS と CloudWatch のコンソールで見られ、あらゆるアプリで同じ形のメトリクスを得られます。Service Connect が結ぶのは ECS のサービス同士だけで、ECS 以外のサービスとは結べません。**サービスディスカバリ**（Cloud Map）単体は名前解決の仕組みで、プロキシを挟まず、負荷分散や再試行は自前です。[^c1-38][^c1-39]

SAMPLE

**VPC Lattice** は、サービス（リスナー・ルール・ターゲットグループで成る）を **サービスネットワーク**に関連付けます。VPC もサービスネットワークに関連付けると、複数 VPC・複数アカウントをまたいで通信させられます。ターゲットグループは ELB のものと似ていますが、同じものではありません。サービスネットワークやサービスには **認証ポリシー**（IAM 形式のリソースポリシー）を付け、呼び出し元を IAM の主体で制御します。アカウントをまたぐ共有は AWS RAM で行い、EC2・ECS・EKS・Lambda を同じ方式で扱えます。[^c1-40][^c1-41]

既定の上限は、サービスあたりリスナー 2・ターゲットグループ 10、リスナーあたりルール 10 です。ターゲットグループあたりターゲットは 1,000、リージョンあたりサービスは 2,000・サービスネットワークは 50、認証ポリシーは 10 KB までです。[^c1-42]

選び分けです。ECS だけで完結するなら Service Connect。EKS・EC2・Lambda が混在する、または複数アカウント・VPC のサービスを IP 重複の影響なく結びたいなら VPC Lattice です。VPC Lattice は CIDR が重なる VPC 同士でも使えます。

### コンテナイメージの管理 [1.1.8]

**ECR** のイメージスキャンは 2 種類です。

| 項目 | 基本スキャン | 拡張スキャン |
|---|---|---|
| 仕組み | CVE データベースでスキャン | Amazon Inspector と連携 |
| 対象 | OS パッケージ | OS とプログラミング言語パッケージの両方 |
| 実行 | プッシュ時または手動 | プッシュ時または継続的 |

拡張スキャンは結果が更新されると Amazon Inspector が EventBridge にイベントを送るので、検出をチケットや通知に自動でつなげられます。基本スキャンと拡張スキャンを切り替えると、それまでに設定したスキャンは使えなくなり、設定し直さなければなりません。[^c1-43]

**ライフサイクルポリシー**は、古いイメージを自動で期限切れにします。ルールは優先度の数字が小さいものから適用され、基準を満たしたイメージは 24 時間以内に期限切れ（またはアーカイブ）になります。意図どおりのイメージが対象になるかは、評価機能で先に確認できます（適用前の確認）。タグ付きイメージは tagPrefixList や tagPatternList で、タグなしイメージは untagged で指定します。基準は件数（imageCountMoreThan）や経過日数（sinceImagePushed）です。[^c1-44]

**複製**は、別リージョンや別アカウントのレジストリへイメージを自動でコピーします。DR や、利用地に近い場所からのプルに使います。**不変（immutable）タグ**を有効にすると、同じタグへの再プッシュが拒否され、デプロイしたタグの中身が後から変わりません。[^c1-45][^c1-46]

クォータは、リポジトリあたりイメージ 100,000、ライフサイクルポリシーあたりルール 50、イメージあたりタグ 1,000、基本スキャンは 24 時間で 100,000 イメージまでです。プルスルーキャッシュのルールはレジストリあたり 50 です。[^c1-47]

### サーバーレス・コンテナのオートスケーリング [1.1.9]

**Lambda の同時実行**は 2 種類の設定で制御します。**予約済み同時実行**は関数の最大値であり最小値でもあります。他の関数はその枠を使えず、追加料金はかかりません。ある関数が暴走しても他をスロットルさせない、または下流

SAMPLE

の DB を守るための上限にも使えます。**プロビジョニング済み同時実行**は事前に初期化した実行環境を用意し、コールドスタートを減らします。確保した分は料金がかかります。同時実行が上限に達した呼び出しは **スロットリング**（429）され、同期呼び出しはエラーを返し、非同期呼び出しはキューに戻って再試行されます。[^c1-48]

**Application Auto Scaling** は、複数のサービスのスケールを共通の仕組みで行います。対象は DynamoDB のテーブルと GSI、ECS サービス、Lambda のプロビジョニング済み同時実行、SageMaker AI のエンドポイントバリアント／推論コンポーネントなどです。ポリシーは **ターゲット追跡**（指標を目標値に保つ）、**ステップ**（しきい値ごとに増減量を決める）、**スケジュール**の 3 つです。[^c1-49]

ECS サービスはターゲット追跡・ステップ・スケジュール・予測スケーリングを使えます。ポリシーには **クールダウン**があり、スケールアウト中はそれ以上の追加が抑えられますが、より大きなスケールアウトが必要なら即座に実行されます。ターゲット追跡では ECSServiceAverageCPUUtilization などの定義済み指標を使います。[^c1-50]

**Kubernetes** のスケールは 2 層です。Pod の数は **HPA**（Horizontal Pod Autoscaler）が CPU やカスタム指標で増減します。ノードの数は、EKS Auto Mode、Karpenter、Kubernetes **Cluster Autoscaler** の 3 つの手段のどれかが増減します。Cluster Autoscaler は Auto Scaling グループのサイズを変え、Karpenter はグループを介さず Pod に合うインスタンスを選んで直接起動します。HPA が Pod を増やしても、ノードが足りなければ Pod は Pending のまま動きません。両方を組み合わせて使います。[^c1-51][^c1-52]

### 入口の型と資源の最適化 [1.1.10]

**ALB** は L7 のロードバランサーで、パス・ホスト・ヘッダーによるルーティング、Cognito などの認証、WAF の関連付けができます。**NLB** は L4 で、TCP・UDP・TLS を扱い、極めて高いスループットと低遅延、AZ ごとの **固定 IP**（Elastic IP を付けられる）を持ちます。固定 IP が要る、TCP/UDP の非 HTTP 通信、PrivateLink で公開する、のいずれかなら NLB を選び、HTTP のルーティングが要るなら ALB を選びます。**TLS の終端**はどちらでもでき、ACM の証明書をリスナーに付けます。[^c1-53]

クォータは、ALB がリージョンあたり 50、ALB あたりリスナー 50・ターゲットグループ 100（アクションあたり 5）・ターゲット 1,000・証明書 25（既定の証明書を除く）です。Lambda を対象にしたターゲットグループは 1 関数のみです。NLB はリージョンあたり 50、リスナー 50（変更不可）、ターゲット 3,000、AZ あたりターゲット 500、証明書 25 です。AZ ごとに 1 つのネットワークインターフェイスを使い、VPC あたり 1,200 が上限です。[^c1-54][^c1-55]

**API Gateway** のスロットルは、アカウント・リージョンごとに全 API 種別（HTTP・REST・WebSocket）の合計で 10,000 RPS です。バケットの最大容量は 5,000 リクエストのバーストです。引き上げ可能なのは RPS で、バーストはサービス側が決めます。一部のリージョンは既定 2,500 RPS・バースト 1,250 です。[^c1-56]

REST API は API キー、クライアントごとのスロットル（**使用量プラン**）、リクエスト検証、AWS WAF 連携を使えます。**プライベート API**（インターフェイスエンドポイント経由で VPC の中だけに公開）も REST API の機能です。HTTP API は機能を絞って低い価格にした製品なので、これらが要らなければ HTTP API を選びます。使用量プランは API キーごとのレートとクォータを決め、課金や取引先ごとの制限に使います。API キーだけでは認証になりません。識別と制限のための手段です。[^c1-57][^c1-58][^c1-59]

**AWS Load Balancer Controller** は、Ingress や Service を監視して Elastic Load Balancing のリソースを作ります。Ingress は ALB、Service（type LoadBalancer）は NLB です。挙動はリソースへのアノテーションで設定します。Pod へ直接つなぐ IP ターゲットモードも選べます。[^c1-60]

SAMPLE

### レイテンシ要件のためのエッジ [1.1.11]

エッジの処理は **CloudFront Functions** と **Lambda@Edge** で選び分けます。

| 項目 | CloudFront Functions | Lambda@Edge |
|---|---|---|
| 実行時間・規模 | 1 ミリ秒未満・毎秒数百万リクエスト | 最大 30 秒・1 リージョンあたり毎秒最大 10,000 |
| 関数メモリ | 2 MB | 128 MB（ビューワー）／10,240 MB（オリジン） |
| コードとライブラリ | 10 KB | 50 MB |
| 言語 | JavaScript（ECMAScript 5.1 準拠） | Node.js と Python |
| イベント | ビューワーリクエスト・レスポンスのみ | ビューワーとオリジンの 4 つ |
| 実行の場所 | CloudFront のエッジロケーション | 世界中に複製され、ビューワーに近い場所で動く（作成は US East (N. Virginia) のみ） |
| ネットワーク・ファイル・本文 | アクセス不可 | アクセス可 |
| KeyValueStore | 使える | 使えない |
| 位置情報・デバイス情報（CloudFront-Viewer-* のヘッダーをポリシーに含めた場合） | 使える | オリジンのイベントでのみ使える |

URL の書き換え、ヘッダーの操作、簡単なリダイレクトや認可トークンの検証など、軽くて大量の処理は CloudFront Functions です。外部 API の呼び出し、リクエスト本文の読み取り、オリジンでのレスポンス生成など、重い処理やネットワークが要るなら Lambda@Edge です。[^c1-61]

**キャッシュの振る舞い**でレイテンシを下げます。キャッシュキーに入れる要素（ヘッダー・クエリ・Cookie）を最小にするとヒット率が上がります。TTL は Cache-Control のヘッダーとキャッシュポリシーの最小・最大・既定で決まります。ビューワーに近いエッジで返せない時は、オリジンの近さが効きます（キャッシュミスの場合）。ビューワーが複数の地域に散らばる場合などは、**Origin Shield** でオリジンの前に中間のキャッシュ層を足し、オリジンへの要求を集約します（リージョンは、オリジンへのレイテンシが最も低いものを選びます）。動的なコンテンツも CloudFront 経由にすると、AWS のネットワークで接続が確立されて速くなります。[^c1-62][^c1-63]

### WebSocket と出版・購読によるリアルタイム処理 [1.1.12]

**API Gateway の WebSocket API** は、クライアントと双方向の接続を保ちます。ルートキーは、ルート選択式（API レベルの routeSelectionExpression 属性）の評価結果で決まります。JSON のメッセージは action などのプロパティで振り分け、JSON でないメッセージは設定した `$default` ルートへ行きます。[^c1-64]

接続が成功すると `$connect` ルートが実行され、認可は `$connect` ルートでのみ行います。切断時には `$disconnect` ルートが動きます。接続の管理として、接続 ID は `$connect` で DynamoDB などに保存し、`$disconnect` で削除します。@connections への DELETE は接続を切るだけで、保存した接続 ID は消えません。バックエンドは次のエンドポイントへ POST して、クライアントへメッセージを送ります。

```
POST https://{api-id}.execute-api.{region}.amazonaws.com/{stage}/@connections/{connection_id}
```

SAMPLE

接続情報の取得や切断も同じエンドポイントでできます（GET と DELETE）。リクエストの認可には IAM を使います（execute-api:ManageConnections）。[^c1-65][^c1-66]

WebSocket のスロットルは、アカウント・リージョンの上限（既定 10,000 RPS・バースト 5,000）を他の API と共有します。HTTP・REST・WebSocket・WebSocket コールバック API の合計に対する上限です。[^c1-56]

**出版・購読**の型は目的で選びます。

| 手段 | 役割 |
|---|---|
| SNS | サーバー側のファンアウト。モバイルプッシュ・メール・SQS・Lambda などへ配る |
| EventBridge | ルールによる振り分け |
| AppSync の GraphQL 購読 | クライアントへ WebSocket でリアルタイムに配信 |
| IoT Core | MQTT のトピックで大量のデバイスと双方向にやり取り |

ブラウザやアプリのクライアントへ配るなら AppSync か API Gateway の WebSocket、デバイスなら IoT Core を選びます。[^c1-67][^c1-68]

### マルチテナントとテナントの分離 [1.1.13]

テナントの分離は 3 つのモデルがあります。

| モデル | 中身 | 分離の強さ |
|---|---|---|
| サイロ | テナントごとに専用リソース（別スタックや別データベース） | 強い・費用大 |
| プール | テナントがリソースを共有する典型的なマルチテナント | 論理的・費用小 |
| ブリッジ | サイロとプールの混在 | 層ごとに選ぶ |

サイロでも、ID・オンボーディング・運用は共有の仕組みで管理します。[^c1-69]

**Cognito** のクォータは AWS アカウントとリージョンごとに適用され、全テナントで共有されます。テナントが増えると他のテナントの認証を巻き込んで制限に達しうるので、規模が大きいなら複数のアカウントやリージョンに分けます。分けるとそれぞれ別のクォータを持て、分離も最も強くなります（費用と運用は増えます）。共有ユーザープールなら、全テナントに同じ脅威保護・MFA・AWS WAF の基準を一括で適用でき、ユーザー管理の入口も 1 つです。WAF の web ACL と Lambda 関数は、関連付ける先と同じリージョンに置く必要があります。**テナント文脈**は、トークンのクレーム（tenant ID など）に持たせ、アプリと認可の判断に引き渡します。[^c1-70]

**IAM のタグ**（属性ベースのアクセス制御）は、プリンシパルのタグとリソースのタグを条件で比べ、テナントごとにポリシーを増やさずに分離します。DynamoDB では、IAM ポリシーの条件キー `dynamodb:LeadingKeys`（パーティションキー）を使います。各テナントのキー値の項目だけにアクセスを絞る、行単位のきめ細かなアクセス制御ができます。[^c1-71]

**テナントごとの暗号化キー**は、KMS のキーをテナントごとに作り、キーポリシーで当該テナントのロールだけに許します。削除や無効化でそのテナントのデータだけを読めなくでき、監査の証跡もテナント別に取れます。リレーショナ

SAMPLE

ル DB のプール型では、**行レベルのセキュリティ**（PostgreSQL の行レベルセキュリティ）で、テナント ID の列に応じて返す行を DB の側で制限します。アプリの WHERE 句を書き忘れても、他のテナントの行は返りません。

たとえば、少数の大口テナントは厳格な分離を求め、多数の小口テナントは費用を抑えたいとします。答えはブリッジです。大口にはサイロ（専用スタックと専用の KMS キー）を、小口にはプール（共有テーブルと LeadingKeys による行単位の制限）を割り当てます。ID の入口は共有の Cognito ユーザープールにして、トークンのテナント ID で認可します（テナント文脈）。

### 生成 AI の組み込み [1.2.1]

**Amazon Bedrock** は、Amazon・Anthropic・OpenAI など 100 以上の基盤モデルを、フルマネージドの 1 つのサービスから使えます。[^c1-72]

**クロスリージョン推論**は、推論プロファイルで、特定の地域（US・EU など）に限る **地理プロファイル**か、商用リージョンを自動で選ぶ **グローバルプロファイル**を選びます。需要の波でスロットルされにくくなります。ルーティングの追加料金はなく、データは AWS ネットワーク内を通り、CloudTrail のログは送信元リージョンに残ります。データの所在地を制限したいなら地理プロファイルです。[^c1-73]

**プロビジョンドスループット**（Provisioned Throughput）は、モデルユニット（MU）の数・モデル・コミット期間（なし・1 か月・6 か月）で時間課金されます。期間が長いほど時間単価が下がります。カスタマイズしたモデルは、プロビジョンドスループットで呼ぶか、対応するモデル（Amazon Nova など）ならカスタムモデルのデプロイでオンデマンド推論で呼びます。推論プロファイルでは Provisioned Throughput を使えません。安定した大量の呼び出しや、予測できる低い遅延が要る時に選びます。[^c1-74]

**Guardrails** には、コンテンツフィルター、拒否トピック、単語フィルター、機密情報フィルター（機微情報フィルター・ブロックまたはマスク）があります。ほかにコンテキストのグラウンディングチェック（根拠チェック）と Automated Reasoning チェックがあります。拒否トピックは話題を自然文で定義して言い回しが変わっても止め、単語フィルターは決まった語だけを止めます。`ApplyGuardrail` API なら、モデルを呼ばずにガードレールだけ評価できます。[^c1-75]

**マルチアカウント**では、基盤モデルの利用を中央の Bedrock アカウントに集め、各アプリのアカウントからクロスアカウントで呼び出す構成が使えます。中央に Guardrails・モデルのアクセス・利用状況の集計・予算を置くと、統制と費用の把握がしやすくなります。呼び出し側のアカウントのロールが、中央アカウントの IAM ロールを AssumeRole して Bedrock を呼びます（信頼ポリシーで許可）。

### ツールと連携する AI エージェント [1.2.2]

**Bedrock Agents** は、モデルが指示を解釈して、アクショングループ（呼び出せる API の集まり）やナレッジベースを使いながら複数ステップの作業を進めます。**アクショングループ**は、OpenAPI スキーマで API を定義し、実行を Lambda 関数に任せます。[^c1-76]

**Amazon Bedrock AgentCore** は、エージェントを本番で動かすための部品の集まりです。

| 部品 | 役割 |
|---|---|
| Runtime | AI エージェントとツールをデプロイ・スケールするサーバーレスのランタイム。速いコールドスタート、非同期エージェント向けの長時間実行、セッションの完全な分離を備え、CrewAI・LangGraph・Strands など任意のフレームワークとモデルで使える |

SAMPLE

| 部品 | 役割 |
| --- | --- |
| Gateway | 既存の API・Lambda 関数・サービスを MCP（Model Context Protocol）対応のツールに変換し、既存の MCP サーバーとも接続して、Gateway のエンドポイント経由でエージェントに提供する |
| Memory | 複数ターンの会話向けの短期記憶と、セッションをまたぐ長期記憶を提供し、エージェント間で共有できる |
| Identity | 既存の ID プロバイダーと互換のエージェント用 ID・認証管理で、ユーザー移行は要らない |
| Code Interpreter と Browser | エージェントが使う組み込みツール（コードの実行と Web の閲覧） |

Harness（1 回の API 呼び出しでエージェントを動かす管理型ループ）は、各セッションを分離された microVM で実行します。[^c1-77]

選び分けです。Bedrock の機能に閉じた管理型のエージェントなら Bedrock Agents。別のフレームワークで書いた既存のエージェントを、セッション分離つきで本番に載せるなら AgentCore Runtime です。社内の API をエージェントのツールとして公開するなら Gateway で MCP 化します。ツール連携の認証と資格情報の管理は Identity に任せます（記録は Observability の役割）。

### ML の推論のアーキテクチャ [1.2.3]

**SageMaker AI** の推論は 4 種類です。

| 種類 | 向く負荷 |
| --- | --- |
| リアルタイム | 低遅延の対話型 |
| サーバーレス | アイドル期間があり、コールドスタートを許容できる負荷 |
| 非同期 | 大きなペイロードや長い処理 |
| バッチ変換 | 永続エンドポイントが要らない大規模データセット |

[^c1-78]

非同期推論は、ペイロードが最大 1 GB、処理時間が最大 1 時間です。入力は S3 に置いて呼び出し、結果は S3 に出ます。リクエストはキューに入り、SNS で成功・失敗を通知でき、リクエストが無い間はインスタンス数を 0 にオートスケールできます。[^c1-79]

サーバーレス推論にプロビジョニング済み同時実行を足すと、確保した分のコンピュートが初期化済みになり、ミリ秒で応答します。確保した同時実行は設定メモリに基づいて課金され、Application Auto Scaling で管理できます。[^c1-80]

バッチ変換は、大規模データセットから推論を得る時や、永続エンドポイントが要らない時に使います。ジョブの間だけ処理し、データの前処理や本番バリアントのテストにも使えます。[^c1-81]

**マルチモデルエンドポイント**は、多数の小さなモデルを 1 つのエンドポイントとインスタンス群で共有し、必要な時にだけモデルを読み込んでコストを下げます。**推論のオートスケーリング**は、Application Auto Scaling でバリアントのインスタンス数を、インスタンスあたりの呼び出し数の目標値などで増減します。[^c1-49]

SAMPLE

### RAG（取り込みと検索の戦略） [1.2.4]

**RAG** は、外部の文書から関連部分を検索し、プロンプトに添えてモデルに答えさせる方式です。社内の最新データを使え、モデルの再学習が要りません。**Bedrock Knowledge Bases** が取り込みから検索までを管理します。

取り込みでは、文書を **チャンク**に分けて **埋め込み**（ベクトル）にし、ベクトルストアに保存します。既定のチャンクは約 300 トークンです。戦略は 3 つです。

| 戦略 | 動き |
|---|---|
| 固定サイズ | トークン数と重複率を指定する |
| 階層（親子） | 検索時に子チャンクで当てて、親チャンクに置き換える |
| セマンティック | 意味のまとまりで分割する |

[^c1-82]

データソースを変更したら、追加・変更・削除をナレッジベースに取り込むため、同期（取り込みジョブ）が要ります。直接取り込み・削除に対応するデータソースでは、同期と更新を別の手順にしなくてよくなります。[^c1-83]

ベクトルストアは、自分で用意するなら Amazon **OpenSearch Serverless**、**Aurora**（pgvector）、Neptune などを選べます。取り込みと検索を自分で制御できます。構造化データは Amazon Redshift のクエリエンジン経由で接続します。既存の Aurora や OpenSearch に文書の検索を同居させたいなら、それを選びます。[^c1-84]

検索は **Retrieve API** で関連チャンクを取り出します。検索の種類はハイブリッド（キーワードとセマンティック）かセマンティックを指定でき、指定しなければ Bedrock がベクトルストアに合う方式を選びます（フルマネージドのナレッジベースは常にハイブリッドです）。既定の結果数は 5 件です。`numberOfResults` で最大件数を指定でき、既定のランカーの代わりに再ランクモデルを使えます。ガードレールは検索結果の参照には適用されません。精度が足りない時は、チャンクの戦略（階層やセマンティック）、結果数、再ランクの順に見直します。[^c1-85]

### 冗長とフェイルオーバーを備えたハイブリッド接続 [1.3.1]

**Direct Connect** の冗長構成は、回復力のモデルで決まります。

| モデル | 構成 | SLA |
|---|---|---|
| Maximum resiliency | 複数のロケーションで、それぞれ別々のデバイスに終端する接続を持つ | 99.99% |
| High resiliency | 複数のロケーションに 1 本ずつの接続を持つ | 99.9% |
| Development and test | 1 つのロケーションで、別々のデバイスに終端する接続 | なし（重要でない負荷向け） |

SLA を得るには、Direct Connect のサービスレベル契約の要件を満たす必要があります。[^c1-86] **LAG**（リンクアグリゲーション）は、同じ Direct Connect エンドポイント（同じ AWS デバイス・同じロケーション）に終端する複数の接続を LACP で 1 本として扱い、帯域を増やします。デバイスやロケーションの障害には耐えないので、LAG だけでは回復力のモデルを満たしません。**BFD** は各仮想インターフェイスで AWS 側が自動で有効にしており、自社のルーターで設定すると効きます。経路の間の切り替えが速まります。[^c1-87]

SAMPLE

バックアップには **Site-to-Site VPN** を使います。1 接続には 2 本のトンネル（各自固有のパブリック IP）があり、片方が止まるともう一方へ自動で切り替わるので、両方の設定が必要です。帯域は標準で 1 トンネル最大 1.25 Gbps（既定）です。Large Bandwidth Tunnel は最大 5 Gbps ですが、Transit Gateway か Cloud WAN にアタッチした接続でしか使えません。Direct Connect と VPN の両方で同じ宛先の経路を受けると、AWS 側の経路の優先順位で Direct Connect の経路が VPN の経路より優先され、Direct Connect が止まると VPN へ流れます。VPN は暗号化されますが、通信はインターネットを通ります（専用線ではありません）。[^c1-88][^c1-89]

接続の構成要素と上限は次の表のとおりです。

| 項目 | 内容・上限 |
|---|---|
| Direct Connect ゲートウェイ | 複数リージョンの VPC や Transit Gateway へ 1 本の接続でつなぐ |
| Transit VIF | Direct Connect ゲートウェイを Transit Gateway につなぐ |
| 専用接続 1 本あたりの VIF | private／public VIF は 50、transit VIF は 4（合計 51、引き上げ不可）。ホスト接続は VIF 1 つのみ |
| Site-to-Site VPN | リージョンあたり接続 50、仮想プライベートゲートウェイあたり接続 10 |
| Transit Gateway | アカウント・リージョンあたり 5、TGW あたりルートテーブル 20、全ルートテーブル合計 10,000 ルート、アタッチメント 5,000。同じ VPC への VPC アタッチメントは 1 つの TGW に 1 つまで |

[^c1-90][^c1-91][^c1-92]

場面ごとの判断は、次の表のとおりに選びます（条件を読み取る練習になります）。

| 場面 | 選ぶもの |
|---|---|
| 専用線が使えない期間の、低コストなバックアップ | Site-to-Site VPN |
| ロケーション障害にも耐え、SLA 99.99% が要る | Maximum resiliency |
| 多数のリージョンの VPC を 1 本でつなぐ | Direct Connect ゲートウェイ＋Transit VIF＋Transit Gateway |

### ハイブリッドのデータ同期 [1.3.2]

**DataSync** は、オンプレミスの NFS・SMB・HDFS と、AWS の S3・EFS・FSx の間でデータを移します。FSx は Windows File Server・Lustre・OpenZFS・NetApp ONTAP が対象です。**エージェント**をオンプレミスに置き、**スケジュール**で定期同期できます。転送中の暗号化とデータ整合性の検証は自動で行われ、利用者が組み立てる必要はありません。メタデータの扱い・除外・帯域制限はタスクで指定します。アカウントとリージョンあたりタスクは 100（引き上げ可）です。Basic モードのタスク実行 1 回で扱えるファイル・オブジェクト・ディレクトリは、オンプレミスなどと AWS の間で 5,000 万まで（AWS のストレージどうしは 2,500 万まで・どちらも引き上げ可）で、Enhanced モードに事実上の上限はありません。S3 のプレフィックスもディレクトリとして数えます。[^c1-93][^c1-94]

**Storage Gateway** は、オンプレミスから AWS のストレージを使う入口です。

| 種類 | 見え方 | 保存先 |
|---|---|---|
| S3 File Gateway | NFS（v3・4.1）／SMB のファイル | S3 のオブジェクト |

SAMPLE

| 種類 | 見え方 | 保存先 |
|---|---|---|
| FSx File Gateway（新規のお客様には提供終了） | SMB のファイル | FSx for Windows File Server |
| Volume Gateway | iSCSI のブロックデバイス | S3（EBS スナップショット） |
| Tape Gateway | 仮想テープライブラリ（iSCSI VTL）。既存のバックアップソフトはそのままに、物理テープを置き換える | S3・Glacier 系 |

S3 File Gateway は、ローカルキャッシュで低遅延アクセスを提供します。オンプレミスの VM／ハードウェアアプライアンス、または EC2 の AMI としてデプロイします（EC2 なら AWS 内にも置けます）。Volume Gateway のキャッシュボリュームは S3 を主ストレージとし、よく使うデータだけ手元にキャッシュします。1 GiB〜32 TiB、1 ゲートウェイで最大 32 ボリューム・合計 1,024 TiB です。ストアドボリュームは、全データを手元に置いて S3 へ非同期にバックアップします。手元に全データが要るならストアド、容量を絞りたいならキャッシュです。[^c1-95][^c1-96][^c1-97]

**Transfer Family** は、SFTP・FTPS・FTP・AS2（とウェブブラウザベース）でファイルを S3 や EFS に直接転送できる（AS2 の保存先は S3 だけ）フルマネージドサービスです。FTP と FTPS はデータ接続のポート範囲に注意が要ります。取引先が既存の SFTP クライアントを使い続ける時や、EDI の AS2 には Transfer Family を選びます。一度きりの大量移行や定期同期は DataSync です。[^c1-98]

### データ変換によるレガシーの近代化 [1.3.3]

古いシステムを段階的に新しくする時は、**API Gateway と Lambda** で、レガシーの API を外に見せる形へ変換します。API Gateway は、メソッドリクエストと統合リクエストでペイロード形式が違う時に変換できます。

**パラメータマッピング**は、統合リクエストの URL パス・クエリ文字列・ヘッダーを変えられますが、ペイロードは変えられません。ペイロードの変換には **マッピングテンプレート**（非プロキシ統合）を使います。パラメータマッピングは統合リクエストならプロキシ・非プロキシのどちらでも使えます。統合レスポンスのマッピングは非プロキシ統合が必要です。[^c1-99]

| 統合タイプ | 値 | 備考 |
|---|---|---|
| Lambda プロキシ | AWS_PROXY | 変換は関数側 |
| Lambda カスタム | AWS | マッピングテンプレートが使える |
| HTTP プロキシ | HTTP_PROXY | 変換なしで転送 |
| HTTP カスタム | HTTP | 変換できる |
| AWS サービス | AWS | 非プロキシのみ |
| モック | MOCK | リクエストに自分で応答する |

[^c1-100]

**Amazon MQ** は、Apache ActiveMQ Classic と RabbitMQ のマネージドメッセージブローカーで、セットアップ・運用・保守を管理します。既存のブローカーは、メッセージングのコードを書き換えずに移行できます（アプリ側の変更は不要）。ActiveMQ のデプロイは、シングルインスタンスブローカー（EFS または EBS ストレージ）と、高可用性のた

SAMPLE

めのアクティブ/スタンバイブローカーの 2 つです。選び分けの軸は、既存のプロトコル（JMS・AMQP・MQTT・STOMP など）を維持するかです。維持するなら Amazon MQ、新規に作り、スケールと運用の軽さを取るなら SQS／SNS です。[^c1-101][^c1-102]

### ハイブリッドの DNS [1.3.4]

**Route 53 VPC Resolver**（旧 Route 53 Resolver）は、VPC の DNS の入口です。VPC では VPC+2 の IP アドレスで接続し、VPC 固有の DNS 名とプライベートホストゾーンのレコードを既定で解決します。ガイドの「Route 53 Resolver」は現在この名前に改名されています。[^c1-103]

| エンドポイント | 向き | 使う場面 |
|---|---|---|
| インバウンド | オンプレミスから VPC への DNS クエリ | オンプレミスが AWS 上の名前を引く |
| アウトバウンド | VPC からオンプレミスなどへのクエリ | AWS がオンプレミスの名前を引く |

**転送ルール**は、ドメイン名ごとに 1 つ作る条件付きフォワーディングです。アウトバウンドエンドポイントは、クエリの送信元になる IP アドレスを VPC のアドレス範囲から選んで指定し、1 つ以上の転送ルールとセットで作ります。転送を始める前に、接続した VPC にインバウンド／アウトバウンドのエンドポイントを作っておきます。エンドポイントの IP アドレスは 2 つ以上が必須で、可用性のため 2 つ以上の AZ に置きます。ルールのターゲットには転送先の DNS サーバーの IP アドレスを指定します（ルールのドメイン名の配下の名前も、このルールで転送されます）。[^c1-104]

**プライベートホストゾーン**は、関連付けた VPC の中だけで引ける DNS ゾーンです。**スプリットビュー**（split-horizon）DNS は、同じ名前のパブリックとプライベートのホストゾーンを作り、VPC をプライベートホストゾーンに関連付けます。パブリックのレコードはインターネット向け、プライベートのレコードは VPC 内の経路を決めます。名前空間が重なる場合は、最も具体的な一致で解決され、広い一致は使われません。VPC では `enableDnsHostnames` と `enableDnsSupport` が必要です。[^c1-105]

たとえば、オンプレミスの端末が AWS 上のプライベートホストゾーンの名前を引き、AWS 上のサーバーは社内ドメインを引きたいとします。インバウンドエンドポイントを作り、オンプレミスの DNS から転送します。アウトバウンドエンドポイントには、社内ドメインの転送ルールを付けます。どちらも複数の AZ に置きます（可用性のため）。

### ハイブリッドのコンピュート [1.3.5]

**Outposts** は、AWS のインフラをオンプレミスに置き、AWS が運用・監視・管理します。形態は 2 つで、Outposts ラックは業界標準の 42U ラック、Outposts サーバーは 1U または 2U のサーバーです。サービスリンクで親リージョンとつながり、ローカルゲートウェイ（LGW）でオンプレミスのネットワークとつながります。データの所在地の制約や、オンプレミスの機器との超低遅延が要る時に選びます（Outposts が向く場面）。[^c1-106]

**Local Zones** は、大都市圏や産業の中心地の近くに計算・ストレージ・データベースなど一部のリソースを置き、低遅延のアプリケーションを動かします。AWS が運用するので、自社の設置場所は要りません。ハイブリッド移行では、低遅延の要件を保ったまま近くの Local Zone へアプリケーションを移せます。選び分けは、設備を自社の施設に置く必要があるなら Outposts、設置不要で都市に近づけるだけなら Local Zones です。[^c1-107]

**ECS Anywhere** は、オンプレミスのサーバーや VM を外部インスタンスとして ECS クラスターに登録し、EXTERNAL 起動タイプでサービスやタスクを動かします。Elastic Load Balancing に非対応なので、受信トラフィックがあるワー

SAMPLE

クロードには効率が悪く、外部インスタンスは同時に 1 つのクラスターにのみ登録できます。受信トラフィックの無いバッチ処理などに向きます。[^c1-108]

オンプレミスで EKS を動かすには 3 つの手段があります。Outposts 上の Amazon Linux ノードを使う方法です。Amazon **EKS Hybrid Nodes** は、自前のインフラをクラスターのノードにします。Amazon **EKS Anywhere** は、自己完結の閉域（air-gapped）環境向けです。[^c1-109]

### 複数 VPC・マルチアカウントの接続 [1.4.1]

**VPC ピアリング**は、2 つの VPC を 1 対 1 でつなぎます。ピア VPC 同士の CIDR ブロックは重複できず、**推移的**な経路は使えません（A-B と B-C があっても A と C は通信できない）。同一リージョンのピアなら、ピア VPC のセキュリティグループをルールで参照できます。IPv6 で通信するには、各 VPC に IPv6 CIDR ブロックが要ります。[^c1-110]

**Transit Gateway** は、多数の VPC と VPN・Direct Connect を 1 つのハブにつなぐ、推移的なルーターです。VPC が数個で 1 対 1 の単純な接続ならピアリング、規模が大きく経路を集中管理するなら Transit Gateway を選びます。ルートテーブルを複数持てるので、アタッチメントを関連付けるテーブルと、経路を伝播させるテーブルを分けられます。環境ごとに **ルートテーブルで分割**でき、本番と開発を互いに見えなくして、共有サービスだけ両方から見えるようにできます（分割の実例）。リージョンをまたぐ接続は TGW のピアリングです。**AWS Cloud WAN** は、コアネットワークを作り、ポリシー・セグメント・アタッチメントで管理するグローバルなネットワークサービスです。アタッチメントには Transit Gateway や VPC などを含みます。コアネットワークを使わない場合は AWS Global Networks for Transit Gateways を使います。[^c1-111][^c1-112]

**PrivateLink** は、サービスを一方向に公開する仕組みです。サービス提供者がロードバランサー（NLB か GWLB）の背後に **エンドポイントサービス**を作ります。既定では利用者に公開されず、許可した利用者がインターフェイスエンドポイントで接続します。接続の開始は利用者側からだけで、サービス側から利用者の VPC へは接続できません。CIDR が重なっていても使えます。Gateway Load Balancer エンドポイントは、ルートテーブル経由で仮想アプライアンス群へ流します。[^c1-113][^c1-114]

| エンドポイント | 対象 | 仕組み |
|---|---|---|
| ゲートウェイ | S3・DynamoDB | ルートテーブルのターゲット。PrivateLink を使わない・追加料金なし |
| インターフェイス | 多くの AWS サービス・自作サービス | ENI（プライベート IP）。PrivateLink を使う |

S3 のゲートウェイエンドポイントは、オンプレミス、別リージョンのピア VPC、Transit Gateway 経由からはアクセスできません。S3 バケットと同じリージョンに作ります。それらから S3 に私的に届けたい時は、インターフェイスエンドポイントを使います。[^c1-115]

### VPC の設計 [1.4.2]

サブネットの IPv4 CIDR は /28〜/16 です。各サブネットの先頭 4 つと末尾 1 つの IP アドレスは使えません（/24 なら 256 から 5 を引いて 251）。同じ VPC のサブネット同士は重複できません。python3 の ipaddress で確かめると、/28 は 16 アドレスで使えるのは 11、10.0.0.0/16 を /20 で分けると 16 個になります。[^c1-116]

設計の型は AZ ごとの配置で、パブリックとプライベートのサブネットを 1 つずつ（高可用なら 2〜3 AZ）置きます。パブリックにはインターネットゲートウェイへの経路を持つロードバランサーや NAT ゲートウェイを置き、アプリとデー

SAMPLE

タはプライベートに置きます。NAT ゲートウェイは AZ ごとに置くと、AZ 障害の波及と AZ 間通信の費用を避けられます（AZ をまたぐ NAT の共用はしません）。将来のサブネット追加とピアリング・接続を見越し、他の VPC やオンプレミスと重ならない CIDR を選びます。

既定のクォータは、リージョンあたり VPC 5、VPC あたりサブネット 200 です。VPC あたり IPv4 CIDR ブロックは 5（最大 50 まで）、AZ あたり NAT ゲートウェイは 5 で、すべて引き上げ可能です。[^c1-117]

**IPAM**（VPC IP Address Manager）を作ると、プライベートスコープとパブリックスコープが自動で作られます。プライベートはインターネットにアドバタイズしない IP 用、パブリックはアドバタイズ可能な IP 用です。スコープは 1 つのネットワークの IP 空間を表し、未接続のネットワークで同じ IP を再利用するには、プライベートスコープを追加で作ります。スコープの中では **プール**が重複しないように割り当てます。IP 空間をルーティング・セキュリティのドメインに分け、使用状況と IP を使うリソースを監視します（Organizations と連携すると、組織全体の CIDR の衝突も防げます）。ビジネスルールに従って VPC へ CIDR を自動で割り当て、IP 割り当ての履歴を見られ、BYOIP のリージョン・アカウント間共有もできます。[^c1-118][^c1-119]

### ネットワークのセキュリティ [1.4.3]

**Network Firewall** は、ステートフルなマネージドのネットワークファイアウォール兼侵入検知・防止サービスです。ステートフル検査にオープンソースの IPS である **Suricata** を使い、Suricata 互換のルールを使えます。プロトコル検出で HTTPS などをポートに依存せず絞れます。ドメイン名での許可リストにも使えます。[^c1-120]

**集中型の検査 VPC** は、Transit Gateway を経由する全トラフィックを、検査用 VPC のファイアウォールへ通す構成です。仮想アプライアンスを使う場合は **Gateway Load Balancer** が、透過的なゲートウェイとして展開・拡張を担います。アプライアンスとは GENEVE プロトコルのポート 6081 でトラフィックをやり取りします。ターゲットへのスティッキーは 5-tuple（既定）・3-tuple・2-tuple から選べ、5-tuple では、送信元と宛先の IP・ポートとプロトコルの 5 つが同じフローを同じアプライアンスに送ります。各 VPC 側の Gateway Load Balancer エンドポイントをルートテーブルのターゲットにしないと、通信は検査を通りません。[^c1-121]

たとえば、複数アカウントの VPC から出るインターネット向けの通信を、1 か所で検査したいとします。検査用 VPC に Network Firewall を置き、各 VPC は Transit Gateway 経由でそこへ向けます。サードパーティのアプライアンスを使うなら Gateway Load Balancer を挟み、台数は Auto Scaling で増減します。行きと戻りを同じアプライアンスに通すため、検査用 VPC のアタッチメントでアプライアンスモードを有効にします（同じ AZ に固定）。[^c1-122]

| 項目 | セキュリティグループ | ネットワーク ACL |
|---|---|---|
| 単位 | インスタンス（ENI） | サブネット |
| 状態 | ステートフル（戻りは自動で許可） | ステートレス（戻りも書く） |
| ルール | 許可だけ | 許可と拒否 |
| 評価 | すべてのルールを合わせて判断 | ルール番号 1〜32766 の小さい番号から、最初に一致したルールを適用 |

特定の IP を拒否したい時はネットワーク ACL、通常の許可はセキュリティグループです。VPC を作ると既定のセキュリティグループが付きます。セキュリティグループは、共有 VPC の参加者のアカウントと AWS Organizations の中で共有したり、同じリージョンの自分の VPC に関連付けたりもできます（既定のセキュリティグループは共有も関連付けもできません）。[^c1-123][^c1-124]

SAMPLE

**AWS WAF** の web ACL は、CloudFront、API Gateway の REST API、ALB、AWS AppSync の GraphQL API に関連付けられます。Cognito ユーザープール、App Runner、Verified Access インスタンスなども対象です。ルールの複雑さは Web ACL capacity units（WCU）で数えます。ルールは自作のほか、AWS **マネージドルール**（SQL インジェクションや既知の不正 IP の群など）を使えます。同じ送信元からの過剰なリクエストは、レートベースのルールで抑えます（一定時間あたりの回数で判定）。**Shield Advanced** は、大規模な DDoS への追加の検知・緩和と、DDoS 対応チームの支援、DDoS による費用の増加の補償を提供します。L7 の防御は WAF、L3/L4 の大規模攻撃には Shield です。[^c1-125]

**VPC フローログ**は、ネットワークインターフェイスの IP トラフィック（送信元・宛先・ポート・プロトコル・許可か拒否か）を記録します。出力先（配信先）は CloudWatch Logs・S3・Amazon Data Firehose の 3 つで、配信経路と権限の設定をサブスクリプションと呼びます。Transit Gateway のフローログは別の機能です。[^c1-126]

### IPv6 の導入とデュアルスタックへの移行 [1.4.4]

既存の IPv4 専用 VPC に IPv6 対応を追加でき、IPv4 と IPv6 の通信は互いに独立します。IPv4 を無効にはできません。IPv4 専用サブネットから IPv6 専用サブネットへの移行経路はありません。段階は、VPC に IPv6 CIDR を足す、サブネットに割り当てる、ルートテーブルに IPv6 の経路を足す、セキュリティグループと ACL に IPv6 のルールを足す、の順です。[^c1-127]

Amazon 提供の IPv6 アドレスは常にインターネットにアドバタイズされます。アドバタイズされず、インターネットから到達できないプライベート IPv6 アドレスも使えます。IPv6 アドレスは 128 ビットで、8 つの 16 進 4 桁のセグメントです。IPv6 では NAT が不要で、インスタンスは固有のグローバルなアドレスを持ちます。[^c1-128]

**Egress-only インターネットゲートウェイ**（Egress-only IGW）は、水平スケールで冗長な、高可用性の VPC コンポーネントです。VPC のインスタンスからインターネットへの IPv6 アウトバウンド通信を許し、インターネット側からの IPv6 接続開始は防ぎます。IPv4 の NAT ゲートウェイに相当します。[^c1-129]

**DNS64 と NAT64** は、IPv6 のみのリソースが IPv4 のみのサービスへ届くための仕組みです。サブネットで DNS64 を有効にすると、Route 53 VPC Resolver が IPv4 のみのレコードから IPv6 アドレスを合成します。合成には既知の /96 プレフィックス `64:ff9b::/96` を使い、経路は NAT ゲートウェイへ向けます。NAT ゲートウェイの NAT64 が、IPv6 から IPv4 への通信を変換します。たとえば 192.0.2.1 は、python3 で確かめると `64:ff9b::c000:201` になります（変換の例）。[^c1-130]

### マルチリージョンのグローバルな経路制御 [1.4.5]

**Route 53 のルーティングポリシー**は次の使い分けです。

| ポリシー | 動き | 使う場面 |
|---|---|---|
| シンプル | 単一リソースを返す | 1 つの宛先 |
| 加重 | 比率で配分する | カナリアや段階移行 |
| レイテンシー | 最良の遅延のリージョンへ | 性能優先のマルチリージョン |
| フェイルオーバー | アクティブ-パッシブ | 障害時の切り替え |
| 位置情報 | ユーザーの所在地で決める | 法令や言語の制約 |

SAMPLE

| ポリシー | 動き | 使う場面 |
|---|---|---|
| 地理的近接性 | 場所の近さ＋バイアスで調整 | 負荷を偏らせたい |
| IP ベース | 送信元の CIDR で決める | 特定の ISP や企業網 |
| 複数値回答 | 最大 8 件の正常な値を返す | 簡易な分散と死活の除外 |

プライベートホストゾーンでも使えます。位置情報は所在地の制約、地理的近接性は距離とバイアスで割合を調整する、レイテンシーは測定した遅延が最小、と覚え分けます。[^c1-131]

**ヘルスチェック**は次の 3 種類です。

| 種類 | 監視の対象 |
|---|---|
| エンドポイント | エンドポイントそのもの |
| 計算されたヘルスチェック | 他のヘルスチェックの結果を束ねる |
| CloudWatch アラーム | アラームの状態 |

アラーム型は、アラームが ALARM 状態になるのを待たず、アラームと同じデータストリームを監視して判定します。エンドポイント型は、確認間隔を 10 秒か 30 秒から選び、連続して失敗する回数（失敗しきい値）を指定できます。ヘルスチェッカーの 18% 超が正常と報告すれば正常、18% 以下なら異常と判定します。プライベートなリソースは、ヘルスチェッカーから直接は確認できません。CloudWatch アラームを監視する型で間接的に確認します。[^c1-132] [^c1-133]

**Global Accelerator** は、既定でエニーキャストの静的 IPv4 アドレスを 2 つ提供し、デュアルスタックなら IPv4 を 2 つと IPv6 を 2 つの計 4 つです。アクセラレーターが存在する限り同じアドレスが割り当てられ、削除するとアドレスを失います（BYOIP も使えます）。標準アクセラレーターは AWS のグローバルネットワークで、ヘルス・クライアントの場所・設定したポリシーから最適なリージョンのエンドポイントへ送ります。カスタムルーティングアクセラレーターは、1 人以上のユーザーを特定の宛先に対応づけます。[^c1-134]

選び分けです。TCP/UDP の非 HTTP、固定 IP が要る、DNS キャッシュに左右されない即時のフェイルオーバーが要るなら Global Accelerator。HTTP/HTTPS のコンテンツをキャッシュして配るなら CloudFront です。DNS で十分で費用を抑えるなら Route 53 のポリシーです。

たとえば、東京と大阪で同じ構成を動かし、東京が止まったら大阪へ自動で切り替えたいとします。DNS だけで足りるなら、Route 53 のフェイルオーバーポリシーにヘルスチェックを付けます（アクティブ-パッシブ）。切り替えが DNS のキャッシュに左右されず、固定の IP アドレスも要るなら、Global Accelerator に両リージョンのエンドポイントを登録します。両リージョンで常時受けたいなら、レイテンシーポリシーです（両方がアクティブ）。

### データレイク・レイクハウス [1.5.1]

**データレイク**は、S3 に生データを置いて用途ごとに処理する構成で、メタデータは Glue Data Catalog に置きます（Lake Formation の権限と合わせて使う）。**Lake Formation** は IAM の権限を補う独自の権限モデルです。grant／revoke により、列・行・セルの粒度で Athena・Redshift Spectrum・EMR・AWS Glue などへ権限を強制します。対象は Data Catalog に登録したデータです。**ハイブリッドアクセスモード**では、Data Catalog のデータを Lake

SAMPLE

Formation の権限と、S3・Glue 向けの IAM ポリシーの両方で保護できます。管理者は Lake Formation の権限を選択的に導入できるので、既存の運用を止めずに少しずつ移せます（段階的な移行）。[^c1-135][^c1-136]

**LF-TBAC**（タグベースのアクセス制御）は、属性に基づく認可の方式で、その属性を **LF-Tag** と呼びます。LF-Tag を Data Catalog のデータベース・テーブル・列に付け、その LF-Tag を使って Lake Formation のプリンシパルに権限を付与します。リソースが増えても、タグを付けるだけで権限が及び、名前での個別付与より運用が軽くなります。[^c1-137]

**Redshift Spectrum** は、Redshift のテーブルへロードせずに、S3 の構造化・半構造化データを直接クエリします。処理は Spectrum 層で行われ、データの大半は S3 に残り、複数クラスターが同じデータセットをコピーなしで同時にクエリできます。頻繁に使う集計は Redshift に置き、まれにしか使わない大量の履歴は S3 に残す、という使い分けをします。[^c1-138]

**Apache Iceberg** のテーブルは、S3 上のデータに ACID とスキーマ進化、タイムトラベルを与えるテーブル形式で、レイクハウスの中核です。Athena は Iceberg テーブルに対して読み取り・タイムトラベル・書き込み・DDL のクエリを実行できます。S3 のテーブルバケットは Iceberg 形式（V3 に対応）でテーブルを保存し、Athena・Redshift・Spark からクエリできます。[^c1-139]

### リアルタイムとバッチの分析パイプライン [1.5.2]

**EMR** は Hadoop や Spark の大規模処理のマネージドサービスです。実行の場は、EC2 上のクラスター、**EMR Serverless**（クラスター管理なし）、**EMR on EKS**（既存の EKS クラスター上で実行）の 3 つです。プライマリノードはクラスターを管理し、YARN ResourceManager と HDFS NameNode を動かします。EMR 5.23.0 以降は、プライマリノードを 3 台にして高可用性にできます。1 台が障害になると自動でスタンバイへフェイルオーバーし、同じ構成の新しいノードで置き換えます（利用者の操作は不要）。コアノードは HDFS の DataNode を動かしてデータを保存し、YARN NodeManager や Spark エグゼキューターも動かします。1 クラスターに置けるコアのインスタンスグループ（またはフリート）は 1 つだけで、複数は持てません。[^c1-140][^c1-141]

**MSK** は Apache Kafka のマネージドサービスです。MSK Provisioned（クラスターを作る）と MSK Serverless（容量管理なしで Kafka を実行）があり、MSK Serverless のクライアント認証は IAM アクセス制御です。MSK Connect でクラスターへ／からデータを流し、MSK Replicator で MSK Provisioned クラスター間を、同じ／別リージョンへレプリケートします。選び分けは、既存の Kafka のアプリや API、エコシステムを使い続けるなら MSK、AWS ネイティブで運用を最小にするなら Kinesis です。[^c1-142]

**Redshift の設計**は、分散キーとソートキーが中心です。分散キーは結合や集計で同じ値の行を同じノードに集め、データの移動を減らします。結合に使う大きい表は KEY 分散、小さい表は ALL、判断がつかなければ AUTO を選びます（AUTO は Redshift が選ぶ）。ソートキーは範囲検索の対象になる列（日時など）に置き、読むブロックを減らします。**RA3** ノードは計算とマネージドストレージを分けて、容量と計算を別々に増減できます。**Redshift Serverless** は、ペタバイト規模のデータウェアハウスを、設定なしで使えるサーバーレス版です。リソースは自動でプロビジョニング・スケールされ、アイドルの間は課金されません。負荷が断続的ならサーバーレス、常時安定した大量処理ならプロビジョンドです。[^c1-143][^c1-144]

**Amazon AppFlow** は、Salesforce などの SaaS と、S3・Redshift などの AWS サービスの間でデータを交換するフルマネージドサービスです。オンデマンドまたはスケジュールでフローを実行します。保存時と転送中を暗号化し、

SAMPLE

PrivateLink で非公開の転送もできます。S3 に送ったデータを Glue Data Catalog に登録でき、SaaS 連携のコードを書かずに取り込めます。[^c1-145]

### 要件に合うデータベースの選定 [1.5.3]

選び方の軸は、アクセスのパターン、整合性の要件、スケールです。

| 要件 | 選ぶもの |
|---|---|
| 結合・トランザクション・既存のエンジン | RDS |
| 同じエンジン互換で高い性能・高可用・読み取りの拡張 | Aurora |
| キー値や単純なアクセスを桁違いの規模・一桁ミリ秒で | DynamoDB |
| 大規模な分析（列指向・SQL） | Redshift |
| ドキュメント（MongoDB 互換） | DocumentDB |
| 高度に連結したグラフ | Neptune |
| Cassandra 互換のワイドカラム | Keyspaces |
| 時系列 | Timestream |
| 耐久性のあるインメモリ | MemoryDB |

[^c1-146]

**DocumentDB** は MongoDB 互換です。クラスターは 0〜16 インスタンスで、書き込みはプライマリのみ、最大 15 のレプリカが同じストレージを共有します。データは 3 つの AZ にコピーされ、ストレージは 10 GB 刻みで自動拡張し、エンジン 8.0 以降は最大 256 TiB（それ以前は 128 TiB）です。[^c1-147]

**Neptune** は、高度に連結したデータ向けのフルマネージドなグラフデータベースです。プロパティグラフの Apache TinkerPop Gremlin と openCypher、RDF の SPARQL に対応し、数十億の関係をミリ秒の遅延でクエリします。[^c1-148]

目的別のデータベースのうち、**Keyspaces** は Apache Cassandra 互換のサーバーレスなワイドカラムデータベースです。**Timestream** は時系列データ向けのマネージドなデータベースで、サーバーレスの Timestream for LiveAnalytics は 2025 年 6 月 20 日から新規のお客様に提供されないので、新規の利用では Timestream for InfluxDB を使います。**MemoryDB** は耐久性のあるインメモリデータベースで、Valkey と Redis OSS に互換です。マイクロ秒の読み取りとミリ秒一桁の書き込みを持ち、Multi-AZ のトランザクションログでデータを複数 AZ に永続化します。キャッシュと別に耐久 DB を持たなくてよく、主データベースとして使えます（キャッシュとは別の位置づけ）。[^c1-149][^c1-150]

**DynamoDB のキー設計**は、アクセスパターンから逆算します。パーティションキーは値の種類が多く、アクセスが均等になる属性にします。偏ると特定のパーティションが熱くなり、スロットルされます（ホットパーティション）。ソートキーは範囲検索や並べ替えに使います。別のキーで引きたい時は GSI を足します。[^c1-151]

**整合性**です。DynamoDB の読み取りは、既定が結果整合性で、強い整合性読み取りの半額です。強い整合性読み取りはテーブルと LSI のみ対応し、GSI と DynamoDB Streams からの読み取りは結果整合性のみです。**トランザクシ**

SAMPLE

**ョン**（TransactWriteItems／TransactGetItems）は、複数の項目をまたいで全部成功か全部失敗にでき、費用は通常の書き込みの 2 倍の容量を使います。[^c1-152][^c1-153]

### 高可用のためのデータベースのスケーリング [1.5.4]

**RDS のマルチ AZ** は 2 形態です。

| 項目 | Multi-AZ DB インスタンス | Multi-AZ DB クラスター |
|---|---|---|
| スタンバイ | 1 台 | 2 台（読み取り可能） |
| 読み取りトラフィック | 受けない（フェイルオーバー専用） | 受けられる |

[^c1-154]

**リードレプリカ**は、非同期でコピーされ、読み取りの負荷を分散します。クロスリージョンのレプリカは災害対策に使え、スタンドアロンに昇格できます。マルチ AZ はフェイルオーバーのための同期、リードレプリカは読み取りの拡張のための非同期、と目的が違います。Aurora クラスターは最大 15 の Aurora レプリカを持てて、共有ストレージなのでレプリカラグは通常 100 ミリ秒よりはるかに小さく、フェイルオーバーの先にもなります。[^c1-155]

**RDS Proxy** は、接続プールを確立して接続を再利用し、すぐに処理できない接続はキューイングまたはスロットリングします。Lambda など接続を大量に開く呼び出しから DB を守り、フェイルオーバー時の接続の切れも短くします。クライアントに IAM 認証を強制でき、DB へは IAM 認証または Secrets Manager の資格情報で接続します。[^c1-156]

**Aurora Global Database** は、1 つのプライマリリージョン（書き込み）と最大 10 の読み取り専用セカンダリリージョンで構成します。セカンダリクラスターが持てる読み取り専用インスタンスは最大 16 で、書き込み用はありません。**ライトフォワーディング**で、セカンダリへの書き込みをプライマリへ転送できます。**スイッチオーバー**（旧「マネージド計画フェイルオーバー」）は、計画的な切り替えで、データ損失なしにリージョンを切り替えます。障害時の切り替えは**フェイルオーバー**（セカンダリをプライマリへ昇格）です。ストレージボリュームのレベルで複製するため低オーバーヘッドで、従来のレプリケーションより RTO・RPO が小さくなります。[^c1-157][^c1-158]

**DynamoDB グローバルテーブル**は、マルチアクティブで、どのレプリカも読み書きに応じます。非同期レプリケーションで、衝突の解決は last-writer-wins が基本です。強い整合性（MRSC）構成もありますが、MRSC は同一アカウント構成のみ対応します。書き込みを複数リージョンで受けたいなら DynamoDB グローバルテーブル、1 か所で書いて他は読み取りと DR なら Aurora Global Database を選びます。[^c1-159]

たとえば、MySQL 互換の DB を、リージョン障害から短い RTO で復旧させたいとします。Aurora Global Database を選び、障害時はセカンダリをプライマリに昇格させます。計画的なリージョン移動なら、スイッチオーバーでデータ損失なしに切り替えます。RDS のクロスリージョンのリードレプリカも昇格できますが、ストレージ層の複製ほど RPO・RTO を詰められません。

### アクセスの型と性能要件に合うストレージ [1.5.5]

**EBS・EFS・FSx・S3** の選び分けです。

SAMPLE

| ストレージ | 種類 | 向く場面 |
|---|---|---|
| EBS | ブロック（単一 AZ・基本は 1 インスタンス） | OS・DB のディスク |
| EFS | NFS のファイル（複数 AZ・Linux） | 共有のファイル領域 |
| FSx | 各種ファイルシステム | 特定のプロトコルや性能 |
| S3 | オブジェクト | 大量の非構造データ・バックアップ・データレイク |

**FSx** は次の 4 種類です。

| 種類 | 特徴 |
|---|---|
| Windows File Server | SMB と Active Directory 連携 |
| Lustre | HPC や機械学習向けの高速な並列ファイルシステム。S3 と連携 |
| NetApp ONTAP | NFS・SMB・iSCSI の複数プロトコルと NetApp の機能 |
| OpenZFS | NFS で ZFS の機能（スナップショットなど） |

[^c1-160][^c1-161][^c1-162][^c1-163]

**EBS** の SSD で耐久性が最も高いのは io2 Block Express（99.999%・年間故障率 0.001%）です。サブミリ秒レイテンシーが要る負荷や 80,000 IOPS／2,000 MiB/s を超える負荷向けです（gp3・gp2・io1 の耐久性は 99.8%〜99.9%）。

| 種類 | サイズ | 耐久性 |
|---|---|---|
| gp3 | 1 GiB〜64 TiB | 99.8%〜99.9% |
| gp2 | 1 GiB〜16 TiB | 99.8%〜99.9% |
| io2 Block Express | 4 GiB〜64 TiB | 99.999% |

io2 Block Express は、Nitro ベースのインスタンスでは最大 256,000 IOPS までプロビジョニングできます。それ以外のインスタンスタイプは最大 64,000 IOPS のボリュームをアタッチでき、実効は 32,000 IOPS までです。io2 Block Express の最大スループットは 4,000 MiB/s です。io1 が最大の 1,000 MiB/s に届くには、64,000 IOPS のプロビジョニングと Nitro ベースのインスタンスが要ります。[^c1-164]

**EFS** は、パフォーマンスモードが General Purpose（推奨）と Max I/O（操作あたりの遅延が高い）、ファイルシステムタイプは Regional か One Zone です。スループットモードは Elastic・Provisioned・Bursting があり、Elastic は既定で最大スループットを自動で上げ下げします。[^c1-165]

**S3 の性能**は、プレフィックスごとのリクエスト数で見ます。パーティション化されたプレフィックスごとに、少なくとも PUT/COPY/POST/DELETE が毎秒 3,500、GET/HEAD が毎秒 5,500 です。バケット内のプレフィックス数に上限はなく、10 プレフィックスに分ければ読み取りは毎秒 55,000（python3 で 5,500×10）まで並列化できます（スケールは徐々に進む）。大きなオブジェクトは **マルチパートアップロード**（1 パート 5 MiB〜5 GiB、最大 10,000 パート、最大 48.8 TiB）で並列に送ります。遠いクライアントからのアップロードは **Transfer Acceleration**（エッジ経由）で速めま

SAMPLE

す。**S3 Express One Zone** は、単一 AZ に置く最小の遅延向けのクラスで、ディレクトリバケットを使います。[^c1-166][^c1-167][^c1-168][^c1-169]

ストレージクラスの課金は次のとおりです。IA 系のクラスは 128 KB 未満のオブジェクトでも 128 KB 分を課金し、最小保存期間は 30 日です（短く削除すると残りを日割りで課金）。Glacier 系は最小保存期間と取り出し料金があり、まれにしか使わないデータ向けです。**S3 Intelligent-Tiering** は、アクセスパターンが変わる、または不明なデータ向けで、取り出し料金はなく、オブジェクト数に応じた小額の監視・自動化料金が課金されます。128 KB 未満のオブジェクトは監視対象外です。耐久性 99.999999999%・可用性 99.9% 設計で、低遅延の 3 つのアクセス階層を自動で行き来します。[^c1-170]

### データのキャッシュ戦略 [1.5.6]

**ElastiCache** は Valkey・Memcached・Redis OSS を提供します。最もシンプルなモデルが要るなら Memcached、豊富なデータ構造・検索・パーシステンスなどが要るなら Valkey または Redis OSS を選びます。キャッシュ作成後に古いエンジンバージョンへは戻せません。戦略は、読み取り中心なら遅延読み込み、常に最新が要るなら write-through、どちらにも TTL を付けて古さを抑えます（1.1.2 の節で述べたとおり）。[^c1-171]

**DAX**（DynamoDB Accelerator）は、DynamoDB 互換（API 互換）のインメモリキャッシュで、結果整合性読み取りをマイクロ秒で返します。強い整合性読み取りを指定したリクエストは、DAX を通り抜けて DynamoDB に渡されるので、キャッシュの恩恵を受けません。アイテムキャッシュの TTL は既定で 5 分です。PutItem などは、DAX を経由して DynamoDB に書く write-through の操作として扱われます。クラスターが過負荷だと ThrottlingException を返してリクエストの受け付け速度を絞り、ThrottledRequestCount メトリクスを CloudWatch に出します。読み取りが多く、同じ項目を何度も読む DynamoDB に向き、強い整合性が必須の読み取りや、書き込みが中心の負荷には効きません。[^c1-172]

選び分けです。DynamoDB の読み取りだけを速くしたいなら、アプリを変えずに済む DAX。複数の DB のクエリ結果やセッションなど汎用のキャッシュなら ElastiCache。キャッシュではなく主データベースとして使うインメモリなら **MemoryDB** です。[^c1-173]

### データカタログとアカウントをまたぐデータ共有 [1.5.7]

**Glue Data Catalog** は、データセットのメタデータ（場所・スキーマ・実行時メトリクス）を集中管理するリポジトリです。メタデータテーブルが 1 つのデータストアを表します。クローラーが自動でデータソースをスキャンして埋めるほか、手動でもテーブルを作れます。[^c1-174]

**アカウントをまたぐカタログの共有**は、Lake Formation で行います。外部アカウントや別アカウントの IAM プリンシパルに Data Catalog リソースの権限を付与すると、**AWS RAM** でリソースを共有します。付与先が同じ組織なら共有リソースをすぐに使えます。同じ組織でなければ、招待を承諾するまで使えません。テーブル・列レベルの保護が使えます。**AWS RAM** は、リソースを AWS アカウント間、Organizations 内の全アカウント、または特定のアカウント ID と共有する仕組みです。リソースを 1 度作れば、他のアカウントから使えるようにできます。受け手のアカウントでは、共有されたテーブルへのリソースリンクを作ってクエリします。[^c1-175][^c1-176]

**AWS Clean Rooms** は、生データを共有・移動せずに、複数の当事者のデータを分析するための安全なコラボレーション環境です。**分析ルール**で SQL クエリを制限し、暗号化したままの計算（Cryptographic Computing）と差分プ

SAMPLE

ライバシーで、利用者の特定を防ぎます。広告主とメディアのように、互いに生データを見せられない相手と共同で集計する時は Clean Rooms、同じ組織内でテーブルを権限つきで見せるなら Lake Formation の共有です。[^c1-177]

---

SAMPLE

## サンプルはここまでです

続き(第2章〜巻末)は、AWS SAP-C03 問題集のご購入で、頻出度別ノート・暗記用ノートと合わせて3点すべてダウンロードできます。

SAMPLE

## AWS SAP-C03 学習用テキスト(無料サンプル)

**版**
2026年11月版(AWS 公式 Exam Guide（SAP-C03・2026年11月17日開始）準拠)

**発行**
IT資格道場(https://www.shikaku-doujou.com)

**お問い合わせ**
info@shikaku-doujou.com

AWS および Amazon Web Services は Amazon.com, Inc. またはその関連会社の商標です。記載されているその他の会社名・製品名・サービス名は、各社の商標または登録商標です。

本教材は公式シラバスの内容を基に作成したもので、Amazon Web Services との提携・承認関係はありません。

本文・図表・設問はすべてオリジナルの著作物です。無断での転載・複製・再配布・二次販売はできません。お気づきの点は info@shikaku-doujou.com までご連絡ください。

SAMPLE