ド事業者との責任分界点や機能・制限が異なり、場合によっては非機能要求事項を満たせない可能性があります。まずは要求事項を満たすために、構成やほかのサービスとの併用によってクリアできるか、その際にかかる費用は許容されるかなど、使用するサービスの機能や制限を詳細に調査しましょう。

またオンプレミスと同様、要求を満たすためにはこの構成にするといったある程度決まったパターンがあります。構成例はAWSが公式に出しているものが多数ありますので、「AWSサービス別資料」で知りたいサービスや構成の情報を確認するとよいでしょう。

経験上、オンプレミスと比べてクラウドが大きく信頼性に欠ける印象はありません。もちろん、システム要件に合致した正しい設計・構築がなされ、正しく運用されていることが前提です。非機能要求は高くなりすぎると、実際にかかる環境のコスト以上に設計・構築・運用コストが高騰します。細かい部分までコントロールしたい場合は「クラウドを使用しない」や「一部をオンプレミスで構築する」といった方針も検討できますが、クラウドを有効活用していくのであれば要求事項をクラウドに合った形へ見なおしていくことも必要でしょう。

参考：「AWSサービス別資料」 https://aws.amazon.com/jp/events/aws-event-resource/archive/?cards.sort-by=item.additionalFields.SortDate&cards.sort-order=desc&awsf.tech-category=*all

2.2 クラウドで考えるセキュリティ

クラウドセキュリティの責任分界点

クラウドを利用するうえでの大きな懸念のひとつとして、「セキュリティリスク」が挙げられます。クラウド事業者との責任分界点を理解すれば、クラウドを利用する私たちが取るべき対応もおのずと見え、セキュリティリスクを低減して安全にクラウド上のサービスを利用できます。

責任分界点とは

責任分界点とはその名のとおり「サービスの提供者と利用者間で責任を分ける点」のことで、**万が一、事故が起こったときや何らかの対応が必要なときに責任の所在を明らかにするもの**であり、クラウドに限った用語ではありません。

たとえば電力会社が一般の家庭へ送電しているケースでいうと、軒先などに取り付けられている引込線取付点が責任分界点です。責任分界点を超えた先、つまりメーターや屋内配線で問題が生じた場合の対応の責任は、契約者にあります。

AWSでも同じく、AWSが責任を持つ領域と、AWSを利用する開発者が責任を持つ領域が責任分界点によって分けられています。

AWSの責任共有モデル

クラウドのサービスを利用してシステムを構築するとき、多くの場合は物理サーバやストレージなどを別途用意する必要はありません。そのため開発者は、たとえばハードウェアのEOSLや、ドライバ・ユーティリティのバージョンアップなどを考慮せずに済みます。しかしOSやアプリケーションのバージョンやバグ対応・ファイアウォール設定などの管理責任は、開発者が持たねばなりません。このように複数の関係者が担当するレイヤ（物理レイヤ、OSレイヤ、アプリケーションレイヤなど）のセキュリティ保護を負い、全体最適することを「責任共有モデル」と呼びます（**図2.2.1**）。

▶**図2.2.1** AWSの責任共有モデル

4.1 AWSのネットワーク設計

クラウドはIaaS、PaaS、SaaSとさまざまなサービス提供形態があります。そのどれを利用する場合でも、物理機器のサービスを利用しない限り、ネットワーク機器やサーバ機器などの物理設計は不要です。しかしクラウドならではの検討事項があります。

本節ではAWSでシステム構築をするにあたり、ネットワークに焦点を当ててオンプレミスと比較した設計観点を記載します。

オンプレミスとは異なる設計の注意点

まず初めにネットワーク設計においてオンプレミスとAWSで異なる点を整理しましょう。最も大きな違いは、ネットワークアドレスやIPアドレスの設計をオンプレミスほど詳細に行わなくてもよいことです。オンプレミスでは「ネットワークアドレスは変更できない」からこそ、慎重に用途を絞り、拡張性も考慮したうえで設計します。

しかしAWSではマネージドサービスが使用するIPアドレス分を大きく確保しなければならないうえ、作成済みのVPCやサブネットはあとから拡張可能（制限あり）なため、ある程度ざっくりと設計しても問題ありません。ネットワークでは通信経路やアクセス制御にフォーカスして設計します。

そのほかの、オンプレミスとAWSでの主なネットワーク設計の違いを**表4.1.1**に示します。

AWSのネットワーク構成要素

リージョンとアベイラビリティゾーン

クラウドといっても必ずどこかに物理のサーバが存在します。AWSではその物理サーバが置かれる**1つ以上のデータセンターの塊をアベイラビリティゾーン（AZ）**と呼び、**複数のAZを含む地理的エリアをリージョン**と呼んでいます（**図4.1.1**）。2023年4月時点では全世界に31リージョン、99アベイラビリティゾーンが存在します。

▶**表4.1.1** オンプレミスとAWS（クラウド）のネットワーク設計比較

設計項目	オンプレミス	AWS（クラウド）	備考
ネットワークアドレスの範囲	構築するシステムによってはIPアドレスを割り当てるホスト数を厳密に算出し、最小限のネットワークアドレスを払い出す	AWSマネージドサービスで使用される分があり、サブネットのサブネットマスクは/24までを推奨（IPv4の場合、VPCもサブネットも/16〜/28が範囲）	マネージドサービスで必要になるIPは、各AWSサービスの説明資料を確認する
セグメントの設計	場所的観点（○階や○○室）、利用用途（サービスLAN、運用LAN、監視LANなど）でセグメントを分ける	インターネットアクセス・外部システムとの接点有無、ルーティングポリシーでセグメントを分ける	帯域は10Gb/sから用意されており、セグメントを分割しなくても十分に通信量を確保できる。必要に応じて帯域の広いインスタンスを選択する
IPアドレスの設計	サーバは若番、ネットワークは老番からIPアドレスを振る、同じサーバやネットワーク機器は第4オクテットのアドレスをそろえるなど、プロジェクトによってルールあり	特定のIPは予約されており使用できない。IPアドレスを固定して使わず、ホスト名によるアクセスを基本とする	マネージドサービスのIPは変わることが前提。また固定していないEC2インスタンスなどのプライベートIPアドレスは変更できない
名前解決	システム内は内部DNSサーバを立てるか、各サーバのhostsファイルを利用し名前解決を行う	AWSでは自動でパブリックのDNS名が割り当てられ、マネージドのDNSで名前解決する。独自に取得したDNS名はRoute 53で管理・名前解決する	AWSで持つDNS名でアクセスするため、ホスト名もそれほど重要視されない
時刻同期	システム内の時刻が一意になるように同一のNTPサーバを割り当てる。インターネットにアクセスできないセグメントのサーバやネットワーク機器は、システム内に建てたNTPサーバへアクセスし、時刻同期する	Amazon Time Sync Serviceと時刻同期する（インターネットアクセス不要）	マネージドサービスへ個別にNTPを設定できないため、EC2インスタンスなどもAWSの用意するAmazon Time Sync Serviceを利用する

参考：「AWSグローバルインフラストラクチャ」 https://aws.amazon.com/jp/about-aws/global-infrastructure/

日本人が作る・日本人向けのシステムは東京リージョン（ap-northeast-1）または大阪リージョン（ap-northeast-3）に作成するのが一般的です。場合によってはほかのリージョンの利用も選択肢になりますが、提供されるサービスや費用はリージョンごとに細かく異なるため選定には注意が必要です。また大阪リージョンはもともとバックアップ用途で作成された経緯があり、機能制限があります。使えるAWSサービスや機能・制限は必ず確認しましょう。

- **バインドマウント**
  - ホスト(EC2、Fargate)上のファイルまたはディレクトリをコンテナからマウントする
  - バインドマウントされているコンテナがすべて停止すると、そのデータは削除される(デフォルトではコンテナが終了してから3時間後)
- **Fargateタスクストレージ**
  - Fargateでホストされている各ECSタスクがバインドマウントするエフェメラルストレージ
  - タスク定義内でvolumes、mountPoints、volumesFromパラメータを使用しているコンテナ間で共有できる

負荷に応じてスケーリングする

EC2をAuto Scalingで増減させるように、ECSでも負荷に応じて実行環境であるEC2やタスク数を増減できます。

実行環境をEC2としている場合、ECS Cluster Auto Scaling(CAS)を設定し、ECSで実行されているタスクが必要とするリソースに基づいてスケールイン・スケールアウトを自動的に行います。スケールイン・スケールアウトはAuto Scalingグループ設定の範囲のため、適切にインスタンスの最小値と最大値を設定します(**図5.4.3**)。

Amazon AuroraやLambdaで設定できるApplication Auto ScalingをECSに設定し、タスクレイヤでもスケールイン・スケールアウトを設定可能です。前述の、実行環境をEC2としている場合はEC2インスタンス数を増減させますが、Application Auto Scalingではタスクの必要数を増減し負荷を分散させます(**図5.4.4**)。

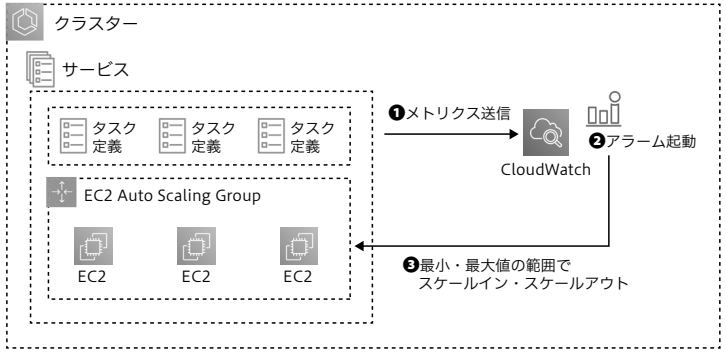
スケールイン・スケールアウトさせるタイミングは、以下の3種類で設定します。

- **ターゲット追跡スケーリングポリシー**
  - 特定のメトリクスのターゲット値に基づいて実行するタスク数を増減させる(複数設定可)
- **ステップスケーリングポリシー**
  - CloudWatchアラームをトリガにスケールイン・スケールアウトする
  - ターゲット追跡スケーリングポリシーが機能しない場合はステップスケーリングを使用する
  - ターゲット追跡スケーリングポリシーと併用可能
- **スケジュールに戻づくスケーリング**
  - 日付と時刻に基づいてサービスが実行するタスクの数を増減させる

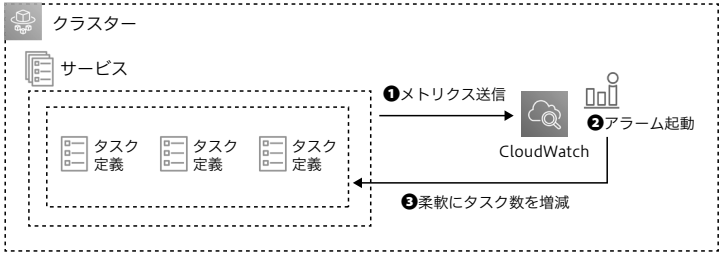
スケールイン・スケールアウトのトリガは、複数設定すると動作が複雑になります。想定した動作になるよう、綿密なテストを計画し、設定値を調整していきましょう。

システムのマイクロサービス化、コンテナ化は、「自動化」と同じように導入するまでのハードルが高く、工数も増大してしまう問題を抱えています。新しいAWSサービスや今まで対応していない技術はまるで魔法のように「今抱えている問題や課題を解決してくれる」と思えるかもしれません。しかし構築・運用するのは「人」です。システムのライフサイクル全体を通して本当に導入する価値があるのなら、そこに関わる人たちの理解を得られるように事前に働きかけるのも重要です。

▶**図5.4.3** Auto ScalingによるECSのスケールイン・スケールアウト



▶**図5.4.4** Application Auto ScalingによるECSのスケールイン・スケールアウト



6.4 Amazon DynamoDB

DynamoDBはフルマネージド型のNoSQLデータベースです。NoSQLは「NO SQL」ではなく「Not Only SQL（SQLだけではない）」を意味し、データをXML形式やJSON形式などのドキュメントとして扱います。

DynamoDBの特徴

DynamoDBは高スループットを実現するデータベースです。主な特徴は以下のとおりです。

- **高信頼性**
  - リージョン内3AZへ同期され、高い可用性と耐障害性を兼ね備えている
- **高スループット**
  - テーブルごとのReadとWriteそれぞれにスループットキャパシティを柔軟に割り当てられる
  - オンライン（無停止）でキャパシティを変更可能
- **サーバレス**
  - VPCの設計が不要で、事前に設定するキャパシティに基づいて自動スケーリングできる
  - 容量が無制限で、データのパーティショニング（分割）も自動で行われる

整合性モデル

読み込みの整合性には、「結果整合性のある読み込み」と「強力な整合性のある読み込み」があります。

- **結果整合性のある読み込み**
  - 読み込みスループットが高い
  - 最新の書き込み結果が反映されていない可能性がある
- **強力な整合性のある読み込み**
  - 必ずすべてデータが更新されてから結果を読み込む

DynamoDBはデフォルトで「結果整合性のある読み込み」を行います。DynamoDBでは少なくとも2AZ（アベイラビリティゾーン）で書き込み完了確認後、およそ1秒以内にAck（確認応答）を返します。書き込み後1秒の間にアクセスした際のデータ不整合（最新のデータとは異なる結果を返す）を許容できる場合は、「結果整合性のある読み込み」のままでよいでしょう。

DynamoDBで「強力な整合性のある読み込み」とするには、DynamoDBへのリクエスト時にConsistentReadパラメータを使用します。「強力な整合性のある読み込み」では必ずすべての更新された結果を読み込みます。ただし「結果整合性のある読み込み」と比べて（課金単位の一つである）キャパシティユニットを2倍消費するため、費用がかかることに注意しましょう。

またDynamoDBトランザクション読み込み／書き込みAPIを使用して、DynamoDBのテーブルに対してトランザクション処理（一括処理）を行えます。複数の項目追加や更新、削除が必要な処理では、データに一貫性をもたせるためにトランザクション読み込み／書き込みAPIを使用してください。

キャパシティモードとテーブルクラス

DynamoDBには「オンデマンドキャパシティモード」と「プロビジョニング済みキャパシティモード」の2種類のキャパシティモードがあります。それぞれの違いを**表6.4.1**に示します。

キャパシティモードはあとから変更も可能です。ただし以下の制約がありますのでご注意ください。

- **24時間に1回のみ変更できる**
- **プロビジョニング済みキャパシティモードからオンデマンドキャパシティモードに変換する場合、マネジメントコンソール経由で実施するとオートスケーリング設定が削除される可能性がある**
- **AWS CLIやAWS SDKを使用して変更すると、再度プロビジョニング済みキャパシティモードへ戻した際に以前の設定を使用できる**

オンデマンドキャパシティモードとプロビジョニング済みキャパシティ

表6.4.1 キャパシティモードの違い

キャパシティモード	課金箇所	ユースケース
オンデマンド	実際にDynamoDBに対して実行されたデータの読み込み／書き込み	トラフィック量が予測できない場合 使った分支払うほうが都合のよい場合
プロビジョニング済み	必要とされる1秒あたりの読み込み／書き込みの回数 格納されたデータ量	トラフィックが予測可能な場合 コストをコントロールしたい場合

8.1 アプリケーション統合サービスの種類と選択

アプリケーション統合サービス

AWSでは疎結合されたコンポーネント間をつなぐサービスをアプリケーション統合サービスと呼んでいます。アプリケーション統合サービスには**表8.1.1**の種類があります。

前段・後段で連携するAWSサービスやその内容は、それぞれのAWSサービスによって変わります。表8.1.1のアプリケーション統合サービスはAWSのマネージドサービスのため、高い可用性とスケーラビリティを備えており、障害の検出やリトライ動作を設計しておくことで障害に強いシステムを構成できます。

▶**表8.1.1** AWSのアプリケーション統合サービス

カテゴリ	AWSサービス	特徴
API管理	Amazon API Gateway	後段にLambdaやDynamoDBなどを配置するWebアプリケーションのAPIを提供
	AWS App Sync	後段にLambdaやDynamoDBなどを配置しGraphQLで開発するバックエンドのAPIを提供
イベントバス	Amazon EventBridge	AWSサービスが発行するさまざまなイベントをトリガに、別AWSサービスと連携する
メッセージング	Amazon Simple Notification Service (SNS)	CloudWatchアラームやLambdaなどからキックされ、pub/sub、SMS、電子メール、モバイルプッシュ通知を行う
	Amazon Simple Queue Service (SQS)	マイクロサービスやサーバレスアプリケーションのデータを疎結合化する
	Amazon MQ	Apache ActiveMQおよびRabbitMQ向けのマネージド型メッセージブローカサービス
コードなしでのAPI統合	Amazon AppFlow	SlackなどのサードパーティSaaSサービスと連携し、RDSなどに格納したデータを分析できるようにする
ワークフロー	AWS Step Functions	複数のAWSサービスを使って視覚的にワークフローを作成できる
	Amazon Managed Workflows for Apache Airflow (MWAA)	Apache Airflowのマネージドオーケストレーションサービス

本章ではシステムを構築するうえで利用頻度の高いAmazon API Gateway（以下、API Gateway）とAmazon EventBridge（以下、EventBridge）に焦点を当てて解説します。AWS Step Functionsは11.3節「AWS Step Functions」で解説していますので、そちらを参照ください。

8.2 Amazon API Gateway

API Gatewayは、簡単にAPIを作成・管理できるフルマネージドなサービスです。

API Gatewayの特徴

API Gatewayは2種類のAPIを用意しています。バックエンドへのプロキシ機能を持つステートレスなRestful APIと、チャットなどのリアルタイム双方向通信を実現するステートフルなWebSocket APIです。

RESTとはREpresentational State Transferの略で、情報をURL（*Uniform Resource Locator*）として定義し、GETやPUTなどのHTTPメソッドで扱います。WebSocketはリクエストを受け取って応答するREST APIと異なり、クライアントアプリケーションとバックエンド間の双方向性通信をサポートし、ひとつのコネクションで継続的にデータ送受信を行う特徴があります。

API GatewayのRestful APIは、HTTP APIとREST APIに分かれます。WAF（*Web Application Firewall*）の統合やクライアントごとのスロットリングなど多機能なREST APIに比べて、HTTP APIでは機能を絞り費用を抑えられます。機能差に関しては「参考」の「REST APIとHTTP API間で選択する」をご覧ください。

ここではバックエンドのシステムやデータをAPI経由で取得する場合を想定し、REST APIの設計について説明します。

参考：「REST APIとHTTP API間で選択する」 `https://docs.aws.amazon.com/ja_jp/apigateway/latest/developerguide/http-api-vs-rest.html`

▶ **失敗または成功**
処理を失敗または成功としてワークフローを停止する

▶ **パス**
入力を単純に出力に渡す、または一部の固定データを出力する

▶ **待機**
一定時間の待機、または指定の日時まで待機する

▶ **並行**
並行なステートを開始する

▶ **マップ**
動的な反復処理を行う

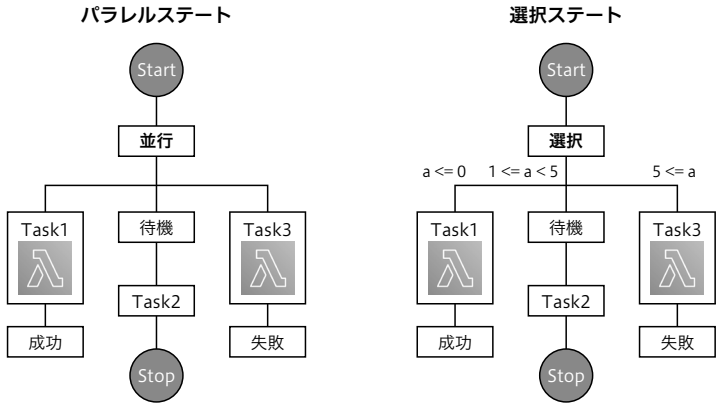
図11.3.1には2つのステートマシンがあります。左側の最初のステートは「並行」なので、Task1と待機、Task3がそれぞれ実行されます。右側の最初のステートは「選択」なのでTask1か待機、Task3のうちaの値によって単一の処理が実行されます。

Step Functionsの入出力

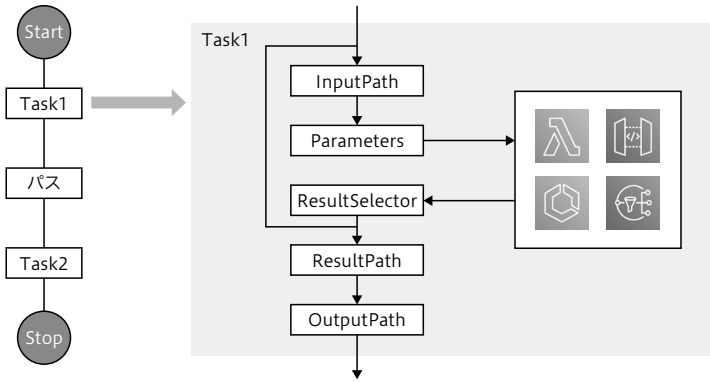
各ステートの入出力はJSONで行い、入出力に使用される5つのフィールドをタイプによって選択できます。たとえば失敗または成功のタイプでは入出力フィールドがありません。なお、どのフィールドも必須ではありません。

図11.3.2にあるタスクタイプの入出力で5つのフィールドを説明します。

▶ **図11.3.1** パラレルステートと選択ステートの例



▶ **図11.3.2** 各ステートの入出力



もともとのタスクに渡される入力を以下のJSONとします。メンバー情報としてAliceとBobのIDが記載されています。

```
{
  "members": {
    "Alice": {
      "id": "ABCDEF"
    },
    "Bob": {
      "id": "FGHIJ"
    }
  }
}
```

▶ **InputPath**
- 与えられた入力からキーを指定し、必要な値のみをそのタスク内の処理へ渡すときに使用
- たとえばAliceのid値のみを処理に渡す場合は、「InputPath」に「$.members.Alice.id」と指定する

▶ **Parameters**
- 入力の一部を使用して新たなJSONを作成し、タスク内の処理へ渡すときに使用
- 入力値からキーと値のペアを使用する場合、設定するキーの名前は「XXX.$」と最後に「.$」で終わるようにする
- たとえば以下のとおり、後続の処理へ実行日の情報(date)を追加して渡したい場合(かつ、"member"ではなく"info"で渡す)を考える

```
{
  "info": {
    "Alice": {
```

12.2 AWS Backup

AWS Backupは、最低料金や初期費用が発生しないフルマネージドなバックアップサービスです。AWSリソースのバックアップ自動化や保存期間が過ぎたデータの削除など、バックアップに関わるタスクの一元管理ができます。なおAWS BackupはAWS環境のみが対象のため、オンプレミス環境も含めて一元管理したい場合は旧来のバックアップまたはジョブ管理ツールを用いて実施します。

参考：「AWS Backupデベロッパーガイド」 https://docs.aws.amazon.com/ja_jp/aws-backup/latest/devguide/whatisbackup.html

「AWS Backupデベロッパーガイド - 開始方法」 https://docs.aws.amazon.com/ja_jp/aws-backup/latest/devguide/getting-started.html

AWS Backupの機能

AWS Backupには以下の機能があります。特徴的なのは法律・規制に対応したバックアッププランを作成できることでしょう。規制要件への準拠を実証する監査レポートも作成できます。

- **バックアップの一元管理**
  - バックアップ要件を満たすバックアップポリシーを一元管理
  - バックアップアクティビティログの一括表示
- **バックアップポリシーの適用**
  - 法律・規制に対応する要件に従ったバックアッププランの作成・適用
- **タグベースのバックアップ**
- **ライフサイクルポリシーの作成**
- **リージョン間のバックアップ**
- **AWS Organizationsとの統合**
- **AWS Backup Audit Managerでの監査とレポート作成**
- **増分バックアップ**
- **ダッシュボードによるモニタリング**
- **フルAWS Backup管理**
- **バックアップ先データの保護**
- **コンプライアンス義務のサポート**
  - FedRAMP High、GDPR、SOC1、2 and 3、PCI、HIPAAなど

AWS Backupの概要

AWS Backupはバックアッププランを作成し、バックアップルールで指定したスケジュール（頻度、開始時間など）でバックアップします（**図12.2.1**）。バックアップ対象のAWSリソースもバックアッププラン内で指定します。指定のしかたはタグやリソースID、特定のテーブルなどで、複数の「リソースの割り当て」をバックアッププランに含められます。バックアップが保存される領域は「バックアップボールト」と呼ばれ、バックアップされたAWSリソースの「復旧ポイント」をARNで確認できます。

なお、すぐにバックアップを取得開始できる「オンデマンドバックアップ」もあります。ただしバックアッププランと異なり、たとえばAmazon DynamoDB（以下、DynamoDB）のテーブルを1つだけといったリソース単位でのバックアップ取得のみ行えます。**複数のリソースに対する一括でのバックアップはできません**。オンデマンドバックアップは早急にバックアップを取得したい際に利用し、その後正式にバックアッププランを作成す

▶ **図12.2.1** AWS Backupの概要