

23 ある開発者は環境ごとに複数のCloudFormationテンプレートを分けて、ネスティッドスタックを構成しようとしています。開発者が実施しなければならない手順の中で誤っているものを選択してください。

- A. 親となるテンプレートを作成し、AWS::CloudFormation::Stackリソースとして、子テンプレートのスタックを定義します。
- B. 子テンプレートを作成し、親テンプレートにそのファイルのローカルパスを指定します。
- C. aws cloudformation packageコマンドを実行し、子テンプレートをS3にアップロードします。
- D. 作成済みの親テンプレートをコンソールからアップロードするか、AWS CLIで実行します。

24 ある開発者はCloudFormationを使ったテスト環境の基盤構築中に、マスターデータを設定するプロセスを組み込みたいと考えています。開発者が実施する手順の中で正しいものを選択してください。

- A. データベースにデータを設定するLambda関数を作成し、CloudFormationでAWS::Lambda::Functionとしてスタックを作成し、関数名を出力します。次にAWS::CloudFormation::Macroリソースを定義し、ServiceTokenにOutputs出力した関数名を指定してこのリソースを実行します。
- B. データベースにデータを設定するLambda関数を作成し、CloudFormationでAWS::Lambda::Functionとしてスタックを作成し、ARNを出力します。次にAWS::CloudFormation::Macroリソースを定義し、ServiceTokenにOutputs出力したARNを指定してこのリソースを実行します。
- C. データベースにデータを設定するLambda関数を作成し、CloudFormationでAWS::Lambda::Functionとしてスタックを作成し、ARNを出力します。次に任意のリソースタイプであるCustom:XXXXを定義し、ServiceTokenにOutputs出力したARNを指定してこのリソースを実行します。
- D. データベースにデータを設定するLambda関数を作成し、CloudFormationでAWS::Lambda::Functionとしてスタックを作成し、関数名を出力します。次に任意のリソースタイプであるCustom:XXXXを定義し、ServiceTokenにOutputs出力した関数名を指定してこのリソースを実行します。

25 ある企業ではアプリケーション環境をAWS CloudFormationを使って構築した後、次回のリリースに向けて環境をメンテナンスしようとしています。推奨される手法の中で正しいものを選択してください。

- A. ダイナミックリファレンス機能を使って環境パラメータを動的に切り替えて再構築すべきです。
- B. チェンジセットを使用して影響を受ける論理IDを特定し、更新対象ではない重要なリソースに影響が出ないことを確認すべきです。
- C. スタックセットを使用して新たに環境を再構築すべきです。
- D. ドリフト検出を使用して更新対象ではない重要なリソースに影響が出ていないことを確認すべきです。

26 ある企業ではサーバレスアプリケーション環境を自動構築するために、SAMを利用するか検討しています。CloudFormationを採用した場合に比べて、メリットとして挙げられるもののうち、誤っている記述を選択してください。

- A. SAMでは、CloudFormationの記述をすべてそのまま実行することができます。
- B. CloudFormationのコマンドでSAMのテンプレートを実行することができます。
- C. SAMでは、LambdaやAPI Gateway、DynamoDB、Step Functionsといったリソースの定義をより簡潔に記述することができます。
- D. SAMでは、作成したアプリケーションを自身のAWSアカウントのリポジトリに登録して、公開することができます。開発者はチームや組織の間で共通のサーバレスコンポーネントをデプロイ、公開、共有できます。

- 27 新たにプロジェクトに参画した新人の開発者が以下のSAMテンプレートを実装しており、あなたは内容をレビューしています。指摘として正しいコメントを2つ選択してください。

```

Resources:
  SampleUpdateDynamoDBFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: src/app.lambda_handler
      Runtime: python3.6
      CodeUri: .
      MemorySize: 512
      Timeout: 30
    Environment:
      Variables:
        TABLE_NAME: !Ref SampleTable
    Policies:
      - Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action:
              - 'dynamodb:PutItem'
              - 'dynamodb:UpdateItem'
        Resource: 'arn:aws:dynamodb:ap-northeast1:xxxxxxxxx:table/SampleTable'
Events:
  BucketEvent:
    Type: S3
    Properties:
      Bucket: !Ref SampleBucket
    Events:
      - 's3:ObjectCreated:*'

SampleReadDynamoDBFunction:
  Type: 'AWS::Serverless::Function'
  Properties:
    Handler: src/app.lambda_handler
    Runtime: python3.6
    CodeUri: .
    MemorySize: 512

```

```

Timeout: 30
Environment:
Variables:
  TABLE_NAME: !Ref SampleTable
Policies:
  - Version: '2012-10-17'
    Statement:
      - Effect: Allow
        Action:
          - 'dynamodb:GetItem'
          - 'dynamodb:Scan'
        Resource: 'arn:aws:dynamodb:ap-northeast1:xxxxxxxxx:table/SampleTable'
Events:
  GetSample:
    Type: Api
    Properties:
      Path: /sample/{sample_id}
      Method: get
SampleBucket:
  Type: 'AWS::S3::Bucket'
SampleTable:
  Type: AWS::Serverless::SimpleTable

```

- 4
- A. 必須要素である「Transform」要素が含まれていません。実行時にエラーとなります。
 - B. 定義されているLambda関数は、ランタイムや実行するハンドラ、使用するメモリサイズ、タイムアウトや環境変数が共通しているので、これらすべて「Globals」セクションで共通定義できます。
 - C. SAM固有の定義としてサポートされている対象は、API Gateway、Lambda、DynamoDBなどであり、S3は含まれていないので、S3に関するこの記述は有効ではありません。
 - D. 定義されているLambda関数では、「DynamoDBCrudPolicy」といったポリシーテンプレートが使われていません。実行時にエラーとなります。
 - E. 定義されているAPI GatewayはHTTP APIに関するものであり、CORSなどの設定も「Globals」セクションから行えます。

28 設問27のテンプレートをデプロイするために、実行する必要があるコマンドの組み合わせを2つ選択してください。

- A. aws cloudformation package と aws cloudformation deploy
- B. sam package と sam deploy
- C. aws cloudformation compile と aws cloudformation deploy
- D. sam build と sam deploy
- E. sam build と sam package と sam deploy

29 ある開発者は、Elastic Beanstalkを使ってDockerコンテナ化したアプリケーションをデプロイしようとしています。次の選択肢の中で、正しいものを選択してください。ただし、Docker Composeは使用しないものとします。

- A. アプリケーションを実装し、Dockerコンテナイメージとしてビルドします。
- B. 実装したアプリケーションのソースディレクトリルートに Dockerfile を保存してファイルをZIPアーカイブします。
- C. Dockerコンテナイメージ名や実行ポート、ターゲットボリュームなどを記載したYAML形式のファイルを作成します。
- D. .ebextension内に作成した設定ファイルを保存します。

解答

1 C

⇒ 問題：330 ページ、本文解説：282 ページ

この問題のユースケースでは、インターネットへの通信、すなわち AWS CodeCommitへのアクセスにHTTP/HTTPSプロキシサーバを経由することが制約として存在します。AWS CodeCommitへのアクセスをSSHを使って行うことはできません。したがって、HTTPS接続によるGit認証情報の設定が必要です。Git認証情報はIAMで作成したユーザに対し、認証情報(IDとパスワード)を作成することができ、HTTPS認証をサポートする任意のツールが利用できます。なお、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：SSH接続を行う場合の設定手順です。
- ・選択肢B：Aと同様、SSH接続を行う場合の設定手順に似ていますが、コンソールからIAMユーザに設定するのは公開鍵です。
- ・選択肢D：AWS CLIでCodeCommit APIを実行する場合の方法であり、HTTPSプロキシを経由してアクセスする設定ではありません。

2 C

⇒ 問題：330 ページ、本文解説：282 ページ

この問題のユースケースでは、GitHubでクローンされている最新のリポジトリをそのままCodeCommitにgit pushすればよいので、gitコマンドを使ったSSHプロトコルを使用したアクセス設定を行う方法が最も簡易的です。したがって、公開鍵/秘密鍵のペアを作成し、公開鍵をAWSコンソール上から開発者のIAMユーザに設定し、秘密鍵を端末に設定すればgitコマンドがCodeCommitに対して使用可能になります。なお、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：GitHubの認証トークンは、AWSサービスがGitHubリポジトリへアクセスする場合に必要となるトークンであり、このケースではローカル端末にすでに最新のリポジトリが存在するので必要ありません。
- ・選択肢B：CLIやSDKが開発者の端末からAWSへアクセスする際に必要な手順であり、この設定ではgitコマンドがCodeCommitにアクセスすることはできません。
- ・選択肢D：SSH接続を行う場合の設定手順に似ていますが、コンソールからIAMユーザに設定するのは公開鍵です。

3 C, E

➡ 問題：331 ページ、本文解説：282 ページ

この問題では、「別のAWSアカウントのCodeCommitにアクセスする際に、必ずしも必要のない設定」を選択します。別アカウントにアクセスする場合は、クロスアカウントアクセス用のIAMロールを作成した上で、端末にgit-remote-codecommitをインストールし、CLIでプロファイル名やロールのARN名といった接続情報を設定します。選択肢CはIAMロールを作成しているアカウントが逆のアクセスする側になっているため、選択肢Eは不要な設定のため誤りです。

4 A

➡ 問題：331 ページ、本文解説：283 ページ

特定のブランチの変更を取り込む場合は、そのブランチをgit fetchしてmergeすれば取り込めます。コンフリクトが発生した場合は必要に応じて修正します。他の選択肢で記載されているプルリクエストは、逆に現在のFeatureブランチをMasterブランチにマージするために発行するものであり誤りです。

5 D

➡ 問題：332 ページ、本文解説：294 ページ

CloudWatch Eventsはリポジトリをモニタリングできますが、対象となるイベントは、リポジトリへの変更やプルリクエスト発生に関するものであり、ソースコードの中身をモニタリングすることはできません(GitHubなどは脆弱性のあるソースコードを検知して通知する機能はありますが、CodeCommitにおいてはCodeGuru Reviewerで、アプリケーション開発中に重大な問題、セキュリティの脆弱性、見つけにくいバグを特定することができます)。

6 D

➡ 問題：332 ページ、本文解説：289 ページ

CodeBuildは継続的インテグレーションを実現するための中核となるサービスで、AWSクラウド環境下でサーバリソースの制約を気にせずにビルドやテスト処理を実行できるのが大きなメリットです。ビルド用のデフォルトのコンテナイメージとして、Amazon Linux 2およびUbuntuを選択でき、任意のカスタムコンテナイメージも選択できます。デフォルトのコンテナイメージには多数のプログラミングランタイムやGit、ブラウザのドライバなどがインストールされています。また、任意の時刻でプロジェクトをイベント実行できるCodeBuildトリガーもサポートされているので、CloudWatch Eventsと連携する必要はありません(CloudWatchのcron式と連携することで、より細かなトリガーの設定ができます)。

7 A

➡ 問題：333 ページ、本文解説：290 ページ

CodeBuildはビルド処理を実行するコンテナのスペックを選択することはできますが、メモリだけを選択的に大きくすることはできません。CPU・メモリセットでスペックを選択できます。なお、他の選択肢の記述はすべてビルドジョブの失敗の原因として考えられる事象です。実行エラーの内容に応じて適切なトラブルシューティングを行う必要があります。

8 A

➡ 問題：333 ページ、本文解説：292 ページ

CodeBuildではプロジェクトを定義する際に、buildspec.ymlのパスを指定することができるので、デフォルトのソースコードルートディレクトリに配置しなくても複数のbuildspec.ymlを用途ごとに切り替えて実行できます。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B：値を参照する際は最新のバージョンの値が選択されます。
- ・選択肢C：CodeBuildに設定するサービスロールにssm:GetParametersアクションを追加する必要があります。
- ・選択肢D：buildspec.ymlにはbatch要素の記述がサポートされており、build-graph、build-list、build-matrixの3タイプで同時実行できます。

9 C

➡ 問題：333 ページ、本文解説：291 ページ

finally要素に記述されたコマンドは各フェーズ中に実行されたコマンドがエラーになっても実行されます。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：on-failure属性をCONTINUEにすると、次のフェーズのコマンドを継続して実行されますが、同一フェーズのコマンドは中断されます。
- ・選択肢B：runtime-versions属性で、使用するプログラミング言語のランタイムやDockerなどのバージョンも指定できます。
- ・選択肢D：アーティファクトは複数指定することができます。

10 D

➡ 問題：334 ページ、本文解説：292 ページ

Privilegedモードが必要なのはキャッシュを利用する場合や、コンテナイメージをビルドする場合であり、CodeBuild Localの実行には必ずしも必要ではありません。その他の選択肢の記述はすべてCodeBuild Localの実行に必要な要素です。

11**B**

➡ 問題：334 ページ、本文解説：297 ページ

CodePipelineではパイプラインのステージでさまざまなプロバイダをサポートしていますが、利用可能なものとユースケースを把握しておく必要があります。Docker HubはソースステージでCodePipelineによってサポートされる対象ではありません。選択肢AではDocker Hubがコンテナリポジトリとして指定されており、ソースステージのプロバイダに含まれていませんが、デプロイステージのターゲットとしてECSを使用する場合、リポジトリの定義はインプットアーティファクトである`imagedefinition.json`に記述されているため、使用可能なオプションです。選択肢のCとDは、サポート対象のプロバイダです。

12**D**

➡ 問題：335 ページ、本文解説：297 ページ

CodePipelineではApprovalステージで手動承認オプションを選択できます。プロバイダとなるSNSは、デフォルトでレビュー用のURLや承認を促すコメントを追記して作成できます。

13**B、D**

➡ 問題：335 ページ、本文解説：298 ページ

CodePipelineのSourceステージでGitHubを選択した場合、OAuth 2.0のWeb IDフェデレーションを利用して、GitHubへアクセスします。GitHub側でリポジトリやブランチに対するWebフックの条件を設定します。なお、Web フックの代わりに、CodePipelineが定期的にポーリングするオプションも選択できます。誤っている選択肢の不正解理由は以下のとおりです。

- ・選択肢B：個人アクセストークンはパスワードの代わりに用いられる新しい手段です。CodePipelineからのアクセスはOAuth 2.0のWeb IDフェデレーションを利用します。
- ・選択肢D：CodePipelineのソースステージの設定でGitHubのWebフック条件を指定することはできません。

14**A**

➡ 問題：335 ページ、本文解説：298 ページ

CodePipelineでコスト最適化を図る使用方法の1つは、必要なときに環境を構築して、終了したら破棄することです。また、CodePipelineは1つのステージで複数のアクションを並列実行できるため、逐次実行はパイプラインの起動時間が余計に発生し、コスト増加要因となります。テスト後にプロダクション向けのコンテナをビルドするのも効率はよくありません。ステージング環境でテストしたコンテナをそのままプロダクション環境で利用するべきであり、コンテナは実行時に環境変数を切り替えることができるため、アクセス先のURLなど環境情報の差分は環境変数を用いて切り替えるのがよりよい方法です。CloudFormationによるステージング環境の構築はソースステージの前に差し込むことができ、この設問の適切な選択肢の1つとなります。

15**A**

➡ 問題：336 ページ、本文解説：298 ページ

CodePipelineをはじめ、CodeCommit、S3、ECRなど、パイプラインから実行されるプロバイダのAWSマネジドサービスを、インターネットに接続しない閉塞的なネットワーク環境で利用することができますが、VPCエンドポイントの設定が必要です。CodeBuildを利用するには、290ページでも解説したとおり、NAT Gatewayを設定する必要があり、このユースケースには適合しません。代わりにJenkinsなどのオープンソースのCI/CDツールを利用するほうがよいでしょう。ただし、オープンソースのGitリポジトリはCodePipelineのサポート外であり、コンテナイメージを保存するためのレジストリも動的にプライベートIPが変わってしまうVPC内に構築するの適切ではありません。VPCエンドポイントを設置して、CodeCommitやECRへ接続するのが適切な構成です。

16**A、D**

➡ 問題：337 ページ、本文解説：299 ページ

CodeDeployでは、実行中のアプリケーションをそのまま更新するIn-Placeデプロイメントと、新たにアプリケーション環境を構築し、リクエストの振り分けを変更するBlue/Greenデプロイメントが選択できます。なお、Canary/LinearはECSやLambdaでサポートされるデプロイ方式であり、Rolling-additional-batchはRollingデプロイメントを追加の新規アプリケーションを使ってリプレースしていく方式で、Elastic Beanstalkでサポートされているものです。

17 D

➡ 問題：337 ページ、本文解説：301 ページ

「デプロイ設定」によりデプロイ戦略を詳細に定義できます。設問のユースケースでは、CanaryタイプのBlue/Greenデプロイメントを指していますが、デフォルトで用意されているデプロイ設定には含まれないので、カスタムで作成する必要があります。

18 C

➡ 問題：338 ページ、本文解説：303 ページ

フック設定で、ApplicationStopはEC2/オンプレミスでサポートされている要素です。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：「os」はEC2/オンプレミスで必須となる要素です。設問の appspec.yml はECS向けのものですので、この要素は不要です。
- ・選択肢B：AppSpecファイルのバージョンは、2023年12月現在、0.0のみを指定する必要があります。
- ・選択肢D：ECSのフック設定では、ライフサイクルイベントで実行する処理をLambda関数で設定しますので、関数名が記載されているのは正しいです。

19 B

➡ 問題：339 ページ、本文解説：305 ページ

CodeDeployエージェントはポート443番を使用したアウトバウンド通信が必要です。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：CodeDeployエージェントはLinuxサーバやWindowsサーバもサポートしています。
- ・選択肢C：CodeDeployエージェントの実行にJava仮想マシンは必要ありません。
- ・選択肢D：オンプレミスのサーバをAWSへ登録するためのCLIインストールや、registerコマンドの実行、割り当てたIAM IDに権限を付与する必要はありますが、エージェントがインストールされているサーバにユーザを作成する必要はありません。

20 B

➡ 問題：339 ページ、本文解説：305 ページ

CodeDeployでは、インターネットを経由しない、VPCから直接接続するためのVPCエンドポイントを提供しています。ただし、EC2へデプロイする場合は、codedeployおよび、codedeploy-commands-secureの2種類のエンドポイントを、LambdaやECSへのデプロイにはcodedeployを設定しておく必要があります。

21 B、E

➡ 問題：340 ページ、本文解説：310-311 ページ

CloudFormationテンプレートでは、各リソースの定義に一意となる論理名を付与しておく必要があります。またクロススタックリファレンスは、あるテンプレートで作成したスタックの情報を別のテンプレートから参照する方法であり、Outputsで必要な情報を出力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：CloudFormationテンプレートにおける必須要素は「Resources」のみです。
- ・選択肢C：組み込み関数内で別の関数を使用することは許容されています。使用可能な対象の詳細は脚注のURL^{*19}を参照してください。
- ・選択肢D：組み込み関数内で擬似変数を使用することは許容されています。使用可能な対象の詳細は脚注のURL^{*20}を参照してください。

*19 https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference-conditions.html#w2aac33c28c21c45

*20 https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/pseudo-parameter-reference.html

22 D

➡ 問題：341 ページ、本文解説：310-315 ページ

CloudFormationを使用してアプリケーション基盤を構築する目的の1つに、環境に応じてテンプレートを再利用して作業コストを軽減することが挙げられます。CloudFormationはConditionsを構成要素として定義することができます、例えば、実行時に指定されたパラメータに応じて条件を設定し、その条件に応じて、設定値や実行対象をコントロールできます。大きくは以下2つの方法で実現します。

- ・Resourcesに定義したリソースの各プロパティでIFなどの条件関数を使って環境ごとに設定するパラメータを切り替える
- ・Resourcesに定義したリソースのConditionプロパティで、Conditionsに定義したパラメータが一致する場合、リソース定義自体の有効可否を一括で切り替える

なお、テンプレートを再利用するのに有効なリソースとしてAWS::CloudFormation::Stackがあります。こちらのリソースには任意のパラメータを渡すことができるため、環境依存値を定義した(親)テンプレートから、環境に依存しない(子)テンプレートに値を渡す構成にすることで再利用性を高めることができます。本設問の解答であるDにおいて、スタックセットは複数のリージョンに同じ環境を展開できますが、同一アカウントで重複が許されないパラメータをLambda関数で切り替える機能はありません。もし実現するのであれば、マクロ機能などを用いて切り替えるように実装する必要があります。

23 D

➡ 問題：342 ページ、本文解説：310 ページ

ネスティッドスタックのテンプレートを実行する手順です。Aから順に実行していく必要がありますが、最後に実行するのは、aws cloudformation packageコマンドで出力される、子テンプレートのファイルパスがS3のものに置き換わった親テンプレートです。

24 C

➡ 問題：342 ページ、本文解説：313 ページ

CloudFormationでサポートされていないリソースや、任意の処理を実行するためカスタムリソースを作成して実行することができます。カスタムリソースの実行対象にはLambda関数を実行するLambda-backedカスタムリソースと、SNS-backedカスタムリソースが選択できます。ここでは、データベースにデータを設定する Lambda関数を作成したのち、AWS::Lambda::Functionリソースタイプでスタックを作成します。また、このLambda関数を実行するカスタムリソースタイプとして、LambdaのARNをServiceTokenプロパティに指定したCustom::XXXX(XXXXは任意の文字列)リソースを含むテンプレートを作成して実行すると、作成したLambda関数が実行されます。

25 B

➡ 問題：343 ページ、本文解説：313 ページ

CloudFormationを使って構築した環境で、新たに更新をかけようとした際に、チェンジセット機能を使って、実行しようとしている新たな更新テンプレートに対し、既存のテンプレートで影響を受けるリソースを確認することができます。意図しない更新が発生しないように、こうした機能を活用してメンテナンスを実行します。なお、他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：ダイナミックリファレンス機能では、Secrets Managerなどと連携して秘匿データなどを参照することが主なユースケースであり、環境差分を設定するパラメータ用途には向きません。環境差分用途でパラメータを更新して、スタックを再実行するには一度、既存のスタックを削除する必要があり、既存のアプリケーション環境との連続性を保つことができません。
- ・選択肢C：スタックセットは、同じアカウントで別のリージョンか、異なるアカウントに同一の環境を展開する機能です。同じく、既存のアプリケーション環境とはアカウントやリージョンなどまったく条件の異なるものができあがってしまうため、適切ではありません。
- ・選択肢D：ドリフト検出は、既存のアプリケーションの設定と構築した際のCloudFormationスタックの設定差分を検出する機能です。ドリフト検出は、現在の実態とテンプレートのずれの検証など、別の用途で環境差分を検出するために使用されます。

26 D

➡ 問題：343 ページ、本文解説：319 ページ

SAMはCloudFormationのマクロ機能を使って実現されているので、SAMテンプレートは、Transform要素が記述されているCloudFormationテンプレートであり、CloudFormationの記述やコマンドがそのまま実行できます。LambdaやAPI Gateway、DynamoDB、Step Functionsといったサービスのリソース定義をより簡潔に定義できます。アプリケーションをServerless Application Repositoryに公開して、組織内の開発者の間や、サードパーティベンダーが提供するサーバレスアプリケーションを利用できますが、リポジトリはAWSアカウント固有のものではなく、AWSがマネージドで管理しているサービスになります。

27 A、B

➡ 問題：344 ページ、本文解説：317 ページ

SAMテンプレートはCloudFormationのマクロ機能を使って実現されているので、Transform要素が必須になります。Globalsセクションで ランタイムや実行するハンドラ、メモリサイズ、タイムアウトや環境変数設定などを共通して定義でき、必要に応じて、個々のリソースごとに設定を上書きして利用できます。その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C：S3は確かにSAM固有の定義としてサポートされてはいませんが、CloudFormationのS3に関するリソース記述がそのまま利用できます（必須属性はありません）。
- ・選択肢D：確かにポリシーテンプレートが使われておらず、より長い記述になっていますが、エラーになることはありません。
- ・選択肢E：このAPI Gatewayの定義はREST APIに関するものであり、HTTP APIに関する設定ではありません。

28 A、B

➡ 問題：346 ページ、本文解説：321 ページ

テンプレートをパッケージ化してS3にアップロードし、デプロイする必要があります。これを実行するためには、以下のコマンドを実行する必要があります。

- ・選択肢A：aws cloudformation package と aws cloudformation deploy
- ・選択肢B：sam package と sam deploy

29 A

➡ 問題：346 ページ、本文解説：326 ページ

Elastic Beanstalkで、Docker ComposeなしでDockerコンテナを実行する場合、以下の3つのオプションがあります。

- ・Dockerfileを作成して、コンソール上からアップロードする
- ・コンテナイメージをビルドして、リポジトリへプッシュする。Dockerrun.aws.jsonという名称のJSON形式のファイルを作成し、コンソール上へアップロードする。ファイルにはイメージ名のほか実行ポート、ターゲットボリュームなどといったコンテナ実行に必要な情報を指定する
- ・DockerfileおよびDockerrun.aws.jsonを含むソースコードルートディレクトリに含むZipアプリケーションアーカイブファイルを作成し、コンソール上からアップロードする

選択肢は、上記の手順の一部を記載したものであり、A以外の選択肢は以下が誤っています。

- ・選択肢B：Zipアーカイブ化した際に含める必要があるのは、DockerfileとDockerrun.aws.jsonの2つです。
- ・選択肢C：作成する設定ファイルはJSON形式であり、YAML形式ではありません。
- ・選択肢D：Dockerrun.aws.json設定ファイルを保存するのはソースコードルートディレクトリであり、.ebextension内ではありません。

第5章

トラブルシューティングと 最適化に関するサービス

- 5-1. Amazon CloudWatch
- 5-2. AWS CloudTrail
- 5-3. AWS X-Ray
- 5-4. Amazon ElastiCache
- 5-5. Amazon SQS
- 5-6. Amazon SNS
- 5-7. Amazon CloudFront
- 5-8. Auto Scaling

5-1 Amazon CloudWatch

Amazon CloudWatchはAWSクラウド監視の基本となるサービスです。システム運用に必要な状態の可視化やオペレーションをサポートするさまざまな機能があります。CloudWatchで扱われる概念や、どのようなユースケースでどういった機能を利用できるかを押さえておきましょう。

1 CloudWatchの概要

CloudWatchは、AWSリソースの状態や、AWS上に構築されたアプリケーションの監視に使用されるサービスです。EC2やECS、Lambdaなどのコンピューティングリソースのメトリクス、アプリケーションログなどを記録・監視し、状況を可視化したり、必要に応じてイベントを起動したりすることができます。

CloudWatchの主な機能は、2023年12月時点で次のとおりです。

【CloudWatchの主な機能と概要】

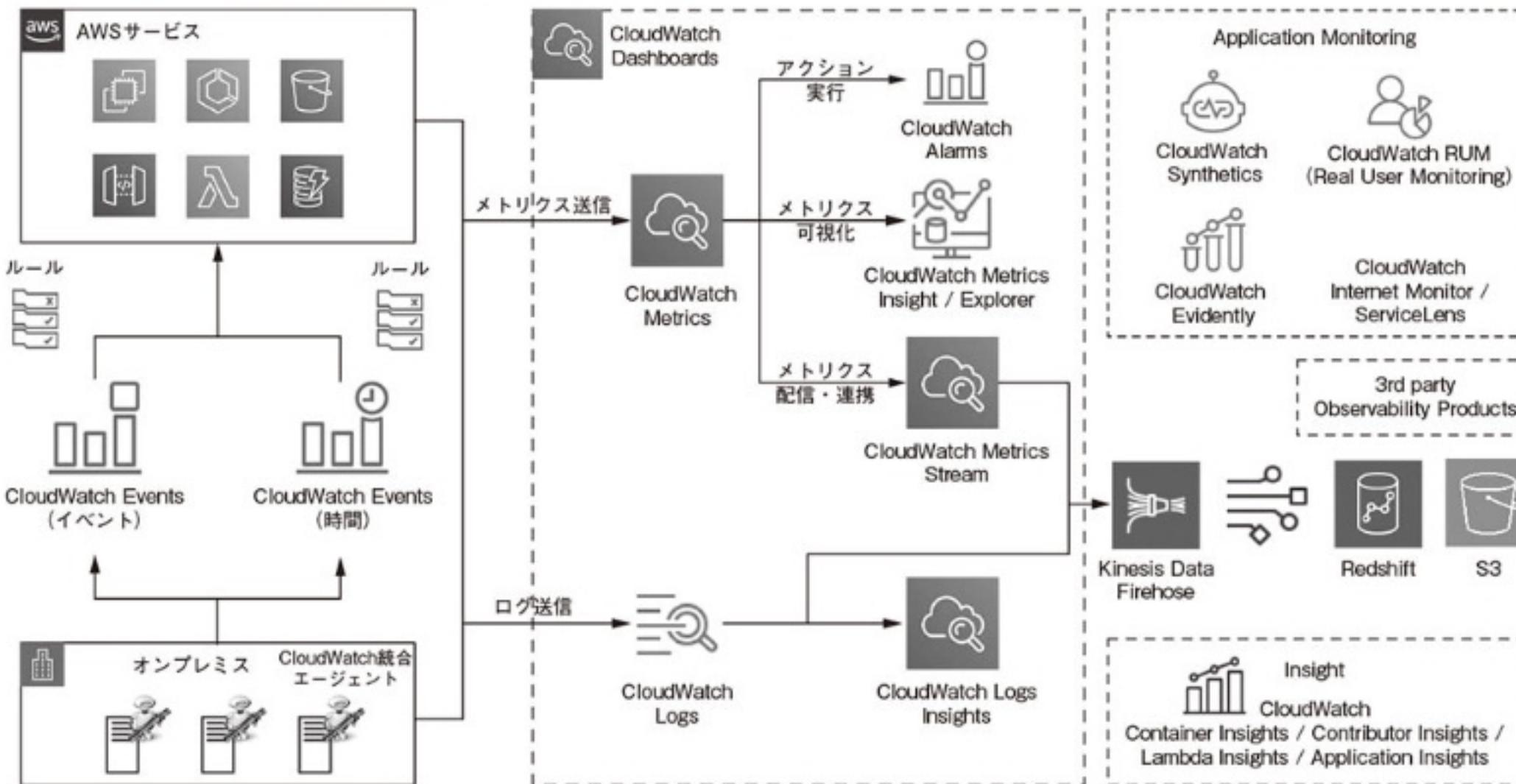
カテゴリ	機能	概要
Infrastructure	CloudWatch Metrics	EC2やECS、Lambdaなどのコンピューティングリソースのパフォーマンスマトリクス（CPU、メモリ）を収集し表示する機能
	CloudWatch Logs	サーバの標準出力に出力されたログを集約して表示する機能
	CloudWatch Alarms	メトリクスの状況にもとづいてアラーム通知やアクションを実行する機能
	CloudWatch Dashboards	さまざまなメトリクスの時系列データをアグリゲーション（集約）して表示するダッシュボード
	CloudWatch Metrics Explorer ^{*1}	タグとリソース、プロパティなどを使ってメトリクスをフィルタリングし、視覚化する機能
	CloudWatch Metrics Stream ^{*2}	CloudWatch Metricsをストリーミング配信し、ニアリアルタイムで連携する機能
	CloudWatch Events (EventBridge)	AWSリソースの変更や時刻起動でイベントを送信し、それを契機として何らかのアクションを実行する機能
	CloudWatch Resource Health ^{*3}	EC2インスタンスの正常・異常、パフォーマンス状況を検出して視覚化する機能

カテゴリ	機能	概要
Insights	CloudWatch Metrics	EC2やECS、Lambdaなどのコンピューティングリソースのパフォーマンスマトリクス（CPU、メモリ）を収集し表示する機能
	CloudWatch Contributor Insights ^{*4}	CloudWatch Logsのログデータを解析し、貢献度合いが高い上位のデータを可視化する機能
	CloudWatch Container Insights ^{*5}	タスクやPod、コンテナレベルでのメトリクスやログを取得し、可視化する機能
	CloudWatch Lambda Insights ^{*6}	Lambda関数のメトリクスを収集・可視化する機能
	CloudWatch Application Insights ^{*7}	リソースグループという単位で包括的にアプリケーション全体の代表的なメトリクス・ログを収集し、可視化する機能
	CloudWatch Logs Insights	CloudWatch Logsで収集されたログを分析・可視化する機能
	CloudWatch Metrics Insights ^{*8}	メトリクスをリアルタイムで集計・可視化し効率よく分析する機能
Monitoring	CloudWatch Synthetics ^{*9}	計測プログラムを用いて、アプリケーションのエンドポイントを外形監視する機能
	CloudWatch Rum (Real-user monitoring) ^{*10}	Webアプリケーションのユーザアクセスに関するデータをニアリアルタイムで収集し、パフォーマンス情報などを可視化する機能
	CloudWatch Evidently ^{*11}	A/Bテストやフィーチャーフラグなどの手法を実施して、アプリケーションの安全性を検証することをサポートする機能
	CloudWatch Internet Monitor ^{*12}	AWS上で実行されるアプリケーションに対し、インターネットアクセス状況の評価やパフォーマンスや可用性を可視化する機能
	CloudWatch ServiceLens ^{*13}	メトリクスおよびログ、X-Rayのトレーシングデータを一元的に結びつけて可視化する機能

- *1 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudWatch_0330_v1.pdf
- *2 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudWatch_0330_v1.pdf
- *3 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudWatch_0330_v1.pdf
- *4 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudWatch_0330_v1.pdf
- *5 <https://aws.amazon.com/jp/blogs/news/aws-black-belt-online-seminar-con247/>
- *6 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudWatch_0330_v1.pdf
- *7 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudWatch_0330_v1.pdf
- *8 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudWatch_0330_v1.pdf
- *9 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudWatch-Synthetics_0331_v1.pdf
- *10 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudWatch-RUM_0430_v1.pdf
- *11 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudWatch_0330_v1.pdf
- *12 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudWatch_0330_v1.pdf
- *13 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudWatch_0330_v1.pdf

CloudWatchではさまざまな機能があり、AWSサービスと相互に連携しています。機能の全体像と各サービスの関連を図に示すと次のようにになります。

【CloudWatchの全体像と機能間の関連】



本書では、試験で問われる可能性が高い、特に基本となるCloudWatch Metrics、Alarms、Logs、Logs Insights、Dashboards、Eventsについて解説を行います。それ以外の機能については、上記の概要に付与している、AWSが公開しているBlack Beltの資料等、適宜参照してください。

2 CloudWatch Metrics

CloudWatch Metricsは、AWSリソースの状態に関するさまざまなデータを収集するサービスです。

● CloudWatch Metrics の基本概念

まずは基本となる「メトリクス」、「名前空間」、「ディメンション」といった概念を押さえておきましょう。

【CloudWatch Metricsの基本的な概念】

概念	説明
メトリクス	システムのパフォーマンスを示す時系列のデータ指標を指す。EC2ではCPU使用率、API Gatewayではリクエスト数といったかたちで、AWSリソースごとに異なるメトリクスが「標準メトリクス」として定められている。AWS CLI 「put-metric-data」 や 「PutMetricData API」 を使用して任意のメトリクスデータをカスタムで送信することもできる。取得間隔はAWSリソースによって異なるが、多くの標準メトリクスは通常1分もしくは5分間隔でデータが収集される。カスタムで1分未満の高解像度収集オプションも選択できる。
名前空間	収集したメトリクスを格納するためのコンテナ。AWSサービスの名前空間は「AWS/<service>」といったフォーマットで命名される。
ディメンション	名前空間別に作成されるリソースを一意に識別するためのキー文字列ペア。

●メトリクスの収集例

特に「名前空間」や「ディメンション」がイメージしづらいと思いますので、CloudWatchで収集されるメトリクスの例をいくつか見てみましょう。例えば、EC2のメトリクスを監視する場合、「AWS/EC2」名前空間には次のメトリクスが含まれます。

【「AWS/EC2」名前空間に含まれるメトリクスの一例】

メトリクス	説明
CPUUtilization	現在のインスタンスでのCPU使用率
DiskReadOps	指定された期間にインスタンストアボリューム（EC2で永続化されない一時ストレージ）で完了した読み取り操作の回数
DiskReadBytes	指定された期間にインスタンストアボリュームで完了した読み取りバイト数
DiskWriteOps	指定された期間にインスタンストアボリュームで完了した書き込み操作の回数
DiskWriteBytes	指定された期間にインスタンストアボリュームで完了した書き込みバイト数
NetworkIn	指定された期間に、インスタンスのすべてのネットワークインターフェースで受信したバイト数
NetworkOut	指定された期間に、インスタンスすべてのネットワークインターフェースで送信したバイト数
EBSReadOps	指定された期間に、インスタンスに接続されたすべてのEBSボリュームからの読み取りバイト数
EBSWriteOps	指定された期間に、インスタンスに接続されたすべてのEBSボリュームからの書き込みバイト数

また、以下のディメンションを選択することで、監視したいメトリクスのターゲットを絞り込めます。

【EC2のディメンションの例】

ディメンション	説明
AutoScalingGroupName	オートスケーリンググループ名を識別子として、対象のメトリクスを観測します。
ImageId	AMIのイメージIDを識別子として、対象のメトリクスを観測します。
InstanceId	EC2インスタンスIDを識別子として、対象のメトリクスを観測します。

EC2の場合、「メモリ使用率」や「ディスク使用率」といった、要件によっては重要な指標が標準メトリクスに含まれていません。標準に含まれていないメトリクスを収集するには、CloudWatchエージェントを使用するか^{※14}、EC2上でデータ収集用のスクリプトを実行し、AWS CLIでカスタムメトリクスとして送信します。CloudWatchエージェントはオンプレミスのサーバにもインストールできるので、AWSとオンプレミスのサーバのメトリクスをCloudWatchでまとめて監視することも可能です。

カスタムメトリクスは、次のようにAWS CLIの「put-metric-data」コマンドで、名前空間やディメンションを指定して送信できます。

```
aws cloudwatch put-metric-data --metric-name Buffers --namespace MyNameSpace --unit Bytes --value 231434333 --dimensions InstanceId=1-23456789,InstanceType=m1.small
```



CloudWatchエージェントを使ったオンプレミスのサーバのメトリクス収集やカスタムメトリクスの送信の方法を問われる場合があります。ユースケースに応じた適切な手法を押さえておくようにしましょう。

【「AWS/ApiGateway」名前空間に含まれるメトリクスの例】

メトリクス	説明
4XXError	指定された期間に取得された、クライアント側に問題があるリクエストエラーの数。
5XXError	指定された期間に取得されたサーバ側に問題があるリクエストエラーの数。
CacheHitCount	指定された期間にAPIキャッシュから配信されたリクエストの数。
CacheMissCount	APIキャッシュが有効になっている状況で、指定された期間にバックエンドから配信されたリクエストの数。
IntegrationLatency	API Gatewayがバックエンドにリクエストを中継してから、バックエンドからレスポンスを受け取るまでの時間。
Latency	API Gatewayがクライアントからリクエストを受け取ってから、クライアントにレスポンスを返すまでの時間。
Count	APIリクエストの合計数。

また、ディメンションを選択することで、監視したいメトリクスのターゲットを絞り込めます。

【API Gatewayのディメンションの例】

ディメンション	説明
ApiName	指定したAPI名を指定してメトリクスを観測します。
ApiName,Stage	指定したAPI名とステージを識別子として、対象のメトリクスを観測します。

5



使用頻度が高いサービスの中で、監視対象メトリクスの適切な選択肢を問われる場合があります。EC2やAPI Gatewayなどいくつかのサービスで代表的なメトリクスを押さえておくとよいでしょう^{※15}。

API Gatewayのメトリクスである「AWS/ApiGateway」名前空間を見てみましょう。この名前空間には、リソース自体のパフォーマンスとは別に、次のようなメトリクスが含まれます。

3 CloudWatch Alarms

CloudWatch Alarmsは、CloudWatch Metricsで収集したメトリクスを監視対象にしてアラームを発行するサービスです。同時に実行するアクションを定義することもできます。

※14 https://docs.aws.amazon.com/ja_jp/AmazonCloudWatch/latest/monitoring/metrics-collected-by-CloudWatch-agent.html

※15 https://docs.aws.amazon.com/ja_jp/AmazonCloudWatch/latest/monitoring/aws-services-cloudwatch-metrics.html

●CloudWatch Alarms の状態

CloudWatch Alarmsで発行されるアラームには以下の状態が定義されており、メトリクスの状態の変化が実行アクションの契機となります。

【CloudWatch Alarmsにおけるアラームの状態】

アラームの状態	説明
OK	定義されたしきい値を下回っており、正常である。
ALARM	定義されたしきい値を上回っており、異常である。
INSUFFICIENT_DATA	アラームが開始直後であるか、メトリクスが利用できないか、データが不足していてアラームの状態を判定できない。

アラームを作成する際は、状態の変化を評価・判定する要素として以下を指定します。

【CloudWatch Alarmsにおけるアラームの評価指標】

アラームの指標	説明
期間	データポイント（メトリクスを構成する個々の値・標本のこと）を評価するタイミング。例えば、期間として1分を選択した場合、アラームはメトリクスを1分あたり1回評価する。
評価期間	状態を決定するための期間または基準とするデータポイントの数。例えば、評価期間として1分を選択した場合、1分間あたりのメトリクスを評価する。データポイントを5と設定した場合、データポイントと期間を乗じた5分間で取得した5つのメトリクスを母数として評価する。
アラームを実行するデータポイント (DataPoints to Alarm)	評価期間中に発生した、状態「ALARM」に遷移するためのしきい値となるデータポイントの値。
しきい値	アラームを実行する境界となるメトリクスの値。

●M out of N

上記の「評価期間」でベースとなるデータポイント数（M）と「アラームを実行するデータポイント」に異なる値（N）を設定すると、「N/M」でアラームを発行するよう設定できます。例えば、「期間」を1分間、「評価期間」を5分間、「データポイントのしきい値」を4と設定した場合で考えてみましょう。このとき、5分間で取得した5つのデータポイントのうち、4つ以上がしきい値を超えると、状態は「ALARM」に移行し、アラームが通知されます。逆に状態が「ALARM」のとき、5分間に取得した5つのデータポイントのうち、しきい値を超えるものが4つ未満になると、状態は「OK」に遷移します。

【CloudWatch Alarmsの設定例】

The screenshot shows the CloudWatch Alarms configuration interface. The top section is titled 'Metrics and Conditions'.

Metric: A line graph showing CPUUtilization over time. The Y-axis ranges from 74 to 76, and the X-axis shows 00:00, 01:00, and 02:00. A red line represents the metric value, which stays below the threshold line for most of the period.

Conditions (展开された状態):

- しきい値の種類**: 静的 (選択済み) - バンドをしきい値として使用
- CPUUtilization が次の時...**: アラーム条件を定義します。
 - より大きい >= しきい値 (選択済み)
 - 以上 >= しきい値
 - 以下 <= しきい値
 - より低い <= しきい値
- ...よりも しきい値を定義します。**: 75
- その他の設定**: アラームを実行するデータポイント: 4 / 5
- 欠落データの処理**: アラームを評価する際に欠落データを処理する方法: 欠落データを見つかりませんとして処理

4 CloudWatch Logs

CloudWatch Logsは、AWSサービスやEC2などのAWSリソースで実行される標準出力を収集し、アプリケーションのログを監視・保存・イベント実行するサービスです。統合CloudWatchエージェントを任意のサーバにインストールすれば、CloudWatch Logsにログを送信することもできます。CloudWatch LogsはAWS PrivateLinkに対応しているため、オンプレミスのサーバログを対象に各種処理を実行することも可能です。

●CloudWatch Logs の形式

CloudWatch Logsで収集されたログは次の形式で保存され、コンソール上から参照できます。収集されたログはS3にエクスポートすることもできます。

【CloudWatch Logsのログの構造】

ログの構造	説明
ロググループ	保持、監視、アクセス制御について同じ設定を共有するログストリームのグループ 例：/aws/lambda/<関数名>
ログストリーム	監視対象のリソースから送信されたタイムスタンプ順のイベント
ログイベント	監視対象のリソースのアクティビティが記録されたログ

●サブスクリプションフィルタ

サブスクリプションフィルタは、指定したフィルタパターンに合致する文字列がログに含まれていた場合に、AWS LambdaやKinesis Data FirehoseなどのAWSサービスをイベント駆動（サブスクリプション）する機能です。ログメッセージを加工・蓄積したり、外部のコミュニケーションツールに通知したい場合に利用できます。

●メトリクスフィルタ

メトリクスフィルタも同様に、指定したフィルタパターンに合致する文字列がログに含まれていた場合に、CloudWatch Metricsにメトリクスデータとして送信する機能です。メトリクスの名前空間やメトリクス名、送信する値を規定しておくと、対象のログの発生状況をCloudWatch Metricsで可視化できます。発生した件数に応じてCloudWatch Alarmsの機能を利用できるので、より細かい発生条件でエラー通知などを行いたい場合に利用するとよいでしょう。

●Embedded Metrics Format (EMF)

EMFはCloudWatch Logsのログの中にカスタムメトリクスを含めたい場合に利用できるフォーマットです。下記のようなフォーマットでログの中に情報出力しておくと、CloudWatchが自動的にカスタムメトリクスとして集計してくれるようになります。

```
{
  "_aws": {
    "Timestamp": 1574109732004,
    "CloudWatchMetrics": [
      {
        "Namespace": "lambda-function-metrics",
        "Dimensions": [["functionVersion"]],
        "Metrics": [
          ...
        ]
      }
    ]
  }
}
```

```
{
  "Name": "time",
  "Unit": "Milliseconds",
  "StorageResolution": 60
}
]
]
},
"functionVersion": "$LATEST",
"time": 100,
"requestId": "989ffbf8-9ace-4817-a57c-e4dd734019ee"
}
```

5 CloudWatch Logs Insights

CloudWatch Logs Insightsは、専用のクエリ言語を使ってCloudWatch Logs内のログを検索・可視化するサービスです。ただし検索の対象となるのは、2018年11月5日以降にCloudWatch Logsに送信されたログのみとなります。

●CloudWatch Logs Insights のフィールド

CloudWatch Logs Insightsでは、送信されるログごとに下記のフィールドが自動生成されます。

【CloudWatch Logs Insightsで自動生成されるフィールド】

フィールド	説明
@message	生の未解析のログイベント
@timestamp	ログイベントがCloudWatch Logsに追加された時間
@ingestionTime	CloudWatch Logsがログイベントを受信した時間
@logStream	ログイベントの追加されたログストリームの名前
@log	account-id:log-group-name の形式のロググループ識別子

さらに、CloudWatch Logsで収集された一部のサービス（VPCフローログやRoute 53、Lambda）のログは、下記のようなフィールドを自動で検出します。

【VPCフローログ、Route 53ログ、Lambdaログで自動検出されるフィールド】

ログ	自動検出されるフィールド
VPCフローログ	@timestamp、@logStream、@message、accountId、endTime、interfaceId、logStatus、startTime、version、action、bytes、dstAddr、dstPort、packets、protocol、srcAddr、srcPort
Route 53ログ	@timestamp、@logStream、@message、edgeLocation、hostZoneId、protocol、queryName、queryTimestamp、queryType、resolverIp、responseCode、version
Lambdaログ	@timestamp、@logStream、@message、@requestId、@duration、@billedDuration、@type、@maxMemoryUsed、@memorySize ※ X-RayトレースIDが含まれている場合は、@xrayTraceIdおよび@xraySegmentId フィールドも自動検出される。
CloudTrail ログ、JSON形式のログ	ネストされたJSONフィールドをドット表記で表示する。 例：JSONオブジェクトuserIdentityのフィールドtypeは、userIdentity.typeと表示される。
その他のログタイプ	@timestamp、@ingestionTime、@logStream、@message、@log

例えば、AWSコンソール上から出力を有効化したVPCフローログのグループを選択し、バイト数を示す「byte」でソートして出力すると、VPCの中でトラフィックデータサイズが大きい順に通信データを出力できます。

【CloudWatch Logs Insightsで通信量が大きいトラフィックデータをVPCフローログの中から検索する例】



ログタイプの中には、CloudWatch Logs Insightsが自動的に検出しないフィールドを持つものもあります。これらについては、後述のparseコマンドを使用して、エフェメラル（一時的な）フィールドを抽出・作成することでクエリで検索・可視化できるようになります。

● クエリ言語の構文

CloudWatch Logs Insightsでは、ロググループに対して独自のクエリ言語を使って検索を行います。下記に示すクエリコマンドをUNIX形式のパイプ文字（|）でつなぎでクエリを実行できます。

【CloudWatch Logs Insightsでサポートされるクエリコマンド】

コマンド	説明
display	クエリ結果に表示するフィールドを指定する。
field	指定したフィールドをログイベントから取得して表示する。
filter	クエリの結果を1つ以上の条件にもとづいてフィルタリングする。
stats	ログフィールドの値にもとづいて集約統計を計算する。
sort	取得したログイベントをソートする。
limit	クエリから返されるログイベントの数を指定する。
parse	ログフィールドからデータを抽出し、1つ以上のエフェメラルフィールドを作成してクエリでさらに処理できるようにする。

クエリ言語では、下記に示すようなオペレーションと関数が利用できます。ここでは代表的なものを示しますので、詳細は開発者ガイドを確認してください^{*16}。

【クエリ言語で利用できるオペレーションと関数（代表的なもののみ記載）】

分類	オペレーションもしくは関数
比較オペレーション	= != < <= > >=
ブール演算子	and、or、not
算術オペレーション	加算：+、減算：-、乗算：*、除算：/、指數：^、剰余：%
数値オペレーション	絶対値：abs(a: number)、上限：ceil(a: number)、下限：floor(a: number)、最大：greatest(a: number, …numbers: number[])、最小：least(a: number, …numbers: number[])、自然対数：log(a: number)、平方根：sqrt(a: number)
一般関数	フィールドの確認：ispresent(fieldName: LogField)、nullでない値の返却：coalesce(fieldName: LogField, …fieldNames: LogField[])
文字列関数	文字列長の返却：strlen(str: string)、文字列の連結：concat(str: string, …strings: string[])など

CloudWatch Logs Insightsは、独自の構文による検索であるため慣れが必要ですが、サンプルクエリが提供されています。これを参考にしつつ、要件に合わせて改変しながら利用するとよいでしょう。

6 CloudWatch Dashboards

CloudWatch Dashboardsは、CloudWatchのコンソール画面にダッシュボードを作成して、各機能での監視状況を集約するサービスです。同一アカウント・同一リージョンでの利用に留まらず、クロスアカウント・クロスリージョンの監視状況を集約できます。

昨今では、マルチアカウントやマルチリージョンにシステムを展開することが一般的となっています。情報集約アカウントにダッシュボードを構成して各アカウントの情報を集約すれば、利用者はそのダッシュボードを確認するだけでシステム全体の稼働状況を確認できます。

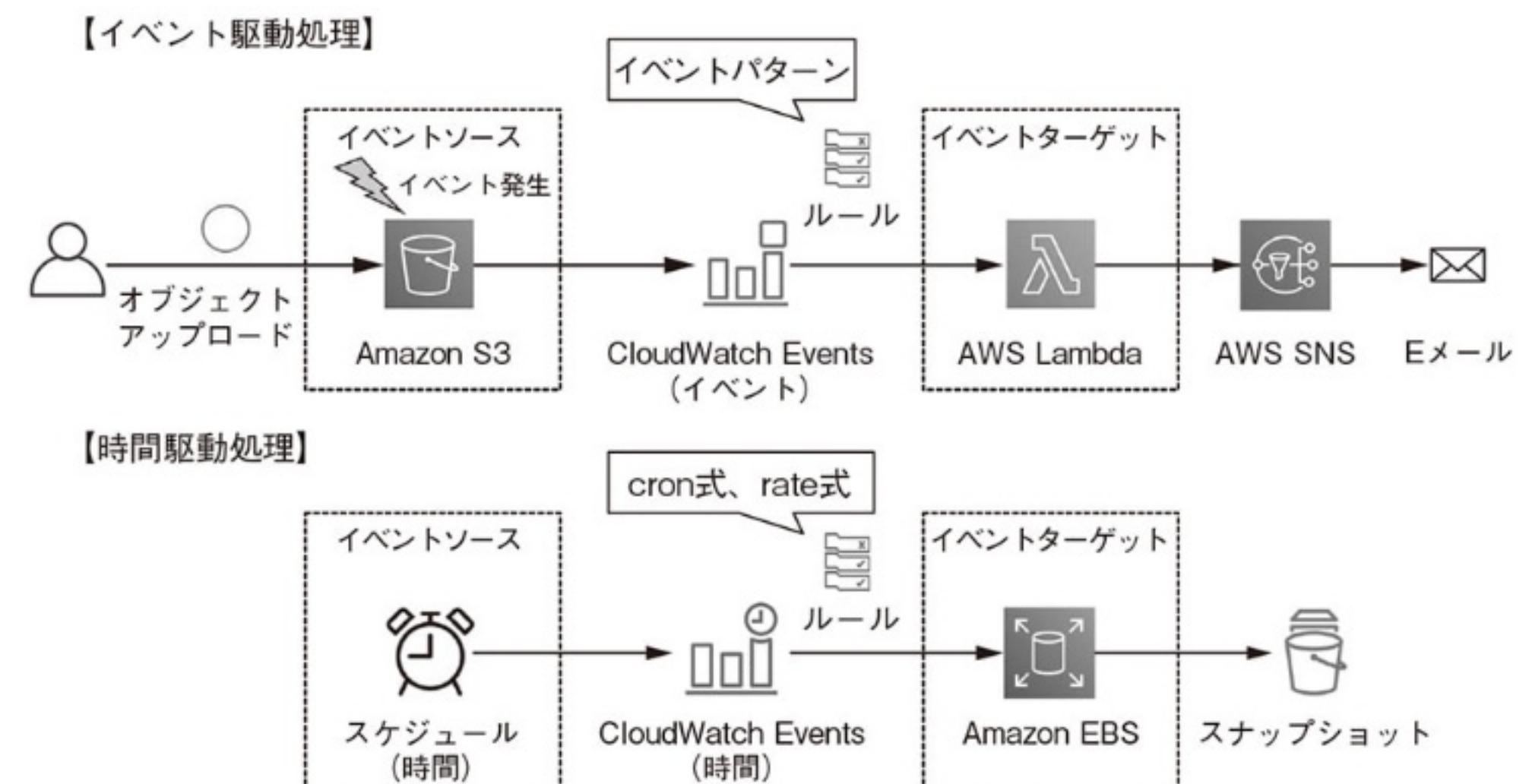
自動更新間隔を設定することも可能です。10秒、1分、2分、5分、15分の5つの選択肢が用意されているので、要件に合わせて選択しましょう。

7 CloudWatch Events

●CloudWatch Events の概要

CloudWatch Eventsは、システムリソースの変更イベントやスケジュールを契機にして、後続処理を実行するサービスです。「イベントソース」から受信したイベント情報を「ルール」に照らし合わせ、ルールに一致したイベントを「イベントターゲット」に振り分けることができます。

【CloudWatch Eventsの全体像と利用例】



*16 https://docs.aws.amazon.com/ja_jp/AmazonCloudWatch/latest/logs/CWL_QuerySyntax.html

●イベントソースとルール

CloudWatch Eventsでは「イベントパターン」もしくは「スケジュール」をもとにしたルールを設定できます。

「イベントパターン」では、イベントソースとしたシステムリソースの変更を契機とし、何らかの処理を実行させる「イベント駆動処理」を実現します。

イベント駆動処理の例として、上図に示した「Amazon S3バケットへのオブジェクトのアップロードを契機にEメール通知を行う」といった処理が考えられます。

このイベント駆動処理では、下記に示すような流れで処理が実行されます。

1. S3バケットにオブジェクトがアップロードされる。
2. 「イベントソース」であるS3バケットがJSON形式のイベント情報をCloudWatch Eventsに送信する。
3. CloudWatch Eventsが「ルール」を確認する。
4. 「Lambda関数にイベント情報を送信する」というルールとの一致を確認して、「イベントターゲット」であるLambda関数にイベント情報を送信する。
5. Lambda関数はJSON形式のイベント情報をもとにして通知メッセージを作成する。
6. Amazon SNS TopicにメッセージをPublishしてEメール通知が行われる。

イベントソースにはここで紹介したS3をはじめ、EC2やCodePipelineといったさまざまなサービスがサポートされています^{*17}。



試験対策

CloudWatch Eventsを使ったユースケースで適切な選択肢を問われる場合があります。イベントソースとして利用できる代表的なサービスを押さえておくとよいでしょう。ただし、2023年12月現在、AWSではイベントを扱う場合、Amazon EventBridge^{*18}が推奨されています。将来的にEventBridgeをより好ましい選択肢として問う問題が出てくる可能性があります。なお、CloudWatch EventsのAPIまたはサービス自体を将来的に廃止する予定はありません^{*19}。

「スケジュール」では、時間を実行契機とした時間駆動処理を実現できます。スケジュールの定義方法として、「cron式」と「rate式」がサポートされています。

スケジュール駆動処理の例として、上図に示した「毎日0:00にAmazon EBSのスナップショットを取得する」といった処理が考えられます。これはcron式を用いて cron(0 0 * * ? *) と設定することで実現できます。

*17 https://docs.aws.amazon.com/ja_jp/AmazonCloudWatch/latest/events/EventTypes.html

*18 https://docs.aws.amazon.com/ja_jp/eventbridge/index.html

*19 <https://aws.amazon.com/jp/eventbridge/faqs/>

cron式とrate式のそれぞれの構文を下記に示します。

【cron式の構文】

```
cron(分 時間 日 月 曜日 年)
```

【cron式のフィールド】

フィールド	設定可能な値	ワイルドカード
分	0-59	, - * /
時間	0-23	, - * /
日	1-31	, - * ? / L W
月	1-12 または JAN-DEC	, - * /
曜日	1-7 または SUN-SAT	, - * ? L #
年	1970-2199	, - * /

【cron式の設定例：毎日午前10:00（UTC）に実行】

```
cron(0 10 * * ? *)
```

【rate式の構文】

```
rate(value unit)
```

【rate式のフィールド】

フィールド	設定可能な値
value	正数
unit	minute, minutes, hour, hours, day, days

【rate式の設定例：10分ごとに実行】

```
rate(10 minutes)
```



cron式、rate式の両方とも引数を「半角スペース区切り」で設定する点に注意してください。



unitに記載する単位は单数形と複数形を区別する必要があります。rate(1 hours) と rate(2 hour) はどちらも無効です。

●イベントターゲット

CloudWatch Eventsのイベントターゲットには、下記に示すAWSサービスを指定できます。

1. Amazon EC2インスタンスの停止、再起動、削除
2. Amazon EBSのスナップショットの作成
3. AWS Lambda関数
4. Amazon Kinesis Data Streamsのストリーム
5. Amazon Kinesis Data Firehoseの配信ストリーム
6. Amazon CloudWatch Logsのロググループ
7. Amazon ECSのタスク
8. Systems Manager RunCommand
9. Systems Manager Automation
10. AWS Batchのジョブ
11. Step Functionsのステートマシン
12. AWS CodePipelineのパイプライン
13. AWS CodeBuildのプロジェクト
14. Amazon Inspectorの評価テンプレート
15. Amazon SNSのトピック
16. Amazon SQSのキュー
17. ほかのAWSアカウントへのイベントバス

上記の「イベントバス」を利用すると、ほかのAWSアカウントとのイベントの送受信を行えます。昨今で一般的になりつつある、マルチアカウントでのシステムの展開に有用な機能です。

例えば、「Amazon EBS バックアップなどの共通的な処理を情報集約アカウントにて一元管理し、個別アカウントに対してクロスアカウントで処理を実行する」といった運用を実現できます。

また、CloudWatch EventsではPrivateLinkがサポートされており、プライベートサブネットやオンプレミス環境からのイベントを起動契機とする処理の実装も行えます。

5-2 AWS CloudTrail

CloudTrailは、AWSで実行されるほぼすべてのAPIの実行履歴を記録するサービスです。監査や不正実行履歴の調査、トラブルシューティングなどで利用されます。

1 CloudTrailの機能

●CloudTrail の概要とイベント

CloudTrailは、AWSアカウント内のユーザ、ロール、サービスなどがマネジメントコンソールやCLI、SDKから実行したAPIアクションをイベントとして記録します。記録の対象は、過去90日間のイベントの履歴となります。プレビュー中や一般提供（GA：General Availability）前のサービス、パブリックAPIがないサービスなどは記録の対象外となりますが、AWSで提供されているほぼすべてのAPIがカバーされています^{※20}。

CloudTrailを有効化すると、API実行履歴がイベントログとして約15分ごとに出力されます。有効化はリージョン単位に行う必要があります。基本的に利用しているリージョンはすべて有効化することが推奨されています。保存期間が90日を超えたログは順次削除されます。より長期間ログを保存する必要がある場合は、S3に保存します。このとき、複数のリージョンのログを1リージョンのバケットに集約することも可能です（クロスリージョンレプリケーション）。また、出力オプションとしてCloudWatch Logsを選択することで、サブスクリプションフィルタによる特定のイベント検知・通知や、CloudWatch Logs Insightsを使ったイベント検索ができるようになります（後述）。

イベントは以下の3種類があり、記録対象を選択できます。デフォルトでは管理イベントが記録対象となります。

●管理イベント

AWSアカウントのリソースで実行される管理オペレーション（コントロールプレーンオペレーション）を対象としたイベントです。IAMロールにポリシーをアタッチする操作や、ネットワーク構成の変更、デバイス登録などがこれにあたります。イベントは証跡として複数出力できますが、管理イベントログは2つ目以降有料です。

^{※20} https://docs.aws.amazon.com/ja_jp/awscloudtrail/latest/userguide/cloudtrail-aws-service-specific-topics.html

●データイベント

AWSのサービスで取り扱われるリソースやデータなどに対する操作オペレーション（データプレーンオペレーション）を対象としたイベントです。S3に保存されたオブジェクトに対する操作や、Lambda関数の実行、DynamoDBテーブルのデータ更新などがこれにあたります。データイベントは出力すると有料となるオプションです。

●インサイトイベント

管理イベントが異常な挙動やエラーを示した場合に記録されるイベントです。アカウントの一般的な使用パターンと大きく異なるAPI呼び出しの変化をCloudTrailが検出した場合にだけログに記録されます。S3バケットが短時間で大量に作成されたりなど、直近7日間で記録されているAPI呼び出し記録の頻度に従って異常値を検出します（CloudTrail Insights）。インサイトイベントも出力すると有料となるオプションです。

参考までに、管理イベントのサンプルを示します。以下は、IAMユーザ「Alice」がCLIからユーザ「Bob」を作成したときに記録されるイベントです。

```
{"Records": [{"eventVersion": "1.0", "userIdentity": {"type": "IAMUser", "principalId": "EX_PRINCIPAL_ID", "arn": "arn:aws:iam::123456789012:user/Alice", "accountId": "123456789012", "accessKeyId": "EXAMPLE_KEY_ID", "userName": "Alice"}, "eventTime": "2014-03-24T21:11:59Z", "eventSource": "iam.amazonaws.com", "eventName": "CreateUser", "awsRegion": "us-east-2", "sourceIPAddress": "127.0.0.1", "userAgent": "aws-cli/1.3.2 Python/2.7.5 Windows/7", "requestParameters": {"userName": "Bob"}, "responseElements": {"user": {"createDate": "Mar 24, 2014 9:11:59 PM", "userName": "Bob", "arn": "arn:aws:iam::123456789012:user/Bob", "path": "/"}}}], "userId": "EXAMPLEUSERID"}]
```

```
"userId": "EXAMPLEUSERID"
}}
}]]}
```

CloudTrailで出力されるイベントログは、SSE-KMSを使って暗号化して保存したり（3章3.3節「AWSで実行される保存データの暗号化の主なユースケース」を参照）、SHA-256ハッシュアルゴリズムを使用してログが改竄されていないことを証明する署名を付与したりするオプションを利用できます。署名はイベント同様、S3の別フォルダに格納されます。CLIからは以下のvalidate-logsコマンドより、指定された時間の範囲でログが改竄されていないことを確認できます。

```
aws cloudtrail validate-logs --start-time 2015-08-27T00:00:00Z --end-time 2015-08-28T00:00:00Z --trail-arn arn:aws:cloudtrail:us-east-2:111111111111:trail/my-trail-name --verbose
```

そのほか、イベントのバージョニング記録や、ログ削除の2段階認証（MFA Delete）などがサポートされています。

2 CloudTrail イベント履歴の調査

収集されたイベントの履歴に対し、次のサービスやツールを使ってイベントを検索できます。

●マネジメントコンソール

マネジメントコンソールで「CloudTrail」サービスから「イベント履歴」メニューを選択すると、過去90日間のイベントが閲覧できます。

【マネジメントコンソールの「イベント履歴】

イベント履歴では複雑な検索はできませんが、以下のような単一属性キーを使ってイベントをフィルタリングできます。

- ・AWSアクセスキー
- ・イベント名
- ・イベントソース
- ・リソース名
- ・リソースタイプ
- ・ユーザ名

●Amazon Athena

Amazon Athenaは、S3内にある構造化されたデータをSQLで分析できるサービスです。Athenaを利用すると、S3に保存されたイベントデータに対し、さまざまな条件式、演算子を組み合わせた高度なクエリを実行できます。特定のユーザの過去の処理解析や、複雑な条件でのイベントデータの抽出に向いています。

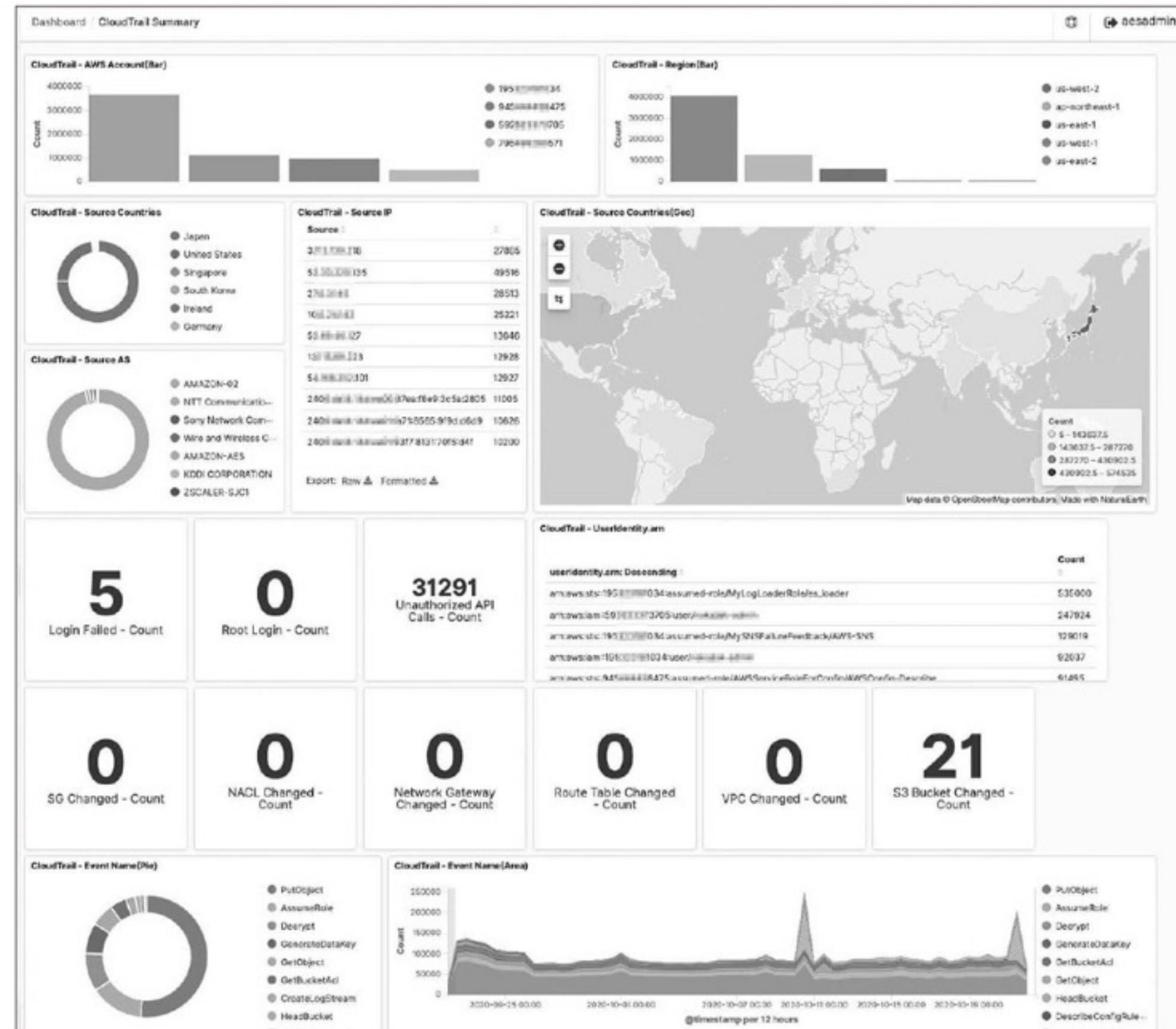
Athenaを利用するには、読み込み対象の構造化データに対応するテーブルを作成する必要がありますが、前述の「イベント履歴」の画面イメージにあるとおり、Athenaのテーブルはコンソール上からも簡単に作成できます。ただし、CloudWatchイベントのデータ数は相当量になるため、Athenaで指定する際は、読み込むデータの対象が多くなりすぎないようターゲットを絞ったパーティション設計を行う必要があります。パーティションはイベントの特定の属性をキーにし

て読み込み対象を選択できます^{※21}。多くの場合、実行日時や、実行されたサービスを示すイベントソースなどでパーティションを作成し、データのスキャン量を削減します。

●Amazon OpenSearch Service

Amazon OpenSearch Serviceは、オープンソースの分散型RESTful検索/分析エンジンであるOpenSearch Serviceとデータを可視化するオープンソースのKibanaを組み合わせたサービスです。インスタンスとストレージの維持に追加コストが発生しますが、解析と可視化に優れたサービスなので、複数のAWSサービスにまたがった情報を集約・加工し、統計データを可視化したい場合などに向いています。次の図は、CloudTrailから収集されたイベントのIPアドレスなどからアクセス統計を分析した例です。

【SIEM on Amazon ESを使ったCloudTrailのログの可視化の例^{※22}】



^{※21} https://docs.aws.amazon.com/ja_jp/athena/latest/ug/cloudtrail-logs.html#create-cloudtrail-table

^{※22} <https://github.com/aws-samples/siem-on-amazon-opensearch-service>

●CloudWatch Logs Insights

5章1.5節で解説した「CloudWatch Logs Insights」を使ってイベントログを解析します。CloudWatch Logs Insights自体の解説は割愛しますが、メリットは前述の解析ツールと比べてイベントデータのロードが不要であることと、イベント履歴と比較して高度な検索機能を持つことです。イベント発生からできるだけリアルタイムに近い解析を行いたい場合などのユースケースが向いています。

5-3 AWS X-Ray

AWS X-Rayは、アプリケーションの処理単位でメトリクスを収集・可視化・分析できるサービスです。開発中にアプリケーション処理の性能ボトルネックを検出したり、商用環境での障害検出したりする用途で利用されます。

1 X-Rayの概要と概念

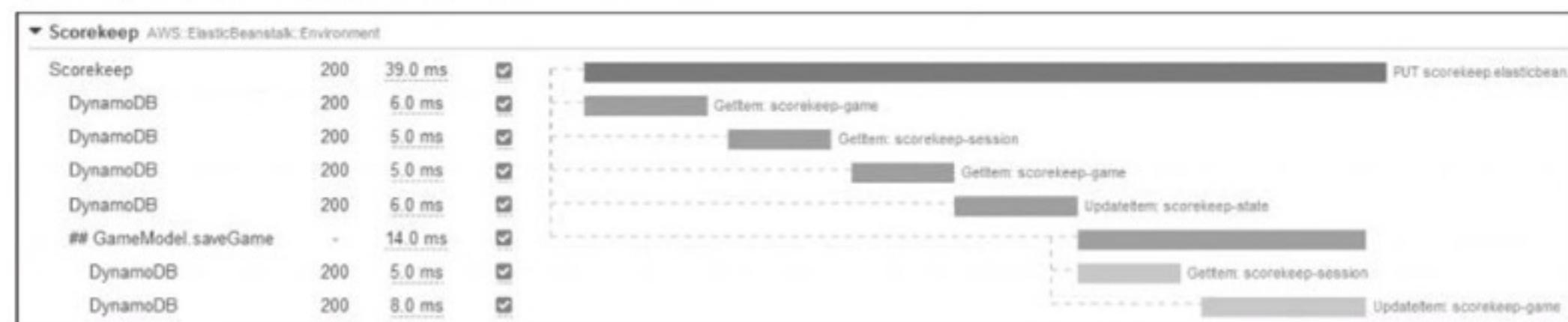
X-Rayは、次のイメージ図のようにアプリケーション内の処理の呼び出し関係、実行可否やレスポンス時間を可視化するサービスです。商用環境で稼働するアプリケーションのエラー監視や、性能試験などで処理の実行状況を可視化することでアプリケーション内で発生している問題を分析できます。マイクロサービスアーキテクチャなど、処理が分散し、さまざまなサービスの通信が発生する場合の分析に適しています。

【X-Rayのメトリクスを可視化したサービスマップイメージ^{※23}】

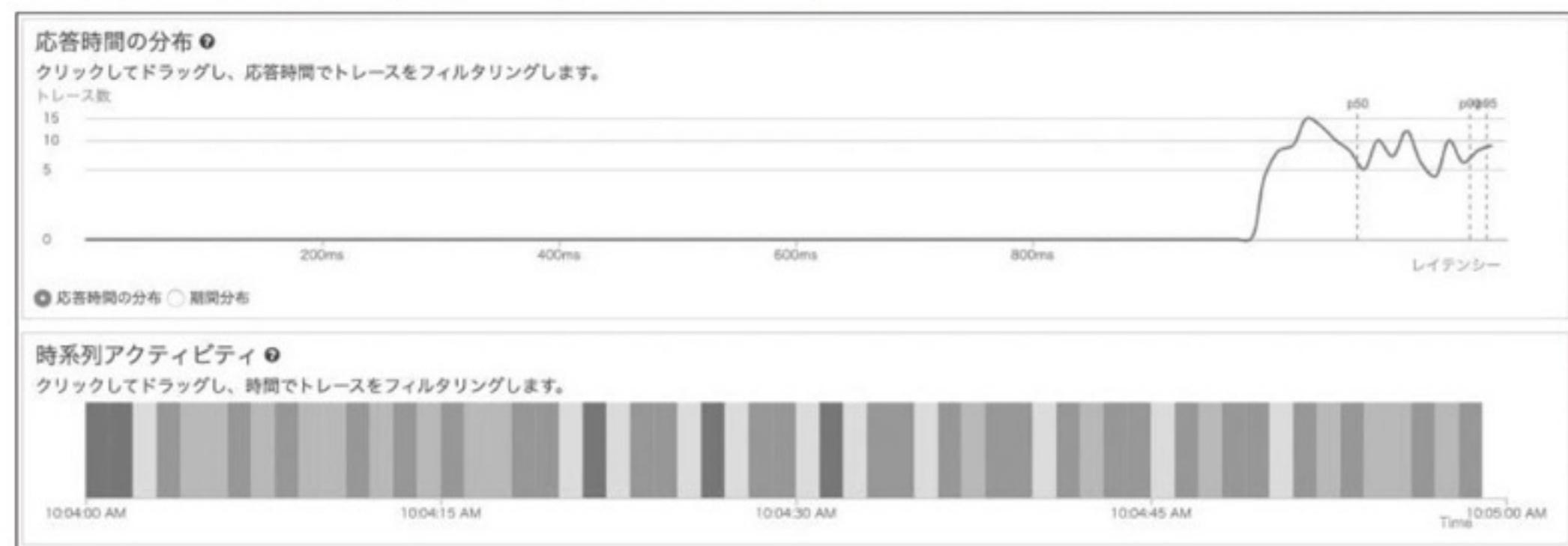


^{※23} <https://docs.aws.amazon.com/xray-sdk-for-ruby/latest/reference/>

【X-Rayのトレースリストイメージ※24】



【X-Rayのアナリティクスツールイメージ※25】



●X-Rayの機能

X-Rayでは次の4つの機能を利用できます。

●サービスマップ

アプリケーションをサービス単位で分割した場合や、RDS、DynamoDB、SQSといったAWSマネージドサービスの呼び出しが含まれる場合に、サービス間の呼び出しの成否や処理にかかった時間を可視化するマップです。

●トレース

あるアプリケーションへの呼び出しの内訳を時間軸で並べてリスト化する機能です。リクエストの受付からレスポンスの返却までのアプリケーションの処理時間や、マネージドサービスの呼び出し時間を並べることで、ボトルネックやエラー箇所を特定できます。

●アナリティクス

アナリティクスは各トレースを統計的に計測した場合のトレンドを可視化する機能です。指定された時間を基準とした応答時間ディストリビューション（分布）と、指定された期間のトレース量を可視化する時系列アクティビティがあります。

X-Rayでは、メトリクス収集のためのSDKライブラリがさまざまなプログラミング言語で提供されており、JavaやNode.js、Python、C#、Ruby、Go言語で利用できます。

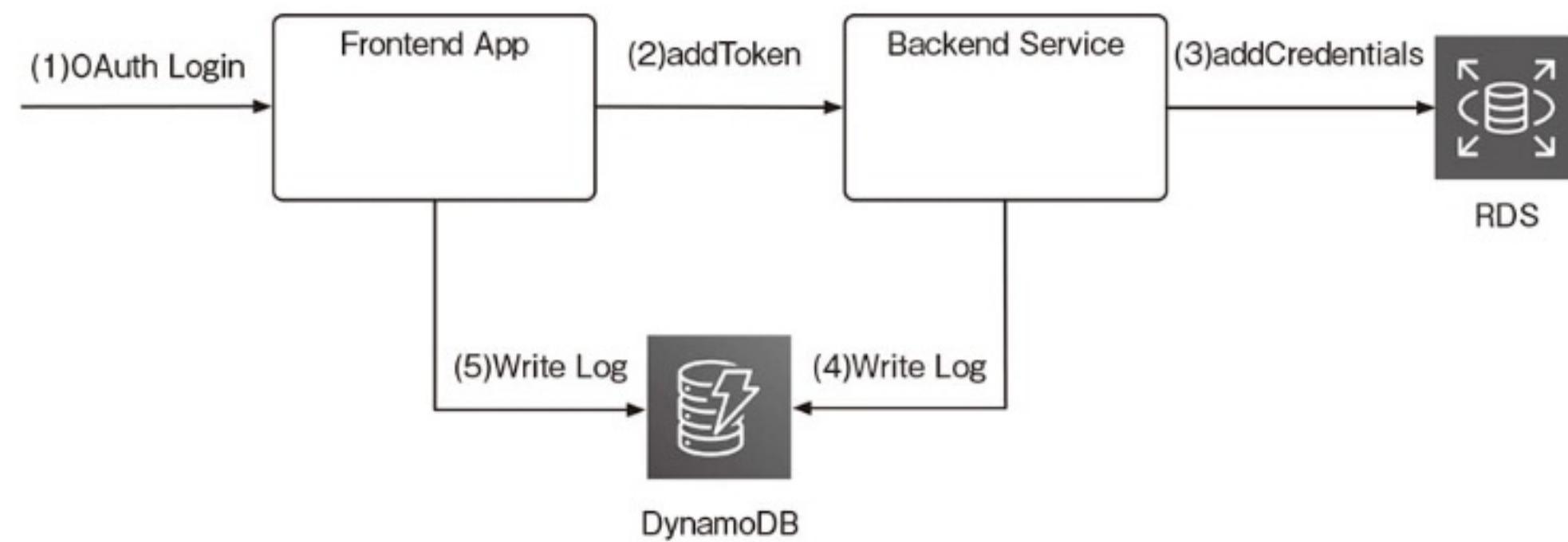
●インサイト

インサイトでは、パフォーマンスやアプリケーション上の問題を継続的に検知できます。

●X-Rayの基本的な概念

具体例を見ながら、X-Rayへの理解を深めましょう。ここでは、あるクライアントがOAuth2トークンを持ってフロントエンドのアプリケーションにログインし、バックエンドに配置されたマイクロサービスを通して、RDSにトークンを保存する例を考えます（下図参照）。フロントエンドのアプリケーションとバックエンドサービスは、それぞれ処理が完了した後にDynamoDBに実行ログを書き込むものとします。

【X-Rayで情報収集するアプリケーションおよびサービスの構成例】

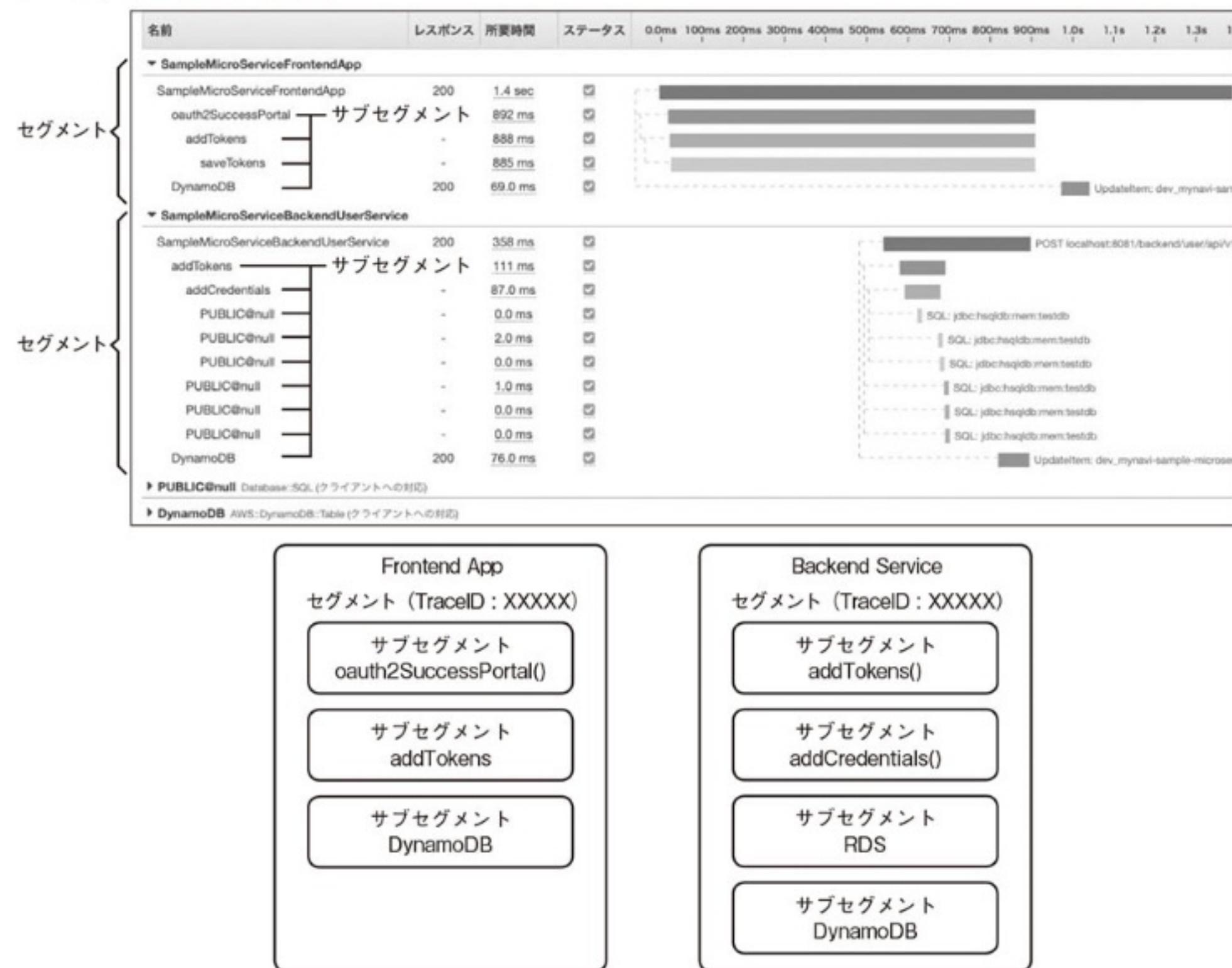


一連の処理が終わると、X-Rayでは、次の図のようにトレースリストが表示されます。この図が表す内容を理解するには「セグメント」「サブセグメント」「TraceID」という3つの概念を理解しておく必要があります。

※24 https://dl.awsstatic.com/webinars/jp/pdf/services/20200526_BlackBelt_X-Ray.pdf

※25 https://dl.awsstatic.com/webinars/jp/pdf/services/20200526_BlackBelt_X-Ray.pdf

【X-Rayの重要な概念】



● セグメント

セグメントはデータを収集する単位です。1つのサービス・アプリケーションに割り振られる識別子として、最初にアクセスしたセグメントにTraceIDが割り振られます。同じセグメントとして、別のアプリケーションを呼び出す場合は、リクエストヘッダ（X-Amzn-Trace-Id）にTraceIDを設定して送信します。ヘッダがセットされたリクエストを受け取ったセグメントは、そのTraceIDを引き継ぎます。

● サブセグメント

セグメントは任意のサブセグメントに分割できます。開発者はアプリケーション内で実行時間を計測したい単位でカスタムで区切るほか、マネージドサービスにアクセスした際の実行時間を計測するよう設定することができます。

● TraceID

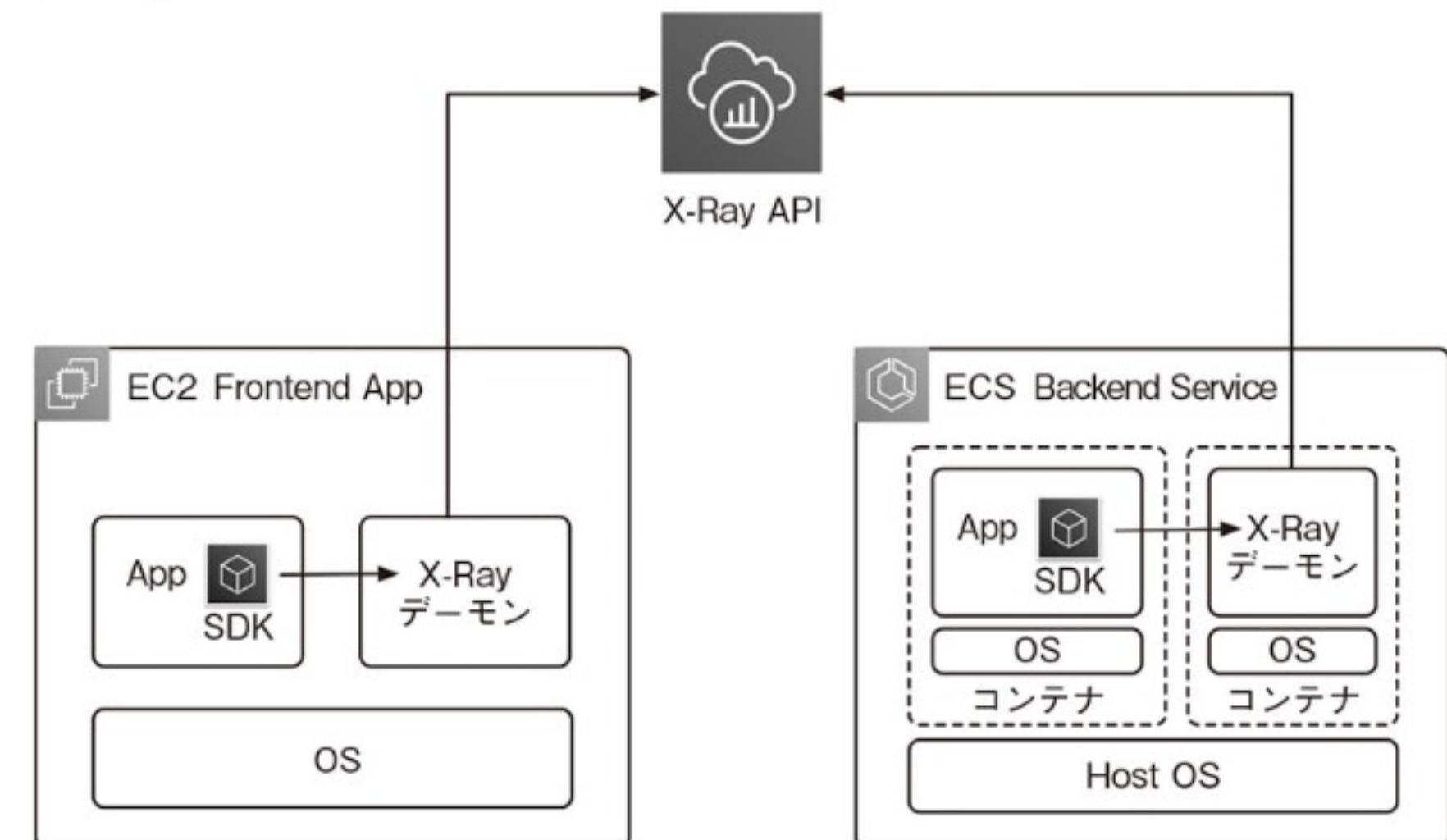
TraceIDはリクエストを識別するIDです。初めてセグメントを開始した際に一意となるUUIDが割り当てられ、リクエストはこのIDを持ち回り、サービスの呼び出し関係を可視化します。

2 X-Rayの利用

● X-Ray デーモンの設定

トレースデータを収集するには、アプリケーションの処理の中でX-RayのSDKライブラリを経由してAPIを呼び出します。アプリケーションから直接API（PutTraceSegments API）を呼び出すこともできますが、この方法を採用することはほとんどありません。アプリケーションの性能に極力影響がないよう、SDKはバックグラウンドプロセスとして動作するX-Rayデーモンを中継し、定期的にデータを送信します。X-RayデーモンはEC2やECSをはじめ、App Mesh、AWS Lambda、API GatewayなどさまざまなAWSリソースでサポートされています。下記の図では、フロントエンドのアプリケーションがEC2、バックエンドのサービスがECSで実行されていた場合のデーモンの実行形態を示しています。

【X-Rayのデーモンの実行形態】



EC2ではデーモンをインスタンスにインストールして実行する必要がありますが、ECSではデーモンプロセスが実行されているコンテナを起動します。Dockerコンテナで動作するデーモンプロセスには、X-Ray APIを呼び出す権限を付与したIAMユーザやロールの認証情報を設定する必要があります。



X-Rayデーモンの環境設定はAWSのリソースであればコンソールなどから行えますが、ローカル端末などでコンテナによるデーモンプロセスを実行する場合は事前に環境構築が必要です。Dockerをインストールした後、公式のDockerfile^{※26}からコンテナイメージを作成し(xxxx/xray-daemon:latest)、以下のように、オプションを使って、実行ホストOSのAWS認証情報を参照してポート2000番で受け付けるように起動しておきます。

```
docker run --attach STDOUT -v ~/.aws:/root/.aws/ --net=host
-e AWS_REGION=ap-northeast-1 --name xray-daemon -p 2000:2000/
udp xxxx/xray-daemon:latest -o
```

```
}
```

なお、実行環境がLambdaの場合はサンプリングルールを変更できません。デフォルトの「1秒ごとに最初のリクエスト、5%のサンプリングレート」で収集されます。

●受信リクエスト

X-Rayのデータ収集対象となるリクエストは、最初のセグメントでTraceIDを付与され、リクエストヘッダにTraceIDが付与されれば、それを引き継いでセグメントを開始します。実装はアプリケーション言語およびSDKの使い方によって異なりますが、まず最初に上記が機能するよう設定が必要です。

●別サービスへのHTTPリクエスト送信

アプリケーションが別のサービスを同期処理で呼び出す際に、トレースを伝播させるのであれば、リクエストヘッダにTraceIDを設定してリクエスト送信する必要があります。こちらもアプリケーション言語およびSDKの使い方によって実装は異なります。

●AWS SDKクライアント

DynamoDB/S3へのアクセスやSQSキュー送信など、SDKのクライアントを使ってリクエスト送信する場合に、X-Rayの設定が必要になります。こちらもSDKの実装に依存しますが、多くはクライアント生成時にX-Rayの設定を行います。以下は、Java言語でDynamoDBのクライアントを生成する際に、X-Rayの設定を行う実装例です。

```
private AmazonDynamoDB client = AmazonDynamoDBClientBuilder.
standard()
.withRegion(Regions.fromName(System.getenv("AWS_REGION")))
.withRequestHandlers(new TracingHandler(AWSXRay.
getGlobalRecorder()))
.build();
```

●SQLクライアント

リレーショナルデータベースアクセスが必要なアプリケーションの場合は、データベースへの接続ドライバやSQLクライアントにX-Rayの設定を行います。

●SDKの利用

アプリケーションではSDKを用いてデーモンに対してトレースデータの送信を行います。SDKの呼び出し方はアプリケーションの実装言語に応じて異なりますが、アプリケーション側で以下の処理を実装しておく必要があります。

●サンプリングルールの設定

実際に行われる処理に対し、どれくらいの割合でトレースデータを送信するかサンプリングのルールを設定します。リクエストのパスやHTTPメソッドなどに応じてサンプリングルールを設定できます。通常以下のような、JSON形式のファイルでサンプリングレートを設定します。なお、コンソールからも同様の設定が可能です。

```
{
  "version": 2,
  "rules": [
    {
      "description": "Player moves.",
      "host": "*",
      "http_method": "*",
      "url_path": "/api/move/*",
      "fixed_target": 0,
      "rate": 0.05
    },
    ],
    "default": {
      "fixed_target": 1,
      "rate": 0.1
    }
  }
```

※26 <https://github.com/aws/aws-xray-daemon>

●カスタムサブセグメント

アプリケーションの任意のタイミングでサブセグメントを開始終了し、実行時間を計測します。以下はPythonでサブセグメントを開始、終了する例です。

```
from aws_xray_sdk.core import xray_recorder

subsegment = xray_recorder.begin_subsegment('annotations')
subsegment.put_annotation('id', 12345)
xray_recorder.end_subsegment()
```

定しておき、フィルタ式のキーワードで、「annotation.key=value」の形式でキーとバリューを指定すると、該当のトレースデータだけをフィルタできます。

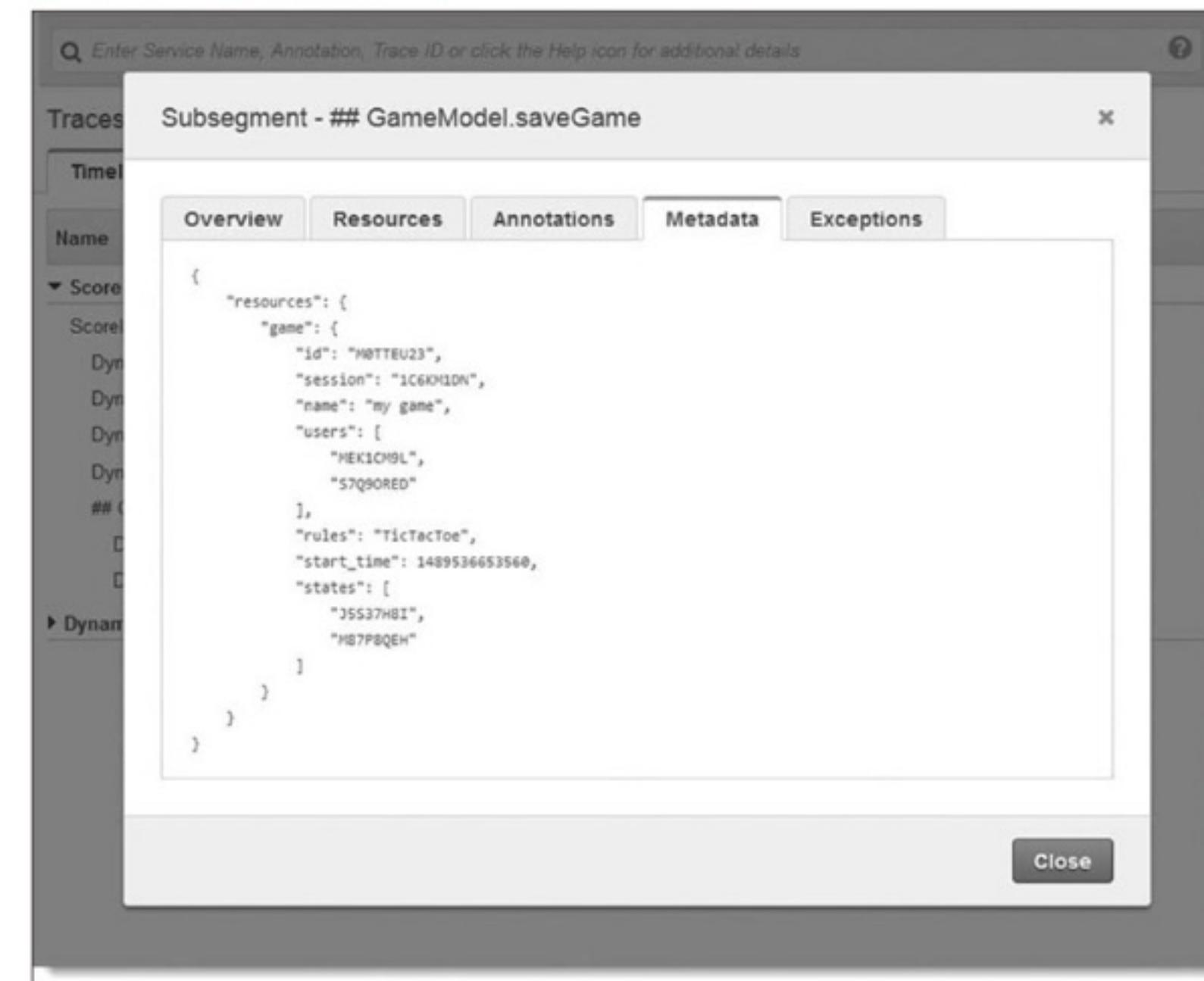
```
Segment segment = AWSXRay.getCurrentSegment();
segment.putAnnotation("gameid", game.getId());
```

トレースごとに最大50個の注釈がインデックス化され、検索キーワードとして利用できます。

メタデータは、トレースに付加的な情報を記録できるキーバリューペアです。トレースの詳細を表示した際にメタデータも一緒に参照できます。解析に必要な情報があれば、次のコード例のようにアプリケーションでサブセグメントのメタデータとして設定しておきます。

```
Subsegment subsegment = AWSXRay.beginSubsegment("## GameModel.
saveGame");
subsegment.putMetadata("resources", "game", gameModel);
// omit
AWSXRay.endSubsegment();
```

【メタデータの利用^{※29}】



●AWS X-Ray のその他の機能

そのほか、X-Rayを活用する上で押さえておきたい機能を解説します。

●フィルタ式

フィルタ式はトレースを抽出するための機能です。マネジメントコンソールなどで、「keyword operator value」形式で条件を指定することによりトレースを抽出できます。以下は「/api/move」のURLでアクセスされるトレースを抽出します。

```
http.url CONTAINS "/api/move"
```

【フィルタ式の利用^{※27}】

ID	Trace status	Timestamp	Response code	Response Time	Duration	HTTP Method
...561513004630e58c75c992ed	OK	3.4min (2023-08-16 17:39:20)	200	0.104s	0.104s	POST
...2e83714b7daac593167d2e73	OK	3.4min (2023-08-16 17:39:19)	200	0.07s	0.07s	POST
...54740787431329383155f154	OK	3.4min (2023-08-16 17:39:18)	200	0.1s	0.1s	POST

使用可能なキーワードや演算子は公式「フィルターエクスプレッションを使用する^{※28}」も参考にしてください。

●注釈 (Annotations) とメタデータ (Metadata)

注釈は上記のフィルタ式で使用するためのインデックス化されたキーバリューペアです。例えばアプリケーションで、次のコードのように注釈に特定の値を設

※27 https://docs.aws.amazon.com/ja_jp/xray/latest/devguide/xray-concepts.html#xray-concepts-filterexpressions

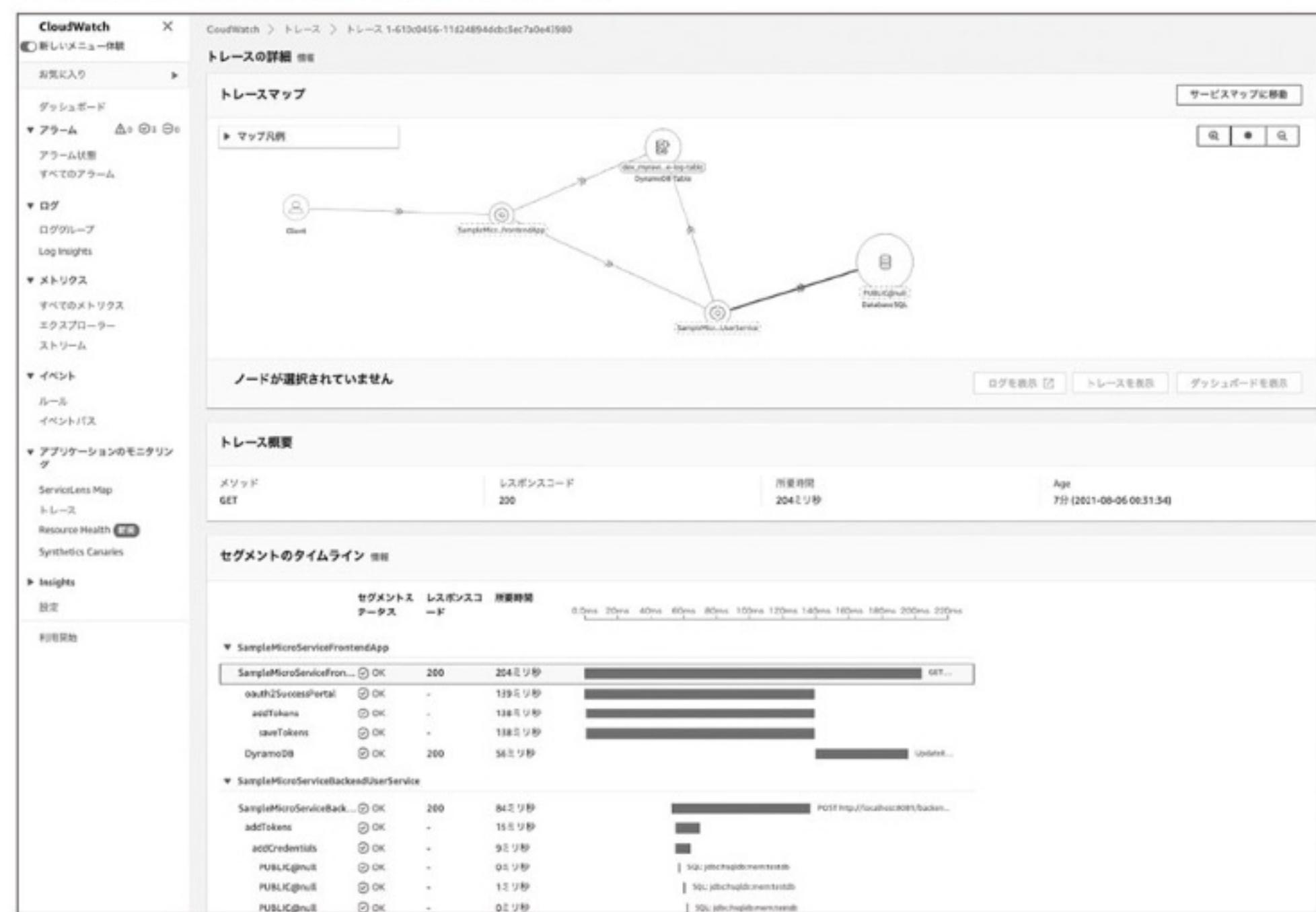
※28 https://docs.aws.amazon.com/ja_jp/xray/latest/devguide/xray-console-filters.html

※29 https://docs.aws.amazon.com/ja_jp/xray/latest/devguide/xray-console-traces.html

- CloudWatchとの統合 (CloudWatch ServiceLens)

X-Rayで収集したデータはCloudWatchとも統合されており、アプリケーションの監視にも活用できます。CloudWatch ServiceLensでは、CloudWatchで収集されるメトリクスやログデータに加えて、X-Rayのデータを使用することで、サービスマップやトレースリストなど、アプリケーションの可視化した情報を1箇所でまとめて確認できます。

【CloudWatch ServiceLensの利用^{※30}】



5-4 | Amazon ElastiCache

Amazon ElastiCacheはセットアップ、運用、拡張が容易なマネージドキャッシュサービスです。

● ElastiCache の概要

Amazon ElastiCacheの主な用途は、RDBに保存してあるマスターデータのキャッシュ化や、複数のAP（アプリケーション）サーバでセッションデータの共有化などです。オープンソースのキャッシュソフトウェアであるMemcachedとRedisをベースとした2種類のエンジンをサポートしており、用途やワークロードに応じて使い分けます。

1 ElastiCache for Memcached

ElastiCache for Memcachedは、Memcachedをベースとしたインメモリ型のキーバリューストアで、以下のような用途で利用されます。

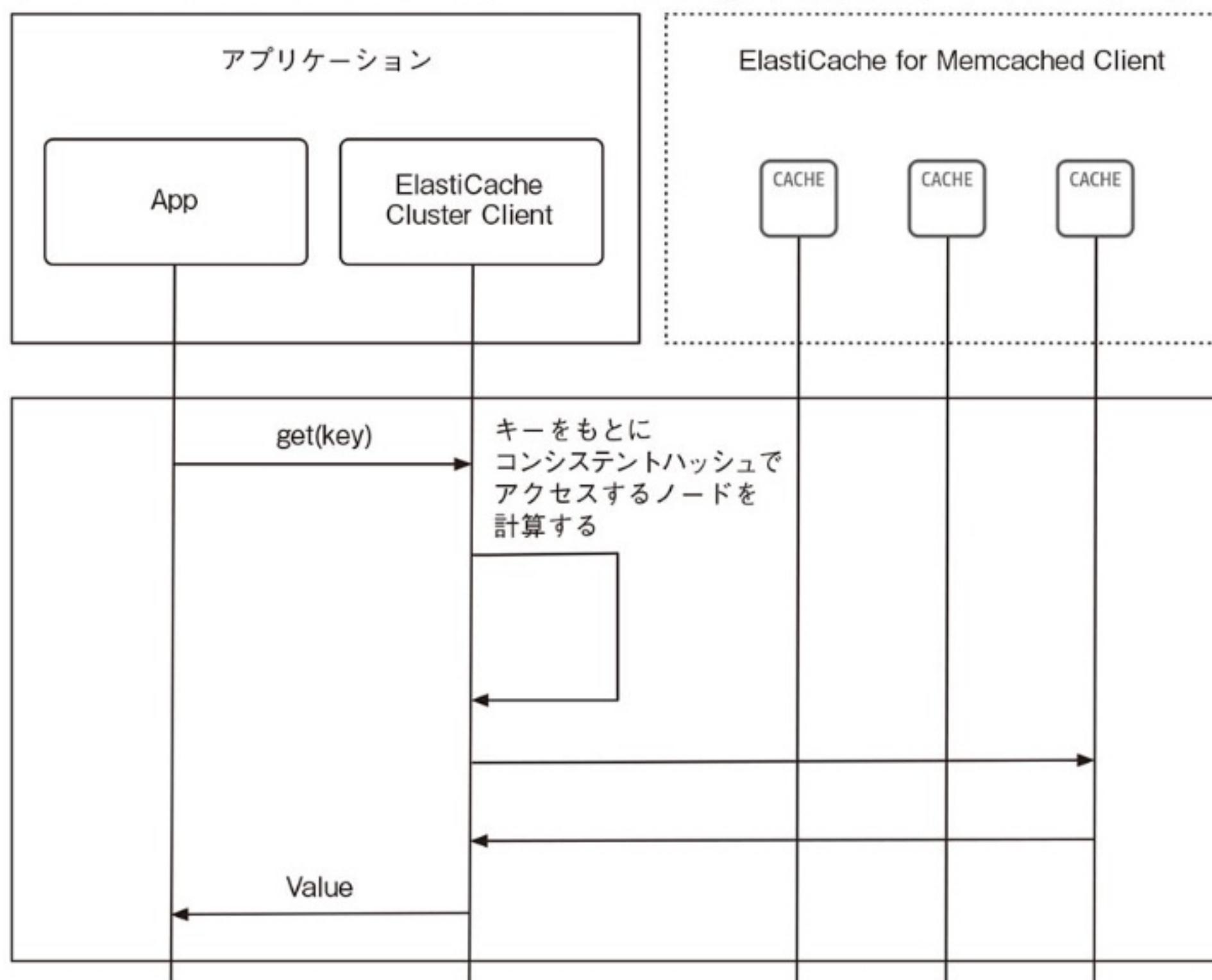
- ・シンプルなキーバリュー型のデータをキャッシュし、低レイテンシで参照する。
 - ・マルチスレッドでアクセス可能な共通データをキャッシュする。
 - ・リクエスト量に応じてノードをスケールアウト・スケールインする。

サポートされているMemcachedのバージョンは、公式ガイド^{※31}を参照してください。ElastiCache for Memcachedはクラスタを構成し、クラスタは複数のアベイラビリティゾーンにまたがって1つ以上のMemcachedキャッシュノードで構成されます。下記の図は、ElastiCacheがデータ保存されているノードにアクセスする仕組みを示したものです。

※30 https://docs.aws.amazon.com/ja_jp/xray/latest/devguide/xray-concepts.html#xray-concepts-annotations

*31 https://docs.aws.amazon.com/ja_jp/AmazonElastiCache/latest/mem-ug/supported-engine-versions.html

【ElastiCache for MemcachedのAuto Discoveryに対応したクライアント^{※32]}



各データはキーのコンシスティントハッシュにもとづいて各ノードに分散して保存されています。クラスタは設定エンドポイントと各キャッシュノードのエンドポイントを持ち、クラスタにはAWSが提供しているAuto Discoveryに対応したクライアント(ElastiCache Cluster Client)を使ってアクセスします。クライアントは設定エンドポイントから指定されたキーの値にもとづいて、格納されているノードのエンドポイントを取得・アクセスし、データを返します。

アプリケーションはまず、指定されたキーを持つデータがキャッシュに存在するかをチェックします。存在すればそのデータを利用し、存在しなければキャッシュ対象のデータソースからデータを取得してキャッシュに保存して、データを利用します。次回以降、同じキーが指定されると、キャッシュからデータを取り出せるようになります。ただし、マルチスレッドで動作するため、データの更新が発生するとスレッド間で不整合が発生する可能性があります。さまざまリクエストから共通的に参照可能なデータに限って利用したほうがよいでしょう。また、ElastiCache Cluster Clientが利用できる言語は、C#、PHP、Javaに限られます。それ以外の言語のアプリケーションについては、どのノードにデータが保存されているかアプリケーション側で判定してから、通常のクライアントライブラリを使ってキャッシュノードに直接アクセスするようにしてください。

^{※32} https://docs.aws.amazon.com/ja_jp/AmazonElastiCache/latest/mem-ug/AutoDiscovery-HowAutoDiscoveryWorks.html

2 ElastiCache for Redis

ElastiCache for Redisは、Redisをベースとした高性能のインメモリ型キーバリューストアです。リードレプリカやマルチアベイラビリティゾーン構成の自動フェイルオーバーといった、AWSのマネージドサービスならではの機能をサポートします。同時に、Redis 3.2からの新機能であるRedis Clusterもサポートし、データをいくつかのグループで分割したシャードと呼ばれる単位で分散保存することで、より高いパフォーマンスや可用性が得られます。

サポートされるバージョンは公式ガイド^{※33}に記載されているので参考にしてください。

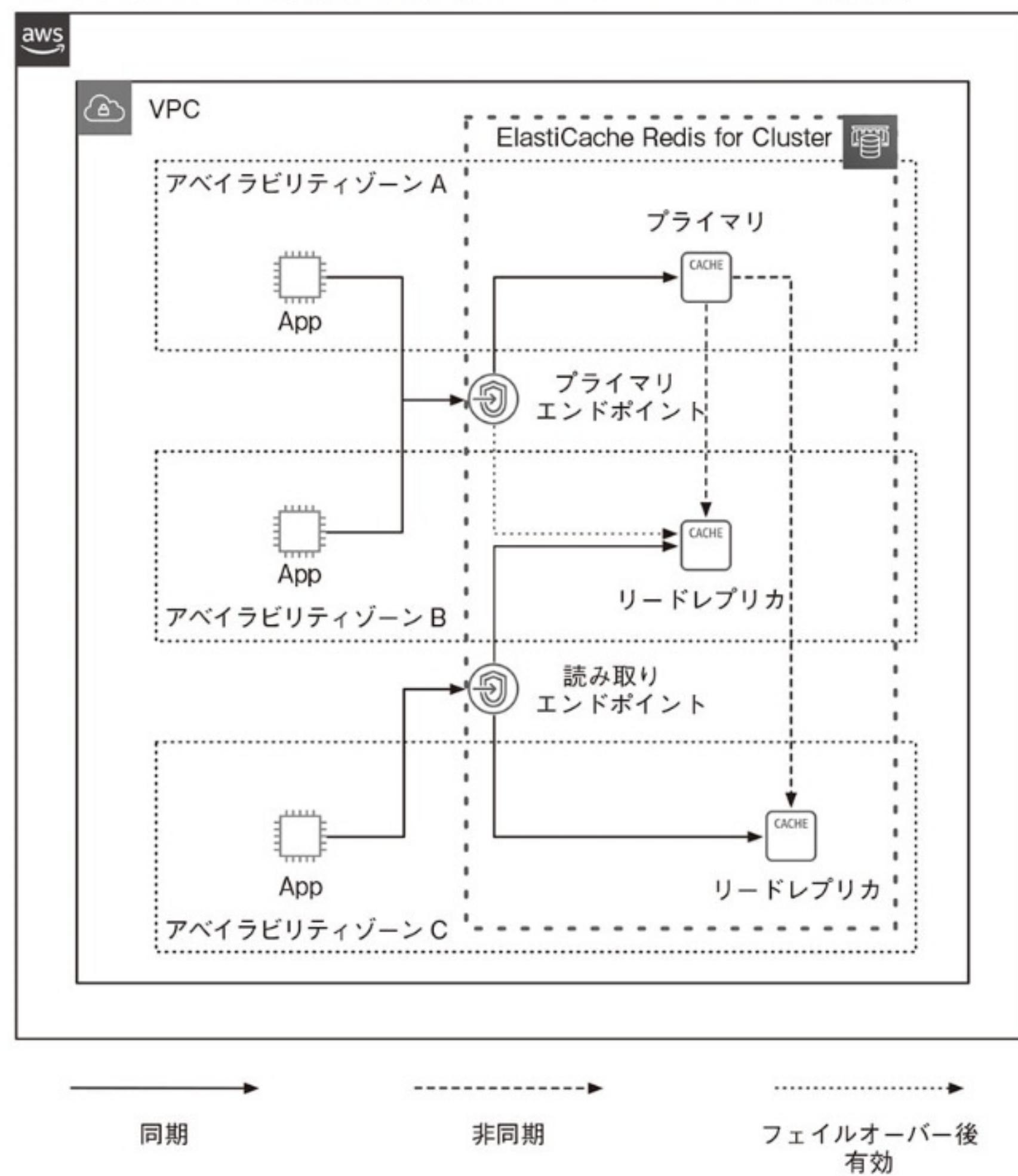
ElastiCache for Redisはクラスタモードの使用有無によって構成が大きく異なります。まずは各パターンの構成を押さえておきましょう。

●クラスタモードが無効のパターン

データをいくつかのグループで分割して保存することで、書き込み・読み取りの負荷を分散する手法をシャーディングといいます。このとき、保存先となる1つ1つのグループ（論理単位）のことをElastiCache for Redisではシャードと呼んでいます。「クラスタモードが無効」のモードでは、ElastiCache for Redisはシャーディングを行いません。その代わりに、データが保存されているプライマリノードとは別に、リードレプリカを別のアベイラビリティゾーンに配置した構成をとることで、読み取り負荷を分散しています。以下にクラスタモードが無効の場合の構成を示します。

^{※33} https://docs.aws.amazon.com/ja_jp/AmazonElastiCache/latest/red-ug/supported-engine-versions.html

【クラスタモードが無効の場合のElastiCache for Redisの構成】



プライマリノードからリードレプリカには非同期にデータが更新されますが、プライマリノードに障害が発生した際は、リードレプリカの1つがプライマリノードに自動的に昇格します(自動フェイルオーバー)。アプリケーションがキャッシュに更新処理を行う場合はエンドポイントとしてプライマリエンドポイントを指定しておきます。これにより、自動フェイルオーバーが行われたとしても、特にアプリケーション側の参照を切り替える必要はありません。また、キャッシュデータの参照しか行わないアプリケーションに対しては、読み取りエンドポイントを指定することで、複数のリードレプリカに分散してアクセスできます。なお、プライマリノードの障害によって昇格したリードレプリカに対しては、その時点で読み取りエンドポイントのディスパッチ先からは除外されます。

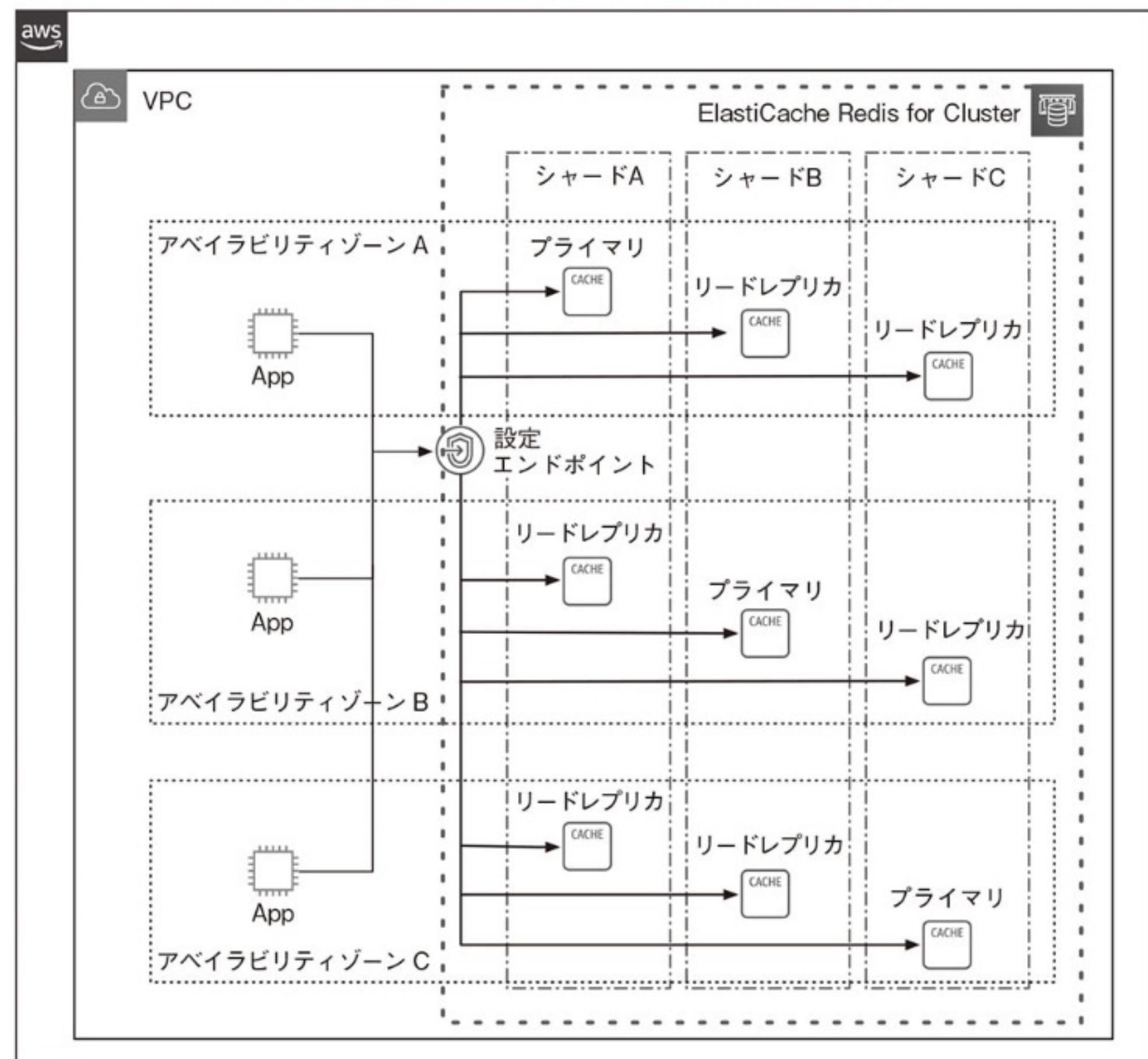
なお、見落とされがちですが、この構成では、フェイルオーバーの最中(数秒から数分程度ですが)、プライマリエンドポイントを使って参照しているアプリケーションはElastiCacheにアクセス不可になります。セッション共有のほか、キャッシュ用途で使用している場合でも、ダウンタイムが許されないのであれば、

アプリケーションの実装でフェイルオーバー中のエラーハンドリングに関する考慮が必要です。

● クラスタモードが有効のパターン

「クラスタモードが有効」なモードの場合は、スロット(キーのハッシュ値)に応じて、データをシャーディングしたかたちで分散して保存します。次の図ではクラスタモードが有効な場合の構成を示しています。

【クラスタモードが有効な場合のElastiCache for Redisの構成】



シャードごとのプライマリノードとレプリカのアベイラビリティゾーンでの配置も任意に設定することができます。

そのほか、ElastiCache for Redisには以下のような特徴が加わります。

●自動バックアップ・リストア

RedisのSnapshotを取得し、S3へエクスポートしたり、クラスタを作成する場合にそのデータをインポートすることができます。

●ダウンタイム0のクラスタリサイズ

クラスタのシャード数やレプリケーション数をダウンタイムなしにスケールアウト・インする機能があります。CloudWatchのアラームとSNS・Lambdaと組み合わせて動的にスケールすることができます。

●グローバルデータストア

最新のRedisエンジンで提供されている、クロスリージョン間のレプリケーション機能です。

●AutoScaling

事前定義されたルールや、CloudWatchメトリクスに従って、クラスタにシャードまたはレプリカを自動的に追加し、スケールアウト・スケールインする機能です。

なお、注意点としてRedisはパブリックアクセスが許可されていません。EC2やECSなどのVPC内のリソースからアクセスする構成にしておく必要があります。



Amazon Memory DB for Redis の登場

2021年の8月にMemoryDB for Redisが発表されました^{*34}。MemoryDBは、複数のアベイラビリティゾーンにまたがるデータを格納する分散トランザクションログを使用して、高速フェイルオーバー、データベースリカバリ、およびノードの再起動を高耐久性で実現します。ElastiCache Redisとの大きな違いとして、MemoryDBは、データの耐久性とマイクロ秒の読み取り、および1桁ミリ秒の書き込みレイテンシを提供するため、キャッシングがいらないアプリケーションの「プライマリデータベース」として安全に利用できます。対して、ElastiCacheは、上述の解説どおり、既存のデータベースからのデータアクセスを高速化する「キャッシング」や複数のアプリケーションでの「共有セッションストア」としての利用に向いています。



ElastiCache (Memcached と Redis) の使い分け

これまで解説してきたとおり、ElastiCacheはデータに高速にアクセスしたい場合に利用されるサービスです。MemcachedとRedisのエンジンは、データ特性に応じて以下のとおり使い分けます。

● Memcached

マルチスレッドでスケールアウトしやすい特性を持つことから、多数のノードから参照されるマスターデータのキャッシング用途

● Redis

- ・セッションなどのシングルスレッドのみから書き込みを許容したいデータ共有用途
- ・シャーディングなどでデータ自体を分散させて高速化したい、多数のキーを持つマスターデータのキャッシング用途

*34 <https://aws.amazon.com/jp/blogs/aws/introducing-amazon-memorydb-for-redis-a-redis-compatible-durable-in-memory-database-service/>

5-5 Amazon SQS

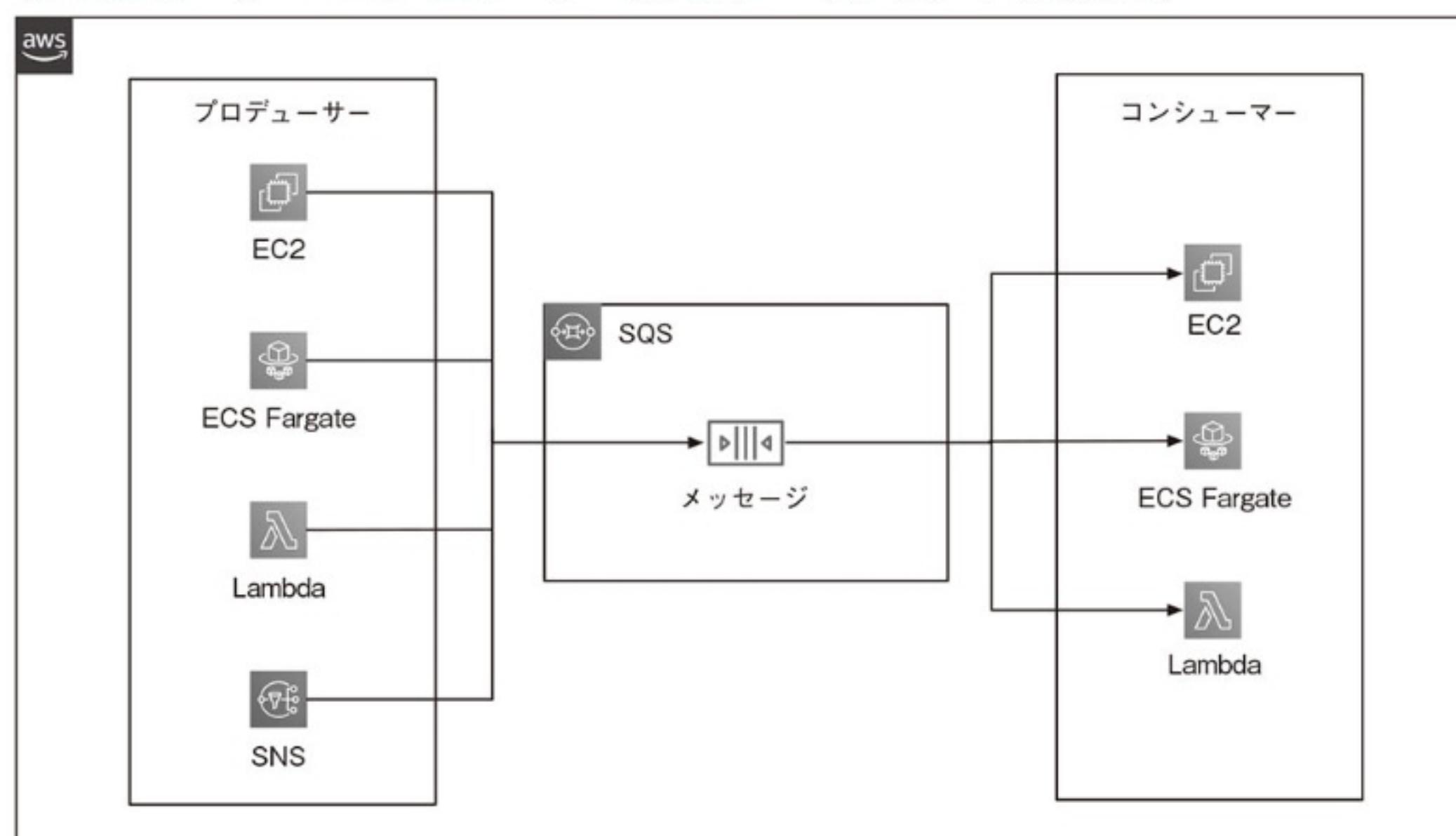
Amazon Simple Queue Service (SQS) は、ほぼ無制限のスケーラビリティを備えたフルマネージド型の分散メッセージキューサービスです。

1 SQSの概要

キューとは、主にアプリケーション間のメッセージ交換として使用されるテキストデータを一時的に保管し、順序性を持って取り出せる仕組みです。アプリケーション間でデータ通信するにはほかにもHTTPリクエストを送信したり、ファイル転送ミドルウェアを利用する方法もあります。しかし、非同期処理によりトランザクションを分離したり、ネットワーク障害が発生しても確実にメッセージを届けたりする用途で、より確実なデータ通信が求められる場合にキューが利用されます。SQSでは、複数のアベイラビリティゾーンにキューを構成できるため、可用性や耐久性に優れています。キューに対するアクセス制御やKMSを利用した保管データの暗号化が可能であるため、セキュアな分散メッセージキューを送信できます。

SQSはプロデューサー・コンシューマーモデルと呼ばれるアーキテクチャで利用されることを前提に構成されています。以下に、プロデューサー・コンシューマーモデルで構成されるアーキテクチャの概念図を示します。

【プロデューサー・コンシューマーモデルアーキテクチャの概念図】



●プロデューサー

メッセージをキューに送信するアプリケーションです。アプリケーション内でAWS SDKやCLIを利用したり、SQSのAPIを呼び出すことによりキューにメッセージを送信できます。

●コンシューマー

キューからメッセージを取得するアプリケーションです。アプリケーション内でAWS SDKやCLIを利用したり、SQSのAPIを呼び出すことによりメッセージを取得します。

2 SQSの利用

●キューの作成

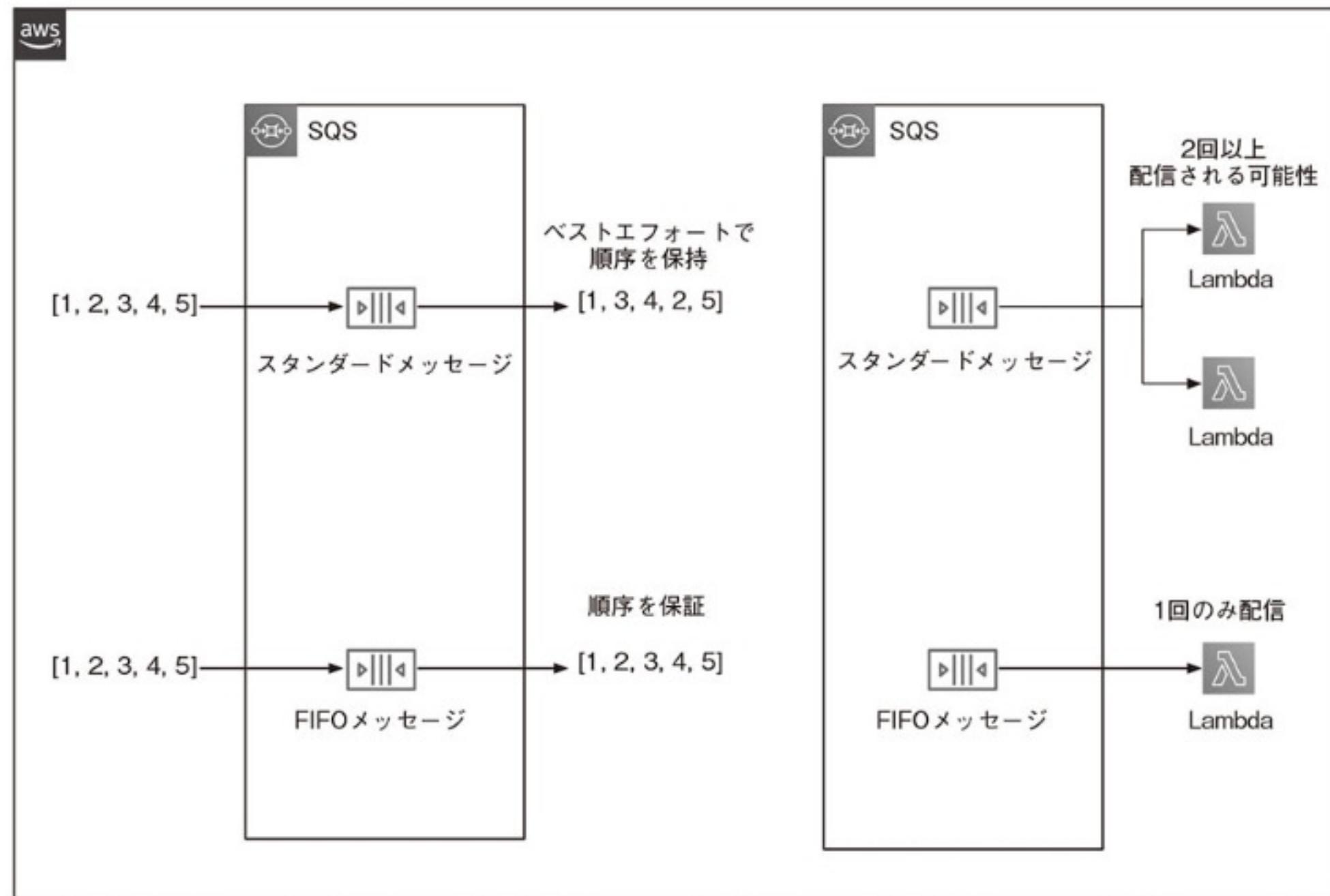
SQSを利用するにはまず、コンソールやCLIなどからキューを作成します。キューの種類は、スタンダードキューかFIFOキューの2つから選択できます。各キューの特徴は以下のとおりです。

【SQSのキューの種類と特徴】

特徴	スタンダードキュー	FIFOキュー
配信順序	概ね時系列処理されるが、順序性はベストエフォート	順序性が保証される
配信方式	少なくとも1回は配信される (2回以上配信され得る)	1回のみ配信される
スループット	ほぼ無制限のスループットを実現可能	最大300件/秒のメッセージの処理が可能

「FIFOキュー」の「FIFO」は「First In First Out」の略であり、「先入れ先出し」を意味します。メッセージをキューイングした順序での取り出しが可能です。配信も厳密に1回だけ行われます。一方で、スタンダードキューは、キューイングしたメッセージに対する順序性の保証が「ベストエフォートかつ2回以上の配信がされることがある」となっています。そのため、コンシューマー側で「幕等性（べきとうせい）」やDynamoDBで処理済みフラグを持つなど、スタンダードキューの特徴を考慮した処理を設計する必要があります。

【スタンダードキューとFIFOキューの違い】



また、キューの作成時には以下の属性を定義します。

● キュー名

キューの名前を定義します。

● 可視性タイムアウト

キューから受信したメッセージが、ほかのコンシューマーに表示されない時間の長さを設定します。「可視性タイムアウト」で詳述します。

● メッセージの保持期間

キューに残っているメッセージをSQSが保持する期間です。デフォルトでは4日間、最小の1分間から最大14日間まで保持できます。MessageRetentionPeriodパラメータとして設定されます。

● 配送遅延

キューに追加されたメッセージを配信する前にSQSが待機する時間です。「遅延キューとメッセージタイマー」で後述します。

● 最大メッセージサイズ

キューの最大メッセージサイズです。MaximumMessageSizeパラメータとして設定されます。

● メッセージ受信待機時間

キューが受信リクエストを取得した後、メッセージが使用可能になるまでSQSが待機する時間です。「コンシューマーにおけるメッセージ取得」で後述します。

● コンテンツベースの重複排除を有効にする (FIFOキューのみ)

メッセージの本文をもとに重複除外IDを作成します。「FIFOキューのメッセージ重複排除」で詳述します。

● 高スループットFIFOキューの有効化 (FIFOキューのみ)

メッセージに対し、高スループット化やスループット制限を設定します。「FIFOキューのメッセージ重複排除」で詳述します。

● 重複排除スコープ (FIFOキューのみ)

メッセージの重複を排除する範囲を指定します。「FIFOキューのメッセージ重複排除」で詳述します。

キューを作成すると、リージョンに応じたサービスエンドポイントが作成されます。アカウントIDとキュー名を組み合わせた以下の形式のURLがSQSキューのエンドポイントとなります。

```
https://sqs.<region>.amazonaws.com/<account-id>/<queue-name>
```

● プロデューサーからの送信

プロデューサーからは上述のSQSのエンドポイントに対し、SDKもしくはCLIを使ってキューを送信します。以下はCLIを使ってメッセージを送信する例です。

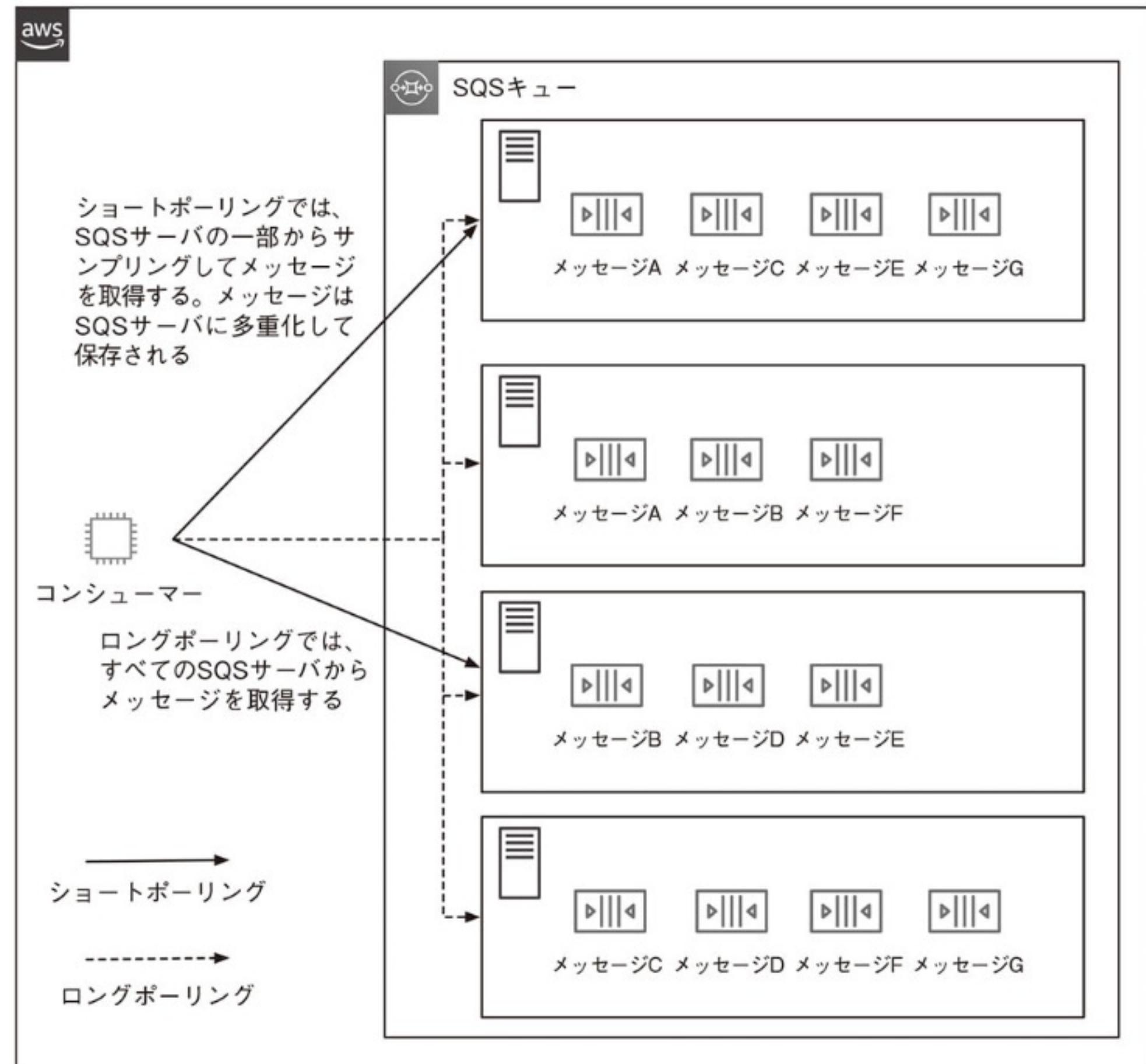
```
aws sqs send-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --message-body "Information about the largest city in Any Region." --delay-seconds 10 --message-attributes file://send-message.json
```

メッセージは本文に加え、文字列型、バイナリ型、数字型のキーバリュー属性を設定することができます。なお、バイナリ型はBase64エンコードされます。メッセージ属性は、以下のように外出しのJSONファイルとして定義することもできます。外出ししたファイルは、上記のCLIコマンド例にあるようにメッセージ送信時に指定します。

【send-message.jsonの例】

```
{
  "City": {
    "DataType": "String",
    "StringValue": "Any City"
  },
  "Greeting": {
    "DataType": "Binary",
    "BinaryValue": "Hello, World!"
  },
  "Population": {
    "DataType": "Number",
    "StringValue": "1250800"
  }
}
```

【ショートポーリングとロングポーリングのメッセージ取得方式の違い】



●コンシューマーにおけるメッセージ取得

コンシューマーからキューに対し、メッセージを取得する場合は、下記の3つのステップで実施します。

1. コンシューマーからキューに対するポーリングを行う。
2. コンシューマーがメッセージを取得して、処理を行う。
3. コンシューマーが処理済みメッセージをキューから削除する。

コンシューマーは、キューをポーリングしてメッセージのキューイング状態を監視する必要があります。ポーリングには、下記の2つの方式があります。

【SQSのキューの種類と特徴】

ポーリング方式	説明
ショートポーリング	分散配置されたSQSのサーバの中から一部をサンプリングしてポーリングを行い、すぐに応答を返す。
ロングポーリング	分散配置されたSQSの全サーバに対してポーリングを行い、最大20秒の待機時間の後に応答を返す。

ショートポーリングとロングポーリングのメッセージ取得方式の違いを図に表すと次のとおりです。

利用者の構築するシステムの特性に応じてポーリング方式を選択します。

ショートポーリング方式では、プロデューサーがキューへメッセージを追加してから、即時にコンシューマーがメッセージを取得するので、即応性が求められる処理が適していますが、キューへの確認回数（APIコール回数）が増えるため費用が高くなりやすい傾向があります。

また、メッセージを保存しているサーバに対し、一部をサンプリングすることによりメッセージ取得するため、1度のポーリングで取得できないメッセージが発生する可能性があります。上図では1つのキューに複数のサーバで分散して保存されているメッセージをショートポーリングで取得していますが、このサンプリングではメッセージFが取得されないことになります。

したがって、ある程度まとまった単位でプロデューサーがメッセージを送信しても、受け取るコンシューマー側では一部データが欠落したりする可能性があることに注意が必要です。もちろんその後欠落したデータも次のポーリングで取得されますが、取得タイミングによっては、同時に複数のコンシューマーから重複してメッセージが取得されることもあり得るので、順序性を問わず、幂等性が担保されるようにアプリケーションを設計する必要があります。

メッセージを可能な限り早めに処理しなければならないなどの制約がなければ、ロングポーリングを選択するとよいでしょう。ロングポーリングではすべてのサーバをクエリした結果を取得します。なお、ショートポーリングは前述の「キューの作成」で解説した「メッセージ受信待機時間 (ReceiveMessageWaitTimeSeconds パラメータ)」を0に設定し、ロングポーリングは0以外(1~20秒)を設定します。

ReceiveMessage APIをSDKやCLIからコールすることにより、キューに保存されたメッセージを最大10件まで取得可能です。以下はCLIでメッセージを取得する例と、アウトプットのサンプルです。

【ReceiveMessage APIをCLIから実行する例】

```
aws sqs receive-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All --message-attribute-names All --max-number-of-messages 10
```

【ReceiveMessage APIのアウトプットサンプル】

```
{
  "Messages": [
    {
      "Body": "My first message.",
      "ReceiptHandle": "AQEBzbVv...fqNzFw==",
      "MD5OfBody": "1000f835...a35411fa",
      "MD5OfMessageAttributes": "9424c491...26bc3ae7",
      "MessageId": "d6790f8d-d575-4f01-bc51-40122EXAMPLE",
      "Attributes": {
        "ApproximateFirstReceiveTimestamp": "1442428276921",
        "SenderId": "AIDAIAZKMSNQ7TEXAMPLE",
        "ApproximateReceiveCount": "5",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        },
        "City": {
          "DataType": "String",
          "StringValue": "Any City"
        }
      }
    }
  ]
}
```

```
}
```

メッセージ取得時に、WaitTimeSecondsパラメータを指定してポーリングタイプを変更することができますが、ロングポーリングのオプション指定した時間によっては、HTTPリクエストタイムアウトになるケースもあるので、使用しているHTTPクライアントライブラリの設定に注意してください。

●メッセージの削除

コンシューマーがメッセージを取得して後続処理を完了した後は、SQSキューからメッセージを削除することを忘れてはいけません。メッセージを削除する際は、上記のアウトプットに含まれる「ReceiptHandle」パラメータが必要になります。

【DeleteMessage APIをCLIから実行する例】

```
aws sqs delete-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBRXT...q2doVA==
```

後述する「可視性タイムアウト」を過ぎると、別のコンシューマーがメッセージを取得してしまう可能性があります。コンシューマーで重複処理が発生する上に、ReceiptHandleパラメータが更新されて削除できない場合が発生するので、キューからメッセージを取得した後のコンシューマーアプリケーションの処理時間を踏まえて可視性タイムアウトを設定するよう設計してください。

●遅延キューとメッセージタイマー

「遅延キュー」と「メッセージタイマー」を利用すると、プロデューサーがキューにメッセージを送信してから、コンシューマーがメッセージを取得可能となるまでの時間を一定時間遅延させることができます。

キュー全体に設定する場合は、前述の「キューの作成」で解説した「配送遅延(DelaySecondsパラメータ)」を設定します。特定のメッセージに対して設定する場合は「メッセージタイマー」を設定します。こちらはメッセージ送信時にオプションとして指定します。

遅延キューおよびメッセージタイマーには、デフォルトで0秒(遅延なし)が設定されており、最大で15分まで設定できます。遅延キューとメッセージタイマーで設定した時間が経過するまでコンシューマーは対象のメッセージを取得できません。遅延キューとメッセージタイマーの両方が設定されている場合は、メッセージタイマーが優先されます。

なお、本書の執筆時点(2023年12月)で、FIFOキューではメッセージタイマーがサポートされていません。

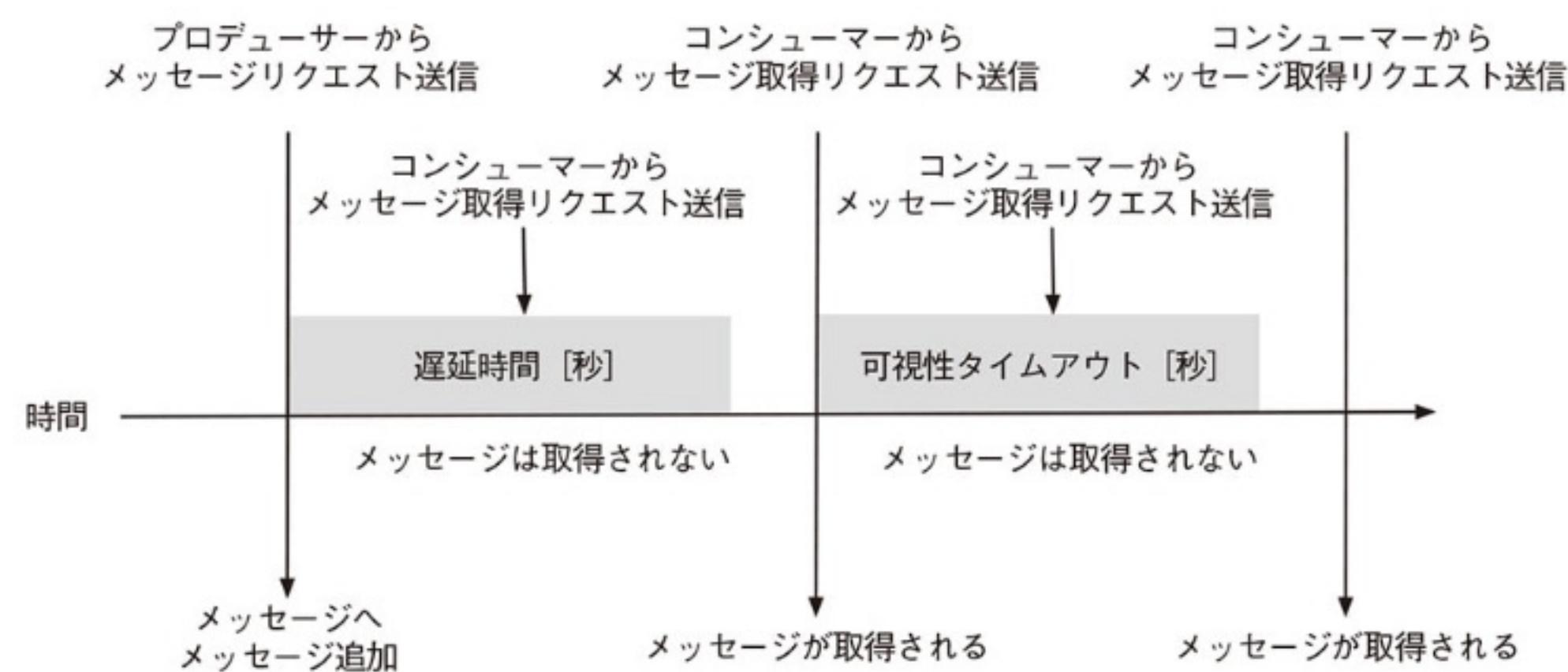
●可視性タイムアウト

前述の「キューの作成」で解説した「可視性タイムアウト（VisibilityTimeoutパラメータ）」では、あるコンシューマーがメッセージを取得してから、ほかのコンシューマーがメッセージを取得できるようにするまでの待ち時間を設定できます。可視性タイムアウトを設定することで、ほかのコンシューマーによる同一メッセージの処理を防止することができます。ただし、スタンダードキューにおいては同一メッセージを複数回処理しないことの保証とはならないため、幂等性を考慮した処理を設計しておく必要があります。

可視性タイムアウトのデフォルト値は30秒であり、0秒（待ち時間なし）から最大12時間までの値を設定できます。

遅延キュー、メッセージタイマー、可視性タイムアウトは、いずれも待機時間を設定するものです。これらの関係を下記に示します。

【「遅延キューとメッセージタイマー」、「可視性タイムアウト」の関係^{※35}】



●デッドレターキュー

「デッドレターキュー」は、正常に処理できないメッセージがキューに滞留し続けることを防止する機能です。

SQSの「Redriveポリシー」にデッドレターキューへ移動させるルールを定義します。このポリシーにメッセージの移動先となる「デッドレターキューのARN」とメッセージの「最大受信数」を設定します。メッセージ処理の失敗回数が最大受信数に設定した回数を超えた場合に、メッセージがデッドレターキューに移動します。

デッドレターキューでのメッセージの保持期間は、キューの設定に従います。ただし、メッセージがキューに追加された際のタイムスタンプが起点となる点に注意が必要です。例えば、デッドレターキューでのメッセージの保持期間がデフォルト値である4日に設定されており、移動元のキューで1日が経過した場合、デッドレターキューに格納されてから3日後にメッセージが削除されます。

※35 https://docs.aws.amazon.com/ja_jp/AWSSimpleQueueService/latest/SQSDriverGuide/sqs-delay-queues.html



2021年の12月に、AWSコンソール上からデッドレターキューに追加されたメッセージを元のキューに戻す機能が追加されました^{※36}。この機能により、アプリケーションの不具合や障害など何らかの理由でコンシューマー側で処理できなかったデッドレターキューを問題解消後に再処理することが可能になります。



デッドレターキューについて試験で問われる場合があります。デッドレターキューの設定方法をよく押さえておくようにしましょう。

●メッセージの暗号化・アクセス制御

SQSでは、3章3節 AWS KMSでも解説したとおり、SSE-KMS（KMSを使ったサーバサイド暗号化）を利用してキュー内に保管されたメッセージを暗号化できます。この際、プロデューサーとコンシューマー双方に、KMSの暗号化鍵へのアクセスが必要である点に注意してください。具体的には、プロデューサーとコンシューマーそれぞれに割り当てるIAMユーザやIAMロールに対し、KMSのアクセス権限を設定しておきます。

またSQSでは、IAMによるアクセス制御だけでなく、キューにアクセスポリシーを付与することもできます。利用者は、IAMとアクセスポリシーのいずれか、もしくは両方によるアクセス制御が行え、セキュアなメッセージキューを構成できます。



キュー内に保持するメッセージの暗号化

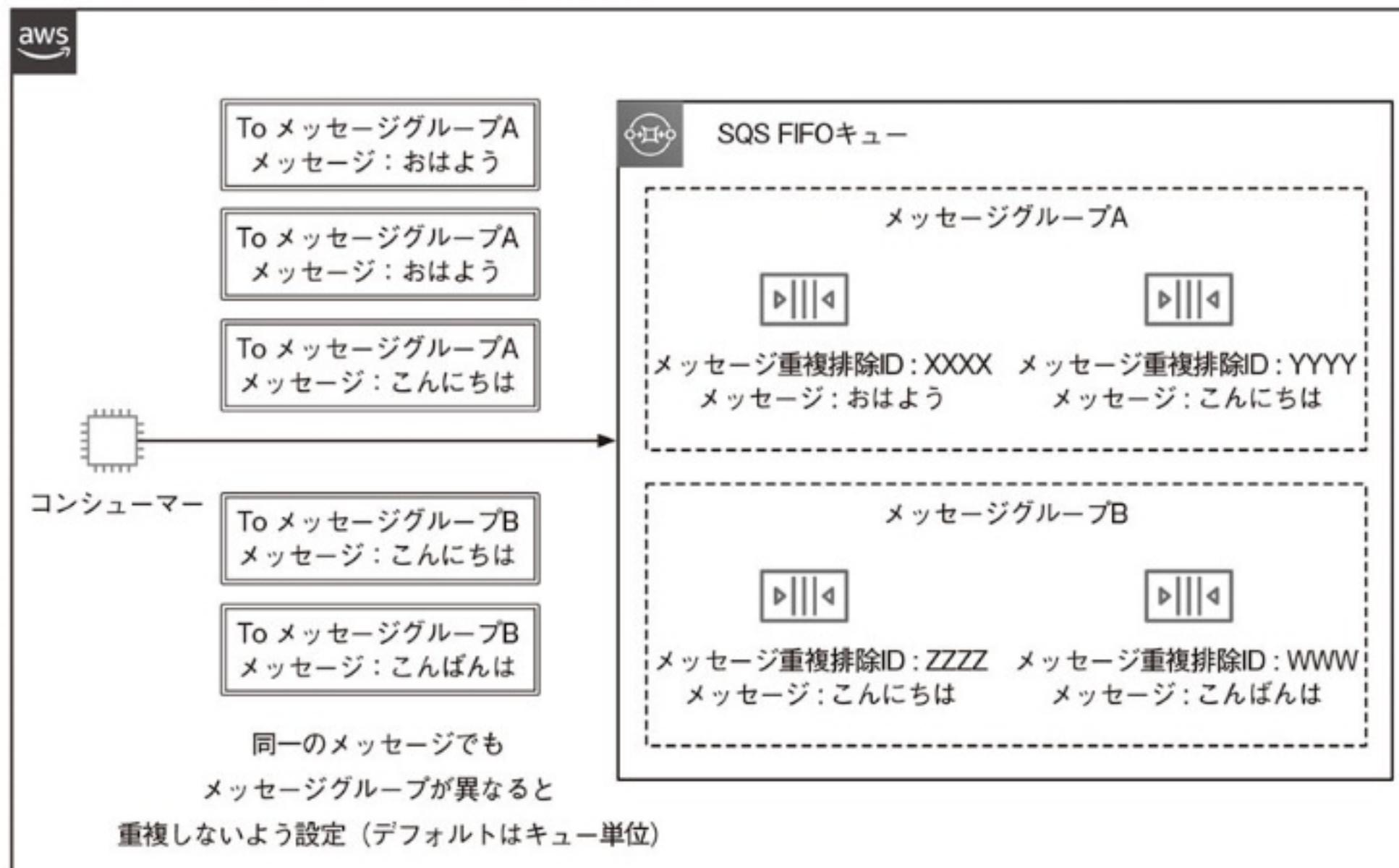
PCI DSSやFISC安全対策基準などのコンプライアンスプログラムでは、蓄積データの暗号化が要件として定められている場合があります。その場合は「SSE-KMS」を使って、キュー内に保持するメッセージを暗号化できます。

●FIFO キューのメッセージ重複排除

FIFOキューでは、メッセージの送信順序での処理が担保されるだけでなく、プロデューサーから意図しないメッセージの2重送信が行われた場合、重複を排除する機能を備えています。FIFOキューには、順序性を保持するために、「メッセージグループ」という論理的なグループが定義されており、このメッセージグループ内でメッセージの処理順序が担保されます。FIFOキューに対し、メッセージを送信するためにこのメッセージグループのID（MessageGroupId）が必要になります。重複排除は、あるメッセージが送信されてから5分間で、同一のメッセージが送信された場合に重複を排除する機能です。

※36 <https://aws.amazon.com/jp/blogs/news/enhanced-dlq-management-sqs/>

【FIFOキューのメッセージ重複排除】



上の図では、FIFOキューのメッセージ重複排除のイメージを示しています。同一のメッセージグループに対して指定された、同一のメッセージは5分間以内であれば、重複であるとみなされ、キューに蓄積されることはありません。各メッセージには一意となる「メッセージ重複排除ID (MessageDeduplicationId)」が割り当てられる必要があります。SQSでは、この「メッセージ重複排除ID」をプロデューサー側で指定して、意図的に一意のメッセージとして識別して送信することもできますし、コンテンツ (メッセージボディ) の内容にもとづいて自動的にIDを振り分けるオプションもあります。この場合、IDはSQSからUUIDを使って、割り振られます。前述の「キューの作成」で解説した「コンテンツベースの重複排除を有効にする」にて、このオプションが指定できます。

FIFOキューでは、「重複排除スコープ」を設定する必要があります。「重複排除スコープ」は「キュー」もしくは「メッセージグループ」いずれかを指定しますが、「キュー」の場合、メッセージグループをまたいでキュー内のすべてのメッセージで重複を排除し、「メッセージグループ」では、メッセージグループ単位に重複を排除します。上の図では「メッセージグループ」で設定した場合の例であり、メッセージが同じ内容でも、メッセージグループが異なれば別のメッセージ重複排除IDが割り当てられます。

また、FIFOキューではこのメッセージグループ単位でスループットを調整できます。デフォルトでは、各メッセージグループ単位で1秒あたり300リクエストがサポートされますが、キューレベルでスループット制限を適用することも可能です。SQSではメッセージグループIDのハッシュ値にもとづいて、メッセージを保存するサーバのパーティションが割り当てられます^{※37}。

※37 https://docs.aws.amazon.com/ja_jp/AWSSimpleQueueService/latest/SQSDeveloperGuide/partitions-and-data-distribution.html

5-6 Amazon SNS

Amazon Simple Notifications Service (SNS) は、マネージド型のメッセージ配信サービスです。SNSを利用すると、「Publish-Subscriber (Pub-Sub)」と呼ばれる非同期なメッセージ配信モデルを実現することができます^{※38}。

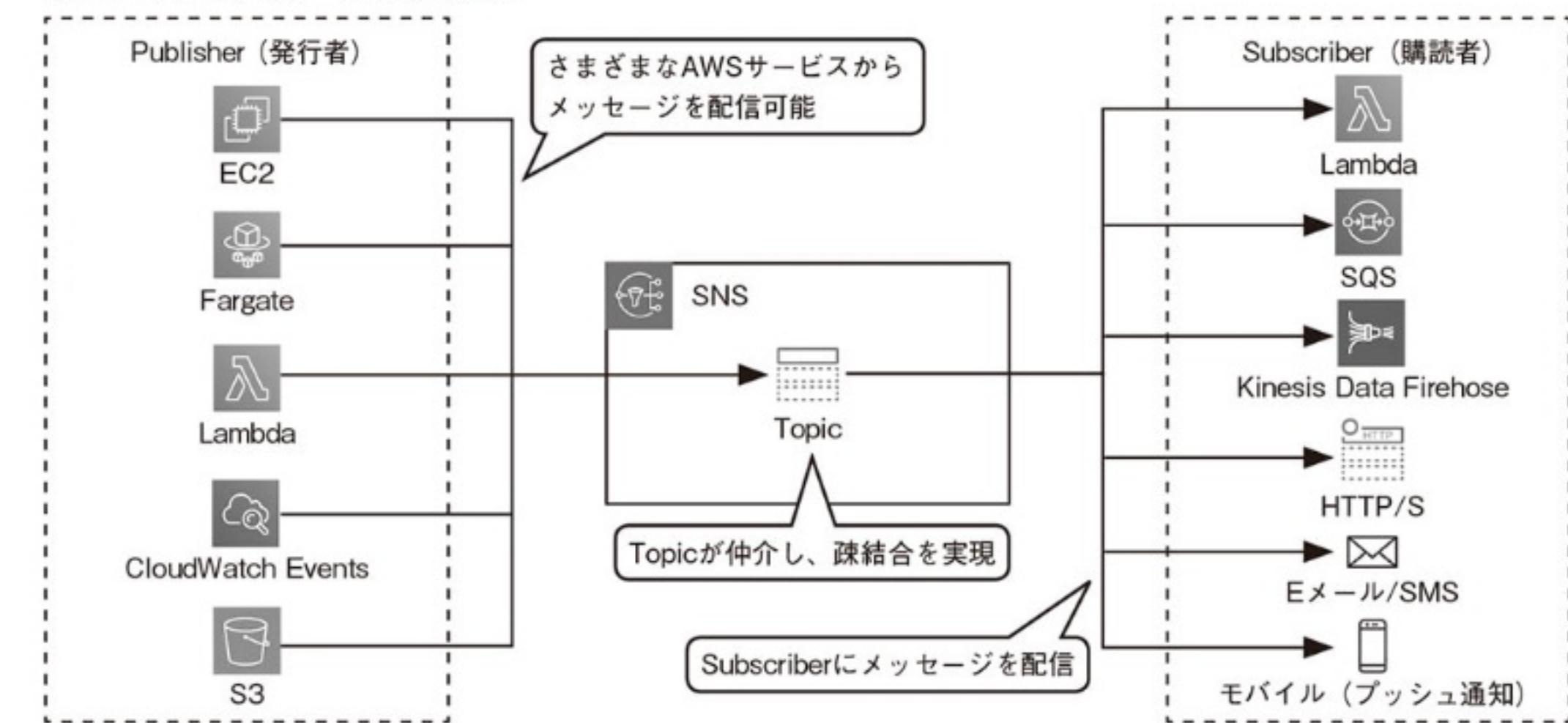
1 SNSの構成要素

Pub-Subでは、メッセージの発行元を「Publisher (発行者)」と呼びます。Publisherは「Topic」という論理的なアクセスポイントに対してメッセージを発行します。「Subscriber (購読者)」は、コンソールへのログインイベントのような興味のあるTopicをあらかじめSubscribe (購読) しておきます。SubscribeしているTopicにメッセージが格納されると、Topicからメッセージが配信されて、メッセージを受信することができます。

Pub-Subの仕組みは「メールマガジンの仕組み」を想像すると理解しやすいです^{※39}。メールマガジンの発行者と購読者がそれぞれPublisherとSubscriber、グルメ情報など特定の話題のメールマガジンがTopicに対応します。メールマガジンの発行者 (Publisher) がメールマガジンを作成してTopicにメールを発行 (Publish) すると、メールマガジンの購読者 (Subscriber) があらかじめ購読 (Subscribe) しておいたメールマガジンを受信することができます。

SNSの全体像・構成要素をまとめます。

【SNSの全体像・構成要素】



※38 PublisherとSubscriberは、それぞれProducerとConsumerと呼ばれることもあります。

※39 『技術用語を比喩から学ぼう！- 第2回「パブサブ」』(https://aws.amazon.com/jp/builders-flash/202004/metaphor-pubsub/) を参考に例え話を作成。

構成要素	説明
Publisher	メッセージの発行元。PublisherはメッセージをTopicに発行（送信）する。SMS（ショートメッセージサービス）は最大140バイト、それ以外は256キロバイトのメッセージを発行することができる。AWS SDKやAPIを用いてメッセージを発行するアプリケーションを実装することや、CloudWatch EventsやS3でのイベント発生を契機にメッセージを発行することが可能である。
Topic	通信チャネルとして機能する論理的なアクセスポイント。Topicに対して発行されたメッセージは即座にSubscriberに配信される。
Subscriber	TopicをSubscribe（購読）するエンドポイントであり、プロトコルとも呼ばれる。Subscriberには下記を設定できる。 <ul style="list-style-type: none"> ・Lambda ・SQS ・Kinesis Data Firehose ・HTTP/S ・Eメール、SMS ・モバイル（プッシュ通知）

SNSを利用することで、Publisherは論理的なアクセスポイントとなるTopicのみを意識すればよく、Subscriberは関心のあるTopicのみを購読すればよい構造となります。結果として、PublisherとSubscriberの結合強度を弱めた疎結合なシステムを構成することができます。

2 SNSのTopicの種類

SNSでは、「スタンダード」と「FIFO」の2種類のTopicが用意されています。「FIFO」は「First In First Out」の略であり、「先入先出し」を意味します。FIFO Topicを利用すると、Topicに到達したメッセージの「順序性」と「重複排除」が保証されます。

下表にそれぞれのTopicの特徴を示します。

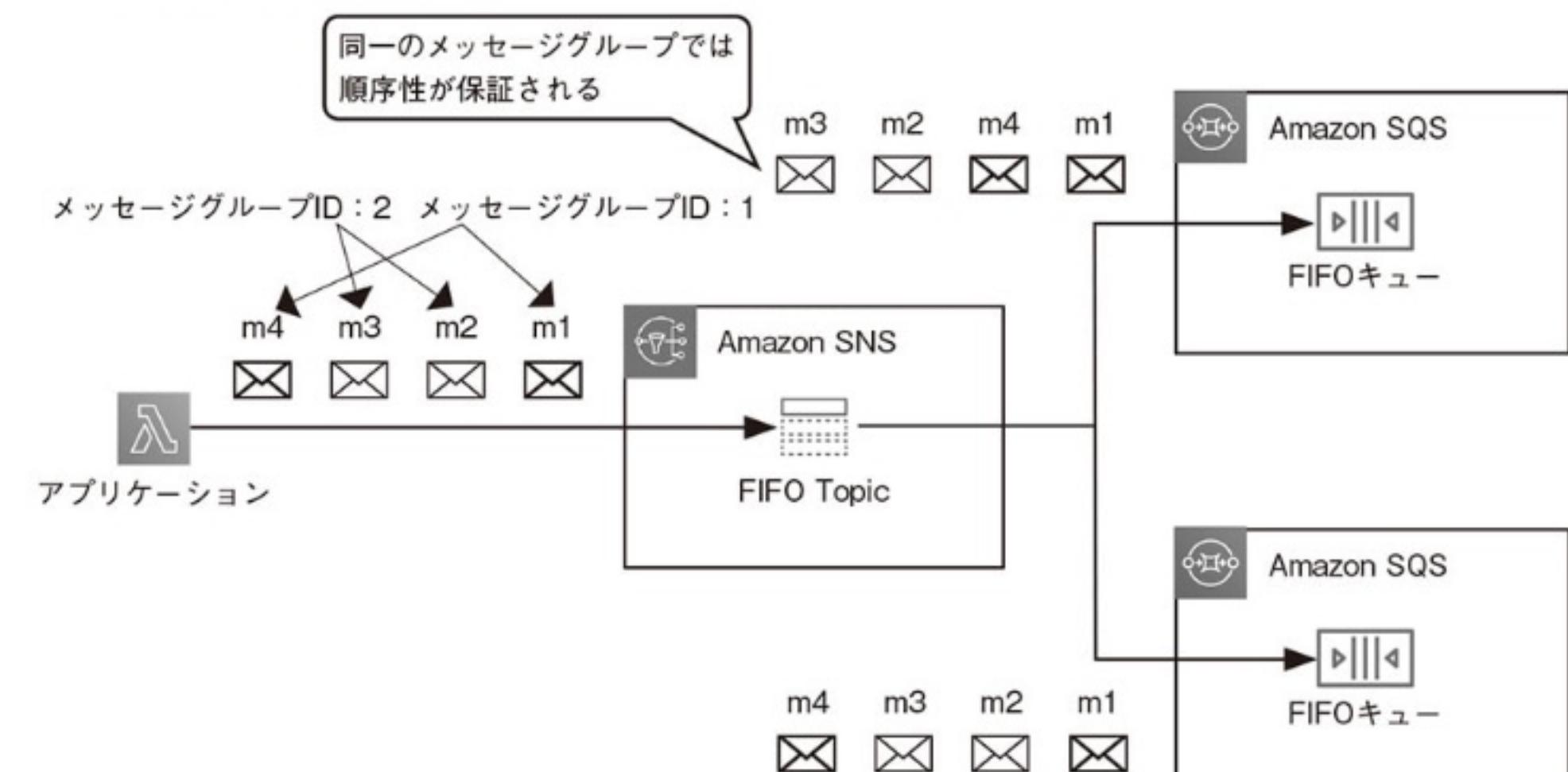
【SNSのTopicの種類と特徴】

特徴	スタンダードTopic	FIFO Topic
配信順序	順序性が保証されない	同一のメッセージグループIDにおいて順序性が保証される
配信方式	メッセージの重複の有無にかかわらず配信する	重複を排除して1回のみ配信する
スループット	ほぼ無制限のスループットを実現可能	最大300件/秒のメッセージの処理が可能

●順序性の保証

FIFO Topicでは、メッセージを発行する際に「メッセージグループID」を指定します。同一のメッセージグループでは、Subscriberへのメッセージの配信順序が保証されます。なお、メッセージのグループ分けを行いたくない場合は、すべてのメッセージに同一のメッセージグループIDを指定します。

【FIFO Topicの順序性^{※40}】



●重複排除の保証

疎結合なシステムを設計する上で気をつけないといけないことが、メッセージの重複です。アプリケーションのリトライ処理などで複数の同じメッセージがSNSに送られてしまうことがあります。銀行の取引履歴の記録のような、順序性が厳しく重複処理が許容されない業務においては、これは大きな問題です。

SNSには重複排除の仕組みが用意されており、こうした業務においても、信頼性の高いシステムを構築することができます。メッセージの重複排除を行う方法として、下記の2つの方法があります。

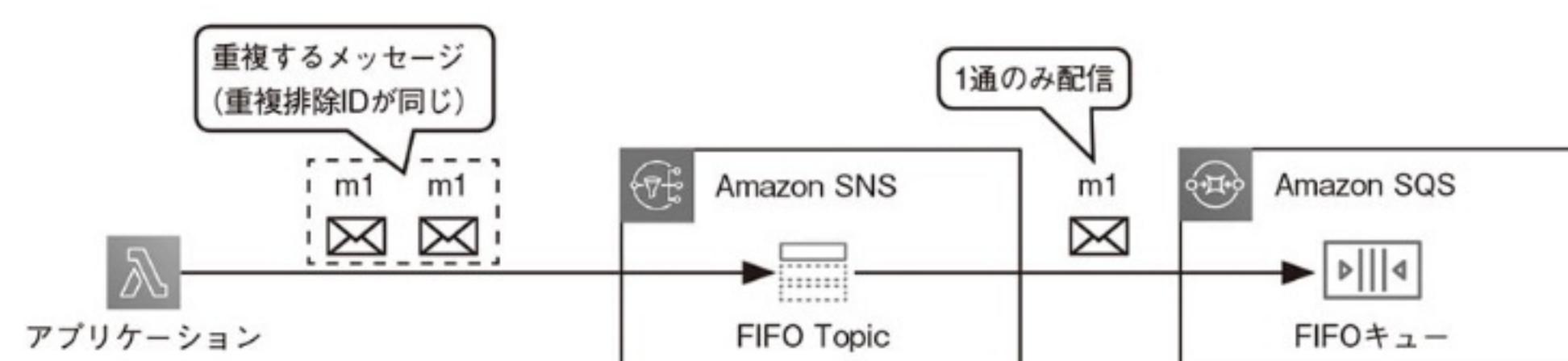
- ・コンテンツベースのメッセージ重複排除を有効化する。
- ・発行するメッセージに重複排除IDを設定する。

コンテンツベースのメッセージ重複排除では、SNSがメッセージの本文の内容（コンテンツ）をもとにSHA-256でハッシュ値を計算し、これを重複排除IDに設定します。SNSは、特定の重複排除IDを持つメッセージが正常に発行されてから5分間は同一IDを持つメッセージの配信を行わないで、これにより重複排除を実現することができます。

コンテンツベースのメッセージ重複排除を無効化した場合は、利用者が独自に重複排除IDを設定する必要があります。

※40 https://docs.aws.amazon.com/ja_jp/sns/latest/dg/fifo-message-grouping.html

【FIFO Topicの重複排除^{※41}】



3 リトライ処理

SNSでは、最初の試行でSubscriber（となるエンドポイント）への配信が正常に行われない場合は、下記の4段階のリトライポリシーに従ってリトライ処理が行われます。

1. 即時の再試行段階
2. バックオフ前段階
3. バックオフ段階
4. バックオフ後段階

「バックオフ」とは配信が失敗した場合に、線形もしくは指数関数的に間隔を徐々に増やしながらリトライ処理を行う手法のことをいいます。下表に示すように、リトライポリシーはエンドポイントによって異なります^{※42}。

なお、HTTP/Sは独自にリトライポリシーを定義して、デフォルトのリトライポリシーを上書きすることができます。

【SNSのリトライポリシー】

プロトコル	1.即時の 再試行段階	2.バックオフ 前段階	3.バックオフ 段階	4.バックオフ 後段階	合計 試行回数
SQS、 Lambda、 Kinesis Data、 Firehose	3回	1秒間隔で 2回	1秒から20秒まで 指數関数的に10回	20秒間隔で 10万回	23日間で 合計10万15回
Eメール、 SMS、 モバイル (プッシュ通知)	0回	10秒間隔で 2回	10秒から600秒まで 指數関数的に10回	600秒間隔で 38回	6時間以上で 合計50回
HTTP/S	0回	0回	20秒から60秒まで 線形に10回	0回	60秒で 合計3回

すべてのリトライ処理が実行されても配信ができない場合はメッセージが破棄されますが、SQSによる「デッドレターキュー」(5章5節)が構成されている場合はデッドレターキューにメッセージが配信されます。

4 セキュリティ

SNSでは、KMS (Key Management Service) を利用してTopicが保持するメッセージを暗号化することができます。ただし、暗号化の対象はメッセージの本文のみであり、Topicのメタデータ（トピック名と属性情報）とメトリクス、メッセージ（件名、メッセージID、タイムスタンプ、属性情報）のメタデータは暗号化の対象外である点に注意してください。

また、SNSでは、IAMによるアクセス制御に加えて、Topicに対してSNS独自のアクセスポリシーを付与することができます。利用者は、IAMとSNS独自のアクセスポリシーのいずれか、もしくは、両方によるアクセス制御を行うことができ、セキュアなシステムを構成することができます。

※41 https://docs.aws.amazon.com/ja_jp/sns/latest/dg/fifo-message-dedup.html

※42 https://docs.aws.amazon.com/ja_jp/sns/latest/dg/sns-message-delivery-retries.html#delivery-policies-for-protocols

5-7 Amazon CloudFront

Amazon CloudFront (CloudFront) は、Content Delivery Network (CDN) の機能を持つマネージドサービスです。CloudFrontをシステムで利用することで、クライアントへのコンテンツの高速配信を実現します。

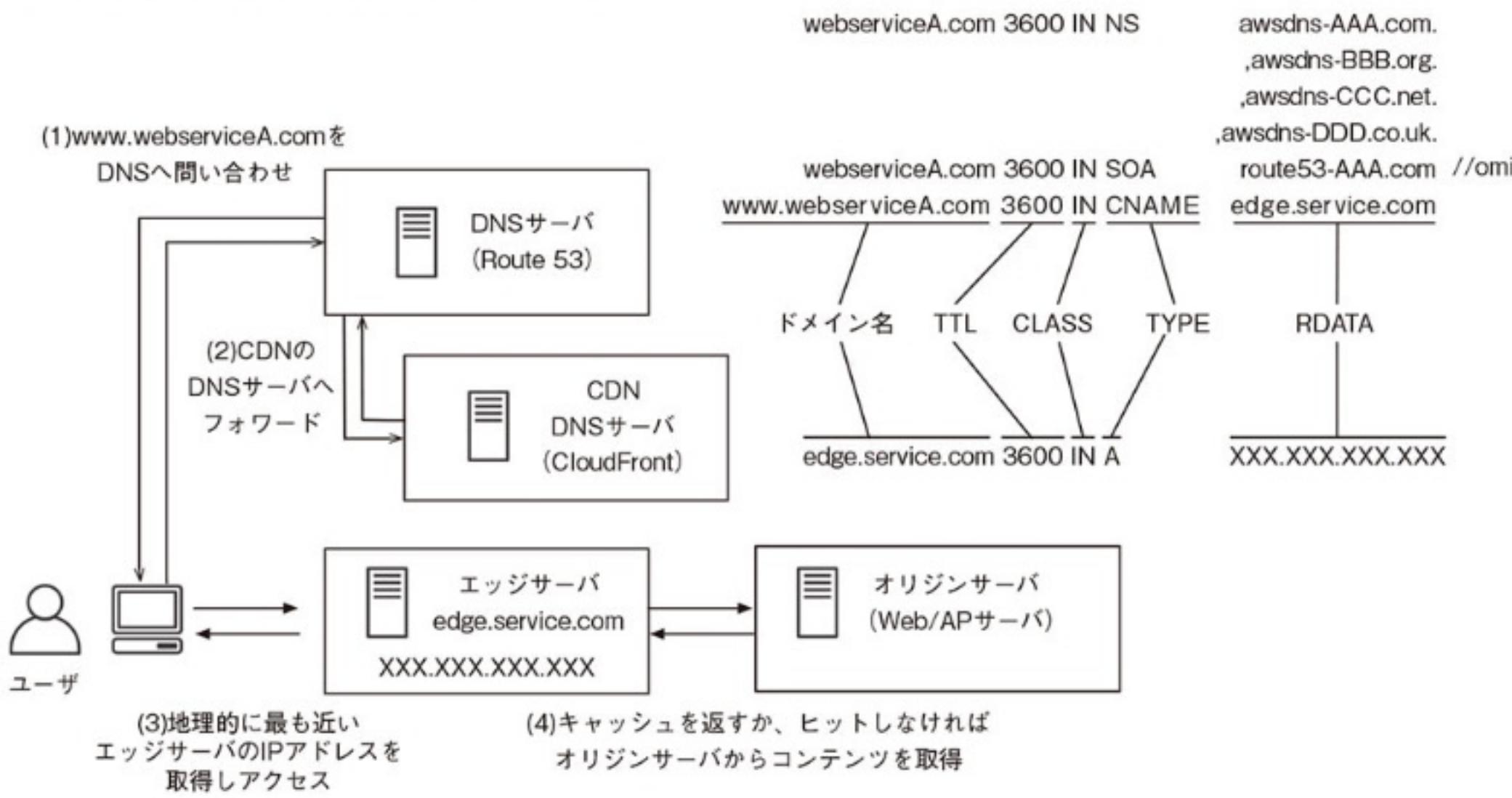
1 Content Delivery Networkとは

CDNとは、システムのサーバの負荷を下げるため、コンテンツ配信を高速化する仕組みです。CDNはオリジンサーバとエッジ(キャッシュ)サーバで構成されます。オリジンサーバとは、コンテンツを配信するサーバ（Webサーバ、APサーバなど）です。エッジサーバは、オリジンサーバのコンテンツの一部を一時的に保持（キャッシュ）するサーバです。

クライアントはシステムにアクセスする際に、まずエッジサーバにアクセスします。エッジサーバの中にリクエストされたコンテンツがあれば、そのままクライアントに返却します。もしエッジサーバにそのコンテンツがなければ、オリジンサーバからコンテンツを取得し、クライアントに配信します。このとき取得したコンテンツはキャッシュサーバに格納されます。

2章5節 Amazon Route 53「DNSの役割」および「ホストゾーン（Hosted Zone）」でDNSの仕組みとWebサービスへのアクセスについて説明しましたが、同じような図を使って、このCDNを実現するためのDNSの構成について説明します。

【CDNを実現するためのDNS構成】



例えば、`www.webserviceA.com`で、あるコンテンツを配信するとします。Route 53などのDNSサーバにはゾーンを管理するSOAレコードやNSレコードが登録されます。加えて、通常サービスを提供するオリジンサーバ（正確にはオリジンとなるWeb/APサーバのフロントに配置されているロードバランサー）が Aレコードで設定されますが、CDNを構成するには、CNAMEレコードでエッジサーバのDNSを登録します。この設定で、エッジサーバを提供するサービス（CloudFront）のDNSサーバにフォワードされ、Aレコードで登録されているエッジサーバのIPアドレス（XXX.XXX.XXX.XXX）が返されます。クライアントは返却されたIPアドレスのエッジサーバにアクセスします。このとき、クライアントへ素早く応答できるように地理的に最も近い場所にあるエッジサーバが一般的に選択されます。

2 CloudFrontの特徴と構成要素

CloudFrontはAWSが提供するCDNサービスです。AWSでは通常のリージョン以外にもエッジロケーションとして、47カ国90都市に、215以上のエッジロケーションと13のリージョン別中間キャッシュから構成されています。リージョン別中間キャッシュとは、オリジンのコンテンツ取得を軽減するために、オリジンサーバとエッジサーバの間にさらにもう1つ設置した中間エッジキャッシュサーバです。

【CloudFrontのエッジロケーション^{※43}】



※43 <https://aws.amazon.com/jp/cloudfront/features/?whats-new-cloudfront.sort-by=item.additionalFields.postDate&whats-new-cloudfront.sort-order=desc>

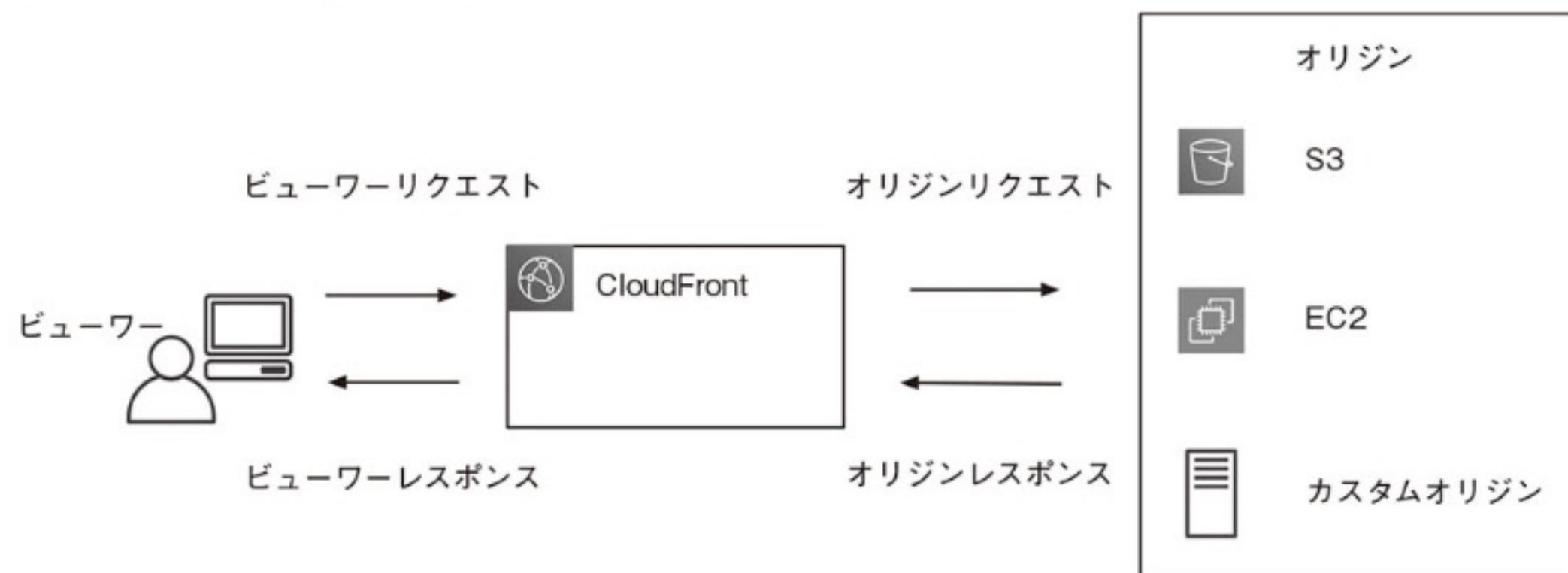
CloudFrontでは、S3のバケットに格納した静的Webサイトホスティングのコンテンツ（2章9節 Amazon S3「その他の機能」を参照）やEC2（フロントはALBなどのロードバランサー）などのAWSリソースをオリジンとして簡単に設定できます。



CloudFrontを使用して、S3に配置した静的コンテンツをキャッシングしてレイテンシを小さくする手法等が試験でたびたび問われます。高速化はエッジロケーションにより実現されていることを押さえておきましょう。

クライアントとなる対象をビューウーと呼び、ビューウーとCloudFrontとの間をビューワリクエスト・レスポンス、CloudFrontとオリジンの間をオリジンリクエスト・レスポンスとして区別します。

【CloudFrontの構成要素】

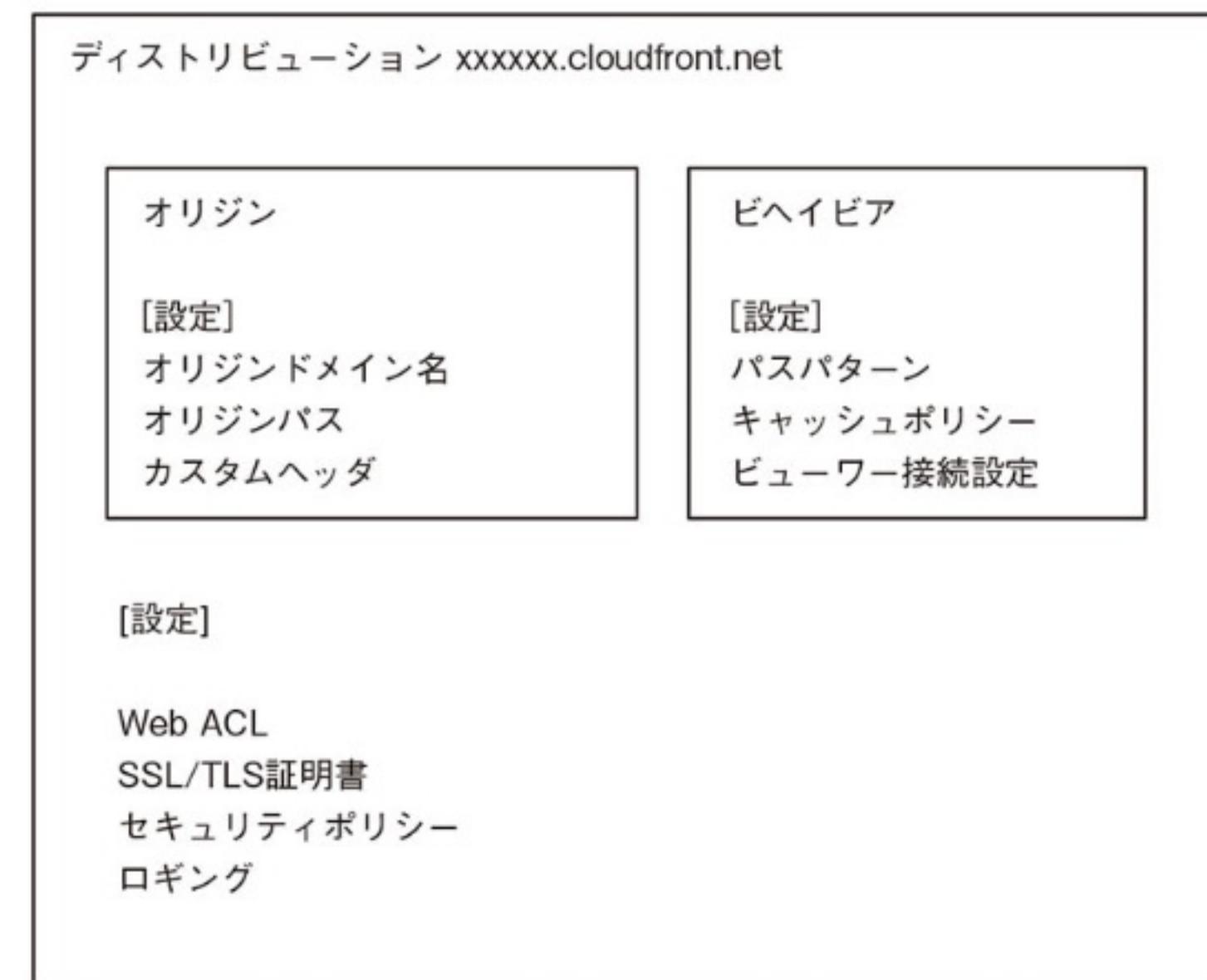


CloudFrontでは、以下の構成要素を設定します。

1. ディストリビューション (Distribution)
2. オリジン (Origin)
3. ピヘイビア (Behavior)

各構成要素の関係は以下の図のとおりです。

【CloudFront の構成要素】



●ディストリビューション

ディストリビューションは、キャッシング対象とするドメイン単位で作成する設定のことです。HTMLリソースなどの静的コンテンツを対象とした「Webディストリビューション」と動画などのストリームデータを対象とする「RTMPディストリビューション」の2タイプがあります。ディストリビューションを作成すると、エッジサーバを指し示す「xxxx.cloudfront.net」がドメイン名として生成されます。直接このドメインを指定しても、エッジサーバへアクセスはできますが、前項で示したように、DNSサーバにCNAMEレコードとして作成することで（前項の図でedge.service.comに相当）、CloudFrontが提供するCDNのDNSサーバにフォワード（転送）されるようになります。

ディストリビューションには複数のオリジンやビヘイビアを設定できます。ドメイン名に続くパスパターンに応じて、参照するオリジンを切り替えたり、キャッシング期間を変えたりすることができます。そのほか、Web ACL (Access Control List) やSSL証明書、ロギングなどエッジサーバ全体の設定もディストリビューションで行います。



CloudFrontで、AWS Certificate Manager (ACM) でのSSL証明書を利用するには「米国東部 (バージニア北部) us-east-1」で作成しておく必要があります。このリージョンに存在しない場合は、証明書を新たに作成し直すか、インポートする必要があります※44。

※44 <https://aws.amazon.com/jp/premiumsupport/knowledge-center/migrate-ssl-cert-us-east/>

●オリジン

オリジンは、オリジンサーバとの接続に関する設定を行います。オリジンサーバとはキャッシュ対象となるコンテンツの元となるデータを持つサーバのことです。オリジンには、Web/APサーバのドメイン名を直接指定したり、S3やALBなどのAWSリソースを指定できます。パスやカスタムヘッダといった設定のほか、オリジンへのアクセス頻度を最適化するOrigin Shieldなどの機能を有効化することもできます。また、S3をオリジンとする場合は、S3のバケットアクセスをCloudFrontからのみに限定してセキュリティ性を高めるOrigin Access Identity(OAI)機能も利用可能です。



2022年の8月にOAIよりも柔軟性やセキュリティを強化したOrigin Access Control (OAC) 機能がリリースされています^{※45}。

●ビヘイビア

ビヘイビアには、エッジサーバの振る舞いに関する設定を行います。オリジンに設定したパスパターンごとに、ビューアーとのプロトコル設定（ビューワープロトコルポリシー）や、キャッシングルール（キャッシングポリシー）を設定できます。このビヘイビアでキャッシングヒット率をいかに高く設定できるかが重要になります。

3 キャッシュ設定

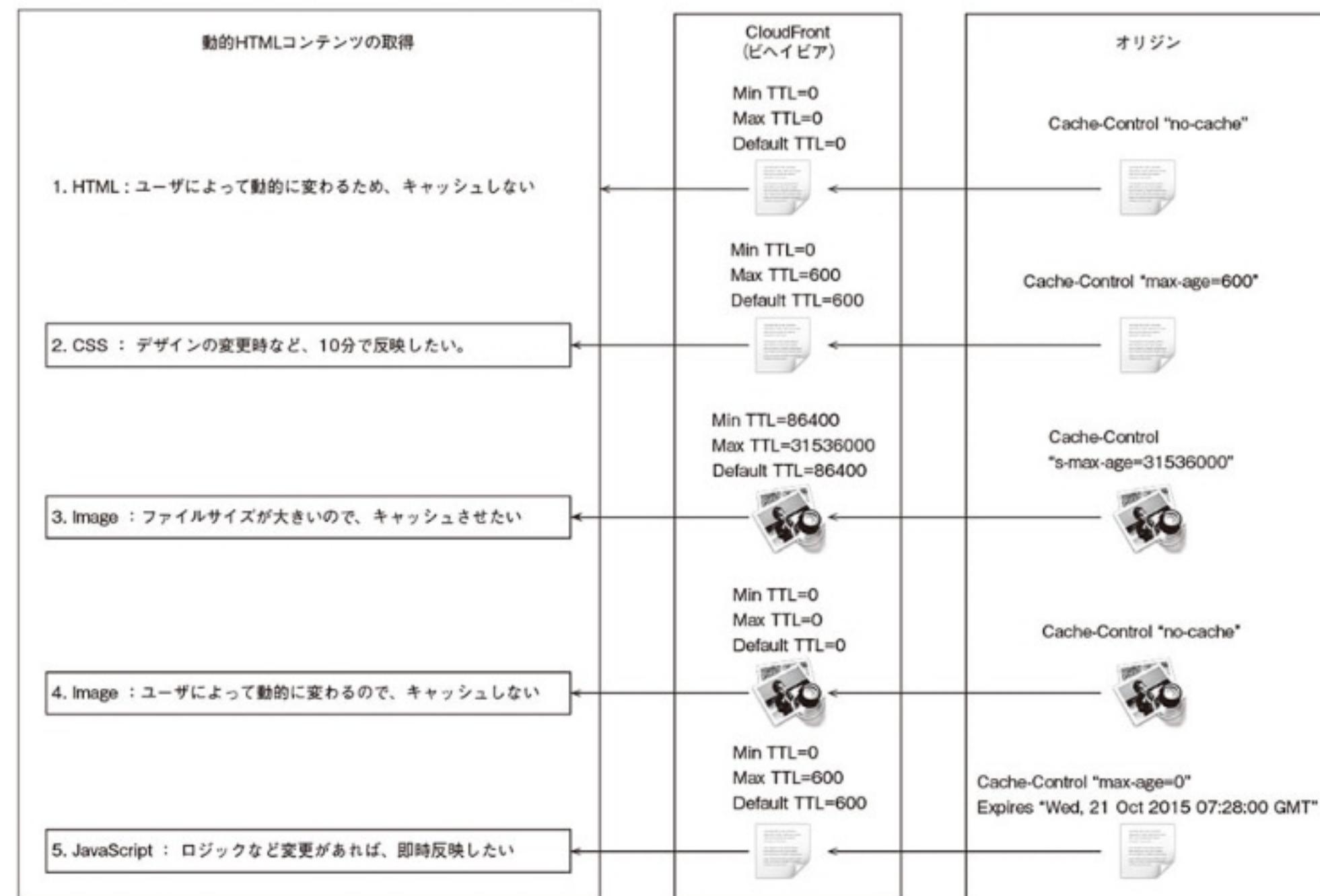
ビヘイビアにおけるキャッシング設定方法について説明します。

●キャッシングコントロール

キャッシングは、URLが一致した場合、またはオプションで指定したHTTPヘッダ、クエリ文字列、Cookieパラメータを任意に加えて完全一致した場合に再利用されるよう構成できます。こうしたキャッシングを“どう扱うか”を指示する設定のことを「キャッシングコントロール」といいます。キャッシングコントロールは通常、HTTPレスポンスヘッダの「Cache-Control」を、コンテンツを提供するオリジンサーバ側で指定することにより（指定の仕方はWebサーバやAPサーバなどオリジンで実行されているミドルウェアにより異なります）、コンテンツ単位でのキャッシングをコントロールします。加えて、CloudFrontのビヘイビアで設定したTTL（Time To Live）の値によって、キャッシングの挙動は変化します。CloudFrontでは、キャッシング期間が最小となるMin TTL、最大となるMax TTL、「Cache-Control」ヘッダがない場合に適用されるデフォルトTTLが秒単位で設定できます。

次の図では、キャッシング設定の例を示しています。動的に生成されるHTMLページのさまざまなコンテンツに対し、キャッシングするのかどうか、キャッシングする期間はどれくらいにするのかといった調整を、キャッシングコントロールで設定しています。この限りではありませんが、オリジンパスは、下記の1~5のようなカテゴリでキャッシングルールを設定できるよう、オリジンで資材のディレクトリを構成したほうがよいでしょう。

【キャッシングコントロールの設定】



1. 動的に生成されるHTMLコンテンツ

動的に生成されるHTMLコンテンツの多くは、オリジン側が持つビューアー（ユーザ）固有のデータが含まれます。そのため、オリジンからは、Cache-Controlで"no-cache"をレスポンスヘッダ指定してキャッシングしないように構成します。Cache-Controlをオリジンで指定しない場合は、CloudFrontのビヘイビアで、Min TTLとDefaultのTTLを0に設定し、キャッシングしないよう構成します。なお、オリジンからCache-Controlで"no-cache"の指定がなく、CloudFront側でMin TTLが0で設定されているにもかかわらず、デフォルトTTLが0より大きく設定されていると、キャッシングされてしまうので注意してください。

※45 <https://aws.amazon.com/jp/about-aws/whats-new/2022/08/amazon-cloudfront-origin-access-control/>

2. 変更時に10分で更新を反映したいCSSなどのコンテンツ

CSSなどビューウェー（ユーザ）固有ではなく、画面デザインの変更反映など、ある程度の時間間隔でキャッシュを更新したい場合は、オリジンからのCache-Controlで更新間隔を指定するか、CloudFrontでMin TTL=0にしてデフォルトTTLを更新間隔に設定するかたちで構成するとよいでしょう。

3. ファイルサイズが大きく、更新頻度も低いコンテンツ

データサイズが大きく、かつビューウェーにとっても同じコンテンツは、まさにCloudFrontで絶好のユースケースとなります。これらのコンテンツは、オリジン側のCache-Controlでmax-ageやs-max-ageを指定するか、CloudFrontでTTLを設定することによりキャッシュ化できます。オリジン側で設定するCache-Controlヘッダの値やCloudFrontでのピヘイビア設定値の組み合わせにより、キャッシュ期間が決まります（後述の表「オリジンで付与されるHTTPヘッダおよびCloudFrontキャッシュの設定内容とキャッシュの有効化」を参照してください）。柔軟性を持たせるためには、オリジン側でCache-Controlを指定しない場合はCloudFrontでのデフォルトTTLが有効になるよう設定します。特定のコンテンツのみキャッシュ期間を変更したい場合に限って、オリジン側で設定したCache-Controlが有効になるように構成するとよいでしょう。

4. ファイルサイズが大きいが、ビューウェーによって切り替わるコンテンツ

この種類のコンテンツをキャッシュ化すべきかはケースバイケースで異なりますが、基本的に1と同様、キャッシュ対象外としたほうがよいでしょう。Cookieパラメータ値などをキャッシュコントロールの対象に含めて、動的コンテンツも個々のビューウェーごとにキャッシュを作成し、アクセスの高速化を図ることもできます。ただしこの方法では、ビューウェーごとにキャッシュができてしまうため、キャッシュ全体の使用データ量や料金に注意する必要があります。

また、Cookieを含めずにキャッシュ化の対象としてしまうと、ほかのビューウェーにも固有のデータが見えててしまうため、設定ミスによるセキュリティ事故を誘発しかねません。安全策をとるのであれば、キャッシュ対象外にするほうがよいでしょう。画像や動画コンテンツなどサイズが大きく、どうしてもパフォーマンスを改善したいデータのみをキャッシュ対象として検討してください。

5. 変更があれば即時変更したいコンテンツ

CSSなどのコンテンツ反映と似ていますが、自開発したJavaScriptのバグなどで改修が発生することに備えて、変更を即時反映できるようにしておきたいコンテンツが一部だけオリジンパスに含まれる場合があるとします。このようなシチュエーションではパスパターンごとにキャッシュルールを設定したり、オリジン側でCache-Controlヘッダのmax-age=0を設定するか、Expiresヘッダに過去日時を設定しておくと、指定された対象のコンテンツはキャッシュが無効化されます。コンテンツにCache-ControlやExpiresヘッダがオリジン側で設定されていなければ、

CloudFrontで設定したTTLの設定値にもとづいてキャッシュが更新されます。

ただしこの方法では、オリジン側でCache-ControlやExpiresヘッダが設定されているものはキャッシュが常に無効になってしまったり、すでにキャッシュされているコンテンツは有効期限が切れるまで更新が反映されなかったりするので、有効期限内に一度だけコンテンツを最新化したい場合は、後述する「キャッシュコンテンツの無効化」を参照してください。

上記のケースを含め整理すると、オリジンで設定するHTTPヘッダとCloudFrontで設定するTTLの設定内容に従って、以下のようなロジックにもとづいてキャッシュが有効になります。

【オリジンで付与されるHTTPヘッダおよびCloudFrontキャッシュの設定内容とキャッシュの有効化】

オリジンHTTP ヘッダ設定	CloudFrontの設定	
	Min TTL=0が 設定されている場合	Min TTL > 0で 設定されている場合
Cache-Control ヘッダにmax- ageを指定する	指定されたmax-ageとMax TTL を比較してより小さい値が キャッシュ期間として有効	max-age < Min TTLの場合、Min TTLが キャッシュ期間として有効 Min TTL < max-age < Max TTLの場合、 max-ageがキャッシュ期間として有効 Max TTL < max-ageの場合、Max TTLが キャッシュ期間として有効
Cache-Control ヘッダにmax- ageとs-maxage を指定する	指定されたs-maxageとMax TTL を比較してより小さい値が キャッシュ期間として有効	s-maxage < Min TTLの場合、Min TTLが キャッシュ期間として有効 Min TTL < s-maxage < Max TTLの場合、 s-maxageがキャッシュ期間として有効 Max TTL < s-maxageの場合、Max TTLが キャッシュ期間として有効
Cache-Control 設定がない	デフォルトTTLがキャッシュ期 間として有効	Min TTLまたは、デフォルトTTLで大き い値がキャッシュ期間として有効
Expiresヘッダ に日付を指定	指定されたExpiresの日付と Max TTLを比較してより小 さい値がキャッシュ期間として 有効	Expiresの日付 < Min TTLの場合、Min TTL がキャッシュ期間として有効 Min TTL < Expiresの日付 < Max TTLの場 合、Expiresの日付がキャッシュ期間と して有効 Max TTL < Expiresの日付の場合、Max TTL がキャッシュ期間として有効
Cache-Control に"no-cache" もしくは "no-store"を 指定	キャッシュは無効	Min TTLがキャッシュ期間として有効



S3がオリジンとなるコンテンツでCache-ControlやExpiresヘッダを指定した場合は、S3オブジェクトのメタデータを使って設定します。また、HTMLファイルのMetaタグに直接Cache-ControlやPragmaを指定しても、CloudFrontは利用しません。CloudFrontのビヘイビアから設定するようにしてください。

●キャッシュポリシーとオリジンリクエストポリシー

キャッシングポリシーとオリジンリクエストポリシーは2020年の7月に発表された比較的新しい機能ですが、すでにビヘイビアへの適用が推奨されています。それまでは、ビヘイビアで上記のキャッシングコントロールをパスパターンごとに1つずつ設定する必要がありました。この設定ではキャッシングルールをポリシー化してアタッチするかたちになります。

加えて、従来では、オリジンに転送するリクエストにヘッダやCookieなど含めて転送していた場合、それらが無条件でキャッシングキーとして扱われていましたが、キャッシングポリシーとオリジンリクエストポリシーが分離されたことで、キャッシングとして扱われるキーと、オリジンに転送するリクエストを分けて設定することが可能になります。なお、キャッシングポリシーに指定したパラメータはオリジンリクエストに自動的に含まれます。

●キャッシングコンテンツの無効化

CloudFrontでは、有効期限が切れる前にキャッシングされているコンテンツを無効化することもできます（Invalidation）。コンソール上やCLIからファイルを指定して、キャッシングを無効化します。ワイルドカードを使用して複数のファイルを同時に無効化することもできます。無効化後は再びオリジンから最新バージョンを取得します。オリジンからCache-Controlヘッダを設定しない場合は、この方法でもキャッシングを最新化できます。



Invalidationを使用せずになるべくコンテンツを最新化したい場合は、ファイル名やディレクトリ名にバージョン情報を付与してリンクを更新する方法もあります^{*46}。Invalidationオペレーションは有料であるため、有効期限内に大量にコンテンツのキャッシングを更新する必要がある場合は、こちらの方法も検討してください。



有効期限が切れる前に、コンテンツを無効化してキャッシングを最新化する方法などを問われる場合があります。キャッシングを最新化する方法をよく理解しておきましょう。

*46 https://docs.aws.amazon.com/ja_jp/AmazonCloudFront/latest/DeveloperGuide/UpdatingExistingObjects.html#ReplacingObjects

4 セキュリティとその他の機能

●通信の暗号化

CloudFrontでは、ビューワー↔CloudFront間の通信は、基本的に暗号化して通信します。暗号化に使用する証明書は、cloudfront.netドメインのSSL/TLS証明書を標準で利用できるほか、AWS Certification Managerで発行された独自の証明書を設定することも簡単に行えます。また、CloudFront↔オリジン間通信の暗号化も可能です。S3やALBなどのAWSリソースへはデフォルトで暗号化され、カスタムオリジンでもオプションで選択できます。

●署名付き URL/Cookie

CloudFrontでは、ビューワー接続設定で「制限付きアクセスオプション」を選択することで、署名付きURL/Cookieを利用した制限付きアクセスが可能になります。署名付きURLとCookieはどちらも同じく、ビューワーに一時的なアクセス許可を付与します。URLは単一のコンテンツに対するアクセス制御になりますが、Cookieはそれを持つビューワーが有効期間中アクセスできます。

第2章9節Amazon S3「セキュリティ保護」でも署名付きURLを解説しましたが、CloudFrontの方式はS3のものとは実装方式が少々異なります（署名自体を作成する実装は同じです）。CloudFrontで署名付きURL/Cookieを利用する準備として、まずローカル端末などで、公開鍵・秘密鍵のキーペアを作成します。署名者として作成された「信頼されたキーグループ」に対し、公開鍵を追加します。



署名者には、ルートユーザーアカウントを使ってキーペアを生成する方法もありますが、「信頼されたキーグループ」を使用する方法が推奨されています^{*47}。

対となる秘密鍵を使って、SDKを使ってアプリケーションの処理などで署名付きURL^{*48}やCookie^{*49}を生成します。署名の文字列の中には、CloudFrontのリソースへのアクセスポリシーが含まれます。

```
{
  "Statement": [
    {
      "Condition": {
        "StringLike": {
          "CloudFrontResourceType": "Object"
        }
      }
    }
  ]
}
```

*47 https://docs.aws.amazon.com/ja_jp/AmazonCloudFront/latest/DeveloperGuide/private-content-trusted-signers.html

*48 https://docs.aws.amazon.com/ja_jp/AmazonCloudFront/latest/DeveloperGuide/private-content-signed-urls.html

*49 https://docs.aws.amazon.com/ja_jp/AmazonCloudFront/latest/DeveloperGuide/private-content-signed-cookies.html

```

    "Resource": "http://d111111abcdef8.cloudfront.net/
horizon.jpg?size=large&license=yes",
    "Condition": {
        "DateLessThan": {
            "AWS:EpochTime": 1426500000
        }
    }
}
]
}

```

アクセスポリシーは接続元のIPアドレスなどを制限したカスタム条件を含めることもできます^{※50}。署名付きURL/Cookieは有効期間内であれば、誰でもアクセスすることができます。有効期限を短めに設定するか、接続条件を厳しくするなど、セキュリティ対策を施した上で利用するようにしてください。

●フィールドレベルの暗号化

フィールドレベルの暗号化は、クレジットカード番号やセキュリティ秘匿データなどサーバ側のログに残さないようにする場合に有効な機能です。ビューアーからのPOSTリクエストに含まれる特定のリクエストパラメータを暗号化します。暗号化には、前述の署名付きURLと同じく、信頼されたキーグループに設定した公開鍵を使用します。プロファイルを作成して設定しておくと、指定したPOSTリクエストのパラメータがCloudFrontで暗号化されます。暗号化したパラメータはオリジン側のアプリケーションでSDKを用いて復号して利用します。

●AWS WAF や AWS Shield との連携

インターネット外部からの脆弱性を狙った攻撃や膨大なトラフィックを送信する攻撃に対して、AWS WAF (Web Application Firewall) で定義したWeb ACL (アクセスコントロールリスト) と、DDoS攻撃を緩和するAWS ShieldをCloudFrontディストリビューションに適用できます。

●ファイル圧縮

CloudFrontは、Amazon S3オリジンとカスタムオリジンの両方のファイルをGzip/Brotli形式で圧縮してビューアーに送信できます。コンテンツを圧縮するとファイルが小さくなるため、ダウンロード時間を短縮できます。場合によっては、元のサイズの4分の1以下になることがあります。キャッシュポリシーのTTL値が0より大きい値に設定されている必要があります。TTL値が0に設定されていると、キャッシュは無効となり、CloudFrontによるコンテンツの圧縮は行われません。

※50 https://docs.aws.amazon.com/ja_jp/AmazonCloudFront/latest/DeveloperGuide/private-content-creating-signed-url-custom-policy.html

●オリジンフェイルオーバー

オリジンは複数作成でき、プライマリオリジン・セカンダリオリジンをまとめたオリジングループを構成できます。オリジングループでは、5XX系など指定したHTTPステータスエラーが発生した際に、別のオリジンへフェイルオーバーするよう構成することもできます。オリジン側の障害を想定して、セカンダリには静的なsorryページなどを設定しておくとよいでしょう。

●地域制限（ホワイトリスト/ブラックリスト）

CloudFrontではビューアーのIPアドレスなどから地域情報を判断しアクセス制御できます。ホワイトリストもしくはブラックリスト形式で指定可能です。アクセス制御はディストリビューション全体に適用されます。

●Lambda@Edge

Lambda@EdgeはCloudFrontのエッジサーバで動作するLambda関数です。コンテンツの整形や、オリジンサーバが持つビューアー固有の情報を使用しない動的なページの生成など、ユーザエクスペリエンスを向上させる処理を実行します。Lambda@Edgeでは、ビューワーリクエスト、オリジンリクエスト、オリジンレスポンス、ビューワーレスポンスにそれぞれ関連付けてLambda関数を実行できます。以下は各タイミングで実行する処理の例です。

●ビューワーリクエスト

ビューアーからCloudFrontがリクエストを受信したときに実行される処理です。A/Bテストなどで異なるオリジンにURLを書き換える場合や、Authorizationヘッダに含まれている認証トークンの検証・アクセス制御、User-Agentに応じた画像コンテンツのURLの切り替えなどを行えます。

●オリジンリクエスト

CloudFrontからオリジンへ転送する際に実行される処理です。CloudFront-Viewer-Countryヘッダの値を使用して、S3バケットのドメイン名を、ビューアーに近いリージョンのバケットのものに更新したりできます。

●オリジンレスポンス

オリジンからCloudFrontがレスポンスを受信したときに実行される処理です。レスポンスに含まれる画像イメージをリサイズしてオリジンに保存したり、レスポンスステータスコードを変更したりできます。

●ビューワーレスポンス

CloudFrontがビューアーにレスポンスを返す前に実行される処理です。Cache-Controlなどレスポンスヘッダの値を更新したりできます。



2021年の5月にLambda@Edgeのさらにフロントで軽量な処理を実行できるCloudFront Functionsがリリースされています。同じくエッジ環境でJavaScriptで実装された関数を実行できる機能ですが、フロントで処理される分高速なレスポンスが期待される反面、実行時間やメモリがLambda@Edgeと比べ制約があります。より軽量な処理を高速に実行したい場合に向いているサービスです。

●レスポンスヘッダポリシー

CloudFrontでは、ビューワーレスポンスの返却時にHTTPレスポンスヘッダを追加する設定機能が2021年の11月からサポートされています^{※51}。CORS (Cross Origin Resource Sharing) やCache-ControlヘッダなどCloudFrontサイドで簡単に追加することが可能です。

●レポートとモニタリング機能

CloudFrontにおける、キャッシュ統計情報やアクセス利用状況などのレポート情報をCloudFrontコンソールから確認することができます。アクセスや利用状況傾向の確認および分析に利用することができます^{※52}。

5-8 Auto Scaling

Auto Scalingは、負荷の上昇や故障などを検知して自動的にリソースを増減させる、クラウドならではの機能の一つです。EC2を対象とするEC2 Auto Scalingと、ECSやLambda、DynamoDBなどのさまざまなAWSサービスを対象とするApplication Auto Scalingがあります。

1 EC2 Auto Scalingの概要と構成要素

EC2 Auto Scalingを利用することで、VPC上に配置したEC2インスタンスを需要に応じて自動的に増減できます。同時に、異常なインスタンスを発見すると、切り離して新しいものに交換するといった処理も自動で行えます。これにより、コストの最適化と可用性の向上を図ることができます。

EC2 Auto Scalingの設定を行うには、以下の構成要素とその関係性を理解しておく必要があります。

●Auto Scaling Group

Auto Scalingの設定の単位であり、以下のようなスケーリングに関わる全般設定を定義したグループです。

- ・起動するインスタンスを配置するVPCおよびサブネット
- ・インスタンス配置数の最小値と最大値および希望値であるDesired Capacity
- ・Scaling Plan（複数設定可能）
- ・ヘルスチェックの方法

●Launch Configuration/Launch Template

Auto Scaling Groupに関連付けられたインスタンスの起動ルールを定めた設定です。1章3.4節「Amazon EC2 (Elastic Compute Cloud)」で解説した「インスタンスの起動フロー」の設定内容とはほぼ同一の内容です。なお、同節で解説した起動テンプレート（Launch Template）を使用することも可能です。

●Scaling Plan

インスタンスをスケールするルールを設定します。Scaling Planは複数種類があり、Auto Scaling Groupへ複数設定することができます。

※51 <https://aws.amazon.com/jp/blogs/news/amazon-cloudfront-introduces-response-headers-policies/>

※52 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AmazonCloudFront-Monitoring-Logging_0801_v1.pdf