章末問題

Q1 Web サーバとして利用している EC2 インスタンスの標準メトリックスとして正しい選択肢はどれか？

- ○ **A** メモリ使用率
- ○ **B** Web ページへのロード時間
- ○ **C** Web サーバのプロセス／スレッド数
- ○ **D** Netowrk I/O

答え

A1 D

EC2 インスタンスの標準メトリックスは、ハイパーバイザが取得できる値で、Network I/O などがあります。これに対し、メモリの使用率は OS で収集する必要があります。

第 9 章

AWS における拡張性と分散／並列処理 (ELB／Auto Scaling／SQS／SWF)

AWS における拡張性と分散／並列処理について

AWS の特徴の 1 つに、伸縮自在性／柔軟性（アジリティ）があります。AWS では、必要なときに必要なだけ IT リソースを用意し、必要が無くなれば解放することで、コストメリットが図れるほか、経営判断から開発／サービス提供開始までの時間を短縮することができます。

そして、この特徴を活かす上で欠かせないのが、本章で紹介する ELB、Auto Scaling、SQS、SWF といったサービス／機能です。これらのサービス／機能は、7 つのベストプラクティスを実践する上でも欠かせないもので、認定試験において様々な分野にまたがって出題されます。

本章では、AWS における拡張性と分散／並列処理について、そこに関わるサービス／機能である ELB と Auto Scaling、SQS そして SWF を中心に説明します。

9-1 密結合と疎結合

図 9-1-1 のように、Web －アプリ－ DB の 3 層構成システムを考えます。ここでは、負荷や冗長性を考慮して、Web サーバとアプリケーションサーバをそれぞれ 3 台ずつと、DB サーバをマスター－スレーブ構成として設計したとします。

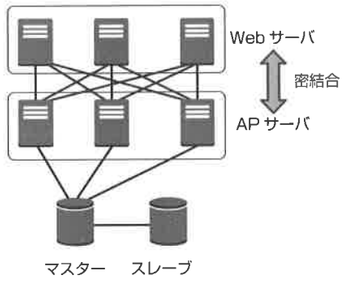

図 9-1-1　Web －アプリ－ DB の 3 層密結合構成

オンプレミスでこのようなシステムを設計する場合、各サーバは、それぞれ可用性を高める工夫がされており、極力ダウンすることがないように設計されています。各サーバの台数や IP アドレスは固定されていることが多く、各サーバは接続先のサーバの IP アドレスを登録し、お互いに密接に関係しています。このような各層のサーバの結びつきが強い構成を**密結合**の構成といいます。

一方、AWS クラウドでは、7 つのベストプラクティスの 1 つの「故障に備えた ”設計” で障害を回避」にあるように、1 台 1 台のサーバ（EC2 インスタンス）の可用性を高めるのではなく、システムとして（構成設計側）で可用性を高めることを考えます。EC2 インスタンスはすぐに新しいものを調達できるので、図 9-1-2 のように、1 台 1 台の EC2 インスタンスは障害が発生してもすぐに新たな EC2 インスタンスに入れ替わります。また、システムの負荷に応じて台数を増減することで、コストメリットを図ることができます。こちらも 7 つのベストプラクティスの 1 つの「伸縮自在性を実装」の考え方になります。

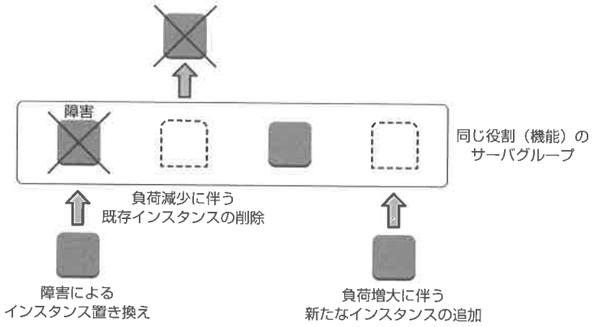

図 9-1-2　EC2 インスタンスの入れ替わりと増減

このように、システム内のサーバが入れ替わったり、増減したりする場合、密結合の構成では運用管理の負担が非常に大きくなり、システムを持続させることが困難になります。そこで、システムの各層の間に負荷分散機能を導入し、各層の結びつきを緩めることを考えます。

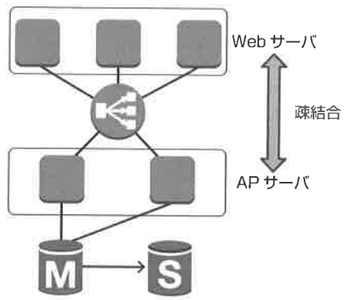

図 9-1-3　Web －アプリ－ DB の 3 層疎結合構成

すると、図 9-1-3 のように、Web 層のサーバは負荷分散装置にトラフィックを送るだけでよく、アプリ層のサーバは負荷分散装置の配下に配置し、負荷分散装置からのトラフィックを受け取るだけでよくなります。たとえ、各サーバが入れ替わったり、増減したとしても、Web サーバは負荷分散装置にトラフィックを送るということ、アプリケーションサーバは負荷分散装置の

配下に配置するという２点を自動化できていれば、システムを効率的に持続させることができます。このような、各層のサーバの結びつきが弱い構成を**疎結合**の構成といい、７つのベストプラクティスの１つである「コンポーネント間を疎結合で柔軟に」の実践になります。

つまり言い換えると、７つのベストプラクティスであり、AWS の特徴／メリットである「故障に備えた設計で障害を回避」や「伸縮自在性を実装」を実現するためには、「コンポーネント間を疎結合で柔軟に」の実践が必要になります。この重要な「疎結合」を実現する上で、この３層構成のシステムでは負荷分散装置が重要になりますが、この負荷分散装置を利用者自身で構築する場合、冗長性の確保や通信／処理のボトルネックとならないように設計／構築することが運用管理の非常に大きな負荷になってきます。AWS では、そういった負荷分散装置の運用管理の手間を AWS 側に任せることができる、Elastic Load Balancing（以下 **ELB**）というマネージド型の負荷分散サービスを提供しています。

重要！
コンポーネント間を疎結合にして伸縮自在性を実装し、AWS のメリットを活かすシステム構成にする！

9-2 ELB

ELB は、マネージド型の負荷分散サービスで、受信したトラフィックを ELB の配下に配置された複数の EC2 インスタンスに分散します。AWS 側で冗長性を確保しており、負荷に応じて自動的に拡張し、通信／処理のボトルネックにならないように設計されています。

ELB には、次のような機能があります。

- 複数の AZ にまたがる負荷分散
- EC2 インスタンスのヘルスチェック
- ELB 自体の自動スケーリング
- SSL のオフロード

- Connection Draining
- アクセスログ記録
- スティッキーセッション

(1) 複数の AZ にまたがる負荷分散

ELB では、図 9-2-1 のように、受信したトラフィックを複数の AZ にまたがって負荷分散することができます。これにより、複数の AZ に EC2 インスタンスを冗長的に配置し、可用性の高いシステムを構築することができます。これは、７つのベストプラクティスの１つである「故障に備えた設計で障害を回避」を実現する重要な要素です。

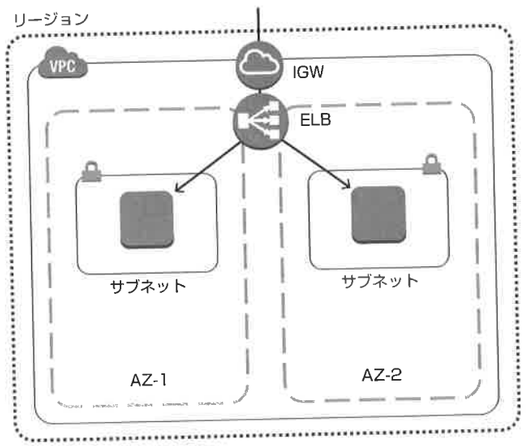

図 9-2-1　ゾーンをまたいだ負荷分散

(2) EC2 インスタンスのヘルスチェック

ELB は、配下に配置されている EC2 インスタンスのヘルスチェックを定期的に実行しています。そして、図 9-2-2 のように、異常を検知するとそのインスタンスに対してはトラフィックの分散をやめ、残りの正常なインスタンスにのみトラフィックを分散します。その際、ELB は異常を検知したインスタンスの分析や復旧作業などは行いません。

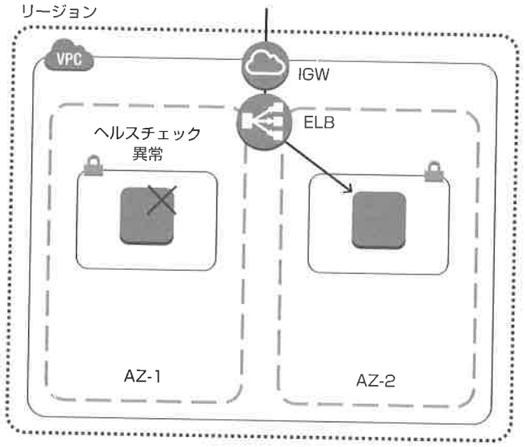

図 9-2-2　ヘルスチェックと負荷分散

補足 ELB の配下に配置している EC2 インスタンスを再起動した際、ELB のヘルスチェック間隔によってはインスタンスの異常と判定され、そのインスタンスへのトラフィックの分散が行われなくなります。以前は、EC2 インスタンスが再起動後に正常に戻っても、その EC2 インスタンスへのトラフィックの送信を再開しませんでしたが、2015 年 12 月に EC2 インスタンスの ELB への自動再登録が可能になりました。

(3) ELB 自体の自動スケーリング

ELB は、受信するトラフィックの流量に合わせて、自動的にその実体を増減させます。ELB を作成すると、図 9-2-3 のように、「example.ap-northeast-1.elb.amazonaws.com」といった DNS 名が作られ、ELB の実体が持つ IP アドレスと関連付け（名前解決）されます。負荷に応じて ELB の実体が増えれば、DNS 名と新たな IP アドレスとの関連付けが行われ、逆に、ELB の実体が減れば、関連付けが削除されます。そのため、ELB を利用したシステムでは、ELB の実体の IP アドレスがわかったとしても、IP アドレスは使わずに DNS 名を使用します。ELB の実体に割り当てられる IP アドレスは、VPC のサブネット内の IP アドレスになります。

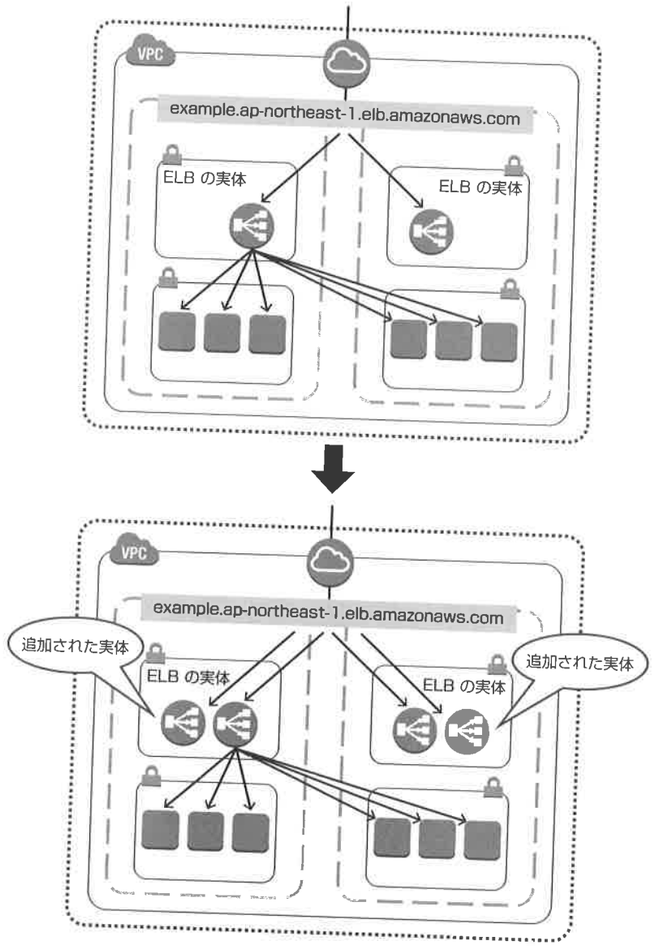

図 9-2-3　ELB の自動スケーリング

図 9-2-3 のように ELB の実体はサブネットの中に作成されますが、システム構成図としては図 9-2-1 のように AZ にまたがる負荷分散を実現して

いるサービスとして、AZ の間に ELB を描くことが多いです。以降の図でも AZ の間に ELB のアイコンを配置します。

(4) SSL のオフロード

クライアントと AWS 上のシステムの間で SSL 通信をしたい場合、SSL 証明書は各 EC2 インスタンスに配置するのではなく、ELB に配置して一元管理することができます。

(5) Connection Draining

ELB には、Connection Draining という機能があります。これは、ELB が配下の EC2 インスタンスの登録解除をするときに、新規のリクエストについてはそのインスタンスへのトラフィックの送信を停止し、登録解除前にそのインスタンスで処理中だったリクエストについては完了まで待つようにする機能です。Connection Draining を有効にしておけば、EC2 インスタンスをメンテナンスのために ELB の配下から登録解除する際や、この後紹介する Auto Scaling で EC2 インスタンスが自動的に削除される際に、クライアントのリクエスト処理が中断されてしまうことがなくなります。

(6) アクセスログ記録

ELB にはアクセスログ収集機能があり、ELB に送信されたアクセスログを S3 バケットに保存することで、アクセスログを一元管理することができます。

(7) スティッキーセッション

ELB には、スティッキーセッションという、システムにアクセスしているクライアントを特定の EC2 インスタンスに紐付けできる機能があります。例えば、図 9-2-4 のように、ユーザ認証が必要な会員 Web サイトを EC2 インスタンス上に構築し、その前段に ELB を配置しているとします。クライアントは、会員 Web サイトにアクセスした際、ELB によって割り振られたある EC2 インスタンスの Web サーバ上でユーザ認証手続きを行います。ユーザ認証に成功すると、Web サーバ側でセッション情報が保持

されるため、クライアントが会員 Web サイト内で別のページを閲覧して再度 Web サーバに要求が送られても、Web サーバはセッション情報を参照することで認証済みであることが確認でき、あらためてユーザ認証処理を行わずに済みます。ところが、ELB を Web サーバである EC2 インスタンスの前段に配置している場合、別ページに遷移する際にアクセス（ページの要求）先が別の EC2 インスタンス（Web サーバ）に割り振られてしまうかもしれません。そうすると、新たにアクセスされた Web サーバはそのクライアントのセッション情報を保持していないため、クライアントは再度ユーザ認証を求められてしまうことになります。この対策として、図 9-2-4 に示す ELB のスティッキーセッション機能を利用することで、クライアントを特定の EC2 インスタンスに紐付けることができ、クライアントが再度認証手続きを求められるような状況が発生するのを防ぐことができます。

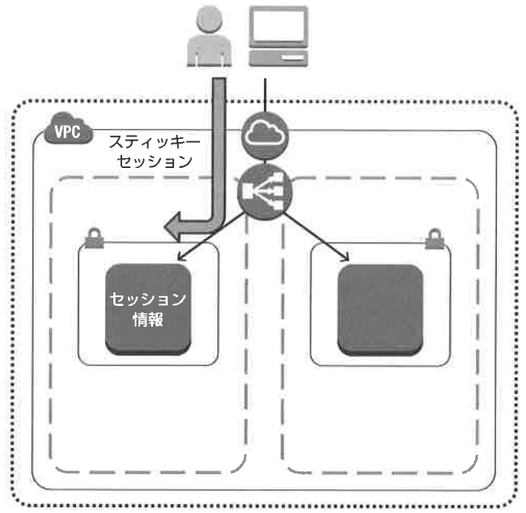

図 9-2-4　ELB のスティッキーセッション

ただし、スティッキーセッションは AWS のメリット／ベストプラクティス

の1つである「伸縮自在性を実装」に影響を及ぼすので、注意が必要です。後述するAuto Scalingを有効にして、負荷の変動に伴いEC2インスタンスが自動的に増減するWebシステムを構築したとします。このとき、スティッキーセッション機能が有効になっていると、新たなインスタンスが追加されても、各クライアントが特定のインスタンスに紐づけられているため、負荷が適切に分散されない場合があります。

AWSのメリット／ベストプラクティスの1つである「伸縮自在性を実装」するためには、「コンポーネント間が疎結合」であることの他に、「各コンポーネントが特定の状態を持たない（**ステートレス**）」であることが重要です。ここでは「セッション情報」という特定の状態を該当のEC2インスタンスが持ってしまっているため、システムの伸縮自在性を阻害しています。そこで、ELBのスティッキーセッション機能を利用するのではなく、7章で紹介したElastiCacheやDynamoDBを利用し、セッション情報をWebサーバのEC2インスタンスの外に出す構成にします。こうすることで、各EC2インスタンスは「ステートレス」になり、クライアントはELBによってどのインスタンスに割り振られてもそれを意識することなくWebシステムを利用でき、伸縮自在性の実装が実現できます。

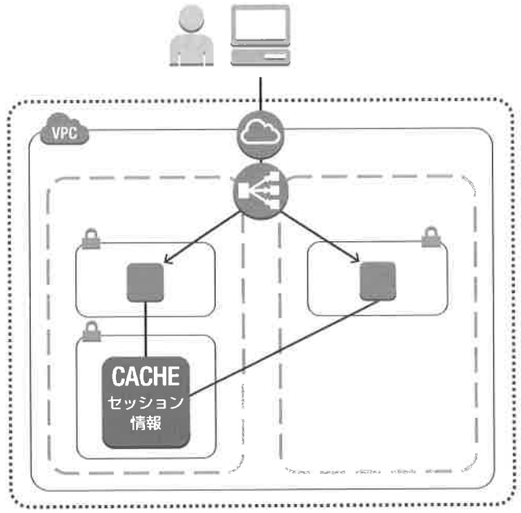

図9-2-5　ステートレス

試験のポイント！

ELBの機能／特徴を理解して、ELBによるシステムの可用性向上メリットを押さえる！

9-3 分散／並列処理

あるインスタンスタイプの EC2 インスタンス 1 台では性能が不足する場合、その対処方法として次の 2 つが考えられます。

(1) スケールアップ：インスタンスタイプを変更し、よりスペックの優れた EC2 インスタンスに変更する（図 9-3-1）。

(2) スケールアウト：既存の EC2 インスタンスはそのままに、同じ機能の EC2 インスタンスの台数を増やして分散処理させる（図 9-3-2）。

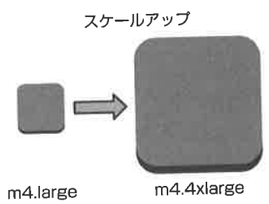

図 9-3-1　スケールアップ

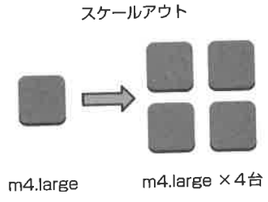

図 9-3-2　スケールアウト

(1) のスケールアップで対応した場合、次の 4 つの問題が発生します。

- インスタンスタイプを変更するには EC2 インスタンスを停止する必要があり、1 台のインスタンスの場合は業務を停止する必要がある。
- インスタンスタイプのサイズは決められており、最終的にはスペックの限界がある。
- EC2 インスタンス 1 台の可用性はどのインスタンスタイプでも同じため、1 台のインスタンスでシステムを構成する場合、7 つのベストプラクティスの「故障に備えた設計で障害を回避」を実践できない構成になり、複数台構成のシステムよりも可用性が下がる。
- システムの負荷が減少した際、スケールアップしたインスタンスタイプではオーバースペックになる。コストメリットを図るためにスケールダウンするには再度 EC2 インスタンスを停止する必要がある。

オンプレミスでは用意できる IT リソースが限られるため、決められたスペックのサーバを決められた台数でシステムを構成することが多くなります。オンプレミス環境でサーバに障害が発生すれば、当然修理して復旧させる必要があります。

一方、AWS では、いくらでも EC2 インスタンスを用意することができます。このため、その業務が分散処理できるものであれば、サーバの負荷が高くなった際には EC2 インスタンスを複数台起動して処理を分散させるスケールアウトの対応をとることで、スケールアップの対応で発生する問題を回避できます。また、図 9-3-3 のように、1 台の EC2 インスタンスで 3 時間かかる処理も、分散処理が可能なら、同じインスタンスタイプの EC2 インスタンス 3 台で分散／並列処理を行って 1 時間で終わらせることができます。EC2 インスタンスの台数を増やしても、その起動時間は変わらないため、EC2 インスタンスの起動にかかる両者のコストは全く同じです。これは、7 つのベストプラクティスの 1 つである「処理の並列化を考慮」にあたり、分散／並列処理が可能なものは、並列化することにより業務を効率化することができます。

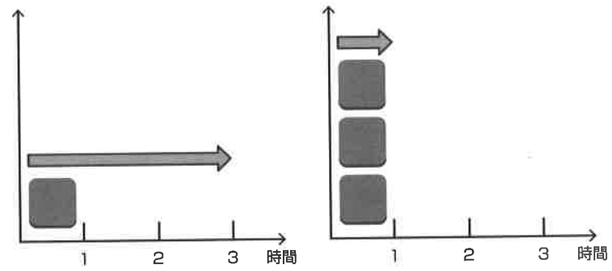

図 9-3-3　並列化

以上のように、AWS を利用する際は、サーバ台数が限られているなどのオンプレミスに存在した様々な制約を恐れずに AWS を活用することで、コストメリットにとどまらない、AWS メリットを享受することができます。逆に、AWS のサービスの仕様上、オンプレミスで利用していた機能が利用できなくなるという制約が存在する場合もあります。ところが、システムの目的であ

る何らかの「サービス」の提供を考えた場合、AWS のサービスでも利用できる別の機能で同じ「サービス」を提供できることがほとんどです。AWS 上にシステムを構築する際、オンプレミス–AWS 双方の制約を恐れず、柔軟な考えでシステムを設計／構築することが重要です。これが 7 つのベストプラクティスの 1 つ「制約を恐れない」です。

試験のポイント！

分散／並列化できる処理は並列化（スケールアップではなくスケールアウト）して、業務を効率化する！

9-4 Auto Scaling

AWS の特徴の 1 つである伸縮自在性を実現する機能が **Auto Scaling** です。Auto Scaling は、EC2 インスタンスで **Auto Scaling グループ**というグループを構成し、設定に従って自動的に EC2 インスタンスの台数を増減させます。分散／並列化できる処理を行う際、処理の負荷が変動するようであれば、負荷に応じてスケールアウト（EC2 インスタンスの台数を増やす）ことで並列処理が進みます。また、負荷が減少した際は**スケールイン**（EC2 インスタンスの台数を減らす）ことで、コストメリットが図れます。Auto Scaling の利用料金は無料で、かかる費用は Auto Scaling によって起動した EC2 インスタンスの利用料金のみです。

Auto Scaling には、次のようなユースケースがあります。

- 負荷に基づいた利用
  - – Web サイトへのアクセス数が増減
  - –ランダムに要求が発生するバッチ処理のジョブ数が増減
- スケジュールに基づいた利用
  - –毎月決まった時間に発生するバッチ処理
  - –チケット販売などで、販売開始時刻が決まっている Web サイト

- 正常な EC2 インスタンスの台数を維持するための利用
  - –数分のダウンタイムは許容される 1 台構成のシステム

ここでは、負荷に基づいた Auto Scaling の利用について説明します。図 9-4-1 のように、ELB の配下に Auto Scaling グループで構成されている Web サイトがあり、CloudWatch によって Auto Scaling グループ（全 EC2 インスタンス）の CPU 利用率の平均をモニタリングします。Auto Scaling グループの CPU 利用率のメトリックスに 70% という閾値を設定しておき、70% を超えたら OK からアラームに状態が遷移し、そのアクションとして Auto Scaling が発動するように設定します。Auto Scaling の発動により EC2 インスタンスの台数が増えれば、Auto Scaling グループの CPU 利用率は減少し、アラームが再び OK に遷移します。CloudWatch によるモニタリングを続け、再びアラームが発生すれば EC2 インスタンスを増やしたり、あるいは減らしたりという増減を自動化します。

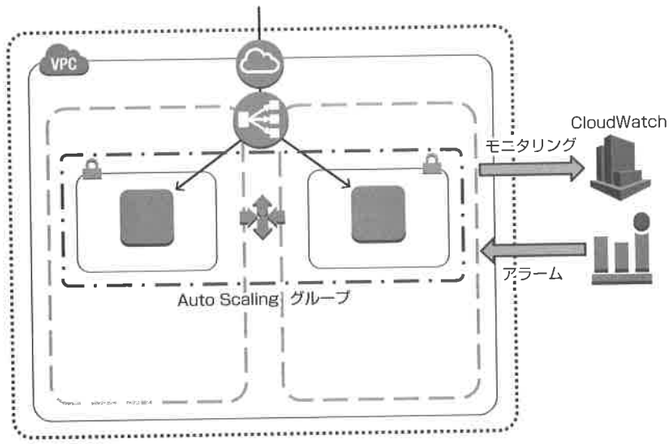

図 9-4-1　Auto Scaling の動作

Auto Scaling には、次の 3 つのコンポーネントが存在します。

(1) 起動設定（Launch Configuration）
(2) Auto Scaling Group（「Auto Scaling グループ」とは異なり、設定項目です）
(3) Auto Scaling ポリシー

(1) 起動設定（Launch Configuration）

「どんな EC2 インスタンスを起動するか？」という設定です。Auto Scaling で EC2 インスタンスを増やす際の EC2 インスタンスの起動設定で、次の項目を設定します。

- AMI
- インスタンスタイプ
- IAM ロール
- CloudWatch 詳細モニタリング
- ユーザデータ
- IP アドレス
- ストレージ（EBS、インスタンスストア）
- セキュリティグループ
- キーペア

など

(2) Auto Scaling Group

「どこに、どんな規模のグループ？」という設定です。EC2 インスタンスが起動するサブネットや、特定の ELB の配下など、どこに？という設定の他、最小／最大台数などグループの規模を決める設定で、次の項目を設定します。

- スタートのグループサイズ（初期 EC2 インスタンス数）
- サブネット（AZ）
- ELB（ヘルスチェック設定も含む）
- 最小／最大グループサイズ（EC2 インスタンス数）

など

(3) Auto Scaling ポリシー

「いつ、何台増減させるか？」という設定です。例えば、負荷に基づくスケーリングであれば、CPU 利用率の CloudWatch アラームを設定しておき、OK からアラームに状態遷移した際に、アクションとして Auto Scaling ポリシーを呼び出します。

- アラーム X が発生した際（OK からアラームに遷移した際）／指定した日時
- N 台追加／削除
- 猶予時間（インスタンスの増減後に、次の増減アクションが発生するまでのクールダウン時間）

補足 2015 年 7 月に Auto Scaling ポリシーに「Step scaling」というタイプが追加され、それまでのポリシーは「Simple scaling」というタイプになり、どちらかを選択できるようになりました。Simple scaling タイプは、1 つの調整値に基づいて EC2 インスタンスを増減させますが、Step scaling は複数の調整値に基づいてインスタンスを増減させます。Step scaling では、例えば、1 分間の CPU 利用率の平均が閾値の 60% を 1 期間連続で上回っている場合にインスタンスを 1 台増やし、その 300 秒後に引き続き CPU 利用率が 70% を上回ればさらにインスタンスを 1 台増やすという、複数ステップでの増減が可能です。

試験のポイント！

Auto Scaling における 3 つの設定項目を押さえる！

Auto Scaling には、次の 2 つの特徴（大原則）があります。

- 正常な EC2 インスタンスを希望する台数（Desired Capacity）維持するため、インスタンスのヘルスチェックをかけている
- Auto Scaling グループが複数の AZ にまたがるとき、AZ 間で EC2 インスタンス数を均等にする

Auto Scaling には「希望する台数（Desired Capacity）」という設定があります。これは、Auto Scaling グループ作成時は「スタートのグループサイズ」

で、その後は Auto Scaling ポリシーや手動設定により Auto Scaling グループの最小～最大台数の範囲で変動します。Auto Scaling は「希望する台数」を正常なインスタンス数で維持するため、EC2 インスタンスのヘルスチェックをかけており、異常なインスタンスを検出するとそのインスタンスを削除し、新たなインスタンスを起動します。また、複数の AZ にまたがる Auto Scaling グループを構成しており、Auto Scaling によって既存の EC2 インスタンスが削除される、あるいは新たな EC2 インスタンスが起動される際、Auto Scaling は複数の AZ 間でインスタンスの数が均等になるように増減させます。例えば、次のような設定を考えてみます。

【例：Auto Scaling Group と Auto Scaling ポリシー設定】

- AZ-1 と AZ-2 の 2 つのサブネットにまたがる Auto Scaling Group
- 最小台数：2 台
- 最大台数：8 台
- スタートのグループサイズ：2 台
- 増加する Auto Scaling ポリシー：CPU 利用率 60% を超えたアラームで 1 台増やす
- 削減する Auto Scaling ポリシー：CPU 利用率 30% を下回ったアラームで 1 台減らす

① Auto Scaling グループを作成する（スタートのグループサイズは 2 台）

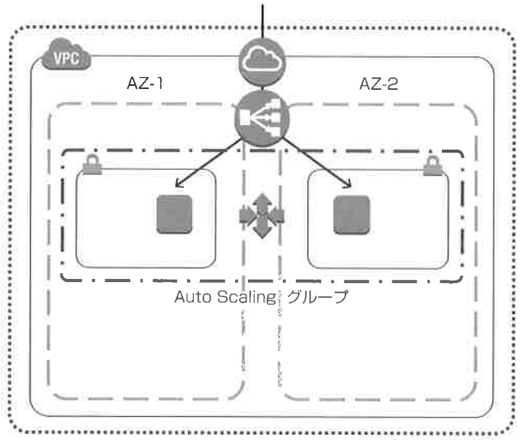

図 9-4-2　Auto Scaling グループ作成時

② 負荷が 60% を超えたため、1 台 EC2 インスタンスを増やす（増やす AZ は任意）

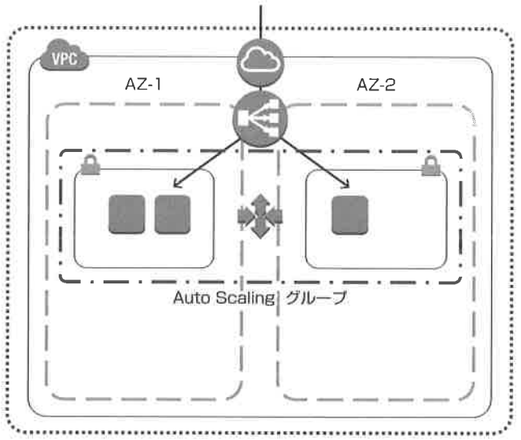

図 9-4-3　Auto Scaling ポリシーの呼び出し①

③ 負荷が再度 60% を超えたため、1 台 EC2 インスタンスを増やす
（必ず台数が少ない AZ 内のサブネットに 1 台増やします）

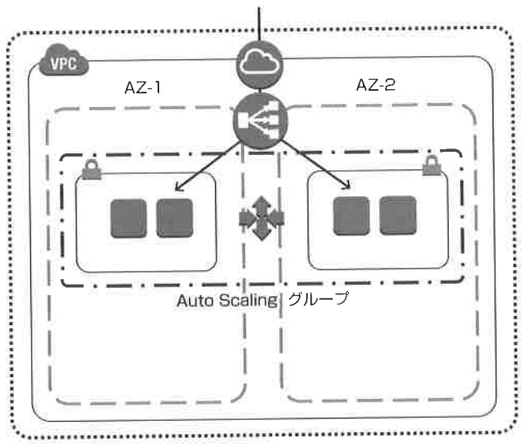

図 9-4-4　Auto Scaling ポリシーの呼び出し②

④ 負荷が 30% を下回ったため、1 台 EC2 インスタンスを減らす
（減らす AZ は任意）

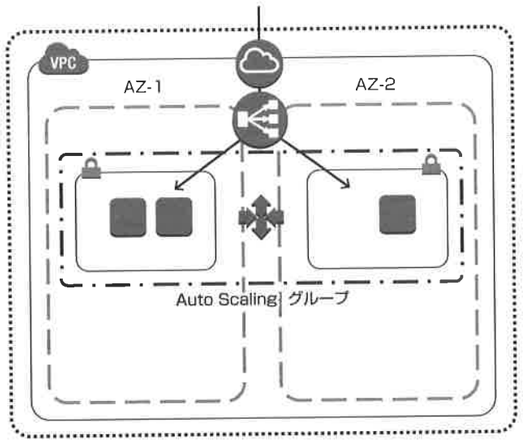

図 9-4-5　Auto Scaling ポリシーの呼び出し③

⑤ 負荷が再度 30% を下回ったため、1 台 EC2 インスタンスを減らす
（必ず台数が多い AZ 内のサブネットの 1 台を減らします）

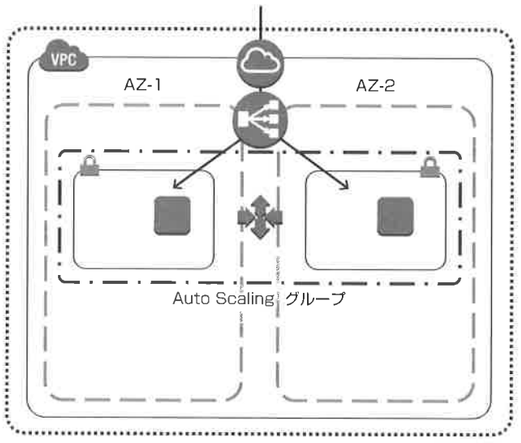

図 9-4-6　Auto Scaling ポリシーの呼び出し④

EC2 インスタンスを減らす際の減らし方は、利用者が設定できますが、デフォルトでは、次の順番に特定された EC2 インスタンスが 1 台削除されます。

① 起動している台数が最も多い AZ のインスタンス（大原則）
② 起動設定（Launch Configuration）が最も古いインスタンス
③ 次の課金タイミングが最も近いインスタンス

試験のポイント！

Auto Scaling における大原則を押さえ、EC2 インスタンスの増減がどのように発生するかを押さえる！

9-5 SQS

AWS には、Amazon Simple Queue Service（以下 **SQS**）というマネージド型のメッセージキューサービスがあります。これは、ELB と並んでコンポーネントを疎結合にする要素で、AWS で分散／並列処理を行う上で重要なサービスです。

例えば、複数の重たいバッチ処理を並列で処理する場合を考えてみます。図 9-5-1 のように、処理を依頼するフロントの EC2 インスタンスとバッチ処理を処理する EC2 インスタンスが密結合している場合、バッチ処理のインスタンスに伸縮自在性はありません。

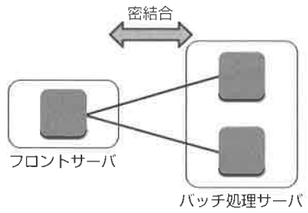

図 9-5-1　密結合バッチ処理

一方、図 9-5-2 のように、フロントの EC2 インスタンスとバッチ処理の EC2 インスタンスの間に SQS を導入し、システムを疎結合にすると、バッチ処理のインスタンスに伸縮自在性を持たせることができ、処理の負荷によってインスタンス数を増減させることができます。

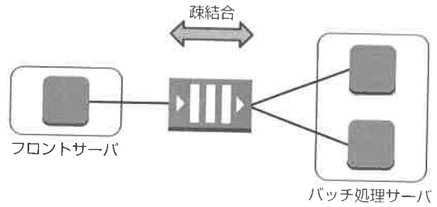

図 9-5-2　疎結合バッチ処理

SQS には、次のような特徴があり、その特徴を押さえて利用する必要があります。

(1) Pull 型（ポーリングされる必要がある）
(2) 順序性の保証はしない（FirstInFirstOut が保証されない）
(3) 最低 1 回配信保証
(4) 可視性タイムアウト
(5) メッセージサイズは最大 256KB

(1) Pull 型（ポーリングされる必要がある）

SQS は Pull 型のメッセージキューであり、図 9-5-3 のようにアプリケーションからポーリングされる必要があります。

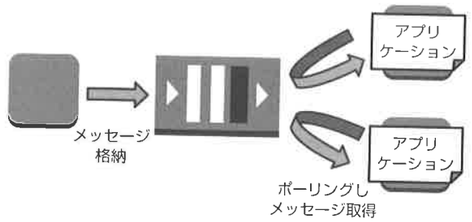

図 9-5-3　Pull 型

(2) 順序性の保証はしない（FirstInFirstOut が保証されない）

SQS はマネージド型のキューサービスで、配信すべきメッセージが失われないように複数のストレージに冗長的にメッセージが保持されています。メッセージの物理的な保持形式とメッセージの取得アルゴリズムによりメッセージの順序性は保証されておらず、後から登録したメッセージが先に登録したメッセージよりも先に取得されることがあります。

(3) 最低 1 回配信保証

メッセージは、あるアプリケーションによって取得されてもキューから削除されることはなく、図 9-5-4 のようにアプリケーションがバッチ処理の最後に明示的に削除する必要があります。この特徴により、メッセージを取得したアプリケーションがバッチ処理中に障害などで停止してしまっても、他のノード（EC2 インスタンス）上のアプリケーションが再度同じメッセージを取得して、バッチ処理を実行できます。

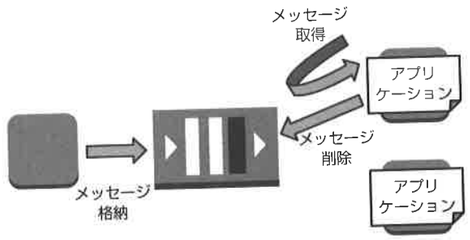

図 9-5-4　最低 1 回配信保証

(4) 可視性タイムアウト

SQS には、あるメッセージを取得したアプリケーションがバッチ処理を実行中に、他のノード（EC2 インスタンス）上のアプリケーションがキューに残っている同じメッセージを取得して同じ処理をしないよう、可視性タイムアウト（図 9-5-5）という機能が備わっています。可視性タイムアウトのデフォルトは 30 秒で、利用者による設定も可能です。

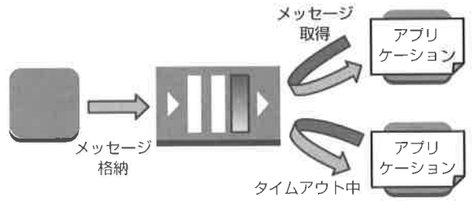

図 9-5-5　可視性タイムアウト

(5) メッセージサイズは最大 256KB

SQS に格納できるメッセージの最大サイズは 256KB であるため、実際の処理対象データが大きい場合は、S3 などの外部ストレージに保存し、SQS にはデータの格納先の情報を格納します。アプリケーションは、SQS からメッセージを取得すると、実際の処理対象データを S3 などの格納先からダウンロードして処理を実行します。（図 9-5-6）

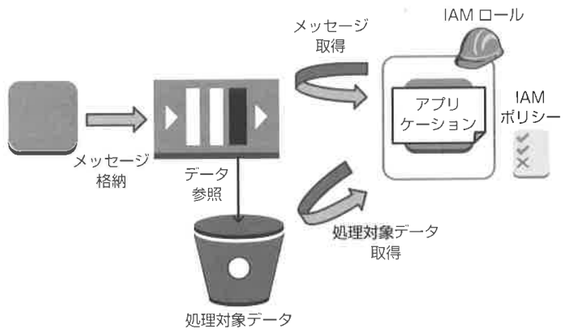

図 9-5-6　SQS のユースケース

SQS は、リージョンサービスであり、プライベートサブネットからキューのポーリングやメッセージの格納といった操作を行うには、NAT インスタンスが必要です。SQS に対する操作は IAM によってアクセス管理を行い、SQS にアクセスするアプリケーションが EC2 インスタンス上で動作する場合は、SQS へのアクセス許可の IAM ポリシーが設定された IAM ロールを EC2 イン

スタンスに割り当てることで、安全に SQS へのアクセスを制御できます。

試験のポイント！

分散／並列処理における SQS のメリット／特徴を理解し、SQS のユースケースを押さえる！

9-6 SWF

Amazon Simple Workflow（以下 **SWF**）は、マネージド型のタスクコーディネータです。これは、商品の発注／請求処理のワークフロー（処理の流れ）のような、重複が許されない、厳密に 1 回限りで順序性が求められる処理のコーディネータとしての利用に適しています。

SWF は、次の 3 つの要素から構成されます。

- **ワークフロースターター**：ワークフローを開始する。
- **ディサイダー**：ワークフロー中の各処理を調整する。
- **アクティビティワーカー**：ワークフロー中の各処理を実行する。

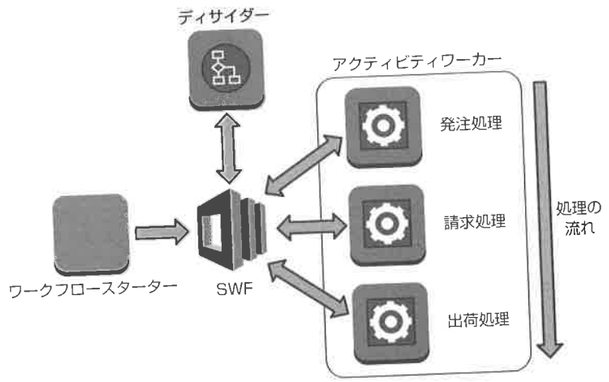

図 9-6-1　SWF を使用した処理の流れ

試験のポイント！

分散／並列処理における厳密に 1 回限りで順序性が求められる処理という SWF のユースケースを押さえる！

章末問題

Q1 ELBについて、その特徴やメリットを正しく説明している選択肢はどれか？

- ○ **A** マルチリージョン（東京リージョンとソウルリージョン）にEC2インスタンスを冗長的に起動し、それらをELBの配下に配置することで、リージョンレベルの障害にも備えた高可用なシステムを構築できる。
- ○ **B** ELBは配下のEC2インスタンスのヘルスチェックを行っており、異常のインスタンスを検出すると、そのインスタンスをターミネートし、配下から削除する。新しいインスタンスの起動については、Auto Scalingと組み合わせて利用する必要がある。
- ○ **C** ELBはPCからのアクセスやモバイル端末からのアクセスなど、受信したトラフィックの種類に応じて特定のEC2インスタンスにトラフィックを分散することができる。
- ○ **D** ELBは負荷に応じてELB自体が動的にスケーリングすることにより、ボトルネックにならないように設計されている。スケーリングに応じてELBの実体のIPアドレスも変化するため、ELBのIPアドレスを直接指定（使用）してはいけない。

Q2 可用性やコストメリットを考慮した上で、適切なシステム構成はどれか？正しい選択肢を**全て**選べ。

- □ **A** システムの負荷テストを1週間かけて行い、予測される最大の負荷に対応できるEC2インスタンスタイプ／数を初期構成とする。予測できない負荷に対しては、Auto Scalingを実装して対応する。
- □ **B** システムの負荷テストは1日で終わらせ、予測される平均の負荷に対応できるEC2インスタンスタイプ／数を初期構成とする。予測できない負荷に対しては、Auto Scalingを実装して対応する。
- □ **C** オンプレミス環境で80時間かかっていた分散／並列処理が可能なバッチ処理がある。この処理をコンピューティング最適化インスタンスファミリーの中で一番性能／利用料金の高いc4.8xlargeを1台使用して40時間で処理する。
- □ **D** オンプレミス環境で80時間かかっていた分散／並列処理が可能なバッチ処理がある。この処理をコンピューティング最適化インスタンスファミリーの中で一番性能／利用料金の高いc4.8xlargeを40台使用して1時間で処理する。

Q3 年末年始の休暇が1ヶ月後に迫り、Auto Scaling設定がされている運航チケット予約WebシステムのEC2インスタンスの最大数を一時的に増やしたい。Auto Scalingのどの設定を変更したらよいか？

- ○ **A** 起動設定（Launch Configuration）
- ○ **B** Auto Scaling Group
- ○ **C** Auto Scalingポリシー
- ○ **D** CloudWatchアラームの閾値

Q4 2つのAZ（AZ-1とAZ-2）内のサブネットが設定されたAuto Scalingグループがある。現在それぞれのAZに2台ずつAuto Scalingグループに所属しているEC2インスタンスが起動している。Auto Scalingポリシーで CPU利用率が70%を超えたら2台インスタンスを増やし、40%を下回ったら1台インスタンスを減らし、さらに30%を下回ったら1台インスタンスを減らす設定をしている。CPU利用率が次のように推移した場合、各AZのインスタンス数の分布として発生しうる選択肢はどれか？正しい選択肢を**全て**選べ。
CPU利用率：50% → 75% → 45% → 35%

- □ **A** AZ-1：3　AZ-2：3
- □ **B** AZ-1：2　AZ-2：2
- □ **C** AZ-1：3　AZ-2：2
- □ **D** AZ-1：2　AZ-2：3

Q5 SQSを導入することで効果が見込まれるシステムはどれか？

- ○ **A** 動画トランスコード
- ○ **B** 動画配信
- ○ **C** ショッピングサイトの買い物かご
- ○ **D** ショッピングサイトの注文ー請求処理

答え

A1 D

A　ELB は AZ をまたがったトラフィックの分散はできますが、リージョンレベルでトラフィックを分散できません。
B　ELB は異常が検知された配下の EC2 インスタンスへはトラフィックの分散を中止するだけで、そのインスタンスに対して他の処理は行いません。
C　ELB は HTTP ヘッダの情報でトラフィックの送信先を分けるような L7 レベルの負荷分散をサポートしていません。

A2 B、D

予測できない負荷に対するテストに時間をかけるのではなく、負荷ベースの Auto Scaling を利用することで、負荷に対応します。運用開始後に最小／最大グループサイズを変更することもできます。
1 台のインスタンスを 40 時間稼働させる利用料金と 40 台のインスタンスを 1 時間稼働させる利用料金は同じため、可能な限り並列処理を実施することで、同じ料金で処理時間を短縮化できます。

A3 B

Auto Scaling において、最大グループサイズを規定しているのは Auto Scaling Group 設定です。

A4 C、D

CPU 利用率が 50% → 75% → 45% → 35% と推移したとき、EC2 インスタンスは 70% を超えた際に 2 台増え、40% を下回った際に 1 台減ります。その結果、マシン台数は 4 ＋ 2 － 1 で 5 台になります。Auto Scaling は各 AZ 間のマシン台数を均等化する仕様ですが、2 つの AZ で 5 台のため、どちらか一方の AZ が 3 台、もう一方の AZ が 2 台になります。

A5 A

A　動画のトランスコード処理は時間がかかる処理です。複数のトランスコード処理の要求が同時に発生した場合、SQS を利用することで、複数のノード（EC2 インスタンス）に分散処理が可能です。
B　後述の章の CloudFront で効率的な動画配信が可能です。
C　ショッピングサイトの買い物かごは参照される買い物リストが格納されるため、NoSQL のサービスである DynamoDB が適しています
D　発注／請求処理のような厳密な順序や回数が求められる処理に SQS は向いていません。厳密な順序や回数が求められる処理には SWF を利用します。

第 10 章

DNS とコンテンツ配信 (Route 53／CloudFront)

DNS とコンテンツ配信について

Route 53 は、AWS のマネージド型の DNS サービスです。Route 53 は、一般的な DNS サーバと同様にドメインを登録／管理することができ、管理するゾーン数やクエリ（問い合わせ）回数に応じた従量課金で低額な利用ができます。

また、AWS には CloudFront というコンテンツ配信のサービスがあり、大きなファイルを低レイテンシーで配信できます。

認定試験においては、どちらもそのユースケースについてよく出題されます。

10-1 エッジロケーション

AWS には、リージョンと AZ 以外に、**エッジロケーション**というデータセンタが世界に 50 カ所以上あります。エッジロケーションでは、EC2 や S3、RDS といったサービスではなく、本章で説明する Amazon **Route 53**（以下 Route 53）の DNS サーバや Amazon **CloudFront**（以下 CloudFront）のキャッシュサーバ、そして AWS WAF（Web Application Firewall）のサーバが動作しています。エッジロケーションは数が多いため、その具体的な場所については AWS の「グローバルインフラストラクチャ」ページで確認してください。

重要！

世界 50 カ所以上のエッジロケーションを利用して、DNS サービスやコンテンツ配信（CDN）サービスが提供されている！

10-2 Route 53

Route 53 はマネージド型の DNS サービスで、DNS サービスが 53 番ポートを利用することからその名前が付けられています。Route 53 では、正／逆引きの名前解決（ゾーン管理）の他、ドメインの登録もできます。

Route 53 を利用してゾーン／ドメイン情報を登録すると、4 カ所のエッジロケーションの DNS サーバにゾーン／ドメイン情報が格納されます。そして、Route 53 が管理しているドメインに対してクエリ（問合せ）が発生すれば、その 4 カ所のうち、クエリを行ったエンドユーザに最も近い DNS サーバが応答します。4 カ所の DNS サーバが同時に停止する確率は限りなく低いため、Route 53 の SLA（Service Level Agreement）は 100% として提供されています。また、利用料金は、管理しているホストゾーン（従来の DNS ゾーンファイル）の数とクエリ回数などの従量課金制になっており、低額で利用することができます。

Route 53 は、次のレコードタイプをサポートします。

- A
- AAAA（IPv6）
- CNAME
- MX
- NS
- PTR
- SOA
- SPF
- SRV
- TXT
- ALIAS（エイリアス；AWS 独自レコード）

Route 53 では、通常の DNS サービスがサポートする A レコードや CNAME レコードをサポートしています。

表 10-2-1

名前	レコードタイプ	値
serv1.hitachi-ia.com	A	123.123.123.123
www.hitachi-ia.com	CNAME	serv1.hitachi-ia.com

表 10-2-1 の例では、serv1.hitachi-ia.com という名前に対するクエリには 123.123.123.123 という IP アドレスを返し、www.hitachi-ia.com という名前に対するクエリには serv1.hitachi-ia.com という別名を返します。

Route 53 は、「**ALIAS レコード**」という独自のレコードもサポートしています。ALIAS レコードは CNAME レコードと同じように機能しますが、CNAME レコードでは対応できない Zone Apex の名前解決をサポートします。**Zone Apex**（ゾーンエイペックス）とは、ゾーンの頂点のことで、表 10-2-1 の例であれば「hitachi-ia.com」を指します。DNS の仕様として、ある名前で CNAME レコードを定義した場合、その名前を別のレコードで名前解決することができません。

いま、例えば、表 10-2-2 のように、ELB の DNS 名と Zone Apex（自ドメイン）を CNAME で名前解決させる例を考えます。

表 10-2-2

名前	レコードタイプ	値
hitachi-ia.com	SOA	nameserv.hitachi-ia.co.jp
hitachi-ia.com	CNAME	example.ap-northeast-1.elb.amazonaws.com

ここでは、hitachi-ia.com という名前（Zone Apex）に対するクエリに example.ap-northeast-1.elb.amazonaws.com という別名を返すようにしたいのですが、そうすると、CNAME の制約から hitachi-ia.com というドメイン名（Zone Apex）に対する SOA など、ゾーン管理に必要な他のレコードを定義することができなくなってしまいます。このことから、CNAME レコードは Zone Apex に対応できません。

ELB は、9 章で説明したように、実体が増減するため IP アドレスを使用することができず、必ず DNS 名を使用する必要があります。同様に、6 章で説明した S3 の静的 Web サイトホスティングでも、S3 上の Web サイトの指定に IP アドレスを使用することができず、mybucket.s3-website-ap-northeast-1.amazonaws.com のようなエンドポイント名で指定する必要があります。したがって、ELB の DNS 名や S3 のエンドポイント名を自ドメインの Zone Apex に関連付けたい場合には、CNAME ではなく、表 10-2-3 のように AWS の独自レコードである ALIAS レコードを利用します。

表 10-2-3

名前	レコードタイプ	値
hitachi-ia.com	ALIAS	example.ap-northeast-1.elb.amazonaws.com
www.hitachi-ia.com	CNAME	hitachi-ia.com

試験のポイント！

Route 53 の独自レコードである ALIAS レコードは、CNAME レコードでは対応できない Zone Apex の名前解決をサポートする！

Route 53 では、各レコードに次のような設定を行えます。

- 加重ラウンドロビン
- レイテンシーベースルーティング
- 位置情報ルーティング
- ヘルスチェックとフェイルオーバー

加重ラウンドロビンは、各レコードに重みづけをし、ある名前に対するクエリに指定された比率で異なる値を応答します。例えば、表 10-2-4 の例では、serv1.hitachi-ia.com に対するクエリが発生した場合、図 10-2-1 のように 7:3 の比率で設定された値を返します。

表 10-2-4　加重ラウンドロビンの設定例

名前	レコードタイプ	値	重みづけ
serv1.hitachi-ia.com	A	123.123.123.123	70
serv1.hitachi-ia.com	A	213.213.213.213	30

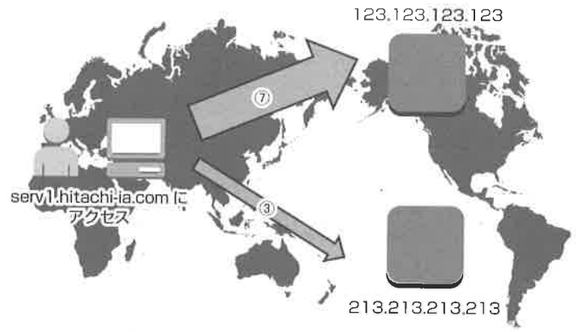

図 10-2-1　加重ラウンドロビン

全世界展開しているようなサービスを複数のリージョンで提供している場合には、レコードに対して**レイテンシーベースルーティング**対応と指定することで、クライアントへのレイテンシー（遅延）を小さくすることができます。表 10-2-5 の例のレイテンシーベースルーティング対応のレコードがあり、クライアントが hitachi-ia.com にアクセスした際、レイテンシーが小さくなるリージョンの ELB の DNS 名が返されます。

表 10-2-5

名前	レコードタイプ	値
hitachi-ia.com	ALIAS	example.ap-northeast-1.elb.amazonaws.com
hitachi-ia.com	ALIAS	example.us-east-1.elb.amazonaws.com

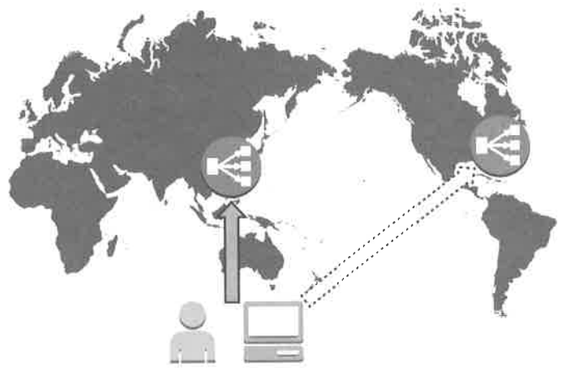

図 10-2-2　レイテンシーベースルーティング

位置情報ルーティングは、レイテンシーベースルーティングと似ていますが、こちらは、クライアントの IP アドレスを元に地理データベースでクライアントの接続元地域を特定し、地理的に近いレコードの値を返します。また、特定のコンテンツを地理情報ルーティングを利用して特定の地域だけに配信することもできます。

Route 53 には**ヘルスチェック**機能があります。名前解決している Web サーバなどが正常に動作しなくなった場合、その Web サーバに対するクエリが発生しても、Web サーバの IP アドレス／名前を返さなくなります。このとき、事前に**フェイルオーバー**先の設定をしていれば、そのフェイルオーバー先の IP アドレス／名前を返します。

例えば、同じ hitachi-ia.com という名前に対して ELB の DNS 名と S3 のエンドポイントを登録しておきます。ELB のレコードにヘルスチェック設定をしておき、ELB をプライマリ、S3 をバックアップに設定しておくと、通常、

Route 53 は hitachi-ia.com に対するクエリの応答として ELB の DNS を返しますが、何らかの障害が発生して ELB が Web ページを返さなくなると、S3 のエンドポイントを返します（図 10-2-3）。

図 10-2-3　DNS フェイルオーバー

以上の Route 53 の機能を利用することで、ELB では実現できないリージョンレベルの負荷分散や冗長構成を実現できます。

試験のポイント！

Route 53 の各種機能による、リージョンレベルのユースケースを押さえる！

10-3 CloudFront

S3 バケットに格納している動画オブジェクトを全世界のエンドユーザに配信することを考えます。6 章で説明したように、S3 バケットにオブジェクトとして格納すれば、次の例のような URL が付与されるため、アクセスコントロールリストなどで適切なアクセス許可を与えれば、動画オブジェクトにアクセスさせること（ダウンロード）ができます。

動画オブジェクト URL の例：
https://s3-ap-northeast-1.amazonaws.com/my-bucket/sp-movie.mp4

S3 はリージョンサービスであり、オブジェクトを格納するバケットは指定したリージョン内に作成します。上記の例であれば、URL に「ap-northeast-1」というリージョン名が付いており、動画オブジェクトが東京リージョンに格納されていることがわかります。この動画オブジェクトをブラジルからアクセス（ダウンロード）する場合、東京リージョンからダウンロードしなくてはいけないため、動画オブジェクトのサイズによっては大きなレイテンシー（遅延）が発生します（図 10-3-1）。

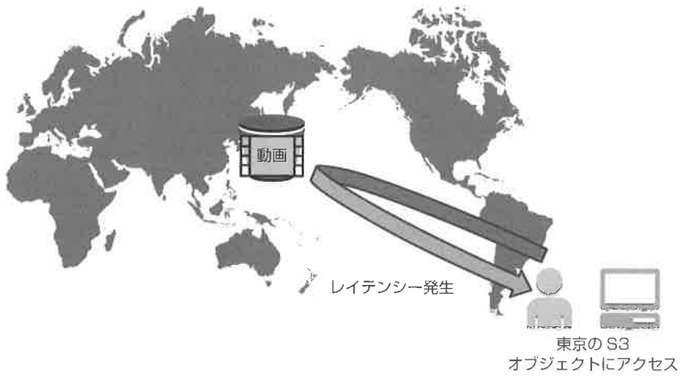

図 10-3-1　ダウンロードの際のレイテンシー

しかし、動画のような「いつ／誰がアクセスしてもコンテンツが変わらない」**静的コンテンツ**であれば、CloudFront を利用することで、レイテンシーを小さくすることができます。CloudFront は **CDN**（Contents Delivery Network）サービスで、全世界に 50 カ所以上存在するエッジロケーションにコンテンツをキャッシュし、コンテンツにアクセスするエンドユーザは地理的に近いエッジロケーションからコンテンツをダウンロードすることで、高速なダウンロードが可能になります。CloudFront はアクセス回数とデータ転送量による従量課金制であり、長期契約や最低利用料金は必要ありません。

CloudFront を利用する流れは、次のとおりです。

① 動画などアクセス（ダウンロード）させたいコンテンツ（ファイル）を S3 バケットや EC2 インスタンス／ELB、あるいはオンプレミスのサーバに格納します。これらの格納先を「オリジンサーバ」といいます（図 10-3-2）。

図 10-3-2　オリジンサーバへの格納

② CloudFront ディストリビューション（以下、ディストリビューションといいます）を作成します（図 10-3-3）。ディストリビューションは「example123.cloudfront.net」のようなドメイン名で、S3 バケットなどのオリジンサーバが設定されています。

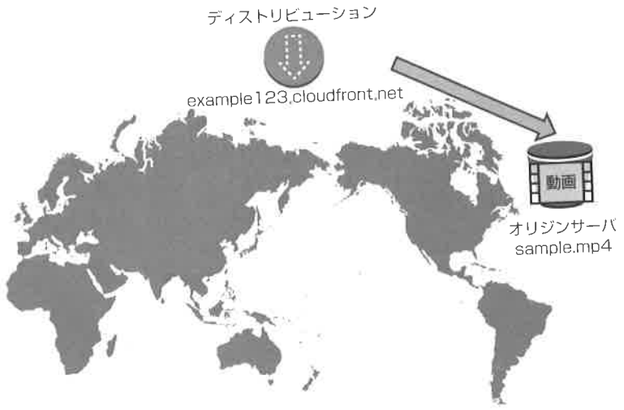

図 10-3-3　ディストリビューションの作成

③ CloudFront ディストリビューションにエンドユーザがアクセスすると、DNS による名前解決の際、地理データベースにより、コンテンツにアクセスしようとしているエンドユーザには、地理的に最も近いエッジロケーションの IP アドレスが返されます（図 10-3-4）。

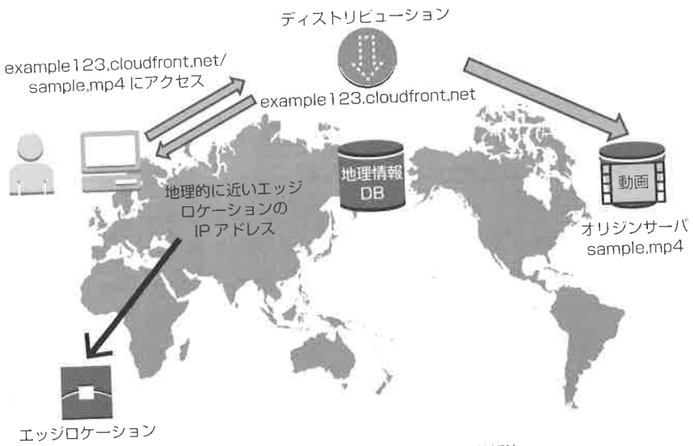

図 10-3-4　ディストリビューションの名前解決

④ エンドユーザは、返された IP アドレスに従い、地理的に最も近いエッジロケーションにアクセスし、コンテンツがキャッシュされていればそこからコンテンツをダウンロードします（図 10-3-5）。コンテンツは、最初にエンドユーザからのアクセスがあった際に、オリジンサーバからダウンロード／キャッシュされます。

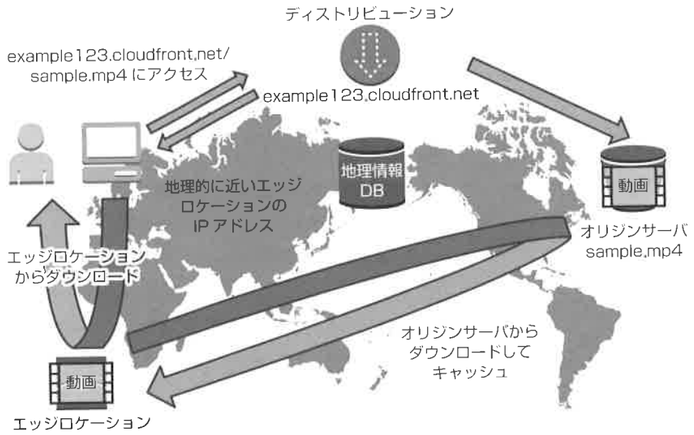

図 10-3-5　エッジロケーションからのダウンロード

以上の①～④の流れにより、コンテンツが最寄りのエッジロケーションにキャッシュされていれば、コンテンツのダウンロードにかかるレイテンシーを小さくすることができます。

1 つのディストリビューションに複数のオリジンサーバを設定して、オリジナルコンテンツの拡張子によってオリジンサーバを S3 バケット／EC2 インスタンスというように振り分けることができます。CloudFront では、エッジロケーションにキャッシュさせる時間を設定することができ、「いつ／誰がアクセスしてもコンテンツが変わらない」**静的なコンテンツ**については S3 バケットに格納してキャッシュ時間を長くし、「サーバサイドプログラムによってコンテンツが変化する」**動的なコンテンツ**については EC2 インスタンスに格納してキャッシュ時間を短くする、といった設定ができます。

また、CloudFront には、エンドユーザのメリットだけではなく、サービス提供側のメリットもあります。CloudFront を利用することで、大量のアクセスが各地のエッジロケーションに分散され、オリジンサーバの負荷が大幅に減少します。これにより、オリジンサーバとして用意しなければいけないリソースを削減することができる他、S3 バケットに直接アクセスさせるよりも AWS の利用料金を抑えられる場合もあります。

試験のポイント！
CloudFront の特徴／メリットを理解し、ユースケースを押さえる！

CloudFront を利用したコンテンツ配信においても、CloudFront の SSL 証明書、あるいは利用者独自の SSL 証明書を利用した暗号化通信が可能です。また、S3 バケットに格納しているオリジナルコンテンツについて、CloudFront 経由でアクセスさせる際にも、6 章で紹介した「署名（期限）付き URL」によるコンテンツの配信が可能です。「署名付き URL」は、S3 バケットに格納しているオブジェクトの「署名付き URL」を生成するのではなく、CloudFront 用のキーペアを使って「署名付き URL」を生成し、その URL にアクセスさせます。

CloudFront 用の「署名付き URL」を利用してアクセス制限する際、S3 バケットに格納しているオブジェクトをエンドユーザに直接アクセスさせるのではなく、CloudFront 経由に限ってアクセスさせるように、バケットポリシーを設定します。バケットポリシーで CloudFront からのアクセスだけを許可するには、CloudFront ユーザを意味する Original Access Identity（OAI）を作成し、OAI からのアクセスだけを許可します。

試験のポイント！
CloudFront を利用したコンテンツ配信におけるセキュリティ／アクセス制限を押さえる！

章末問題

Q1 S3 の静的ウェブサイトホスティング機能を利用した Web ページを、「example.com」という名前でアクセスさせるよう、Route 53 で名前解決したい。どのレコードを使用すればよいか？

- ○ **A** A レコード
- ○ **B** AAAA レコード
- ○ **C** CNAME レコード
- ○ **D** ALIAS レコード

Q2 2 つのリージョンにそれぞれ同じ Web システムを構成し、世界のどこからアクセスしても同じ URL で Web システムにアクセスできるように Route 53 で名前解決している。このとき、利用すべき Route 53 の機能として、適切なものはどれか？正しい選択肢を**全て**選べ。

- □ **A** 加重ラウンドロビン
- □ **B** レイテンシーベースルーティング
- □ **C** ヘルスチェック／フェイルオーバー
- □ **D** クロスリージョンルーティング

Q3 CloudFront を利用した際のメリットはどれか？正しい選択肢を**全て**選べ。

- □ **A** 1 年単位の契約で CDN サービスが利用できる。
- □ **B** コンテンツを世界各地のエッジロケーションにキャッシュすることで、動画などの大きなコンテンツを高速にダウンロードできる。
- □ **C** アクセス（ダウンロード）させるオリジナルのデータ（コンテンツ）格納先として、AWS 上の S3 バケットや ELB（EC2 インスタンス）の他、オンプレミスのサーバも指定でき、コンテンツ配信サーバの負荷を下げることができる。
- □ **D** CloudFront では、ディストリビューション作成時にオリジナルデータ（コンテンツ）が世界各地のエッジロケーションにキャッシュされるため、最初にアクセスしたエンドユーザから高速アクセス（ダウンロード）が可能となる。

Q4　年度初めに行われた期首方針の社外秘の説明動画を、全世界の支店／事務所に配信したい。どのように配信するのが最も適切か？

○ **A**　全てのリージョンに動画配信用の S3 バケットを作成し、その中に動画をコピーする。全従業員を IAM ユーザとして登録し、IAM ポリシーで各従業員の最寄りのリージョンの S3 バケット内の動画へのアクセスを許可する。

○ **B**　全てのリージョンに動画配信用の S3 バケットを作成し、その中に動画をコピーする。動画オブジェクトの URL から署名付き URL を作成し、各支店／事務所のイントラサイトに署名付き URL を掲載する。

○ **C**　本社近くのリージョンに動画配信用の S3 バケットを作成し、その中に動画を格納する。CloudFront を利用して、その S3 バケットをオリジンサーバに設定し、全社のイントラサイトに CloudFront のディストリビューション URL を掲載する。

○ **D**　本社近くのリージョンに動画配信用の S3 バケットを作成し、その中に動画を格納する。OAI を設定し、本社の従業員を含め、S3 バケットからの直接動画配信を禁止し、全社のイントラサイトに CloudFront の署名付き URL を記載する。

答え

A1　D

Zone Apex の別名名前解決を行うには、Route 53 独自の ALIAS レコード使用する必要があります。

A2　B、C

B　レイテンシーベースルーティングの設定をすることで、Route 53 に問い合わせた際、エンドユーザからのレイテンシーが低くなるリージョン（サイト）の名前／IP アドレスが返されます。

C　2 つのリージョン（サイト）でヘルスチェック／フェイルオーバー設定をしておくことで、片方のリージョン（サイト）がダウンしているときは、Route 53 への問い合わせに対し、正常に動作しているリージョン（サイト）名前／IP アドレスのみが返されます。

D　クロスリージョンルーティングという名前の機能はありません。

A3　B、C

A　CloudFront は、長期契約が不要な、従量課金制のサービスです。

C　オリジンサーバとして、オンプレミスのサーバも設定可能です。

D　コンテンツがエッジロケーションにキャッシュされるのは、そのエッジロケーションに初回アクセスが発生したときです。

A4　D

AWS リソース／運用管理にかかるコストと、社外秘の動画配信の安全性を考慮した場合、D が最も適切です。OAI は CloudFront を意味するユーザに相当し、S3 オブジェクトへのアクセスを CloudFront に限定することができます。

第 11 章

AWS サービスのプロビジョニング／デプロイ／構成管理 (CloudFormation／Elastic Beanstalk／OpsWorks)

AWS のプロビジョニング／デプロイ／構成管理サービスについて

AWS は柔軟性に富んだ従量課金制のクラウドサービスのため、必要なときに、必要なだけ、リソースを提供／配置（プロビジョニング／デプロイ）することで、利用者はコストメリットが得られます。そして、必要なときに、必要なだけ、リソースを提供／配置する上で重要なのは、自動プロビジョニング／デプロイサービスです。

AWS には、CloudFormation というプロビジョニングサービスや、Elastic Beanstalk というデプロイサービス、OpsWorks という構成管理サービスがあります。認定試験においては、これらのサービスのユースケースについてよく出題されます。

11-1 CloudFormation

AWS は、柔軟性に富んだ従量課金制のクラウドサービスです。開発／検証環境が必要になったときに、必要な数を迅速にプロビジョニング（提供）したり、災害発生時には一時的なサイトを本番環境サイトが稼働していたリージョンとは異なるリージョンに迅速にプロビジョニングしたりすることで、コストメリットを図りつつ、連続的なサービスを提供できます。このとき、手動によるプロビジョニングでは、誤りが発生したり、プロビジョニングまでにかかる時間が長くなったりする問題点があります。

AWS には、AWS **CloudFormation**（以下 CloudFormation）というプロビジョニングサービスがあり、利用者が用意した定義（コード）に従って AWS リソースを自動的にプロビジョニングします。自動化により、AWS リソースの構築／管理を効率化できる他、インフラストラクチャをコード化して、インフラのバージョン管理が可能になります。

CloudFormation 自体の料金は無料で、CloudFormation によってプロビジョニングされたリソースの利用料金のみ発生します。

CloudFormation を利用するには、次の 2 つの用語を押さえる必要があります。

- **テンプレート**：プロビジョニングするリソースを規定する JSON 形式のテキストファイル
- **スタック**：CloudFormation によってプロビジョニングされるリソースの集合／管理単位

CloudFormation は、JSON 形式で記述された、テンプレートと呼ばれる設定ファイルに従って、様々な AWS サービスをプロビジョニングします。テンプレートを元にプロビジョニングされるリソースの集合をスタックといい、スタック単位でリソースの更新や削除が可能です。そのため、ある時点でのスタックを定義するテンプレートや現時点でのスタックを定義しているテンプレートなど、テンプレートのバージョン管理ができ、これによりインフラストラクチャをあたかもソフトウェアのようにコード化して、バージョン管理をすることができます。

試験のポイント！

CloudFormation を利用したインフラストラクチャのバージョン管理イメージを押さえる！

テンプレートは、AWS から提供されるサンプルテンプレートを元に利用者が編集したり、利用者が独自に作成したりできる他、**CloudFormer** というツールを利用して作成することもできます。CloudFormer は、利用者のアカウントで現在作成されている AWS リソースを元にテンプレートを作成することができるツールで、利用者がテンプレートを自作する際の開始点として利用できます。

例えば、EC2 インスタンスをプロビジョニングするテンプレートは、次のようになります。

<例：東京リージョンかオレゴンリージョンで NAT インスタンスを起動する>

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
    "Parameters" : {
          "MyKeyPair" : {
                "Description" : "Key Pair Name",
                "Type" : "String"
          },
          "MyInstanceType" : {
                "Description" : "EC2 Instance Type",
                "Type" : "String"
                "Default" : "t2.micro",
                "AllowedValues" : ["t2.micro", "t2.small"]
          }
    }
    "Mappings" : {
          "AWSRegionToAMI" : {
                "ap-northeast-1" : {
                      "AMI" : "ami-12345678"
                },
                "us-west-2" : {
```

```
                        "AMI" : "ami-abcdefgh"
                }
    },
    "Resources" : {
          "NATInstance" : {
                "Type" : "AWS::EC2::Instance",
                "DependsOn" : [ "RDSInstance"],
                "Properties" : {
                        "ImageId" : {
                              "Fn::FindInMap" : [
                                    "AWSRegiontoAMI",
                                    {
                                      "Ref" : "AWS::Region"
                                    },
                                    "AMI"
                              ]
                        },
                        "InstanceType" : {"Ref" : "MyInstanceType"},
                        "KeyName" : {"Ref" : "MyKeyPair"},
                        "NetworkInterfaces" : [
                        {
                              "DeviceIndex" : "0",
                              "AssociatePublicIpAddress" : "true",
                                    ⋮
                        }
                        ],
                        "SourceDestCheck" : "false",
                        "UserData" : { "Fn :: Base64" : { "Fn :: Join" : ["", [
                              "#!/bin/bash ¥n",
                              "yum -y update ¥n",
                                    ⋮
                              "/opt/aws/bin/cfn-signal -s true '",
                              {
                                    "Ref" : "WaitHandle1"
                              },
                                    ⋮
```

利用者がスタックを作成する都度指定する項目については、Parameters セクションで規定でき、上記の例では、キーペアとインスタンスタイプをスタック作成時に指定できます。

インスタンスタイプについては、t2.micro か t2.small を選択します。AMI ID のように、リージョン固有の値を持つリソースをテンプレートに記載する際、どのリージョンでも同じテンプレートを利用できるように、リージョンごとの値（ここでは AMI ID）を返すマッピングテーブルを作成しておき、リソース作成時に値を参照します。こうすることで、東京リージョンでスタックを作成すれば東京リージョンの AMI ID が参照され、オレゴンリージョンでスタックを作成すればオレゴンリージョンの AMI ID が参照されます。なお、CloudFormation はスタックの作成途中で指定されている AMI が見つからないなどのエラーが発生すると、デフォルトでロールバックし、そこまで作成したすべてのリソースを削除します。

CloudFormation は、リソースの依存関係などを判断して、できるだけ並列にリソースを起動するため、利用者が明示的に作成順序を指定したい場合には DependsOn セクションで規定します。上記の例では、テンプレートの別の場所に記載のある RDS インスタンスを作成してから NAT インスタンスを作成します。また、テンプレートにはユーザデータを記述でき、ソフトウェアのインストールなど、起動時に行う様々な処理を指定できます。ユーザデータの処理に時間がかかりそうな場合、ユーザデータの最後にシグナルを送る処理を記載し、そのシグナルを受け取ってから残りのスタックの作成を続行するという指定も可能です。テンプレートには条件を記載することもでき、スタック作成時に選択した条件によって、例えば本番／開発／検証環境を作成することができ、環境ごとにテンプレートを個別に作成する必要はありません。

試験のポイント！

CloudFormation で設定できる項目や、エラー発生時の動きなどを押さえる！

11-2 Elastic Beanstalk／OpsWorks

AWS Elastic Beanstalk（以下 Elastic Beanstalk）は、アプリケーションのデプロイツールです。これを使うことで、開発者が自身で開発環境を AWS 上に構築し、そこにアプリケーションをデプロイ（配置）することができます。構築できる AWS リソースは、次の通りです。

- ELB
- EC2 インスタンス（Auto Scaling Group）
- S3 バケット
- RDS（オプション）

など

開発者は、Elastic Beanstalk でアプリケーションのバージョン管理ができ、既存の環境を以前のバージョンに戻すことができます。CloudFormation と同様に、Elastic Beanstalk 自体の料金は無料で、プロビジョニングされたリソースの利用料金のみ発生します。

AWS OpsWorks（以下 OpsWorks）は、AWS 上のアプリケーションサーバの構成管理ツールで、ELB や EC2 インスタンスを作成し、その後に Chef のレシピを実行してソフトウェアのインストールや設定などを自動化できます。

前述の Elastic Beanstalk は、アプリケーション管理プラットフォームで、開発者がコーディングしたアプリケーションを簡単にデプロイ／管理できるサービスであり、サーバの構成管理に焦点を当てた OpsWorks とは異なります。また、11-1 で紹介した CloudFormation は、AWS 上に VPC（ネットワーク）から構築できる、インフラストラクチャ部分のプロビジョニングサービスであり、やはりサーバの構成管理に焦点を当てた OpsWorks とは異なります。

Elastic Beanstalk や OpsWorks は CloudFormation から呼び出すことができ、CloudFormation と Elastic Beanstalk を組み合わせて、VPC を含めたインフラ部分の作成からアプリのデプロイまでを自動化したり、CloudFormation と OpsWorks を組み合わせて、アプリの設定までを自動化することができます。

試験のポイント！

CloudFormation／Elastic Beanstalk／OpsWorks のユースケースを押さえる！

章末問題

Q1 CloudFormation で実施できることとして、誤っているものはどれか？

- ○ **A** インフラストラクチャをコードとして記述でき、バージョン管理できる。
- ○ **B** リソースを作成するリージョンごとに異なるテンプレートを作成する必要がある。
- ○ **C** 本番環境と開発環境で EC2 インスタンスの台数が異なるが、1 つのテンプレートで本番環境と開発環境の設定が記述できる。
- ○ **D** CloudFormation で作成したリソースを一括して更新／削除できる。

Q2 Q2.CloudFormation の特徴として、正しいものはどれか？

- ○ **A** CloudFormation はテンプレートに記述された順番にリソースを作成していくため、依存関係のあるリソースは記載順序に気をつける。
- ○ **B** CloudFormation でスタックの作成途中にエラーが発生した場合、デフォルトでは、たとえそれまでに課金が発生するリソースが起動していたとしても、そのリソースを削除してロールバックする。
- ○ **C** CloudFormer というツールを利用し、作成したテンプレートに間違いがないかを確認することができる。
- ○ **D** CloudFormation は EC2 インスタンスや RDS インスタンスなどの実体を作成するツールであり、VPC は作成できないため、事前に VPC を作成しておく必要がある。

Q3 CloudFormation／Elastic Beanstalk／OpsWorks の使い方として、適切なものはどれか？

- ○ **A** CloudFormation のテンプレートにバージョン番号をつけ、アプリケーションのバージョンアップに合わせてスタックの更新を行う。
- ○ **B** 複数のリージョンで本番環境とは異なる VPC で開発環境と検証環境を作成するため、Elastic Beanstalk を利用して環境をデプロイする。
- ○ **C** OpsWorks から Chef のレシピを実行し、ELB の配下に Auto Scaling 設定がされた EC2 インスタンスが配置される構成を作成する。
- ○ **D** Web-DB 連携アプリケーション開発環境を複数用意するため、

CloudFormation で ELB と EC2 インスタンスと RDS インスタンスを作成し、EC2 インスタンスに必要なソフトウェアをインストールする。CloudFormation から OpsWorks を呼び出して、アプリケーションソフトウェアの接続先の DB として RDS を設定する。

答え

A1 B

マッピングテーブルを利用することで、リージョンごとに異なる項目／値をスタックが作成されるリージョンに合わせ、1 つのテンプレート内で指定することができます。

A2 B

A　CloudFormation は、テンプレートの記載順ではなく、並列で作成できるリソースは並列に作成していくため、依存関係があるリソースを作成する際は、DependsOn 属性を指定する必要があります。

C　CloudFormer は、現在のアカウント上で作成されているリソースを元にテンプレートを作成するツールです。

D　CloudFormation は、VPC を含んだほぼすべての AWS リソースを作成することができます。

A2 D

A　アプリケーションの管理には、Elastic Beanstalk が適しています。

B　VPC からリソースを作成するには、CloudFormation を利用します。

C　Chef のレシピを実行して構成を管理するのは AWS リソースの設定ではなく、アプリケーションの設定です。

第 12 章

EC2 の料金モデル（オンデマンドインスタンス／リザーブドインスタンス／スポットインスタンス）

EC2 の料金モデルについて

AWS の特徴／メリットとして、利用した分だけコストが発生する従量課金制があり、これにより、IT リソースにかかるコストを抑えることができます。しかし、サーバの中には 24 時間 365 日稼働し続けるものもあり、そういった用途向けに年間契約の料金モデルも AWS では用意しています。さらに、需要と供給のバランスに応じて価格が設定される料金モデルも用意されており、これらの料金モデルを組み合わせることで、サービスを十分な IT リソースで提供しながら、IT リソース全体のコストを最適化することができます。

認定試験では、料金モデルの最適な組合せが出題されます。本章では、EC2 の料金モデルであるオンデマンドインスタンスとリザーブドインスタンス、そしてスポットインスタンスについて説明します。

12-1 オンデマンドインスタンス

オンデマンドインスタンスは、EC2インスタンスのデフォルトの課金方式で、EC2インスタンスが起動している時間だけ、1時間単位で支払いが発生します。長期契約は不要で、必要なときに、必要なだけEC2インスタンスを用意することでコストを抑えることができ、システムの負荷の増減に対応できます。

オンデマンドインスタンスの1時間あたりの料金は、次の3つの要素で決まります。

- リージョン
- インスタンスタイプ
- OS (Amazon Linux／RHEL／Windows Server など)

オンデマンドインスタンスは、開発／検証環境のサーバや、Auto Scalingグループで増減するサーバなど、1年を通して常時稼働することが求められていない用途での利用が向いています。

12-2 リザーブドインスタンス

リザーブドインスタンスは1年あるいは3年契約を結ぶことにより、オンデマンドインスタンスよりも割安にEC2インスタンスやRDSインスタンス、ElastiCacheノードやRedshiftノードを利用できる課金方式で、**RI** (Reserved Instance) と略した名称で呼ばれることがあります。DynamoDBやCloudFrontについても同様の割引方式がありますが、こちらはキャパシティ(テーブル容量やデータ転送量)を事前に予約するリザーブドキャパシティといいます。本書では、EC2のリザーブドインスタンスを元にして説明します。

リザーブドインスタンスでは、EC2インスタンスの起動／停止に関わらず、

利用料金が発生します。料金の支払い方法と契約期間は、表12-2-1のとおりです。

表12-2-1　リザーブドインスタンスの料金の支払い方法と契約期間

支払方式	契約期間
全額前払い	1年あるいは3年
一部前払い	1年あるいは3年
前払いなし	1年

オンデマンドインスタンスに対する割引率は、支払方式別では「前払いなし」、「一部前払い」、「全額前払い」の順で高く、契約期間別では「1年」、「3年」の順で高く、最大で75%ほどの割引になります。全額前払いの場合、1年あるいは3年分のリザーブドインスタンスの料金を一括で前払いします。一部前払いの場合は少額を前払いし、その後の契約期間中、リザーブドインスタンスの割引利用単価に支払い月の時間数(24時間×支払い月日数)をかけた額を支払います。前払いなしについても、毎月の支払は一部前払いと同様ですが、割引利用単価が一部前払いよりも高くなります。

補足　2014年11月以前は、リザーブドインスタンスには「軽度使用RI」「中度使用RI」「重度使用RI」という3種類の支払い方式がありましたが、2014年12月より、現在のシンプルな支払い方式に変わりました。現在の「一部前払い」が以前の「重度使用RI」に相当します。

例えば、東京リージョンでAmazon Linux、m4.largeの場合のリザーブドインスタンスの料金とオンデマンドインスタンスに対する割引率は、表12-2-2のようになります。(2016年4月現在)

表12-2-2　リザーブドインスタンスの料金とオンデマンドインスタンスに対する割引率

期間	支払い方式	前払い	毎月平均	実質的時間単価	オンデマンドに対する割引率	オンデマンド(毎時)
1年	全額前払い	$799	$0	$0.0912	48%オフ	$0.174
	一部前払い	$408	$34.31	$0.0936	46%オフ	
	前払いなし	$0	$79.57	$0.109	37%オフ	
3年	全額前払い	$1666	$0	$0.0634	64%オフ	$0.174
	一部前払い	$886	$24.09	$0.0667	62%オフ	

オンデマンドインスタンスは、起動していた時間だけ課金が発生しているため、稼働率が低ければ1年あるいは3年間の合計利用料金がリザーブドインスタンスよりも低くなります。一般に、年間稼働率が70%を超えるようであれば、リザーブドインスタンスの方が料金を低く抑えることができると言われています。また、1年契約よりも3年契約の方が割引率が高くなりますが、3年間の契約期間中に新しい世代のインスタンスタイプが発表されたり、現行インスタンスの値下げが行われてもリザーブドインスタンスはその恩恵を受けることができません。

リザーブドインスタンスは、利用料金の割引だけではなく、ITリソースキャパシティの予約（リザーブ）にもなります。AWSのITリソースの需要が高まっても、リザーブドインスタンスとして予約しているインスタンスは、いつでも確実に起動できます。

12-3 スポットインスタンス

スポットインスタンスは、入札形式のEC2インスタンスの利用／支払い方式で、需要と供給のバランスによって決まるスポット価格（市場価格）を入札価格が上回ると、EC2インスタンスを利用できます。スポット価格は、次の3つの項目ごとに設定されています。

- アベイラビリティゾーン（AZ）
- インスタンスタイプ
- OS（Amazon Linux／SUSE Linux／Windows Serverなど）

各AZの各インスタンスタイプの需要に基づいて、そのAZ／インスタンスタイプ／OSのスポット価格が決まります。スポット価格は需要と供給のバランスによって変動し、最大でオンデマンドの90%近い割引価格になります。これから新規に起動するインスタンスであれば、入札価格がスポット価格を上回るとスポットインスタンスが起動します。そのとき、実際のスポットインスタンスの利用価格は入札価格ではなく、スポット価格になります。一方、

既にある入札価格で起動中のスポットインスタンスがあり、その入札価格をスポット価格が上回った場合、インスタンスはターミネート（終了）されます。そのため、スポットインスタンスは、計算クラスタノードの一部や、Auto Scalingの増加分のインスタンスなど、スポットインスタンスが突然削除されてしまっても問題ないところに利用します。また、スポットインスタンス上のデータについては、頻繁にチェックポイントを設けてS3やEBS、DynamoDBといった不揮発性のストレージに書き出す必要があります。あるいは、スポットインスタンスのターミネートは2分前にEC2インスタンスのメタデータに通知されるため、その通知をトリガーにして外部ストレージに書き出します。なお、スポット価格の高騰によってスポットインスタンスがターミネートされた場合、その最後の1時間分の利用料金は発生しません。

補足　2015年10月にスポットブロックというオプションがリリースされました。スポットブロックオプションを有効にしたスポットインスタンスは、1〜6時間の間で、スポット価格の変動にかかわらず継続して利用することができます。ブロック価格はスポット価格よりも少々高めですが、オンデマンド価格よりは低額です。

重要！

スポットインスタンスは、単独で使用するのではなく、オンデマンドやリザーブドインスタンスと組み合わせて利用する！

試験のポイント

業務（提供サービス）の継続とEC2のコスト最適化の両方を考慮して、オンデマンド／リザーブド／スポットインスタンスそれぞれのユースケースを押さえる！

章末問題

Q1 24時間365日サービスを提供するチケット販売Webシステムを ELBと配下のAuto Scalingグループの EC2インスタンス、及びRDS で構成している。初期EC2インスタンス数は2台だが、チケット販売開始時刻にはアクセスが集中するため、販売開始時刻の5分前にもう2台手動で追加しておき、販売開始後はAuto Scalingで自動拡張するように設定した。リザーブドインスタンスの使いどころとして適切なものはどれか？

- ○ **A** 初めから起動している2台のインスタンス
- ○ **B** 販売開始時刻の5分前に追加する2台のインスタンス
- ○ **C** 販売開始後にAuto Scalingによって追加されるインスタンス
- ○ **D** このシステムにリザーブドインスタンスを使う必要はない。

Q1 24時間365日サービスを提供するチケット販売Webシステムを ELBと配下のAuto Scalingグループの EC2インスタンス、及びRDS で構成している。初期EC2インスタンス数は2台だが、チケット販売開始時刻にはアクセスが集中するため、販売開始時刻の5分前にもう2台手動で追加しておき、販売開始後はAuto Scalingで自動拡張するように設定した。スポットインスタンスの使いどころとして適切なものはどれか？

- ○ **A** 初めから起動している2台のインスタンス
- ○ **B** 販売開始時刻の5分前に追加する2台のインスタンス
- ○ **C** 販売開始後にAuto Scalingによって追加されるインスタンス
- ○ **D** このシステムにスポットインスタンスを使う必要はない。

答え

A1 A

Bはオンデマンドインスタンスが適しています。
Cはスポットインスタンスが適しています。

A2 C

Q1.と同じです。

索引

■著者プロフィール

大塚 康徳（おおつか やすのり）

株式会社日立インフォメーションアカデミー　システム研修部所属
AWS Authorized Instructor
AWS 認定ソリューションアーキテクト–プロフェッショナル

日立インフォメーションアカデミーに入社後 UNIX や Linux、オープンソースクラウド基盤の研修を担当。豊富な AWS 導入経験を有する日立ソリューションズに出向し、AWS の導入案件に従事。提案から設計、構築までを担当し AWS の 7 つのベストプラクティスの重要性を肌で感じる。
その後 AWS の認定インストラクターである「AWS Authorized Instructor」の認定を取得し、「Amazon Web Services 実践入門 1」「Amazon Web Services 実践入門 2」「Architecting on AWS」「Systems Operations on AWS」を担当。現在に至る。
休日の昼間から呑むビールが楽しみ。

■日立インフォメーションアカデミーについて

株式会社日立インフォメーションアカデミーは研修の企画支援から開発支援、研修実施、研修定着支援までを提供する「人財育成のトータルソリューション」企業です。
「AWS 認定トレーニングパートナー」として入門レベルだけではなく、AWS 認定アソシエイトレベルに対応するすべての AWS の認定トレーニングを提供しています。
https://www.hitachi-ia.co.jp/