EC to や IELB など、 AWS リソースの環境構築を設定ファイルをもとに自動化できるサービスです。うん。設定ファイルのことをテンプレートと呼び、 JSON や YAML 形式で構築したい AWS リソースの設定内容を記述します。うん。テンプレートで定義され、クラウドフォーメーションによって作成された AWS リソースの集合体を STAC という単位でまとめて管理することができます。はい。また、クラウドフォーメーションは追加料金なしでごいただけます。クラウドフォーメーションは、インフラストラクチャーズコード IAC を実現するサービスです。インフラストラクチャをコードで定義・作成し、コードの変更を通じてインフラストラクチャを更新することで、複製や再構築、再利用が容易になり、エラーの原因となるような手動のオペレーションを排除することができます。

のアプローチを用いることで、ソフトウェアの開発手法をインフラストラクチャに適用できるようになります。例えば、エディターの機能を使った構文チェックや、静的解析ツールを使ってインフラのセキュリティチェックを実施できるようになったり、バージョン管理システムを使ってデプロイ前に行動をレビューするといったことができるようになります。

クラウドフォーメーションの基本機能について、 AWS リソースの作成と変更、削除のシーンに分けてご紹介します。 EC to や Rdselb など、必要とする AWS リソースを含むテンプレートを作成すれば、そこに書かれている定義に従ってクラウドフォーメーションが自動的に作成や設定をします。リソース間に依存関係がある場合は、クラウドフォーメーションが自動的に解決します。メリットとしては、構築作業を迅速化できること、手作業によるオペレーションミスを排除できること、そして、一度テンプレートを作成すれば、検証環境の作成や別リージョンへの展開など。環境の複製にかかる時間を効率化できることです。

次に変更機能ですが、同じテンプレートを何度使用しても同じ結果が得られる適当性が担保されています。つまり、クラウドフォーメーションが現在のインフラの状態とテンプレートに示されているあるべき姿を確認して債を埋めるように働きますので、結果としてインフラが目的の状態に収束します。

マネジメントコンソールで設定を変更するときと同じように、変更の内容によってはリソースの再起動や再作成が伴いますが、うした影響があるかどうか、変更セットと呼ばれる変更前後の差分情報で事前に確認することができます。うん。スタックの変更によってアプリケーションに何らの障害が発生すると、クラウドフォーメーションは前回の状態にスタックをロールバックします。うん、リソースの変更時発生しうる更新動作は、こちらにお示ししている通りの 3 種類です。 EC の設定変更を例に挙げてまとめておりますのでご確認ください。最後に削除ですが、依存関係を解決しつつ、スタックに含まれるすべてのリソースを一括で削除できます。うん、 RDS のようなデータストアは、削除前にスナップショットを取得したり、スタックの管理下から切り離し保持することもできます。うん。

クラウドフォーメーションの基本機能が分かったところで、次にクラウドフォーメーション特有の用語についてご説明いたします。うん、テンプレートスタック偏光セットですまずは、テンプレートついてご説明いたします。テンプレートはクラウドフォーメーションの心臓部で、スタックの設計図と呼べるものです。どのリソースをどのような設定で起動するのか記述します。詳細設計書やパラメータシートに記載する項目と似ているなとお感じになられる方もいらっしゃるのではないでしょうか。記述方法はJSON、 YAML の両方に対応しています。

次はスタックです。スタックとは、単一のユニットとして管理できる AWS リソースの集合体です。スタック内のリソースを一度に作成・削除が可能で、リソース間に依存関係がある場合は自動ティサイト削除の順番を判断し。します。変更の際、操作が行われるのは現在の状態とテンプレートで定義された状態に違いのあるリソースだけで、変更操作の必要のないリソースについては操作を行いません。

最後に変更セットについてご説明します。スタックを更新することで、現在稼働しているリソースにどのような影響が出るのか事前に確認することができます。この影響を確認し、スタックの更新を続行するか、または別の変更セットを作り直すか検討することができます。ここまでで、基本機能と用語について押さえていただきました。ここからは、実際にクラウドフォーメーションを使った環境構築の流れを見ていきたいと思います。こちらが構築の流れの全体像です。まず、ヤムルジェイソンでテンプレートを記述し、そのテンプレートをアップロードします。その後、マネジメントコンソールでスタックを作成すると、エリソースが作成され、以降はスタックの更新を通じて AWS リソースを管理・運用していきます。それでは、これらの流れを詳しく見ていきましょう。

公式ドキュメントを参考に、任意のエディターでテンプレートを記述します。こちらのスライドには、エディターの一例として、ビジュアルスタジオコードとクラウドフォーメーションデザイナーを紹介しています。うん、テンプレートに記述する内容については後述いたします。

テンプレートが完成したら、マネジメントコンソールでローカルのファイル、つまりテンプレートをアップロードします。すでに S にアップロードされている場合はそちらを選択することもできます。アップロードしたら、マネジメントコンソールでスタックの作成手順に進みます。作成と直前のレビューページで設定した内容を確認します。テンプレートに IAM リソースが含まれている場合は、それを知る警告を承認してスタックを作成してください。このとき、 IAM リソースに関連付けられた権限が意図したものであるかどうか確認することをお勧めします。

うん。スタック作成処理が完了すると、図のようにリソースタブでリソースの ID やステータスを確認することができます。うん。とこれ手動で作成し、稼働中の AWS リソースをクラウドフォーメーションで管理したくなった場合はどうすればいいのか疑問に思われる方がいらっしゃるかもしれません。

クラウドフォーメーションには、インポート機能というものがあり、インポートしたいリソースの実際の設定と一致するようにテンプレートを定義すれば、後からクラウドフォーメーションのスタックにインポートできます。このインポートにかかる追加料金はありません。また、 AWS 公式のものではありませんが、フォーマーツーというツールを使って既存の AWS リソースを読み取り、テンプレートを生成することができますのでお試しください。

SPK 0。

AWS クラウドフォーメーションを使用して、テンプレートに記述したリソースを作成する際、あるリソースの 1 つ以上のリソースに依存する場合があります。これをリソースの依存関係と言います。

なお、一般的にリソースの構築順は、テンプレートで定義したリソース間の依存関係から、クラウドフォーメーションが自動的に決定します。例えば、 EC インスタンスは、 EC インスタンスに使用するセキュリティグループに依存します。クラウドフォーメーションスタックにおいて、 EC インスタンスがセキュリティグループを参照するように記述することで、最初にセキュリティグループが作成され、次に EC インスタンスが作成されます。また、テンプレートで定義したリソース間に依存関係がない場合、クラウドフォーメーションはリソースの作成を並行して開始します。なお、デペンジョン属性または組み込み関数を利用することで、リソースの依存関係を指定することが可能です。ディペンゾーン属性を使用してリソースの作成順序を定義した場合、クラウドフォーメーションはリソースを定義された順序へ処理します。うん。また、 if 関数、GET、 AT 関数などの組み込み関数を使用して、プロパティの値として別のリソースを参照すると、参照先のリソースから作成することが可能です。

では、ディペンゾーン特性および組み込み関数によるリソースの依存関係について、もう少し詳しく見ていきましょう。まずはじめに、ディベンジョン属性によるイメージ的な依存関係についてご説明いたします。 Dipenzon 属性を使用することで、明示的にリソースの依存関係を定義することが可能です。例えば、左下のテンプレートでは、上にエスリーバケットを、下に SNS トップを定義しています。うん、また、 SNS トピックの利便ゾーン属性には、エススリーバケットを定義しています。そうすることで、右下のイベントのように、クラウドフォーメーションスタックはエスリーバペットの作成が完了するのを待ってから SNS トピックの作成を開始し。ます。

次に、組み込み関数による暗黙的な依存関係についてご説明します。 TIF 関数や getat 関数など、組み込み関数を使用して別のリソースを参照することで、暗黙的に依存関係を定義することが可能です。まずは if 関数について例えを用いて説明いたします。 AWS SNS サブスクリプションリソースタイプを作成する場合、プロパティのトピックアーンにはサブスクライブするトピックの Amazon リスイムが必要となります。その際に、 if 関数を使用して AWS SNS トピックリソースタイプを参照すると、 SNS トピックの Amazon リソースネームを取得することが可能です。なお、クラウドフォーメーションは FIF 関数で Amazon リソース電話を取得するために、 SNS トピックの作成が完了するのを待ってから、 SNS トピックサブスクリプションの作成を。紹介します。このように LIF 関数を用いることで、暗黙的にリソースの作成順序を定義することが可能です。うん。

なお、スタックを削除する場合には、削除される順序が作成される順序とは逆になります。今回の場合では、最後に作成された SNS トピックサブスクリプションが最初に削除されます。その後に SNS トピックが削除される運びとなります。次に、 getat 関数について例を用いてご説明いたします。 getat 関数も lif 関数同様に、参照されるリソースがクリエイトコンプリートステータスになってから、参照するリソースの作成が開始されます。例えば、 AWS EC セキュリティグループイングレスリソースタイプを作成する場合には、プロパティのグループ ID にセキュリティグループの ID が必要となります。その際に、 AWS EC ツールセキュリティグループリソースタイプは、 GET AT 関数を使用してグループ ID 特性を参照すると、セクシュリティグループの ID を参照することが可能です。

なお、クラッドフォーメーションは、 getat 関数でセキュリティグループの ID を取得するために、セキュリティグループの作成が完了するのを待ってからセキュリティグループイングレスの作成を開始します。うん、このように getat 関数を用いることでも、暗黙的にリソースの複製順序を定義することが可能です。なお、スタックを削除する場合には、 RIF 関数と同様に、削除される順序が作成される順番と逆になります。今回の場合では、最後に作成されたセキュリティグループイングレスが最初に削除されます。その後にセキュリティグループが削除される運びとなります。

次に動的参照についてご説明いたします。クラウドフォーメーションの動的な参照を利用することで、 AWS システムズ・マネージャー、パラメーター・ストア、 AWS シークレッツ・マネージャーを含む AWS サービスに保存されている値を参照することが可能です。

うん。 AWS Systems マネージャーではパラメータストアに格納されている平文で保存されているデータ、または暗号化されているデータを参照することが可能です。一方、 AWS シークレッツマネージャーでは、保存されているシークレット情報を参照することが可能です。

うん。それでは AWS システムズマネージャーおよび AWS シークレッツマネージャーにて動的参照を行う場合の考慮事項などについて見ていきましょう。まずは、 SSM パラメータストアにて保存されている平文のデータを自動的参照する場合についてご説明いたします。うん。例では、 SSM の動的な参照を使用して、 SD バケットのアクセス制御を SSM パラメータストアに格納されているパラメータ値に設定します。

今回のテンプレートでは、クラウドフォーメーションは、スタック操作およびエンコンセット操作とにバージョン 2 のエススリーアクセスコントロールパラメータを使用します。また、特出する考慮事項といたしましては、 4 点ございます。中でも、はじめに記載しておりますように、 SSM のパラメータ内容を変更した場合は、クラウドフォーメーションにパラメータの変更を反映させるためにスタックの更新が必要になること。および次に記載しておりますように、バージョンを指定しない場合、スタックを作成・更新するたびに最新バージョンのパラメータを使用することにご留意なさい。

次に、 SSM パラメータストアにて暗号化され保存されているデータを動的参照する場合についてご説明いたします。うん。例では、 IAM ユーザーパスワードを SSM パラメータストアに格納されている安全のモデルと。に設定しています。今回のテンプレートでは、クラウドフォーメーションは、スタック操作および変更セット操作にバージョン 10 の IAM ユーザーパスワードパラメータを使用します。特設する考慮事項といたしましては、 4 点ございます。また、記載がございますように、 SSM セキュアストリングパラメータは、サポートしているリソースプロパティに対してのみ使用可能となっております。こちらが 2023 年 11 月現在でサポートしている 1 つの一覧でございます。最後に AWS シークレッツマネージャーのシークレットを動的に参照する方法についてご説明いたします。はい。まずはじめに、 AWS シークレットマネージャーの場合には、 AWS システムズマネージャーとは違い、シークレットID、シークレットの値、キー、バージョン、ステージ。およびバージョン ID を含む複数のセグメントで構成されています。うん。

なお、バージョンステージおよびバージョン ID を使用しない場合には、デフォルトで AWS カレントというバージョンが指定されます。それでは、同様に d を用いて、 AWS シークレッツマネージャーについて見ていきましょう。例では、 Myrds シークレットシークレットに保存されているユーザー名、パスワードの値を取得するために、シークレット ID セグメントと JSON キーセグメントを使用しています。なお、今回のテンプレートではバージョンステージ及びバージョン ID を指定していないため、取得されるシークレットのバージョンは AWS カレントの値となり。ます。

次はゲストされたスタックについてご説明いたします。ネストされたスタックを使用することで、大きいクラウドフォーメーションテンプレートを再利用可能な小さなテンプレートに分割することができます。そして、他のテンプレート内から分解した小さなテンプレートを参照することでスタックを作成することが可能です。うん。なお、ネストされたスタックに対しても、他のスタックをネストすることが可能です。

次に、ネストサイトスタックにて使用される用語について説明します。テストされたすべてのスタックが属している最上位のスタックをルートスタックと呼びます。また、ネストされたスタックにはそれぞれ直接の親スタックが存在します。図を用いて説明いたしますと、スタック B から見ると、 Stack A は親スタックであると同時に、ルートスタックでもあります。うん、またスタック E から見ると、スタック D が親スタックとなります。

具体的なニュースケースとしては、複数のスタックに使用しているリソースの構成がある場合、同じ構成のテンペットを複数用意するのではなく、使用している縫製の専用のテンプレートを作成し、再利用することが考えられます。また、大きなテンプレートを分割するため、リソース数の制限を回避することも可能です。なお、ネストされた Stack のテンプレートをあらかじめ SD バケットに保存しておく必要があることにご留意ください。うん。例えば次あるように、VPC、IAM、 EC それぞれのリソースを作成する小さなテンプレートを用意し、あらかじめ S バケットに保存しておくことで、親となるスタックの一部のスタックとして作成することができます。

うん、それでは親スタックのテンプレートにどのようにネストされたスタックを参照するのか見ていきましょう。まずは、 AWS クラウドフォーメーションスタックというソースタイプにて、コスタックを定義する必要があります。うん。そして、テンプレート URL プロパティにて、 S 林にあらかじめ保存しておいたネストされたスタックのテンプレートの URL を指定します。またば、パラメータ図プロパティを使用することで、ネストされた Stack のテンプレートにパラメータを渡すことが可能です。なお、 GET AT 関数を利用することで、他のリストされたスタックのテンプレートのアウトプットセクションで出力した値を参照することもできます。

続いて、クロススタック参照についてご説明します。クロロススタック参照は、あるスタックでデプロイしたリソースを他のスタックで参照したい場合に使用される機能です。うん。なお、スタックを分けることで、スタックの役割やスタックの管理責任などを分けることにもつながります。うん。例えば 1 室インスタンスを作成したい場合に、セキュリティチームが管理する IOM 用のスタック。うん、ネットワークチームが管理する VPC 用のスタックを参照するといったことが可能です。ただし、クロススタック参照を利用する上で 3 つほど留意点がございます。 1 つ目に、スタックを分けて管理することが可能ですが、その分管理が煩雑になる可能性がございます。うん。

次に、別のスタックによって参照されている出力値は変更を削除することができません。最後に、 SAC の出力値が参照されている場合、そのスタックを削除することができません。それでは早速クロスタック参照の使い方について見ていきましょう。まず、参照したいリソースを持っている参照される側のスタックでは、アウトプットセクションのエクスポートフィールドに値を指定します。うん。次に参照する側のスタックでは、組み込み関数インポートバリュー関数にて取得したい値を指定することで参照することが可能です。うん。

さて、これまでリストされたスタックおよびクラスタック参照についてご説明いたしました。ここで、一度ネストされたスタックとクロススタック参照の違いについて確認してみましょう。テストされたスタックは、テストされたスタックのグループ内でのみパラメータなどの情報を共有したい場合に使用することを推奨しております。うん。一方でクロスタック参照では、ネストされたスタックのグループ内に限らず、他のスタックと情報を共有したい場合に使用することを推奨しておりますし、では最後にスタックセッツについて見ていきましょう。

まず、スタック設置の概要といたしましては、 1 つのテンプレートを使用して、複数の AWS アカウント、複数のリージョンに対して同様のスタックを作成、更新、削除することが可能な機能です。次に、スタック設置でよく使用される用語を 4 つご説明いたします。 1 つ目に、管理社運と、これはスタックセットを作成する AWS アカウントです。 2 つ目、ターゲットアカウント。こちらはスタックセットの 1 つ以上のスタックを作成、更新、削除する AWS アカウントです。うん。次にスタックセットは、スタックの作成に使用するテンプレート及びパラメーター、スタックを作成するターゲットアカウントリージョンドなどの定義を指しています。最後に、スタックインスタンス。これは、ターゲットアカウントのスタックへのリファレンスです。スタックがない場合にも存在することが可能となっております。なお、スタックインスタンスは 1 つのスタックセットにのみ関連付けられます。

えっ。  
次は、ゲストされたスタックについてご説明いたします。ネストされたスタックを使用することで、大きいクラウドフォーメーションテンプレートを再利用可能な小さなテンプレートに分割することができます。うん。そして、他のテンプレート内から分解した小さなテンプレートを参集することで、スタックを作成することが可能です。うん。なお、テストされたスタックに対しても他のスタックをリストすることが可能です。

次に、ネストサイトスタックにて使用される用語について説明します。テストされたすべてのスタックが属している最上位のスタックをルートスタックと呼びます。また、ネストされたスタックにはそれぞれ直接の親スタックが存在します。図を用いて説明いたしますと、 Stack B から見ると Stack A は親スタックであると同時に、 Rootstack でもあります。うん。また、 Stack E から見ると、 Stack D が親 STAC となります。

具体的なユースケースとしては、複数のスタックに使用しているリソースの構成がある場合、同じ構成のテンプレートを複数用意するのではなく、使用している構成の専用のテンプレートを作成し。再利用することが考えられます。また、大きなテンプレートを分割するため、リソース数の制限を回避することも可能です。

なお、 REST された Stack のテンプレートをあらかじめ SD バケットに保存しておく必要があることにご留意ください。はい。例えば図にあるように、VPC、 imec それぞれのリソースを作成する小さなテンプレートを用意し、あらかじめエスリーバケットに保存しておくことで、親となるスタックの一部のスタックとして作成することができます。うん、それでは親スタックのテンプレートにどのようにネストされたスタックを参照するのか見ていきましょう。まずは、 AWS クラウドフォーメーションスタックというソースタイプにてコスタックを定義する必要があります。はい。そしてテンプレート URL プロパティにて、レスリンにあらかじめ保存しておいたテストされたスタックのテンプレートの URL を指定します。またはパラメーターズプロボティを使用することで、ネストされた Stack のテンプレートにパラメータを渡すことが可能です。はい。なお、 getat 関数を利用することで、他のテストされた Stack のテンプレートのアウトプットセクションで出力した値を参照することもできます。

続いて、クロススタック参照についてご説明します。クロススタック参照は、あるスタックでデプロイしたリソースを他のスタックで参照したい場合に使用される機能です。はい。なお、スタックを分けることで、スタックの役割やスタックの管理責任などを分けることにもつながります。たとえば、 EC インスタンスを作成したい場合に、セキュリティチームが管理する IOM 用のスタックはいネットワークチームが管理する VPC 用のスタックを参照するといったことが可能です。ただし、クロススタック参照を利用する上で 3 つほど留意点がございます。 1 つ目に、スタックを分けて管理することが可能ですが、その分管理が煩雑になる可能性がございます。うん。

次に、別のスタックによって参照されている出力値は変更を削除することができません。最後に、スタックの出力値が参照されている場合、そのスタックを削除することができません。それでは早速クロスタック参照の使い方について見ていきましょう。まず、参照したいリソースを持っている参照される側のスタックでは、アウトプットセクションのエクスポートフィールドに値を指定します。うん。次に参照する側の Stack では、組み込み関数、インポートバリュー関数にて取得したい値を指定することで参照することが可能です。

さて、これまで REST されたスタックおよびクラスタック参照についてご説明いたしました。ここで、一度ネストされたスタックとクロススタック参照の違いについて確認してみましょう。 REST された Stack はネストされた STAC のグループ内でのみパラメータなどの情報を共有したい場合に使用することを推奨しております。一方で、クロスタック参照では、ネストされたスタックのグループ内に限らず、他のスタックと情報を共有したい場合に使用することを推奨しております。

では最後にスタックセットについて見ていきましょう。まず、 Stacksets の概要といたしましては、 1 つのテンプレートを使用して、複数の AWS アカウント、複数のリージョンに対して同様のスタックを作成、更新、削除することが可能な機能です。うん。

次に、 Stacksets でよく使用される用語を 4 つご説明いたします。 1 つ目に、管理者アカウント、これはスタックセットを作成する AWS アカウントです。 2 つ目にターゲットアカウント。こちらは、スタックセットの 1 つ以上のスタックを作成、更新、削除する AWS アカウントです。次に、 Stack セットは、 stac の作成に使用するテンプレートおよびパラメータスタックを作成するターゲット、アカウント、リージョンなどの定義を指しています。

最後に、スタックインスタンス。これはターゲットアカウントの Stack へのリファレンスです。スタックがない場合にも存在することが可能となっております。なお、スタックインスタンスは 1 つのスタックセットにのみ関連付けられます。ここからは、スタック設置のオペレーションについてご案内いたします。

まず、セルフマネージド型またはサービスマネージド型、どちらのアクセス許可モデルとするかを選択し、初回のみ設定を行う必要があります。セルフマネージドでは、管理アカウント、ターゲットアカウント双方に指定された名称・内容で IAM ロールを手動で作成する必要があります。なお、バイアムロール作成用のクラウドフォーメーションテンプレートが配布されておりますので、必要に応じてご利用ください。一方、サービスマネージドの場合には、 AWS Organizations で Stacksets の信頼されたアクセスを有効化することで、管理アカウントとターゲットアカウントの双方に必要な IAM ロールが自動的に作成されます。

次のオペレーションは、スタックセットの作成です。 Sack の作成に使用するテンプレート、ターゲットアカウント、リージョンの指定を行うことで、スタックセットを作成できます。うんタックセットの作成は大きく分けると 4 つのステップがあります。 1 つ目はテンプレートの作成ですまず、はじめにサービスマネージドもしくはセルフマネージドどちらのアクセス許可にするかを選択いただきます。うん。そして Stack のテンプレートを指定します。次にスタックセットの詳細を指定します。そして、スタックセットオプションにてタグや実行設定を設定します。最後にデプロイオプションにて、デプロイ先やリージョン、自動デプロイオプションなどを設定します。ここで自動デプロイオプションについて解説させていただきます。自動デプライオプションとは、 AWS Oranizations の組織単位へアカウントを追加・削除した際のスタックインスタンスの挙動を設定する機能です。なお、自動デプロイオプションは、サービスマネージドのアクセス許可モデルの場合のみ設定することが可能です。うん。加えて、後ほど説明する Stacksets の Stack に上書きしたパラメータは、設定時に指定されたアカウントのみ適用され、自動デプロイオプションで追加されるアカウントには適用されないことをご留意ください。

次に、デプロイオプションについてご説明いたします。まずはじめに、同時アカウントの最大数は、スタックセットの作成、更新、削除に適用され、リージョン別に一度にオペレーションを実行するターゲットアカウントの最大数または割合を指定することが可能です。うん、たとえば、割合 50% で 2 つのリジオンの 10 個のターゲットアカウントにデプロイする場合を見てみましょう。うん。 10 個のターゲットアカウントの 50% のため、一度にオペレーションが実行されるターゲットのアカウント数はとなります。そして、リージョン別に実施されるため、図のような流れになります。まず初めにリージョン A で 5 つのアカウント。次に同じリージョン A でその他の 5 つのアカウント。その後にリージョン B で 5 つのアカウント。最後に T ション B で、その他 5 つのアカウントといった形です。

次に、デプロイオプションの障害体制についてご説明いたします。こちらは、スタックセットの作成、更新、削除時に各リージョンでスタックオペレーションの失敗を許容するターゲットアカウント数を設定し、設定することが可能です。例えば、割合 20% で 3 つのリージョンの 10 個のターゲットアカウントにデプロイした場合を想定してみましょう。うん、まずリージョン A ではデプロイに 1 つのターゲットアカウントで失敗しました。その場合は、割合が 10% のため、次のリージョンへとストラックのオペレーションが進みます。その次に、リージョンビリーでは、 3 つのターゲットアカウントでデプロイに失敗しました。その場合には、割合が 30% となり、指定した 20% を超えるため、タックオペレーションは自動的に停止します。

最後に同時実行モードについてご説明いたします。同時実行モードは、スタックセット操作中の同時実行レベルの動作を選択できるパラメータです。なお、選択できるオプションは、厳密な障害耐性とソフト障害体制の 2 つがあります。はい。デフォルトで設定されている厳密な障害耐性は、失敗したアカウントの数が障害体制で設定した数プラス 1 の値を超えないように、同時実行レベルが動的に下がります。つまり、デプロイに失敗した場合、同時実行レベルである実際の同時実行数は、デプロイしたデプロイの数に比例して検証します。そのため、デプロイが失敗するたびにデプロイ速度が低下します。

一方で、ソフト障害体制は、デプロイに失敗しても同時実行レベルである。実際の同時実行数は一定。これにより、デプロイの失敗数に限らず、スタックセットの操作を同時アカウントの最大数で実行することができます。しかし、実際の障害数を考慮せずに同時にデプロイを実施するため、障害耐性で設定した値よりもデプロイに失敗したスタックインスタンスが多くなる可能性があることにご留意ください。うん。そのため、厳密な障害耐性よりもデプロイ速度を優先したい場合に使用することを推奨いたします。

うんでは、例を用いて厳密な障害耐性について説明いたします。例では障害耐性の値を後、同時アカウントの最大数を 10 とします。その場合、実際の同時実行数は障害耐性の数の 5 に 1 を足した 6 となります。なお、下の図は障害対数の値が同時アカウントの最大数の値に与える影響と、それらがスタックセット操作の実際の同時実効性に与える影響を示しています。

では早速、デプロイが失敗した場合の挙動について見ていきましょう。 2 つ目の図では、スタックセットが 2 つのスタックインスタンスのデプロイに成功し、 1 つのデプロイに失敗しています。その場合、 3 番目の図のように、実際の同時実行数は 6 から 5 に減少します。うん、その後さらに 2 つのスタックインスタンスのデプロイに失敗すると、実際の同時実行数は 5 から 3 に減少します。うん。なお、失敗したスタックインスタンスの合計は 3 つとなります。続いてスタックセットが 3 つのデプロイに失敗すると、失敗したスタックインスタンスの合計は 6 つとなります。うん、その場合、失敗したスタックインスタンスの数が障害体制で設定していた 5 と等しくなるため、スタックセットは操作に失敗します。今回の例では。スタックセットは、操作を停止するまでに 9 つのスタックインスタンスをデプロイし、 6 つのスタックインスタンスにて失敗しました。

次に、ソフト障害体制について例を用いてご説明いたします。例では、傷害耐性の値を後、同時アカウントの最大数を重としています。なお、ソフト障害体制では、実際の同時実行数は同時アカウントの最大数と同じ 10 となります。

うん。では、ソフト障害体制にてデプロイが失敗した場合の挙動について見ていきましょう。厳密な障害体制と異なり、ソフト障害耐性の場合はスタックの操作が失敗しても、実際の同時実行数は 10 のまま変わりありません。うん、さらにスタックインスタンスのデプロイが 2 回失敗しても、実際の同時実行数に変化はございません。加えて、 6 つのスタックの操作に失敗し、障害耐性の値である 5 に達した場合には、スタックセットは操作に失敗します。うん。しかし、同時実行キュに残りの操作がある場合には、操作を完了するまでスタックセットはデプロイを続けます。うん。そのため、最終的には障害体制数に達した際に残っていた 7 つのスタックインスタンスにデプロイを実施し、合計 8 つのスタックインスタンスが失敗しております。このように、ソフト障害体制は、障害が発生したスタックインスタンスの数が障害対数の数より多くなる可能性がございます。

次に、スタックセットの更新方法についてご説明します。スタックセットを更新する場合、 4 つの手順にて更新することが可能です。まずはじめに更新したいスタックセットを選択し、スタックセットの詳細を編集を押下します。そして、現在のテンプレートを使用するか、新しくテンプレートを置き換えるかを選択します。次にスタックセットの説明やパラメータなどの詳細を変更します。次に、作成時と同様にタグや実行設定を指定することができます。最後にこちらも作成時と同様にデプロイ先やリージョンなどを指定することが可能です。

ここで、サービス Monaged のアクセス許可モデルの場合に設定できるアカウントフィルタータイプについてご説明します。うん。カルンとフィルタータイプにはなし、共通集合、差分和集合の 4 種類があり、これらを適宜に用いることで指定された組織単位を使用してデプロイを個々のターゲットアカウントに制限することなどが可能となります。

うん、続いて Stack インスタンスのパラメータの上書きについてご説明します。こちらはアカウントおよびリージョン別に Stack インスタンスのパラメータを上書きすることが可能です。うん、 3 つの手順にて Stack インスタンスのパラメータを上書きすることが可能です。

はじめに任意のスタックセットを選択し、アクションからスタックセットのパラメータを上書きを押下します。うん。そしてデプロイ先やリージョン、デプロイオプションを指定します。そして、上書きしたいパラメータを選択し、アクションからスタックセット値の上書きを押下して内容を入力することで上書きできます。また、留意点としては、上書きされたパラメータをスタックセットで指定された値に戻すためには、スタックセット値に設定を選択する必要がございます。なお、上書きされたパラメータは、設定したアカウントにのみ適用されることもご留意ください。

次にスタックの削除について説明いたします。 Stack の削除は、指定したターゲットアカウントからスタックを削除を実行することが可能です。なおあわせて対象のスタックインスタンスも。削除されます。実際に削除する方法としては、初めに削除するスタッフを選択し、アクションのスタックセットからスタックを削除を押下します。次にデプロイ先やデプロイオプションなどを選択することで削除することができます。最後にスタックセットの削除についてご説明いたします。なお、スタックセットは、スタックセット内にスタックインスタンスがない場合にのみ削除することが可能です。はじめに削除するスタックセットを選択し、アクションからスタックセットの削除を押下します。次に削除ボタンを押下することでスタックシットを削除することができます。

以上でこのセミナーを終了します。最後に、 AWS ブラックビルとオンラインセミナーの説明と内容についての注意を表示します。本セミナーへの感想は XQ Twitter までお寄せください。内容についての注意点もご確認ください。