グレー障害の検出と軽減

マルチ AZ の高度なレジリエンスパターン

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

マルチ AZ の高度なレジリエンスパターン: グレー障害の検出と軽減

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは、お客様に混乱を招く可能性がある態様、または Amazon の信用を傷つけたり、失わせたりする態様において、Amazon のものではない製品またはサービスに関連して使用してはなりません。Amazon が所有しない他の商標はすべてそれぞれの所有者に帰属します。所有者は必ずしも Amazon との提携や関連があるわけではありません。また、Amazon の支援を受けているとはかぎりません。

Table of Contents

要約と序章 i
序章 1
グレー障害 3
視点別のオブザーバビリティ 3
グレー障害の例 6
グレー障害への対応 7
マルチ AZ の可観測性 10
CloudWatch 複合アラームによる障害検出 15
1 つのアベイラビリティーゾーンで影響を検知する 15
影響がリージョンのものではないことを確認する 17
影響が 1 つのインスタンスによるものではないことを確認する 17
まとめ 19
外れ値検出による障害検出 20
単一インスタンスのゾーンリソースの障害検出 25
[概要] 28
アベイラビリティーゾーンの退避パターン 29
アベイラビリティーゾーンの独立性 30
コントロールプレーンとデータプレーン 36
データプレーン制御の退避 37
Route 53 Application Recovery Controller (ARC) のゾーンシフト 37
Route 53 ARC 39
セルフマネージド型 HTTP エンドポイントの使用 40
コントロールプレーン制御による退避 47
まとめ 50
結論 52
付録 A — アベイラビリティーゾーン ID の取得 53
付録 B — カイ二乗計算の例 55
寄稿者 61
ドキュメントの改訂 62
注意 63
AWS 用語集 64
..... lxv

マルチ AZ の高度なレジリエンスパターン

出版日:2023 年 7 月 11 日 (ドキュメントの改訂)

多くのカスタマーは、高可用性のマルチアベイラビリティーゾーン (AZ) 設定でワークロードを実行しています。このアーキテクチャは、バイナリ障害が発生しても良好に動作しますが、グレー障害が発生して問題を起こす場合があります。この種の障害の兆候は微妙で、迅速かつ確実に検出できない場合があります。このホワイトペーパーでは、1 つのアベイラビリティーゾーンに分離されたグレー障害の影響を検出し、そのアベイラビリティーゾーンでの影響を軽減するアクションを実行するように、ワークロードをインストルメント化する方法について説明します。

序章

本書の目的は、回復力のあるマルチ AZ アーキテクチャをより効果的に実装できるようにすることです。Amazon 仮想プライベートクラウド (VPC) ネットワークで回復力のあるシステムを構築するためのベストプラクティスの 1 つは、複数のアベイラビリティーゾーンに各ワークロードをデプロイすることです。

アベイラビリティーゾーンは、冗長電源、ネットワーク、および接続を備えた 1 つ以上の個別のデータセンターです。複数のアベイラビリティーゾーンを使用すると、単一のデータセンターを使用した場合よりも、可用性、耐障害性、スケーラビリティが高いワークロードを運用できます。

Amazon Elastic Compute Cloud (EC2) Auto Scaling や Amazon Relational Database (Amazon RDS) など、多くの AWS のサービスでは、マルチ AZ 設定を利用できます。これらのサービスでは、追加のオブザーバビリティツールやフェイルオーバーツールを構築する必要はありません。これらのサービスでは、単一のアベイラビリティーゾーンに影響する AWS リージョン 内で簡単に検出できるバイナリ障害モードに対して、ワークロードの回復性を実現します。バイナリ障害は、大部分のリソースに影響を与える完全に物理的なハードウェア障害、停電、または潜在的なソフトウェアバグである場合があります。

しかし、グレー障害と呼ばれる別のカテゴリの障害があります。この障害の兆候は微妙で、迅速かつ確実な検出が困難です。その結果、障害による影響を軽減するまでの時間が長くなります。このホワイトペーパーでは、グレー障害がマルチ AZ アーキテクチャに与える影響、その検出方法、そして最後に軽減方法に焦点を当てて説明しています。

> このホワイトペーパーに記載されているガイダンスは、主に次のような特定のクラスのワークロードに適用されます。

- 主にゾーン AWS サービスを使用している
- 単一リージョンの回復力を改善する必要がある
- 必要なオブザーバビリティと回復力のパターンを構築するために多額を投資する意思がある

このようなワークロードでは、??? に示されているトレードオフの一部または全部を行いたくない場合があります。または、複数のリージョンを使用するオプションがないかもしれません。これらのタイプのワークロードは、ポートフォリオ全体のごく一部である可能性が高いため、このガイダンスはプラットフォームレベルではなくワークロードレベルで検討する必要があります。

グレー障害

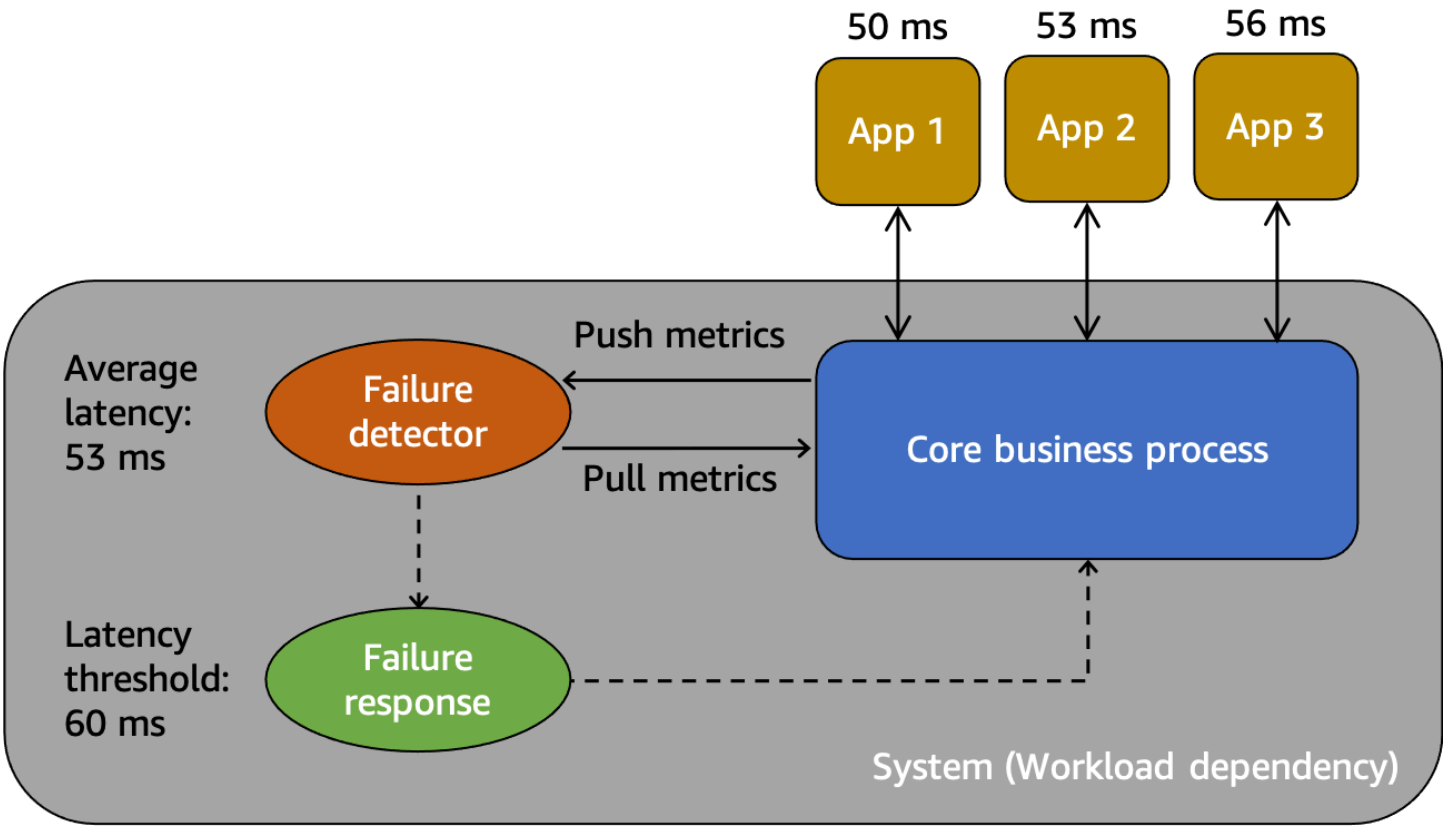
グレー障害は、視点別のオブザーバビリティという特性によって定義されます。これは、異なるエンティティは異なる視点で障害を観測することを意味します。これが何を意味するのかを定義しましょう。

視点別のオブザーバビリティ

通常、操作するワークロードには依存関係があります。例えば、これらは、ワークロードの構築に使用する AWS クラウドサービス、またはフェデレーションに使用するサードパーティの ID プロバイダー (IdP) の場合があります。これらの依存関係は、ほとんどの場合、独自のオブザーバビリティを実装し、エラー、可用性、レイテンシーなどに関するメトリクスを記録して、カスタマーの使用状況によって生成されます。これらのメトリクスのいずれかでしきい値を超えると、通常、依存関係は何らかのアクションを実行してそれを修正します。

これらの依存関係には通常、サービスのコンシューマーが複数存在します。また、コンシューマーは独自のオブザーバビリティを実装し、依存関係とのやり取りに関するメトリクスとログを記録し、ディスク読み取りのレイテンシー、失敗した API リクエストの数、データベースクエリに要した時間などを記録します。

これらのやり取りと測定については、次の図の抽象モデルに示されています。

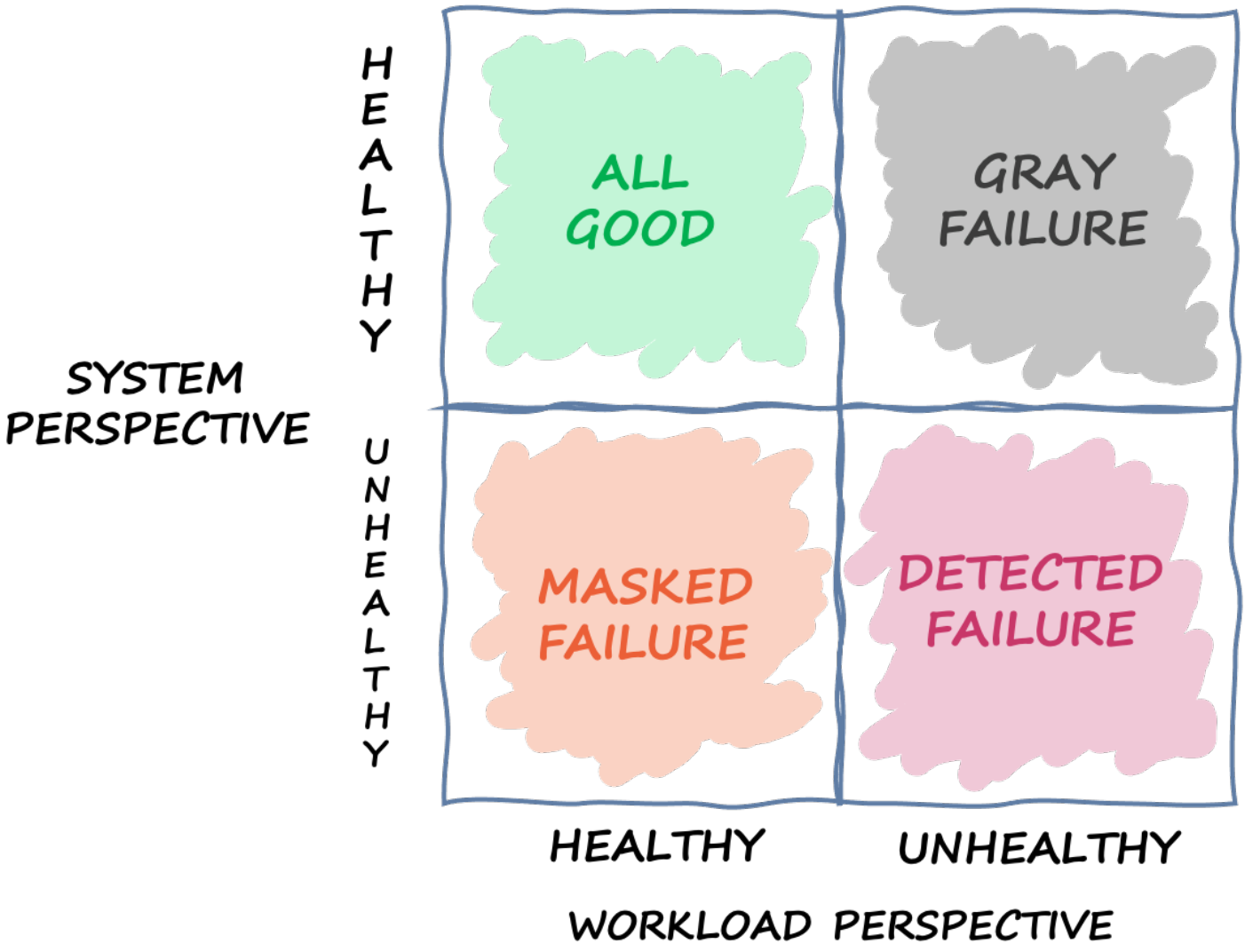


グレー障害を理解するための抽象モデル

まず、このシナリオでは、コンシューマーアプリ 1、アプリ 2、アプリ 3 の依存関係であるシステムが存在します。このシステムには、コアビジネスプロセスから作成されたメトリクスについて検証する障害ディテクターがあります。また、障害ディテクターで検出された問題を軽減または修正する障害対応メカニズムも備えています。システム全体の平均レイテンシーは 53 ミリ秒で、平均レーテンシーが 60 ミリ秒を超えると障害応答メカニズムを呼び出すためのしきい値が設定されています。アプリ 1、アプリ 2、アプリ 3 もシステムとのやり取りについて独自のオブザベーションを行い、それぞれ 50 ミリ秒、53 ミリ秒、56 ミリ秒の平均レイテンシーを記録しています。

視点別のオブザーバビリティとは、システムコンシューマーの 1 つがシステムの異常を検出しても、システム自体の監視では問題が検出されないか、影響がアラームのしきい値を超えない状況です。アプリ 1 の平均レイテンシーが 50 ミリ秒ではなく 70 ミリ秒になったとしましょう。アプリ 2 とアプリ 3 の平均レイテンシーに変化は見られません。これにより、基礎となるシステムの平均レイテンシーは 59.66 ミリ秒に増加しますが、障害対応メカニズムを有効化するレイテンシーのしきい値を超えることはありません。ただし、アプリ 1 のレイテンシーは 40% 増加しています。これにより、アプリ 1 に設定されたクライアントタイムアウトを超えることで可用性に影響が出たり、より長い一連のやり取りで影響が連鎖的に発生する可能性があります。アプリ 1 から見ると、依存し

ている基盤となるシステムに異常があるものの、アプリ 2 とアプリ 3 と同様にシステム自体から見た場合、システムは正常です。次の図は、これらの異なる視点をまとめたものです。



異なる視点に基づいてシステムがどのような状態になるかを定義する象限

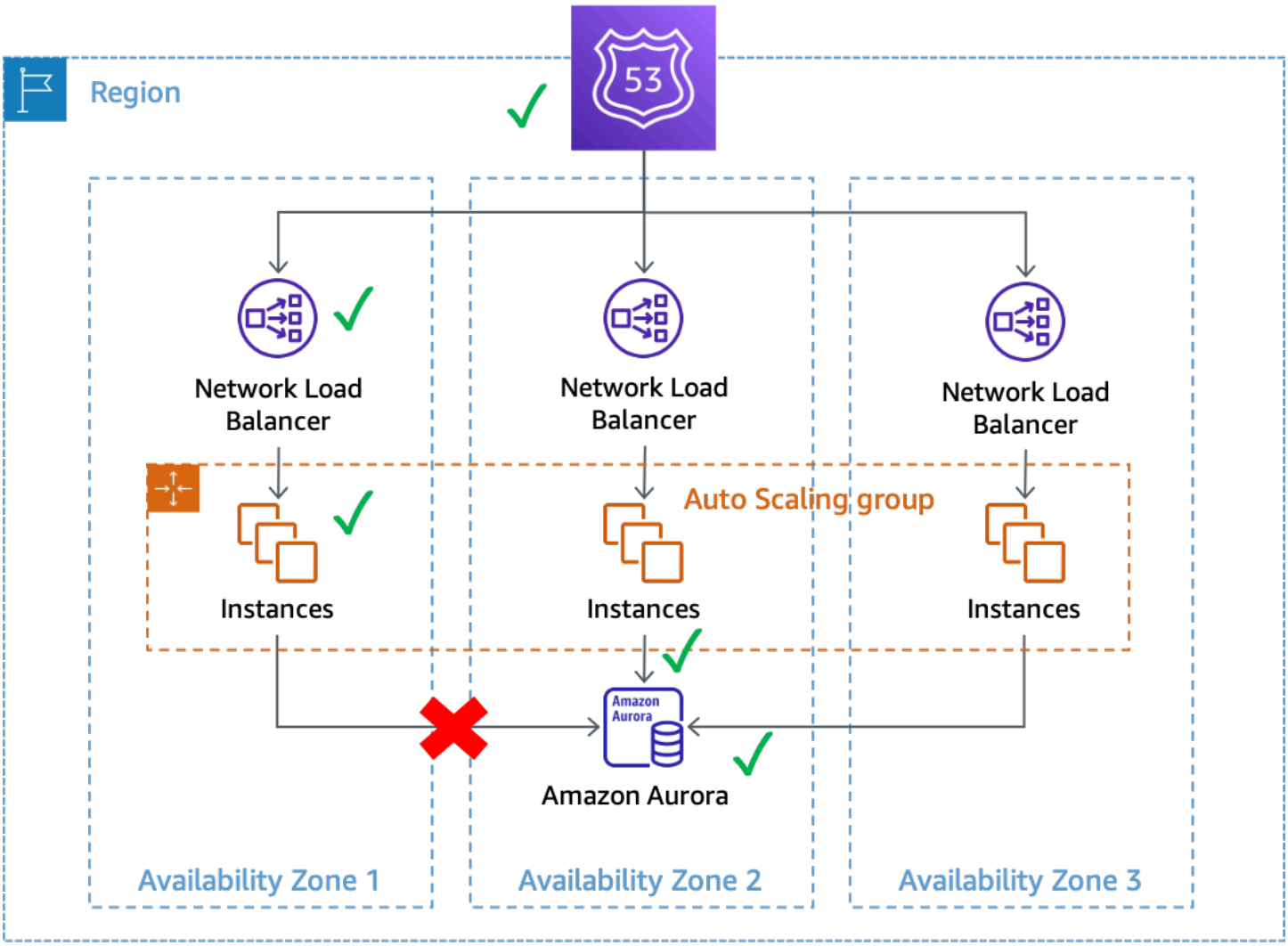
障害はこの象限を超えることもあります。イベントは、グレー障害として始まり、次に検出された障害になり、その後マスクされた障害に移行してから、グレー障害に戻ることもあります。サイクルが決まっているわけではなく、根本原因が解決されるまで、ほとんどの場合、障害が再発する可能性があります。

このことから導き出された結論は、ワークロードは、障害を検出し、軽減する基盤となるシステムに常に依存できるとは限らないということです。基盤となるシステムがどれほど洗練され、回復力に優れていても、障害が検出されなかったり、反応しきい値を下回る可能性が常にあります。アプリ 1 など、このシステムのコンシューマーには、グレー障害が引き起こす影響を迅速に検出し、軽減する

機能が必要です。これには、こうした状況に対応するオブザーバビリティおよび復旧のメカニズムを構築する必要があります。

グレー障害の例

グレーの障害は、AWS のマルチ AZ システムに影響する可能性があります。例えば、3 つのアベイラビリティーゾーンにデプロイされた Auto Scaling グループの Amazon EC2 インスタンスのフリートがあるとします。これらはすべて 1 つのアベイラビリティーゾーンの Amazon Aurora データベースに接続しています。そこで、アベイラビリティーゾーン 1 とアベイラビリティーゾーン 2 の間のネットワークに影響するグレー障害が発生したとします。この障害の結果、アベイラビリティーゾーン 1 のインスタンスからの新規および既存のデータベース接続の一部が失敗します。このアーキテクチャを次の図に示します。



アベイラビリティーゾーン 1 のインスタンスからのデータベース接続に影響するグレー障害

この例では、Amazon EC2 は、システムとインスタンスのステータスのチェックを引き続きパスしているため、アベイラビリティーゾーン 1 のインスタンスは正常と見なします。また、Amazon EC2 Auto Scaling はどのアベイラビリティーゾーンへの直接的な影響も検出せず、設定されたアベイラビリティーゾーンでキャパシティを起動し続けます。また、Network Load Balancer (NLB) は、NLB エンドポイントに対して実行される Route 53 ヘルスチェックと同様に、その背後にあるインスタンスも正常であると見なします。同様に、Amazon Relational Database Service (Amazon RDS) は、データベースクラスターが正常であると見なし、自動フェイルオーバーをトリガーしません。多くの異なるサービスが存在し、それらすべてがサービスとリソースを正常と見なしていても、ワークロードはその可用性に影響する障害を検出します。これがグレー障害です。

グレー障害への対応

AWS 環境にグレー障害が発生したとき、一般的に次の 3 つのオプションを使用できます。

- 何もしないで、障害が終了するのを待ちます。
- 障害が 1 つのアベイラビリティーゾーンに分離されている場合は、そのアベイラビリティーゾーンから退避します。
- 別の AWS リージョン にフェイルオーバーし、AWS リージョンの分離を活用して影響を軽減します。

多くの AWS カスタマーは、ほとんどのワークロードでオプション 1 を使用しても問題ありません。カスタマーは、追加のオブザーバビリティや回復力のソリューションを構築する必要はなくなる代わりに、目標復旧時間 (RTO) が長くなる可能性を受け入れます。中には、3 番目のオプションである マルチリージョンのディザスタリカバリ (DR) を、さまざまな障害モードに対する緩和策として実装することを選択するカスタマーもいます。マルチリージョンアーキテクチャは、このようなシナリオでうまく機能します。ただし、この方法を使用する場合にはいくつかのトレードオフがあります (マルチリージョンの考慮事項についての詳しい説明は「AWS マルチリージョンの基礎」を参照)。

まず、マルチリージョンアーキテクチャの構築と運用は困難かつ複雑で、コストがかかる可能性があります。マルチリージョンアーキテクチャでは、どの DR 戦略を選ぶかについて慎重に検討する必要があります。ゾーンの障害に対処するためだけにマルチリージョンのアクティブ/アクティブ DR ソリューションを実装するのは、経済的に実現可能ではないかもしれません。一方、バックアップと復元の戦略は、回復力の要件を満たさない可能性があります。さらに、必要なときに確実に機能するように、マルチリージョンのフェイルオーバーを本番環境で継続的に実施する必要があります。これらすべてにおいて、構築、運用、テストに多くの時間とリソースが必要です。

次に、現在 AWS のサービスを使用した AWS リージョン 間のデータレプリケーションは、非同期で行われます。非同期レプリケーションに伴ってデータが失われる可能性があります。つまり、リージョン間のフェイルオーバー中に、ある程度のデータ損失や不整合が発生する可能性があります。データ損失量に対する許容範囲は、目標復旧時点 (RPO) として定義します。確かなデータ整合性が必須であるカスタマーは、プライマリリージョンが再び利用可能になったときに、これらの一貫性の問題を解決するための調整システムを構築する必要があります。または、独自の同期レプリケーションシステムや二重書き込みシステムを構築する必要があり、これらは応答遅延、コスト、複雑さに大きな影響を与える可能性があります。また、セカンダリリージョンがすべてのトランザクションに対してハード依存関係になるため、システム全体の可用性が低下する可能性があります。

最後に、アクティブ/スタンバイアプローチを使用する多くのワークロードでは、別のリージョンへのフェイルオーバーの実行に要する時間はゼロではありません。場合によっては、ワークロードのポートフォリオをプライマリリージョンで特定の順序で切断したり、接続をドレインしたり、または特定のプロセスを停止する必要があります。その後、特定の順序でサービスを再開しなければならない場合があります。また、新しいリソースをプロビジョニングする必要がある場合や、サービス開始前に必要なヘルスチェックをパスするまでに時間がかかる場合もあります。このフェイルオーバー処理は、完全に利用できない期間として行われる可能性があります。これが RTO が懸念されることです。

リージョン内では、AWS の多くのサービスが強力な整合性のあるデータの永続性を提供します。Amazon RDS マルチ AZ 配置では同期レプリケーションを使用します。Amazon Simple Storage Service (Amazon S3) は、強力な書き込み後の読み取り整合性を提供します。Amazon Elastic Block Storage (Amazon EBS) は、マルチボリュームの Crash-consistent スナップショットを提供します。Amazon DynamoDB では、強力な整合性のある読み込みを実行できます。これらの機能により、マルチリージョンアーキテクチャで達成できるよりも低い RPO (ほとんどの場合 RPO はゼロ) を単一のリージョンで達成できます。

インフラストラクチャとリソースはすでにアベイラビリティーゾーン全体でプロビジョニングされているため、アベイラビリティーゾーンを退避する方がマルチリージョン戦略よりも RTO が低くなる可能性があります。マルチ AZ アーキテクチャは、アベイラビリティーゾーンに障害が発生しても、サービスの停止とバックアップの順序付けを慎重に行ったり、接続をドレインする必要はなく、静的に動作し続けることができます。リージョン間のフェイルオーバーでは完全に利用不能になる期間が発生する場合がありますが、アベイラビリティーゾーン間の退避中は、作業が残りのアベイラビリティーゾーンにシフトされるため、多くのシステムではわずかな機能低下で済む場合があります。システムがアベイラビリティーゾーンの障害に対して静的に安定するように設計されている場合 (この場合、これは負荷を吸収するために他のアベイラビリティーゾーンにキャパシティがあらかじめプロビジョニングされていることを意味します)、ワークロードのカスタマーにはまったく影響がない可能性があります。

> 1 つのアベイラビリティーゾーンの障害がワークロードに加えて 1 つまたは複数の AWS リージョンサービスに影響を与える可能性があります。リージョンへの影響が観測された場合は、その影響の原因が単一のアベイラビリティーゾーンにあったとしても、そのイベントをリージョンサービスの障害として扱う必要があります。アベイラビリティーゾーンを退避させても、この種の問題は軽減されません。これが発生した場合は、リージョンのサービス障害に対して準備している対応計画を使用してください。

このドキュメントの残りの部分では、シングル AZ のグレー障害に対して RTO と RPO を低く抑える方法として、2 つ目のオプションであるアベイラビリティーゾーンからの退避に焦点を当てます。これらのパターンは、マルチ AZ アーキテクチャの価値と効率を高めるのに役立ち、ほとんどのクラスのワークロードで、これらの種類のイベントに対処するためにマルチリージョンアーキテクチャを構築する必要性を軽減できます。

マルチ AZ の可観測性

1 つのアベイラビリティーゾーンに分離されたイベント中にアベイラビリティーゾーンから退避するには、まず、障害が 1 つのアベイラビリティーゾーンに実際に分離されていることを検出できる必要があります。そのためには、各アベイラビリティーゾーンでシステムの動作を忠実に可視化する必要があります。AWS の多くのサービスには、リソースに関する運用上のインサイトを提供する、すぐに使えるメトリクスが用意されています。例えば、Amazon EC2 には、CPU 使用率、ディスクの読み取りと書き込み、ネットワークトラフィックの送受信など、多数のメトリクスがあります。

ただし、これらのサービスを使用するワークロードを構築する場合は、これらの標準メトリクスだけでなく、追加の可視性が必要となります。ワークロードが提供するカスタマーエクスペリエンスへの可視性が必要です。さらに、メトリクスをそれらが生成されるアベイラビリティーゾーンに合わせる必要があります。これこそが、視点別の観測可能なグレー障害を検出するために必要なインサイトです。このレベルの可視性には、インストルメント化が必要です。

インストルメント化では、明示的なコードを記述する必要があります。このコードは、タスクにかかった時間の記録、成功または失敗した項目の数のカウント、リクエストに関するメタデータの収集などを行う必要があります。また、事前にしきい値を定義して、何を正常と見なし、何を正常と見なさないかを定義する必要があります。ワークロードのレイテンシー、可用性、エラー数の目標とさまざまな重要度のしきい値を定義する必要があります。Amazon Builders' Library の記事「運用の可視性を高めるために分散システムを装備する」には、さまざまなベストプラクティスが記載されています。

メトリクスは、サーバー側とクライアント側の両方から生成する必要があります。クライアント側のメトリクスを生成してカスタマーエクスペリエンスを理解するには、ワークロードを定期的に調査してメトリクスを記録するソフトウェアである canary を使用するのがベストプラクティスです。

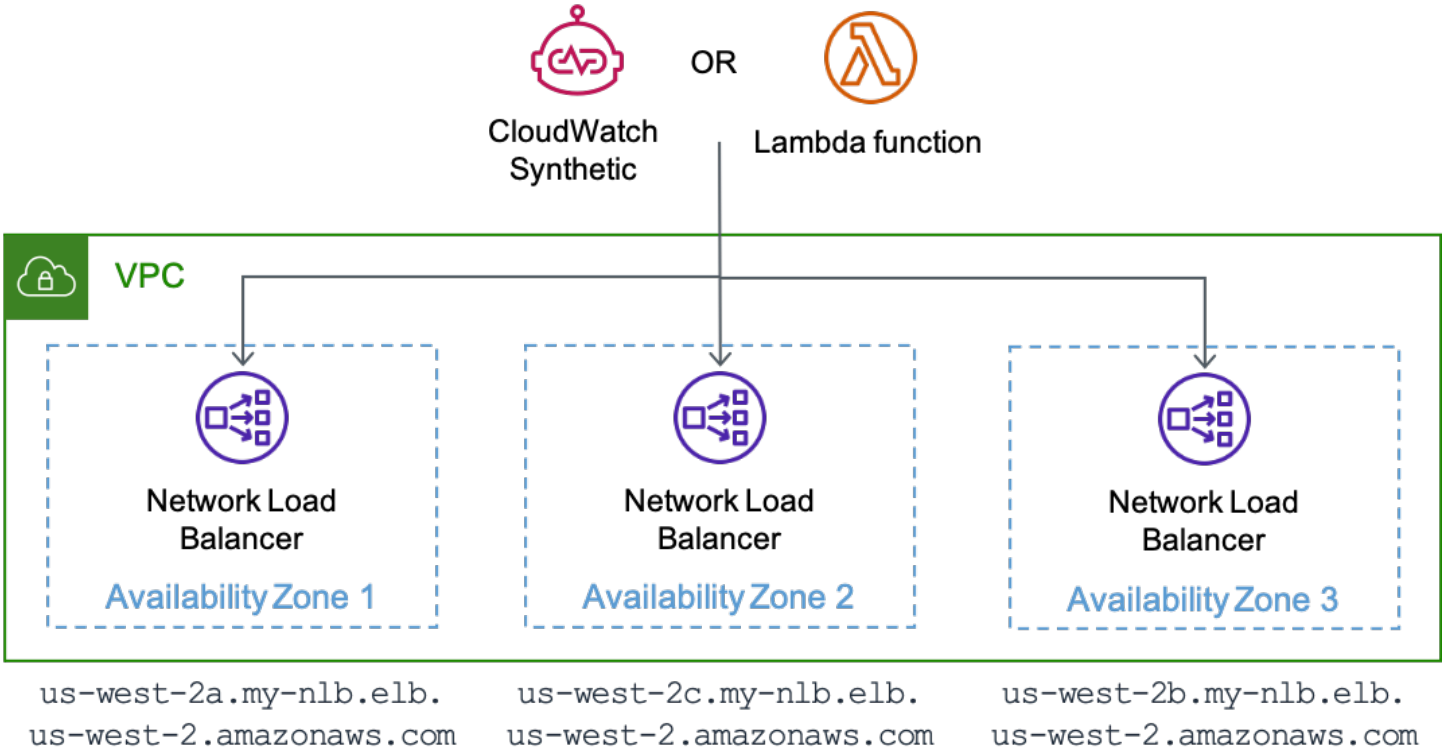
これらのメトリクスを作成することに加えて、そのコンテキストも理解する必要があります。その 1 つの方法は、ディメンションを使用することです。ディメンションは、メトリクスに独自のアイデンティティを与え、メトリクスが何を伝えているかを説明するのに役立ちます。ワークロードの障害を特定するためのメトリクス (レイテンシー、可用性、エラー数など) には、障害分離境界に合わせたディメンションを使用する必要があります。

例えば、モデルビューコントローラー (MVC) の Web フレームワークを使用して、1 つのリージョンで複数のアベイラビリティーゾーンにまたがる Web サービスを実行している場合、ディメンションセットのディメンションとして `Region`、`Availability Zone ID`、`Controller`、`Action`、および `InstanceId` を使用する必要があります (マイクロサービスを使用している場合は、コント

ローラ名とアクション名の代わりにサービス名と HTTP メソッドを使用します)。これは、さまざまなタイプの障害が、これらの境界で分離されることが望ましいためです。ウェブサービスのコードのバグが商品の出品に影響を与えている場合、このバグがホームページにも影響を与えることは望ましくありません。同様に、1 つの EC2 インスタンスの EBS ボリューム全体が、他の EC2 インスタンスによるウェブコンテンツの提供に影響を与えることは望ましくありません。アベイラビリティーゾーンの ID ディメンションを使用すると、AWS アカウント全体でアベイラビリティーゾーンに関連する影響を一貫して特定できます。ワークロード内のアベイラビリティーゾーン ID は、いくつかの方法で確認できます。一部の例については、「付録 A — アベイラビリティーゾーン ID の取得」を参照してください。

このドキュメントでは、例の中で主に Amazon EC2 をコンピューティングリソースとして使用していますが、`InstanceId` は、ディメンションのコンポーネントとして Amazon Elastic Container Service (Amazon ECS) コンピューティングリソースおよび Amazon Elastic Kubernetes Services (Amazon EKS) コンピューティングリソースのコンテナ ID に置き換えることができます。

ワークロードにゾーンエンドポイントがある場合、canary では、`Controller`、`Action`、`AZ-ID`、`Region` をメトリクスのディメンションとして使用することもできます。この場合、テスト中のアベイラビリティーゾーンで実行するように canary を調整します。これにより、分離されたアベイラビリティーゾーンのイベントが canary を実行しているアベイラビリティーゾーンに影響を与えている場合でも、テスト中の別のアベイラビリティーゾーンに異常があるように見せるメトリクスは記録されなくなります。例えば、canary では、Network Load Balancer (NLB) または Application Load Balancer (ALB) の背後にあるサービスの各ゾーンのエンドポイントを、その ゾーンの DNS 名を使用してテストできます。



CloudWatch Synthetics で実行している canary またはNLB の各ゾーンのエンドポイントをテストする AWS Lambda 機能

これらのディメンションを使用してメトリクスを生成することで、これらの境界内で可用性やレイテンシーに変化が生じたときに通知する Amazon CloudWatch アラームを設定できます。また、ダッシュボードを使用してそのデータをすばやく分析することもできます。メトリクスとログの両方を効率的に使用するために、Amazon CloudWatch にはカスタムメトリクスをログデータに埋め込むことができる埋め込みメトリクスフォーマット (EMF) が用意されています。CloudWatch はカスタムメトリクスを自動的に抽出するため、これらを視覚化してアラームを発行できます。AWS には、EMF を簡単に開始できるように、さまざまなプログラミング言語用のクライアントライブラリがいくつか用意されています。これらのライブラリは、Amazon EC2、Amazon ECS、Amazon EKS、AWS Lambda、およびオンプレミス環境で使用できます。ログにメトリクスが埋め込まれていると、Amazon CloudWatch Contributor Insights を使用してコントリビューターデータを表示する時系列グラフを作成することもできます。このシナリオでは、AZ-ID、InstanceId、または Controller などのディメンション、および SuccessLatency または HttpResponseCode などのログ内の他のフィールド別にグループ化されたデータを表示できます。

```
{
  "_aws": {
    "Timestamp": 1634319245221,
```

```
    "CloudWatchMetrics": [
      {
        "Namespace": "workloadname/frontend",
        "Metrics": [
          { "Name": "2xx", "Unit": "Count" },
          { "Name": "3xx", "Unit": "Count" },
          { "Name": "4xx", "Unit": "Count" },
          { "Name": "5xx", "Unit": "Count" },
          { "Name": "SuccessLatency", "Unit": "Milliseconds" }
        ],
        "Dimensions": [
          [ "Controller", "Action", "Region", "AZ-ID", "InstanceId"],
          [ "Controller", "Action", "Region", "AZ-ID"],
          [ "Controller", "Action", "Region"]
        ]
      }
    ],
    "LogGroupName": "/loggroupname"
  },
  "CacheRefresh": false,
  "Host": "use1-az2-name.example.com",
  "SourceIp": "34.230.82.196",
  "TraceId": "|e3628548-42e164ee4d1379bf.",
  "Path": "/home",
  "OneBox": false,
  "Controller": "Home",
  "Action": "Index",
  "Region": "us-east-1",
  "AZ-ID": "use1-az2",
  "InstanceId": "i-01ab0b7241214d494",
  "LogGroupName": "/loggroupname",
  "HttpResponseCode": 200,
  "2xx": 1,
  "3xx": 0,
  "4xx": 0,
  "5xx": 0,
  "SuccessLatency": 20
}
```

この口グには 3 つのディメンションのセットがあります。これらは、インスタンス、アベイラビリティーゾーン、リージョンの粒度の順に進んでいきます (この例では、Controller と Action が常に含まれます)。これらは、1 つのインスタンス、1 つのアベイラビリティーゾーン、または AWS リージョン全体で、特定のコントローラーのアクションに影響が生じた場合に知らせるア

ラームを、ワークロード全体にわたって作成することをサポートします。これらのディメンションは、2xx、3xx、4xx、5xx の HTTP 応答数のメトリクスと、成功したリクエストのレイテンシーのメトリクスに使用します (リクエストが失敗した場合、失敗したリクエストのレイテンシーのメトリクスも記録されます)。またログには、HTTP パス、リクエスト元のソース IP、このリクエストでローカルキャッシュを更新する必要があるかどうかなど、その他の情報も記録されます。これらのデータポイントを使用して、ワークロードが提供する各 API の可用性とレイテンシーを計算できます。

> 可用性のメトリクスで HTTP 応答コードを使用することに関する注意。
>
> 通常、2xx と 3xx の応答は成功、5xx は失敗と見なすことができます。4xx 応答コードは中間のいずれかの問題を示します。通常、これらはクライアントエラーが原因で生成されます。パラメータが範囲外であるために 400 応答になったり、存在しないものをリクエストしたために 404 応答になったりします。これらの応答は、ワークロードの可用性に対してカウントされません。ただし、これはソフトウェアのバグが原因である可能性もあります。例えば、より厳格な入力検証を導入したために、以前なら成功していたはずのリクエストが拒否された場合、400 応答は可用性の低下としてカウントされる可能性があります。または、顧客のスロットリングにより、429 応答が返される場合があります。顧客のスロットリングでサービスの可用性を維持することはできますが、顧客の観点からは、サービスが顧客のリクエストを処理できないことになります。4xx 応答コードを可用性の計算に含めるかどうかを決める必要があります。

このセクションでは、CloudWatch を使用してメトリクスを収集して分析する方法について説明しましたが、使用できるソリューションはこれだけではありません。Amazon Managed Service for Prometheus および Amazon Managed Grafana、Amazon DynamoDB テーブルにメトリクスを送信したり、サードパーティのモニタリングソリューションを使用したりすることもできます。重要なのは、ワークロードが生成するメトリクスには、ワークロードの障害分離境界に関するコンテキストが含まれている必要があるということです。

ワークロードで生成するメトリクスのディメンションを障害分離境界に合わせることで、アベイラビリティーゾーンの分離された障害を検出するオブザーバビリティを構築できます。以下のセクションでは、1 つのアベイラビリティーゾーンの障害から生じるエラーを検出するための 3 つの補完的な方法について説明します。

トピック

- CloudWatch 複合アラームによる障害検出
- 外れ値検出による障害検出

- 単一インスタンスのゾーンリソースの障害検出
- [概要]

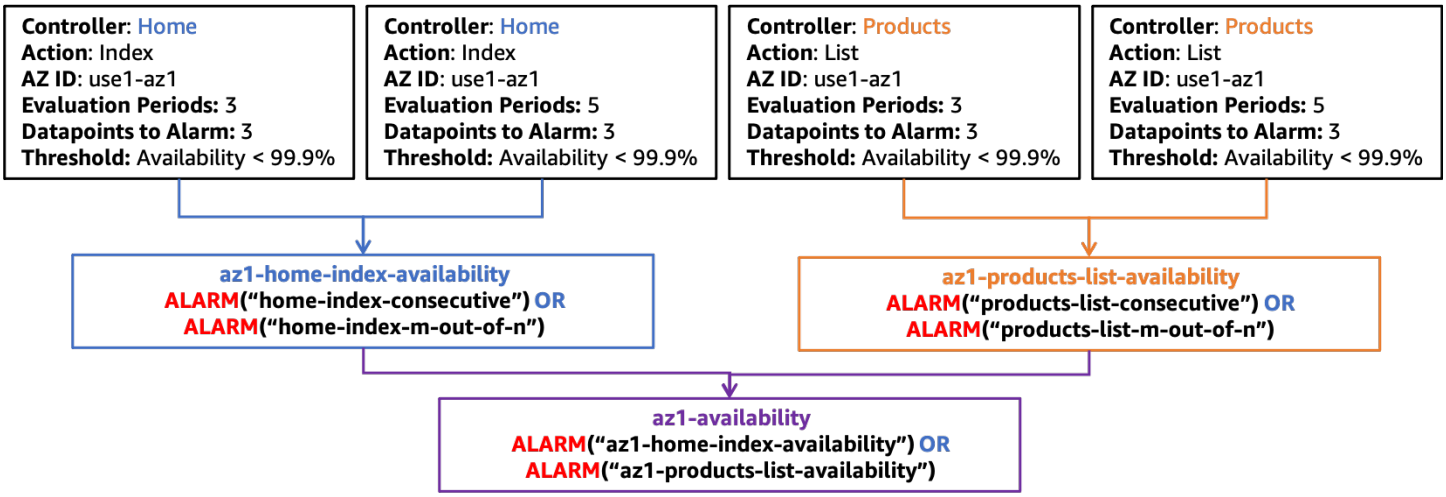
CloudWatch 複合アラームによる障害検出

CloudWatch メトリクスの場合、各ディメンションセットは一意のメトリクスであり、それぞれに CloudWatch アラームを作成できます。次に、Amazon CloudWatch 複合アラームを作成し、これらのメトリクスを集約できます。

影響を正確に検出するために、このホワイトペーパーの例では、アラームの対象となるディメンションセットごとに 2 つの異なる CloudWatch アラーム構造を使用します。各アラームは 1 分間のピリオドを使用します。つまり、メトリクスは 1 分に 1 回評価されます。1 つ目の方法では、評価期間およびアラームへのデータポイントを 3 に設定して (合計 3 分間の影響)、3 つの連続する違反データポイントを使用します。2 つ目の方法では、評価期間を 5、アラームへのデータポイントを 3 に設定して、5 分間の時間枠内で 3 つのデータポイントが違反している場合に、「M out of N」という値を使用します。これにより、一定の信号だけでなく、短時間で変動する信号も検出できます。ここに記載されている期間とデータポイントの数はあくまでも目安です。ワークロードに適した値を使用してください。

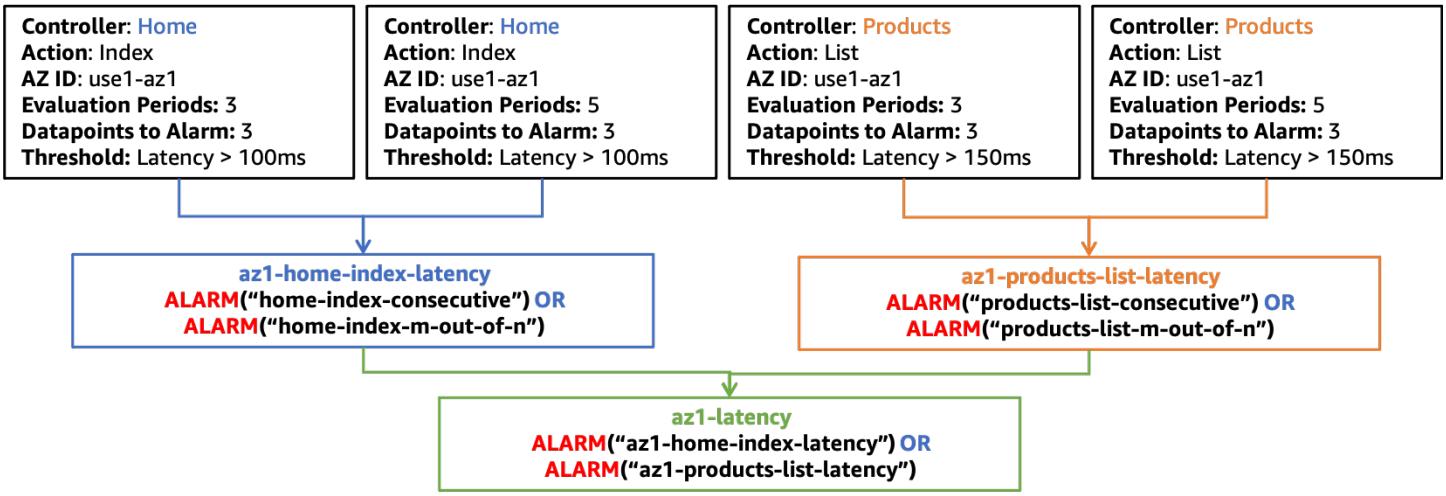
1 つのアベイラビリティーゾーンで影響を検知する

このコンストラクトを使用して、Controller、Action、InstanceId、AZ-ID、および Region を使用するワークロードについて考えてみましょう。ワークロードには、Products と Home の 2 つのコントローラーがあり、コントローラーごとに 1 つのアクション、List と Index があります。このワークロードは、us-east-1 リージョンの 3 つのアベイラビリティーゾーンで動作します。各アベイラビリティーゾーンの Controller と Action の各組み合わせの可用性について 2 つのアラームと、それぞれのレイテンシーについて 2 つのアラームを作成します。次に、オプションで、それぞれの Controller およびAction の組み合わせの可用性について複合アラームを作成することもできます。最後に、アベイラビリティーゾーンのすべてのアベイラビリティアラームを集約する複合アラームを作成します。これについては、1 つのアベイラビリティーゾーン use1-az1 に関する次の図で示しています。ここでは、それぞれの Controller と Action の組み合わせに対してオプションの複合アラームを使用しています (同様のアラームが use1-az2 と use1-az3 アベイラビリティーゾーンでも存在する場合がありますが、わかりやすくするために表示されていません)。



use1-az1 可用性に対する複合アラーム構造

また、次の図に示すように、レイテンシーについても同様のアラーム構造を構築することになります。

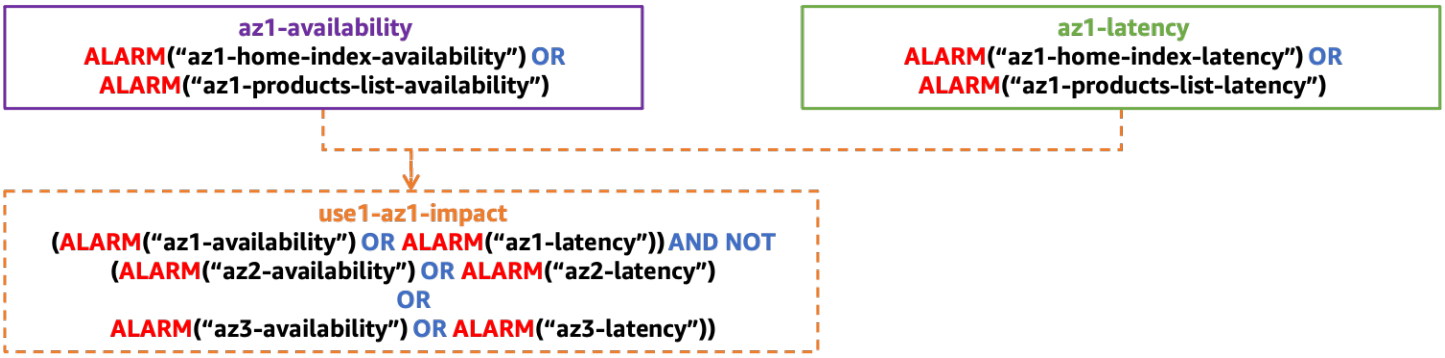


use1-az1 のレイテンシーの複合アラーム構造

このセクションの残りの図では、az1-availability と az1-latency の複合アラームのみがトップレベルに表示されます。これらの複合アラーム az1-availability および az1-latency は、ワークロードのあらゆる部分において、可用性が特定のアベイラビリティーゾーンで定義済みのしきい値を下回るか、またはレイテンシーがそれを上回った場合に通知します。また、スループットを測定して、1 つのアベイラビリティーゾーンのワークロードが作業を受け付けるのを防止する影響を検出することも検討してください。canary によって発行されたメトリクスから生成されたアラームを、これらの複合アラームに統合することもできます。これにより、サーバー側またはクライアント側のどちらかで可用性やレイテンシーへの影響が確認された場合に、アラームによってアラートが作成されます。

影響がリージョンのものではないことを確認する

別の複合アラームのセットを使用して、分離されたアベイラビリティーゾーンのイベントのみにより、アラームをアクティブにすることができます。これを行うには、1 つのアベイラビリティーゾーンの複合アラームを ALARM 状態にし、その他のアベイラビリティーゾーンの複合アラームを OK 状態にします。これにより、使用するアベイラビリティーゾーンごとに 1 つの複合アラームが生成されます。次の図に例を示しています (use1-az2、use1-az3、az2-latency、az2-availability、az3-latency、および az3-availability のレイテンシーと可用性に関するアラームがあることに注意してください、ここでは簡略化のため画像は掲載していません)。



1 つの AZ に隔離された影響を検出する複合アラーム構造

影響が 1 つのインスタンスによるものではないことを確認する

1 つのインスタンス (またはフリート全体のごく一部) が可用性とレイテンシーのメトリクスに不均衡な影響を与える可能性があり、これによって、実際はそうではないのに、アベイラビリティーゾーン全体が影響を受けているように見えることがあります。アベイラビリティーゾーンを評価するよりも、問題のあるインスタンスを 1 つ削除するほう早く、効果的です。

通常、インスタンスとコンテナは一時的なリソースとして扱われ、AWS Auto Scaling などのサービスによく置き換えられます。新しいインスタンスが作成されるたびに新しい CloudWatch アラームを作成することは困難です (ただし、Amazon EventBridge または Amazon EC2 Auto Scaling ライフサイクルフックを使用すると、確実に可能です)。代わりに、CloudWatch Contributor Insights を使用して、可用性とレイテンシーのメトリクスへの寄与要因の数を特定できます。

例えば、HTTP Web アプリケーションの場合、各アベイラビリティーゾーンで 5xx HTTP 応答に関する上位の寄与要因を特定するルールを作成できます。これにより、どのインスタンスが可用性の低下の原因となっているかが特定されます (上記で定義した可用性メトリクスは、5xx エラーの存在によって決まります)。EMF ログの例を使用して、InstanceId のキーでルールを作成します。次に、HttpResponseCode フィールドでログをフィルタリングします。次の例は use1-az1 アベイラビリティーゾーンのルールです。

```
{
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.InstanceId",
                "IsPresent": true
            },
            {
                "Match": "$.HttpStatusCode",
                "IsPresent": true
            },
            {
                "Match": "$.HttpStatusCode",
                "GreaterThan": 499
            },
            {
                "Match": "$.HttpStatusCode",
                "LessThan": 600
            },
            {
                "Match": "$.AZ-ID",
                "In": ["use1-az1"]
            },
        ],
        "Keys": [
            "$.InstanceId"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/loggroupname"
    ],
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    }
}
```

CloudWatch アラームは、これらのルールに基づいて作成することもできます。Metric Math と INSIGHT_RULE_METRIC 関数を UniqueContributors メトリクスで使用して、Contributor

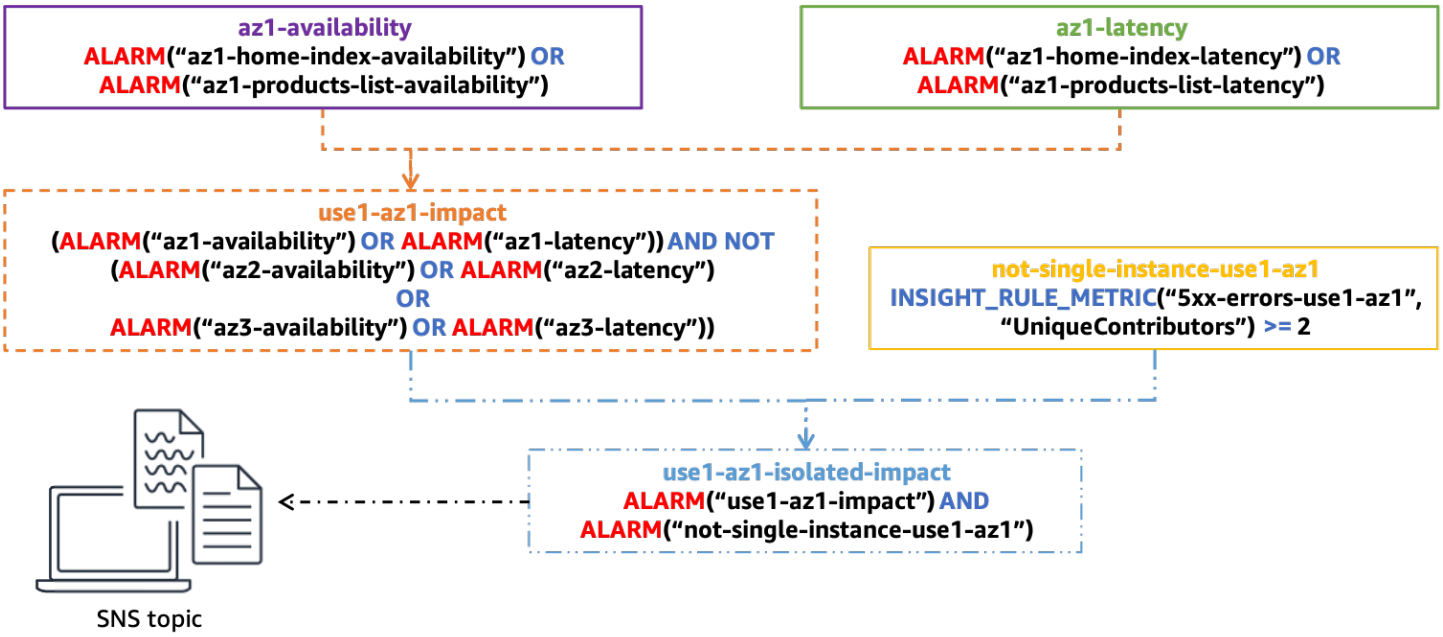
Insights のルールに基づくアラームを作成できます。また、可用性のメトリクスに加えて、レイテンシーやエラー数などのメトリクスに対する CloudWatch アラームを使用して、追加の Contributor Insights ルールを作成することもできます。これらのアラームを、分離されたアベイラビリティーゾーンに対する影響の複合アラームと組み合わせて使用すると、1 つのインスタンスがアラームをアクティブにしていないことを確認できます。use1-az1 のインサイトルールの メトリクスは、次のような場合があります。

```
INSIGHT_RULE_METRIC("5xx-errors-use1-az1", "UniqueContributors")
```

このメトリクスがしきい値 (この例では 2) を超えた場合のアラームを定義できます。5xx 応答に対する一意の寄与要因がこのしきい値を超えると、アラームがアクティブになり、2 つを超えるインスタンスから影響が生じていることを示します。このアラームで「より小さい」比較ではなく「より大きい」比較を使用する理由は、一意の寄与要因の値が 0 である場合にアラームを始動しないようにするためです。これにより、影響が単一のインスタンスからのものではないことがわかります。このしきい値は、ワークロードごとに調整します。一般的な目安は、この数をアベイラビリティーゾーンの総リソースの 5% 以上にすることです。サンプルのサイズが十分であれば、全体の 5% を超えるリソースが影響を受けた場合、統計的有意性を示します。

まとめ

次の図は、1 つのアベイラビリティーゾーンの複合アラーム構造全体を示しています。



1 つの AZ の影響を判断するための複合アラーム構造全体

最後の複合アラーム「use1-az1-isolated-impact」は、レイテンシーまたは可用性からの分離されたアベイラビリティーゾーンの影響を示す use1-az1-aggregate-alarm が ALARM の状態であり、同じアベイラビリティーゾーンの Contributor Insights ルールに基づくアラーム not-single-instance-use1-az1 も ALARM の状態である場合 (つまり、複数のインスタンスから影響が生じている場合) にアクティブになります。このアラームスタックは、ワークロードが使用するアベイラビリティーゾーンごとに作成します。

この最終アラームに、Amazon Simple Notification Service (Amazon SNS) アラートをアタッチできます。以上のすべてのアラームは、アクションなしで設定しています。アラートは、手動調査を開始するよう E メールでオペレーターに通知できます。また、アベイラビリティーゾーンから退避するためのオートメーションを開始することもできます。ただし、これらのアラートに対応するためのオートメーションの構築には注意が必要です。アベイラビリティーゾーンからの退避が起きると、エラー率の増加が緩和され、アラームが OK 状態に戻ります。別のアベイラビリティーゾーンで影響が発生すると、オートメーションは 2 番目や 3 番目のアベイラビリティーゾーンからの退避も起こす場合があり、ワークロードの使用可能なすべての容量が削除される可能性があります。オートメーションでは、アクションを実行する前に、退避が既に実行されていないかどうかを確認する必要があります。退避を成功させるには、他のアベイラビリティーゾーンのリソースをスケールすることが必要になる場合もあります。

MVC Web アプリ、新しいマイクロサービス、または一般として個別に監視したい追加機能に新しいコントローラーやアクションを追加する場合、この設定ではいくつかのアラームを変更するだけで済みます。その新しい機能に新しい可用性アラームとレイテンシーアラームを作成し、それらを適切なアベイラビリティーゾーンに合わせた可用性とレイテンシーの複合アラームに追加します。ここで使用している例では、これらは az1-latency と az1-availability です。残りの複合アラームは、設定後も変化しません。これにより、この方法による新機能のオンボーディングプロセスがより簡単になります。

外れ値検出による障害検出

複数のアベイラビリティーゾーンで相関性のない理由で発生するエラー率が高くなっている場合、前述の方法とのギャップが 1 つ生じることがあります。3 つのアベイラビリティーゾーンに EC2 インスタンスをデプロイしていて、可用性アラームのしきい値が 99% であるシナリオを想像してみてください。その後、1 つのアベイラビリティーゾーンで障害が発生し、多くのインスタンスが分離され、そのゾーンの可用性が 55% まで低下します。同時に、別のアベイラビリティーゾーンでは、1 つの EC2 インスタンスが EBS ボリュームのストレージをすべて使い果たし、ログファイルを書き込めなくなります。これにより エラーが返され始めますが、ロードバランサーのヘルスチェックはログファイルの書き込みをトリガーしないため、それでもパスします。その結果、そのアベイラビリ

ティーゾーンの可用性が 98% に低下します。この場合、複数のアベイラビリティーゾーンで可用性の影響が見られるため、1 つのアベイラビリティーゾーンの影響アラームは有効になりません。ただし、障害のあるアベイラビリティーゾーンを退避させることで、影響のほぼすべてを軽減することはできます。

ワークロードの種類によっては、前の可用性メトリクスが役に立たない可能性があるすべてのアベイラビリティーゾーンで、エラーが一貫して発生することがあります。例えば AWS Lambda の場合について検討してみましょう。AWS は、カスタマーが独自のコードを作成して Lambda 関数で実行できるようにします。このサービスを使用するには、依存関係を含むコードを ZIP ファイルにアップロードし、関数へのエントリポイントを定義する必要があります。しかし、カスタマーはこの部分を間違えることがあります。例えば、ZIP ファイルの重要な依存関係を忘れたり、Lambda 関数定義のメソッド名を間違えたりすることがあります。これにより、関数が呼び出されず、エラーが発生します。AWS Lambda では、この種のエラーは常に発生しますが、必ずしも異常であることを示すものではありません。ただし、アベイラビリティーゾーンの障害などが原因でこれらのエラーが表示されることもあります。

この種のノイズに含まれるシグナルを見つけるには、外れ値検出を使用して、アベイラビリティーゾーン間でエラー数に統計的に有意な偏りがあるかどうかを判断できます。複数のアベイラビリティーゾーンでエラーが見られるものの、そのうちの 1 つで本当に障害が発生した場合、そのアベイラビリティーゾーンでは他のアベイラビリティーゾーンに比べてエラー率が大幅に高くなるか、はるかに低くなる可能性があります。しかし、どれくらい高い、または低いのでしょうか。

この分析を行う方法の 1 つは、カイ二乗 (χ^2) 検定を使用してアベイラビリティーゾーン間のエラー率の統計的に有意な差を検出することです (外れ値検出を実行するためのさまざまなアルゴリズムがあります)。それでは、カイ二乗検定の仕組みを見てみましょう。

カイ二乗検定では、何らかの結果の分布が生じる確率を評価します。ここでは、一連の定義済みの AZ 全体にわたるエラーの分布を調べます。この例では、計算を簡単にするために、4 つのアベイラビリティーゾーンを考えてみます。

まず、帰無仮説を立てます。これにより、何をデフォルトの結果と信じるかを定義します。この検定では、帰無仮説として、エラーは各アベイラビリティーゾーンに均等に分布していると予想します。次に、エラーは均等に分布しておらず、アベイラビリティーゾーンの障害が示唆されているという対立仮説を立てます。これで、メトリクスのデータを使用して、これらの仮説を検定できるようになります。この目的のために、5 分間のウィンドウからメトリクスをサンプリングします。このウィンドウに 1,000 個の公開されたデータポイントがあり、合計で 100 個のエラーが確認されたとします。均等な分布では、4 つのアベイラビリティーゾーンのそれぞれで 25% の確率でエラーが発生すると予想されます。次の表は、予想した結果と実際の結果の比較を示しているものとします。

表 1: 予想したエラーと実際に確認されたエラー

AZ	予想	実際
use1-az1	25	20
use1-az2	25	20
use1-az3	25	25
use1-az4	25	35

したがって、実際には分布が均等ではないことがわかります。ただし、サンプリングしたデータポイントにある程度のランダム性があったために、このような結果になったとも考えられます。この種の分布がサンプルセットで発生する可能性がある程度あると、帰無仮説がまだ成り立つ余地があります。これは次の質問につながります。少なくともこれくらいの極端な結果が得られる確率はどれくらいであるか。その確率が定義したしきい値を下回っている場合は、帰無仮説を棄却します。統計的に有意であるためには、この確率は 5% 以下でなければなりません。[1]

[1] Craparo, Robert M. (2007). 「有意水準」。Salkind, Neil J、Encyclopedia of Measurement and Statistics3. Thousand Oaks、CA: SAGE Publications。889〜891ページ。ISBN 1-412-91611-9.

この結果の確率は、どのように計算すればよいでしょうか。よく研究された分布を提供する χ^2 統計を使用します。これは、この公式を使ってこれくらい極端な、あるいはもっと極端な結果が得られる確率を決定できます。

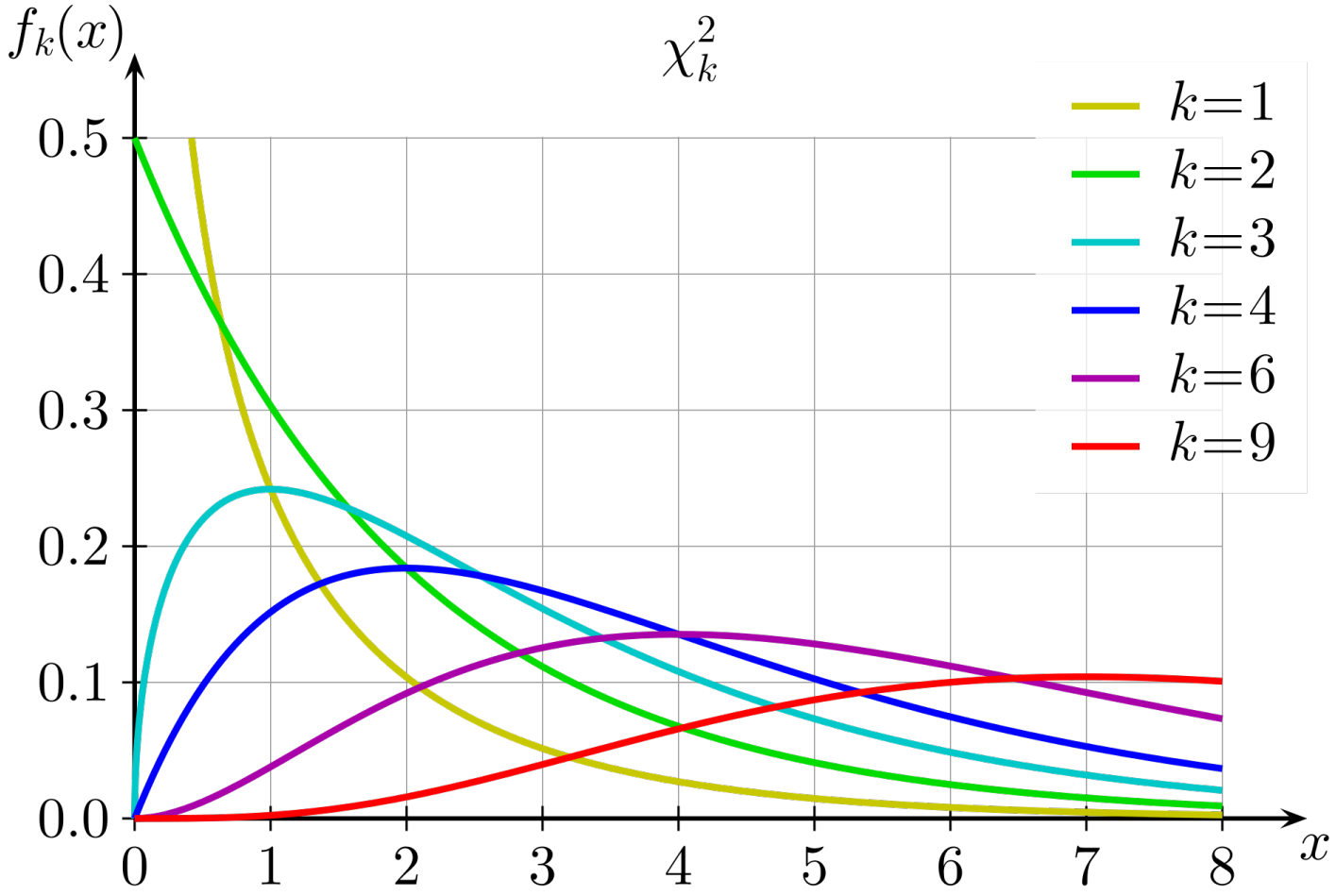
$$E_i = expected\ observations\ of\ type\ i$$
$$O_i = actual\ observations\ of\ type\ i \qquad (1)$$

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

この例では、次のような結果になります。

$$\chi^2 = \frac{(20-25)^2}{25} + \frac{(20-25)^2}{25} + \frac{(25-25)^2}{25} + \frac{(35-25)^2}{25}$$
$$\chi^2 = \frac{-5^2}{25} + \frac{-5^2}{25} + \frac{0^2}{25} + \frac{10^2}{25} \qquad (2)$$
$$\chi^2 = 1 + 1 + 0 + 4$$
$$\chi^2 = 6$$

それでは、確率の観点から 6 は何を意味するでしょうか。カイ二乗分布は、適切な自由度で調べる必要があります。次の図は、さまざまな自由度でいくつかの異なるカイ二乗分布を示しています。



さまざまな自由度でのカイ二乗分布

自由度は、テストの選択肢の数より 1 つ少ない数として計算します。この場合、アベイラビリティーゾーンは 4 つあるため、自由度は 3 になります。次に、k = 3 プロットで x ≥ 6 の曲線の下の

面積 (積分) を知りたいとします。一般的に使用される値を含む、事前に計算された表を使用して、その値を概算することもできます。

表 2: カイ二乗の臨界値

自由度	臨界値を下回る確率				
	0.75	0.90	0.95	0.99	0.999
1	1.323	2.706	3.841	6.635	10.828
2	2.773	4.605	5.991	9.210	13.816
3	4.108	6.251	7.815	11.345	16.266
4	5.385	7.779	9.488	13.277	18.467

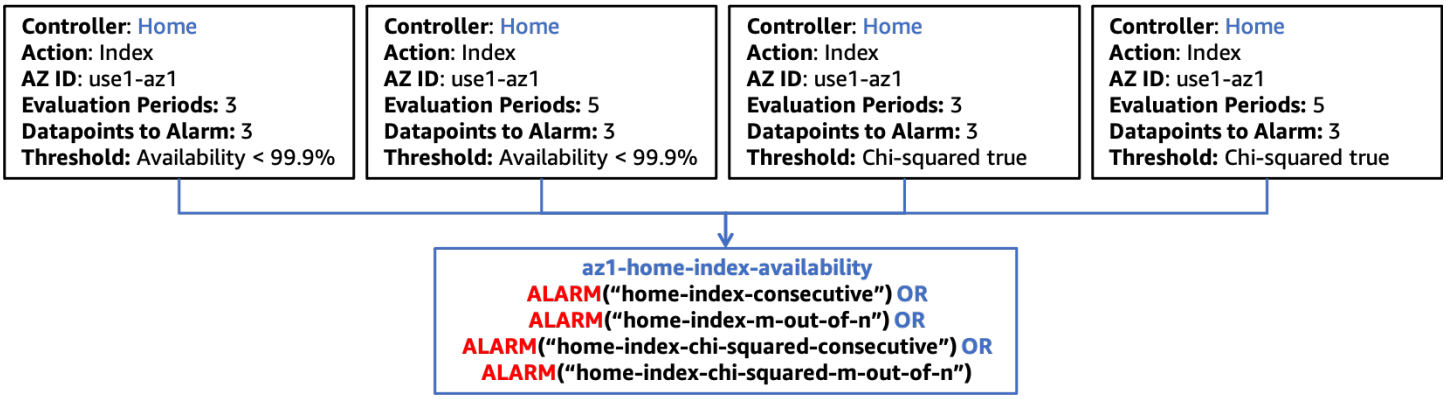
自由度が 3 の場合、カイ二乗値 6 は、確率列 0.75 および 0.9 の間になります。つまり、この分布が発生する可能性は 10% を超え、しきい値の 5% を下回らないということです。したがって、帰無仮説を受け入れ、アベイラビリティーゾーン間のエラー率には統計的に有意な差はないと判断します。

カイ二乗統計テストの実行は CloudWatch メトリクスの計算ではネイティブにサポートされていないため、CloudWatch から該当するエラーメトリクスを収集し、Lambda などのコンピューティング環境でテストを実行する必要があります。このテストは、MVC コントローラー/アクション、個々のマイクロサービスレベル、またはアベイラビリティーゾーンレベルなどで実行するかを決めることができます。アベイラビリティーゾーンの障害が各コントローラー/アクションまたはマイクロサービスに等しく影響するのか、それとも DNS 障害のようなものが高いスループットのサービスではなく低いスループットのサービスに影響を与えてるのか、つまり集約したときにその影響を隠すことができるのかをについて検討する必要があります。いずれの場合も、適切なディメンションを選択してクエリを作成します。詳細度のレベルは、作成する結果の CloudWatch アラームにも影響します。

指定された時間枠内の各 AZ とコントローラー/アクションのエラー数メトリクスを収集します。まず、カイ二乗検定の結果を true (統計的に有意な偏りがあった) または false (なかった、つまり帰無仮説が成立する) のいずれかで計算します。結果が false の場合は、各アベイラビリティーゾーンの結果をカイ二乗するため、メトリクスストリームに 0 データポイントをパブリッシュします。結果が true の場合は、エラーが予想値から最も遠いアベイラビリティーゾーンの 1 データポイント、その他には 0 データポイントをパブリッシュします (Lambda 関数で使用できるサンプルコードについては 付録 B — カイ二乗計算の例 を参照)。Lambda 関数によって生成されるデータポイントに基づ

いて、3 つ連続の CloudWatch メトリクスアラームと 5 つのうち 3 つの CloudWatch メトリクスアラームを作成することで、以前の可用性アラームと同じアプローチをとることができます。前の例と同様に、この方法を変更して、短い枠または長い枠で使用するデータポイントを増減することができます。

次に、以下の図に示すように、コントローラーとアクションの組み合わせの既存のアベイラビリティーゾーンにこれらのアラームを追加します。



カイ二乗統計検定と複合アラームの統合

前述のように、ワークロードに新しい機能をオンボードするとき、その新機能に固有の適切な CloudWatch メトリクスアラームを作成し、複合アラーム階層の次の層を更新してそれらのアラームを含めるのみになります。残りのアラーム構造は変化しません。

単一インスタンスのゾーンリソースの障害検出

場合によっては、ゾーンリソースのアクティブなインスタンスが 1 つしかないことがあります。最も一般的には、リレーショナルデータベース (Amazon RDS など) や分散キャッシュ (Amazon ElastiCache (Redis OSS) など) などの単一のライターコンポーネントを必要とするシステムが該当します。1 つのアベイラビリティーゾーンの障害がプライマリリソースのあるアベイラビリティーゾーンに影響する場合、そのリソースにアクセスするすべてのアベイラビリティーゾーンに影響が及ぶ可能性があります。これにより、すべてのアベイラビリティーゾーンで可用性のしきい値を超える可能性があります。つまり、最初の方法では、影響のソースである 1 つのアベイラビリティーゾーンを正しく特定できません。さらに、各アベイラビリティーゾーンで同様の エラー率が見られる可能性が高いため、外れ値分析でも問題は検出されません。つまり、このシナリオを具体的に検出するには、追加のオブザーバビリティを実装する必要があります。

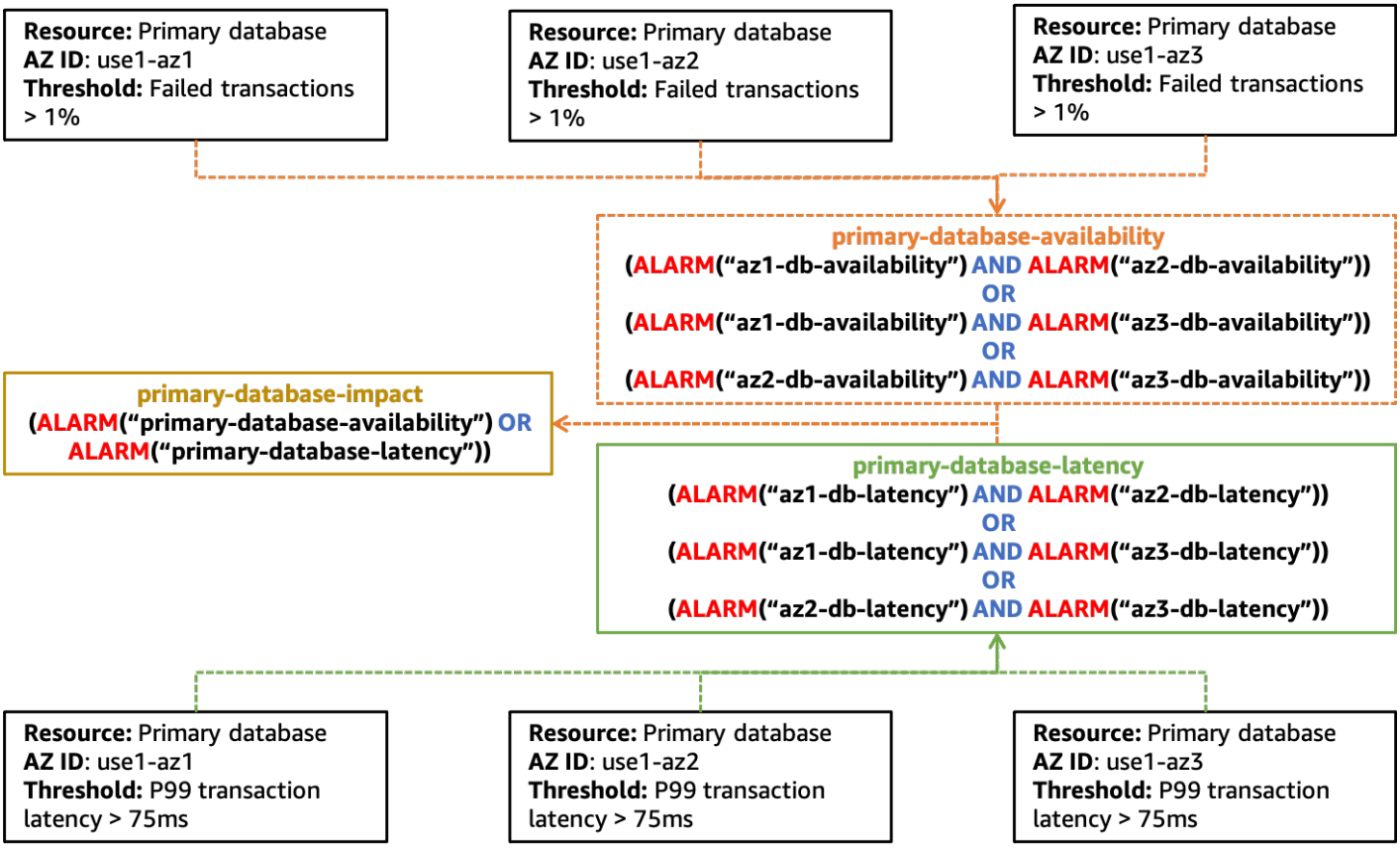
懸念するリソースがその状態に関する独自のメトリクスを生成する可能性が高いですが、アベイラビリティーゾーンに障害が発生している間は、そのリソースはそれらのメトリクスを提供できない場合

があります。このシナリオでは、アラームを作成または更新して、いつ盲目飛行しているのかを知る必要があります。すでに監視してアラームを設定している重要なメトリクスが存在する場合は、欠如しているデータを違反として処理するようにアラームを設定できます。これにより、そのリソースがデータの報告を停止したかどうかを知ることができ、そのリソースを以前に使用した同じ in a row および m out of n アラームに含めることができます。

また、リソースの状態を示す一部のメトリクスでは、アクティビティがないときにゼロ値のデータポイントがパブリッシュされることもあります。障害によってリソースとのやり取りが妨げられているのであれば、この種のメトリクスには欠損データのアプローチを使用することはできません。また、値がゼロになってもおそらく警告する必要はありません。なぜなら、それが通常のしきい値の範囲内である場合があるからです。この種の問題を検出する最善の方法は、この依存関係を使用するリソースによって生成されるメトリックを使用することです。この場合、複合アラームを使用して複数のアベイラビリティーゾーンでの影響を検出する必要があります。これらのアラームでは、リソースに関連する重要なメトリクスカテゴリをいくつか使用する必要があります。以下にいくつかの例を示します。

- スループット — 受け取る作業単位の割合。これには、トランザクション、読み取り、書き込みなどが含まれます。
- 可用性 — 成功した作業単位と失敗した作業単位の数を測定します。
- レイテンシー— 重要なオペレーション全体にわたって正常に実行された作業のレイテンシーのパーセンタイルを複数測定します。

  ここでも、測定する各メトリクスカテゴリのメトリクスに対して in a row および m out of n メトリクスアラームを作成できます。前と同様に、これらを複合アラームにまとめて、この共有リソースがアベイラビリティーゾーン全体に影響を与えるソースであるかどうかを判断できます。複合アラームを使用して複数のアベイラビリティーゾーンへの影響を特定できるようにしたいと思うものの、必ずしもその影響がすべてのアベイラビリティーゾーンである必要はありません。次の図は、この種のアプローチにおける高レベルの複合アラーム構造を示しています。



1 つのリソースによる複数のアベイラビリティーゾーンへの影響を検出するアラームの作成例

この図は、使用するメトリクスアラームのタイプと複合アラームの階層について、より規範的ではないことに気付くでしょう。これは、この種の問題を発見するのは難しい場合があり、共有リソースに適したシグナルに細心の注意を払う必要があるためです。これらの信号は、特定の方法で評価しなければならない場合もあります。

さらに、`primary-database-impact` アラームは特定のアベイラビリティーゾーンに関連付けられていない点にも注意してください。これは、プライマリデータベースインスタンスは、使用するように設定されているどのアベイラビリティーゾーンにも配置でき、その場所を指定する CloudWatch メトリクスがないためです。このアラームが有効になったら、リソースに問題がある可能性を示すシグナルとして使用し、自動的に実行されていない場合は、別のアベイラビリティーゾーンへのフェイルオーバーを開始する必要があります。リソースを別のアベイラビリティーゾーンに移動したら、隔離されたアベイラビリティーゾーンのアラームが有効になるかどうかを確認するか、またはアベイラビリティーゾーンの退避計画を先制的に呼び出すかを選択できます。

[概要]

このセクションでは、1 つのアベイラビリティーゾーンの障害を特定するのに役立つ 3 つの方法について説明しました。ワークロードの状態を全体的に把握するには、それぞれの方法を組み合わせて使用する必要があります。

CloudWatch 複合アラームの方法では、例えば、単一の共有リソースが原因ではない 98% (アベイラビリティーゾーンの障害)、100%、99.99% の可用性など、可用性の偏りが統計的に有意でない場合に問題を検出することができます。

異常値検出は、複数のアベイラビリティーゾーンで相関関係のないエラーが発生し、そのすべてがアラームのしきい値を超えている場合に、1 つのアベイラビリティーゾーンの障害を検出するのに役立ちます。

最後に、単一インスタンスのゾーンリソースの劣化を特定すると、アベイラビリティーゾーンの障害がアベイラビリティーゾーン間で共有されているリソースにいつ影響を与えるかを発見するのに役立ちます。

これらのパターンのそれぞれから生成されたアラームを CloudWatch 複合アラーム階層にまとめることで、1 つのアベイラビリティーゾーンの障害が発生し、ワークロードの可用性やレイテンシーに影響が及ぶときを検出できます。

アベイラビリティーゾーンの退避パターン

1 つのアベイラビリティーゾーンで影響を検出したら、次のステップでは、そのアベイラビリティーゾーンを退避させます。退避によって達成すべき成果は 2 つあります。

まず、影響を受けているアベイラビリティーゾーンへの作業の送信を停止する必要があります。これは、アーキテクチャによって意味が異なる可能性があります。リクエスト/レスポンスのワークロードでは、カスタマーからの HTTP や gRPC リクエストなどが、ロードバランサーやアベイラビリティーゾーンの他のリソースに送信されるのを停止することになります。バッチ処理またはキュー処理システムでは、影響を受けているアベイラビリティーゾーンでのコンピューティングリソースの処理が停止する可能性があります。また、影響を受けていないアベイラビリティーゾーンのリソースが、影響を受けているアベイラビリティーゾーンのリソースと相互作用しないようにする必要があります。例えば、影響を受けているアベイラビリティーゾーン内のインターフェイス VPC エンドポイントにトラフィックを送信したり、またはデータベースのプライマリインスタンスに接続中の EC2 インスタンスなどです。

2 つ目の成果は、影響を受けているアベイラビリティーゾーンに新しいキャパシティがプロビジョニングされないようにすることです。影響を受けているアベイラビリティーゾーンでプロビジョニングされる EC2 インスタンスやコンテナなどの新しいリソースは、既存のリソースと同じ影響を受ける可能性が高いため、これは重要です。また、最初の成果によって作業がそれらに送信されなくなるため、処理するようにプロビジョニングされた負荷を吸収できません。これにより、既存のリソースへの負荷が増加し、最終的にはワークロードがブラウンアウトしたり、完全に利用できなくなったりする可能性があります。AWS には、この対策に使用できる自動スケーリングサービスとして、Amazon EC2 Auto Scaling、Application Auto Scaling、AWS Auto Scaling などがあります。さらに、Amazon ECS、Amazon EKS、AWS Batch などのサービスは、通常のオペレーションの一環として、VPC 内のすべてのアベイラビリティーゾーンにわたってホストでの作業をスケジュールできます。

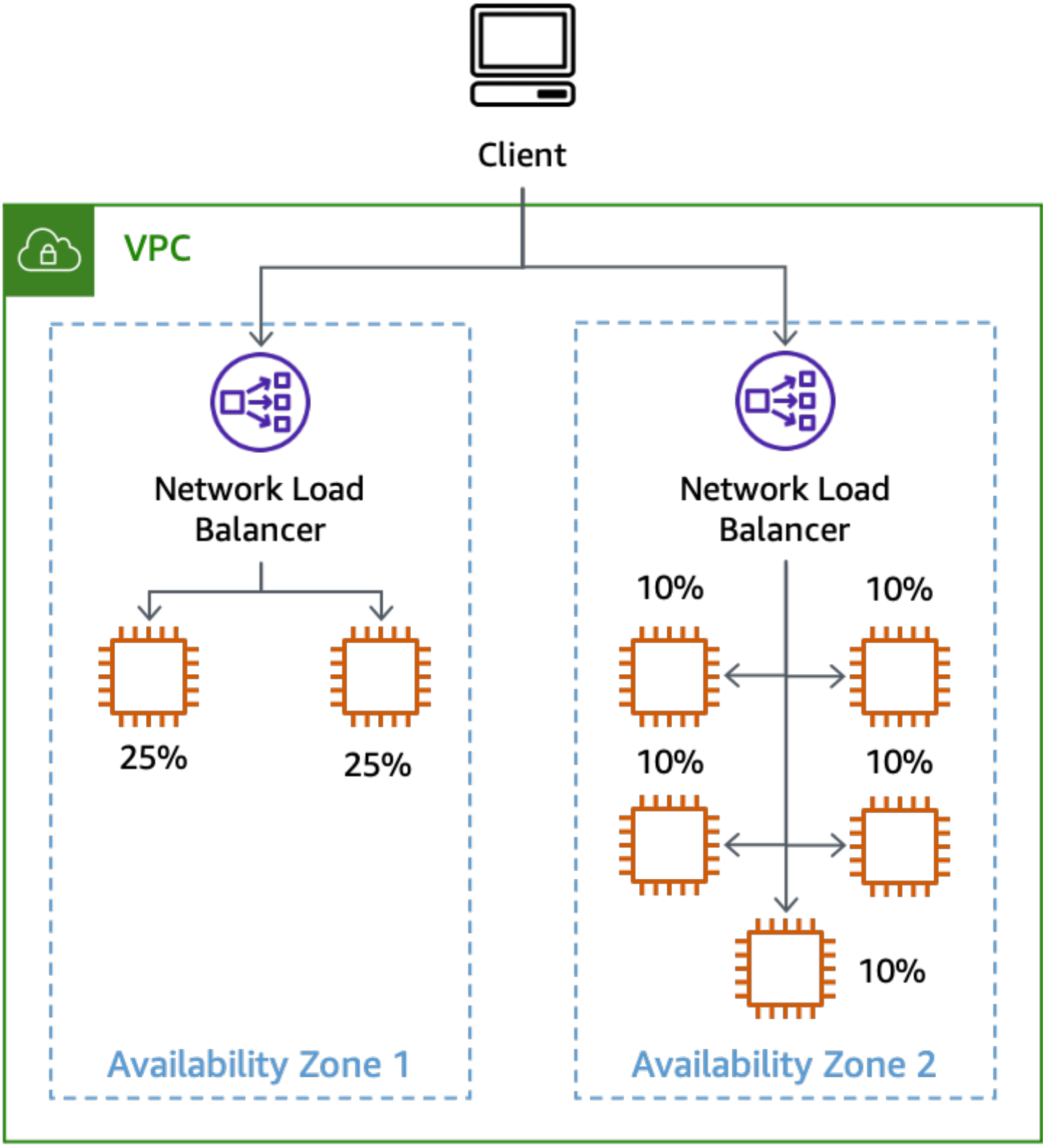
トピック

- アベイラビリティーゾーンの独立性
- コントロールプレーンとデータプレーン
- データプレーン制御の退避
- コントロールプレーン制御による退避
- まとめ

アベイラビリティーゾーンの独立性

最初の成果として、影響を受けているアベイラビリティーゾーンへの作業の送信を停止するには、アベイラビリティーゾーンの独立性 (AZI) (アベイラビリティーゾーンのアフィニティと呼ばれることもあります) を実装する必要があります。このアーキテクチャパターンは、別のアベイラビリティーゾーンのプライマリデータベースインスタンスへの接続など、絶対に必要な場合を除いて、アベイラビリティーゾーン内のリソースを分離し、異なるアベイラビリティーゾーンのリソース間のやり取りを防止します。

リクエスト/レスポンスタイプのワークロードでは、AZI を実装するには Application Load Balancer (ALB)、Classic Load Balancer (CLB)、および Network Load Balancer (NLB) (NLB では、クロスゾーン負荷分散はデフォルトで無効) のクロスゾーン負荷分散を無効にする必要があります。クロスゾーン負荷分散を無効にすることには、いくつかのトレードオフがあります。クロスゾーン負荷分散を無効にすると、各インスタンスにあるインスタンスの数に関係なく、トラフィックは各アベイラビリティーゾーンに均等に分配されます リソースや Auto Scaling グループが不均衡な場合、他よりもリソースの少ないアベイラビリティーゾーンのリソースにさらに負荷をかける可能性があります。これを次の図に示します。アベイラビリティーゾーン 1 の 2 つのインスタンスはそれぞれ 25% の負荷を受け取り、アベイラビリティーゾーン 2 の 5 つのインスタンスはそれぞれ 10% の負荷を受け取っています。



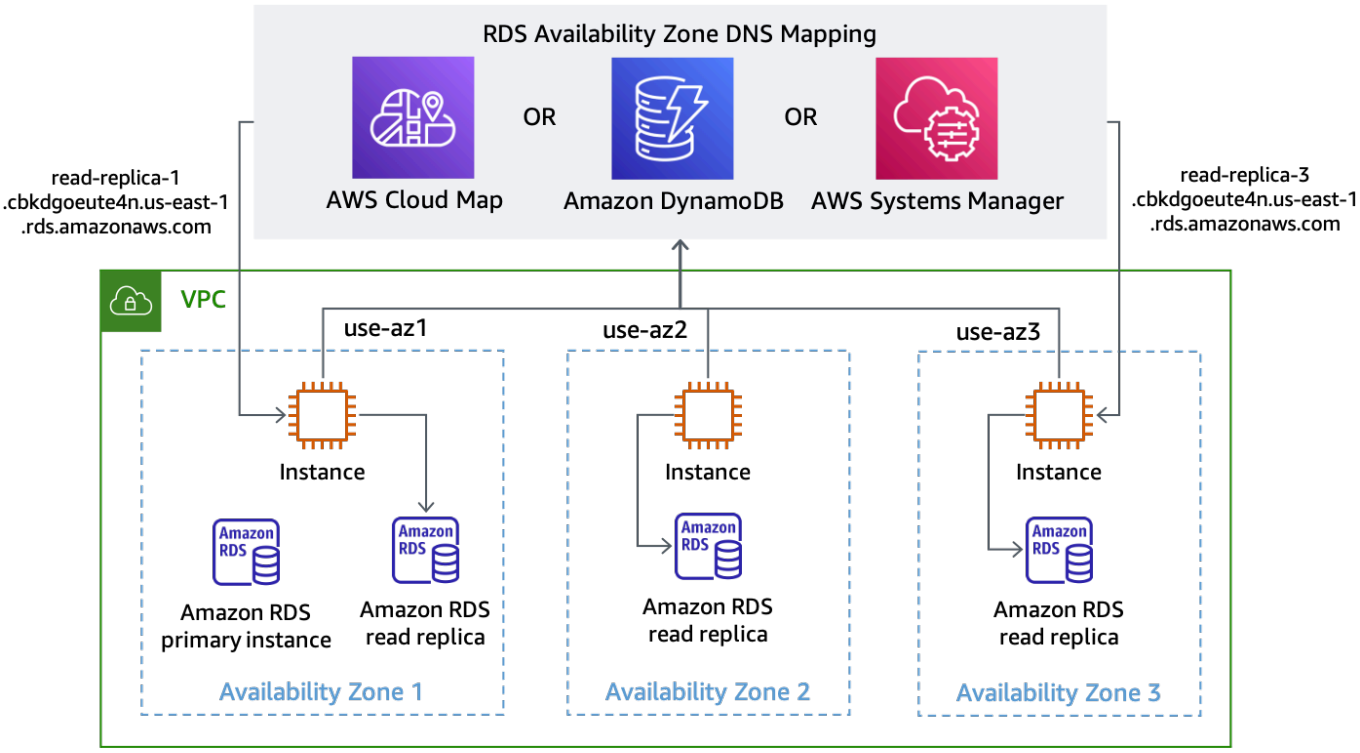
不均衡なインスタンスでクロスゾーン負荷分散を無効にした場合の影響

アベイラビリティーゾーンからの効果的な退避をサポートするには、使用する他のゾーンサービスも AZI パターンを使用して実装する必要があります。例えば、インターフェイス VPC エンドポイントは、インターフェイスエンドポイントを使用可能にするアベイラビリティーゾーンごとに特定の DNS 名を提供します。

AZI を実装する場合の課題の 1 つはデータベースです。特に、ほとんどのリレーショナルデータベースがサポートしているのは常に 1 つのプライマリライターのみだからです。プライマリインスタンスと通信するとき、アベイラビリティーゾーンの境界を越える必要がある場合があります。AWS データベースサービスの多くは、ユーザー定義のマルチ AZ 設定をサポートしており、Amazon RDS や Amazon Aurora などのマルチ AZ フェイルオーバー機能が組み込まれています。多くの障害シナリオでは、サービスが影響を検出し、問題が発生したときにデータベースを別のアベイラビリティーゾーンに自動的にフェイルオーバーできます。ただし、グレー障害の場合、サービスがワークロードに影響を及ぼしている影響を検出しなかったり、その影響がデータベースとはまったく関係ない場合があります。このような場合、アベイラビリティーゾーンで影響が検出されたら、手動でフェイルオーバーを呼び出してプライマリデータベースを移動できます。これにより、1 つのアベイラビリティーゾーンの障害に効果的に対応できます。

これらのデータベースでリードレプリカを使用している場合は、プライマリデータベースのようにリードレプリカを別のアベイラビリティーゾーンにフェイルオーバーできないため、それらのデータベースに AZI を実装する必要もあります。アベイラビリティーゾーン 1 に 1 つのリードレプリカがあり、3 つのアベイラビリティーゾーンのインスタンスがそれを使用するように設定されている場合、アベイラビリティーゾーン 1 に影響する障害は他の 2 つのアベイラビリティーゾーンのオペレーションにも影響を及ぼします。防ぐ必要があるのはその影響です。

RDS インスタンスの場合、特定のアベイラビリティーゾーンのレプリカにアクセスするための DNS エンドポイントを受け取ります。AZI を実現するには、アベイラビリティーゾーンごとにリードレプリカを用意し、そのアベイラビリティーゾーンでどのレプリカエンドポイントを使用するかをアプリケーションが判断する方法が必要です。考えられる方法の 1 つは、`use1-az1-read-replica.cbkdgoeute4n.us-east-1.rds.amazonaws.com` など、アベイラビリティーゾーン ID をデータベース識別子の一部として使用することです。また、サービス検出 (AWS Cloud Map など) を使用したり、またはAWS システムマネージャーパラメータストアまたは DynamoDB テーブルに保存されている簡単なマップを検索して行うこともできます。この概念を次の図に示します。

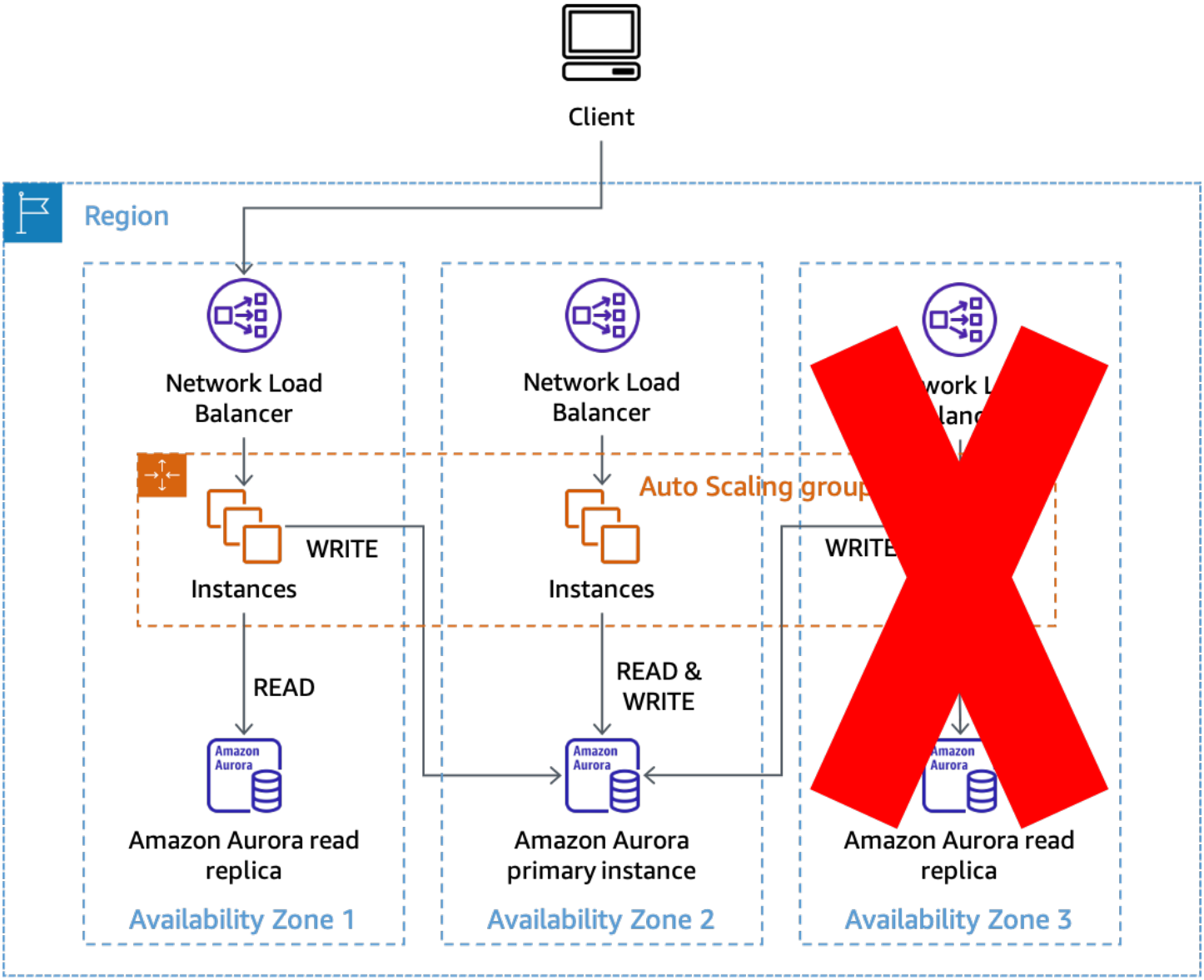


各アベイラビリティーゾーンの RDS エンドポイント DNS 名の検出

Amazon Aurora のデフォルト設定では、使用可能なリードレプリカ間でリクエストの負荷が分散されるシングルリーダーエンドポイントを提供します。Aurora を使用して AZI を実装するには、ANY タイプを使って各リードレプリカにカスタムエンドポイントを使用します (それにより、必要に応じてリードレプリカをプロモートできるようにします)。レプリカがデプロイされているアベイラビリティーゾーン ID に基づいてカスタムエンドポイントに名前を付けます。次に、カスタムエンドポイントから提供された DNS 名を使用して、特定のアベイラビリティーゾーンの特定のリードレプリカに接続できます (次の図を参照)。

Aurora リードレプリカにカスタムエンドポイントを使用する方法

システムがこのように設計されているとき、アベイラビリティーゾーンの退避作業ははるかにシンプルになります。例えば、以下の図では、アベイラビリティーゾーン 3 に影響する障害が発生しても、アベイラビリティーゾーン 1 と 2 の読み取りおよび書き込みオペレーションはどちらも影響を受けません。



AZI を使用して Amazon Aurora リードレプリカによる影響を防止する方法

あるいは、アベイラビリティーゾーン 2 が影響を受けた場合でも、アベイラビリティーゾーン 1 と 3 で読み取りオペレーションは成功します。その後、Amazon Aurora がプライマリデータベースを自動的にフェイルオーバーしていない場合は、手動で別のアベイラビリティーゾーンへのフェイルオーバーを呼び出して、書き込み処理機能を回復できます。この方法により、アベイラビリティーゾーンを退避させる必要があるときに、データベース接続の設定を変更する必要がなくなります。必要な変更を最小限に抑え、プロセスをできるだけシンプルに保つことで、信頼性が向上します。

コントロールプレーンとデータプレーン

アベイラビリティーゾーンの退避に使用できる実際のパターンについて説明する前に、コントロールプレーンとデータプレーンの概念について説明する必要があります。AWS は、サービスでコントロールプレーンとデータプレーンを区別します。コントロールプレーンは、システムに変更 (リソースの追加、削除、変更) を行い、これらの変更を必要な個所に伝播して ALB のネットワーク設定の更新や AWS Lambda 関数の作成などを行うために必要なメカニズムです。

データプレーンは、EC2 インスタンスの実行、Amazon DynamoDB テーブルからの項目の取得、同テーブルへの項目の追加など、これらのリソースの主な機能を担います。コントロールプレーンとデータプレーンの詳細については、「可用性ゾーンを使用した静的安定性」と「AWS 障害分離境界」を参照してください。

このドキュメントの目的において、コントロールプレーンは、データプレーンよりも変動要素と依存関係が多い傾向があることを考慮します。このため、データプレーンに比べてコントロールプレーンは障害を受ける可能性が統計的に高くなります。これは、Amazon EC2 や EBS などの AZI を提供するサービスに特に当てはまります。これらのサービスの一部には、ゾーンごとに独立しており、シングル AZ イベント中に影響を受ける可能性があるコントロールプレーンがあるためです。

以上の情報に基づいてコントロールプレーンのアクションを使用して AZI 退避を実行することはできますが、成功する確率は (特に障害発生時には) 低くなる可能性があります。影響の軽減に成功する確率を高めるには、2 つの異なるパターンを使用できます。最初のパターンでは、データプレーンのアクションのみに依存して、影響を受けたアベイラビリティーゾーンに作業がルーティングされないようにするか、影響を受けたアベイラビリティーゾーンでの作業を停止することで、影響を軽減します。次に、2 番目のパターンでは、コントロールプレーンのアクションでリソースの設定を更新することで、影響を受けたアベイラビリティーゾーンでの容量のプロビジョニングを阻止するとともに、そのアベイラビリティーゾーンと他のアベイラビリティーゾーンとの通信を停止できます。

このセクションで説明する復旧パターンは赤い緊急ボタンです。これは、組立ラインでのアンドンのひもを引っ張るのと同じように、大規模なアクションをすばやく実行するためのメカニズムです。このメカニズムでは、ワークロードが一時的なエラーを克服するために、コード内でジッターを伴うエクスポネンシャルバックオフによる再試行などの戦略を既に試行したことを前提としています。これは、分離されたアベイラビリティーゾーンの影響が検出された場合、可用性やレイテンシーに与える影響が深刻であり、影響を効果的に軽減するにはアベイラビリティーゾーンから退避する必要があることを意味します。

データプレーン制御の退避

データプレーンのみのアクションを使用してアベイラビリティーゾーンの退避を実行するのに実装できるソリューションはいくつかあります。このセクションでは、そのうちの 3 つと、それらのいずれかを選択する必要のあるユースケースについて説明します。

これらのソリューションのいずれかを使用する場合は、移行するアベイラビリティーゾーンの負荷を処理するのに十分なキャパシティが残りのアベイラビリティーゾーンにあることを確認する必要があります。最も回復力の高いのは、各アベイラビリティーゾーンで必要なキャパシティを事前にプロビジョニングしておくことです。3 つのアベイラビリティーゾーンを使用している場合、ピーク負荷を処理するのに必要なキャパシティの 50% を各ゾーンにデプロイできます。そのため、1 つのアベイラビリティーゾーンが失われても、コントロールプレーンに頼って追加のプロビジョニングを行うことなく、必要なキャパシティの 100% を確保できます。

さらに、EC2 Auto Scaling を使用している場合は、シフト中に Auto Scaling グループ (ASG) がスケールインしないようにします。そうすれば、シフトが終了しても、グループにはカスタマーのトラフィックを処理するのに十分なキャパシティが残ります。そのためには、ASG に必要な最小キャパシティが現在のカスタマーの負荷に対応できるようにする必要があります。また、P90 や P99 のような外れ値の多いパーセンタイルメトリクスではなく、メトリクスに平均値を使用することで、ASG が誤ってスケールインすることを防ぐこともできます。

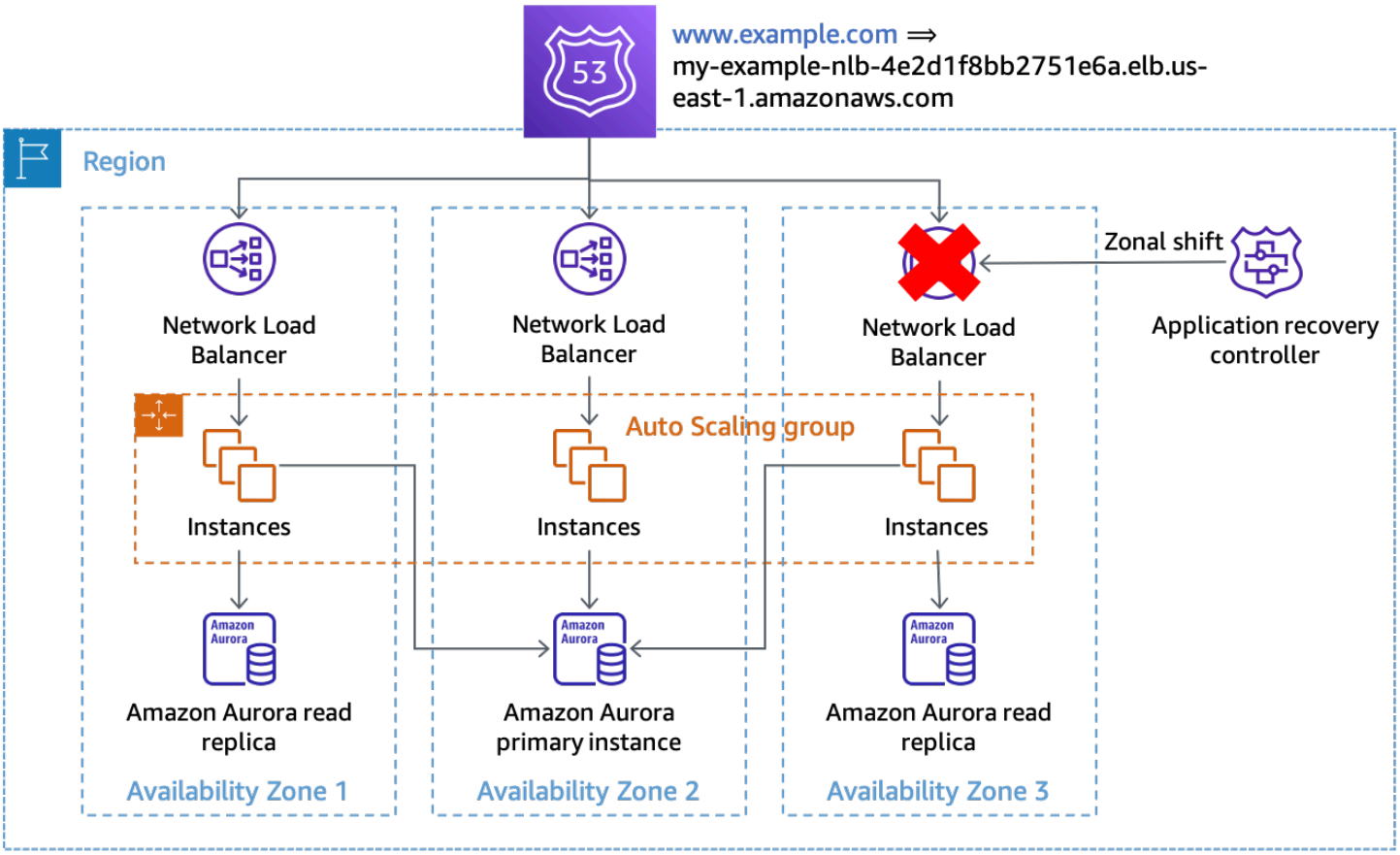
シフト中は、トラフィックを処理しなくなったリソースの使用率は非常に低くなるはずですが、他のリソースは新しいトラフィックに合わせて使用率を高め、平均値がほぼ一定に保たれるため、スケールインアクションが防止されます。最後に、ALB および NLB に対してターゲットグループの健全性設定を使用し、DNS フェイルオーバーを正常なホストの割合または数で指定することもできます。これにより、正常なホストが十分にないアベイラビリティーゾーンにトラフィックがルーティングされるのを防ぎます。

Route 53 Application Recovery Controller (ARC) のゾーンシフト

アベイラビリティーゾーンの退避の最初のソリューションでは、Route 53 ARC のゾーンシフトを使用します。このソリューションは、NLB または ALB をカスタマートラフィックの入力ポイントとして使用するリクエスト/レスポンスワークロードに使用できます。

アベイラビリティーゾーンに障害が発生したことを検出すると、Route 53 ARC でゾーンシフトを開始できます。このオペレーションが完了し、既存のキャッシュされた DNS レスポンスの有効期限が切れると、新しいリクエストはすべて残りのアベイラビリティーゾーンのリソースにのみルーティングされます。次の図は、ゾーンシフトの仕組みを示しています。以下の図では、`my-example-`

nlb-4e2d1f8bb2751e6a.elb.us-east-1.amazonaws.com を指定する www.example.com の Route 53 エイリアスレコードがあります。ゾーンシフトはアベイラビリティーゾーン 3 に対して実行されます。



ゾーンシフト

この例では、プライマリデータベースインスタンスがアベイラビリティーゾーン 3 にない場合、最初の退避の成果を得るために必要なアクションはゾーンシフトの実行のみで、影響を受けているアベイラビリティーゾーンでは作業が処理されません。プライマリノードがアベイラビリティーゾーン 3 にあった場合、Amazon RDS がまだ自動的にフェイルオーバーしていなければ、ゾーンシフトと連携して (Amazon RDS コントロールプレーンに依存する) 手動で開始したフェイルオーバーを実行できます。これは、このセクションのすべてのデータプレーン制御ソリューションに当てはまります。

退避の開始に必要な依存関係を最小限に抑えるために、CLI コマンドまたは API を使用してゾーンシフトを開始する必要があります。退避プロセスが簡単であればあるほど、信頼性が向上します。特定のコマンドは、オンコールエンジニアが簡単にアクセスできるローカルランブックに保存できます。アベイラビリティーゾーンの退避には、ゾーンシフトが最も好ましく、最もシンプルなソリューションです。

Route 53 ARC

2 つ目のソリューションは、Route 53 ARC の機能を使用して特定の DNS レコードの状態を手動で指定する方法です。このソリューションには、可用性の高い Route 53 ARC クラスターデータプレーンを使用できるという利点があり、最大 2 つの異なる AWS リージョン の障害に対する耐障害性が得られます。これには追加コストがかかるというトレードオフがあり、DNS レコードの追加設定が必要になります。このパターンを実装するには、ロードバランサー (ALB または NLB) によって提供されるアベイラビリティーゾーン固有の DNS 名にエイリアスレコードを作成する必要があります。これを次の表に示します。

表 3: ロードバランサーのゾーン DNS 名に設定された Route 53 エイリアスレコード

ルーティングポリシー: 加重	ルーティングポリシー: 加重	ルーティングポリシー: 加重
名前: `www.example.com`	名前: `www.example.com`	名前: `www.example.com`
タイプ:A (エイリアス)	タイプ:A (エイリアス)	タイプ:A (エイリアス)
値: `us-east-1b.load-balancer-name.elb.us-east-1.amazonaws.com`	値: `us-east-1a.load-balancer-name.elb.us-east-1.amazonaws.com`	値: `us-east-1c.load-balancer-name.elb.us-east-1.amazonaws.com`
加重: `100`	加重: `100`	加重: `100`
ターゲットの正常性の評価: true	ターゲットの正常性の評価: `true`	ターゲットの正常性の評価: `true`

これらの DNS レコードごとに、Route 53 ARC ルーティング制御に関連付けられた Route 53 ヘルスチェックを設定します。アベイラビリティーゾーンの退避を開始するとき、ルーティング制御状態を `Off` に設定します。AWS では、アベイラビリティーゾーンの退避を開始するのに必要な依存関係を最小限に抑えるために、CLI または API を使用してこれを行うことをお勧めしています。ベストプラクティスとして、Route 53 ARC クラスターエンドポイントのローカルコピーを保存しておくと、退避が必要なときに ARC コントロールプレーンからそれらを取得する必要がなくなります。

この方法を使用する際のコストを最小限に抑えるために、1 つの Route 53 ARC クラスターとヘルスチェックを 1 つの AWS アカウント に作成し、組織でヘルスチェックを他の AWS アカウント と共有することができます。この方法を行う場合は、ルーティングの制御にアベイラビリティーゾーン名 (例えば `us-east-1a`) ではなく、アベイラビリティーゾーン ID (AZ-ID) (例えば `use1-az1`) を使

用する必要があります。これは、AWS は、物理的なアベイラビリティーゾーンを各 AWS アカウント のアベイラビリティーゾーン名にランダムにマッピングためです。AZ-ID を使用すると、同じ物理的位置を一貫して参照できます。アベイラビリティーゾーンの退避を開始すると、例えば use1-az2 に対して、各 AWS アカウント の Route 53 レコードセットは、それらが AZ-ID マッピングを使用して、各 NLB レコードに適切なヘルスチェックを設定するようにします。

例えば、Route 53 のヘルスチェックが use1-az2 に対する Route 53 の ARC ルーティング制御に ID 0385ed2d-d65c-4f63-a19b-2412a31ef431 で関連付けられているとします。このヘルスチェックを利用したい別の AWS アカウント で、us-east-1c が use1-az2 にマップされている場合、レコード us-east-1c.load-balancer-name.elb.us-east-1.amazonaws.com に use1-az2 のヘルスチェックを使用する必要があります。ヘルスチェック ID 0385ed2d-d65c-4f63-a19b-2412a31ef431 をそのリソースレコードセットで使用します。

セルフマネージド型 HTTP エンドポイントの使用

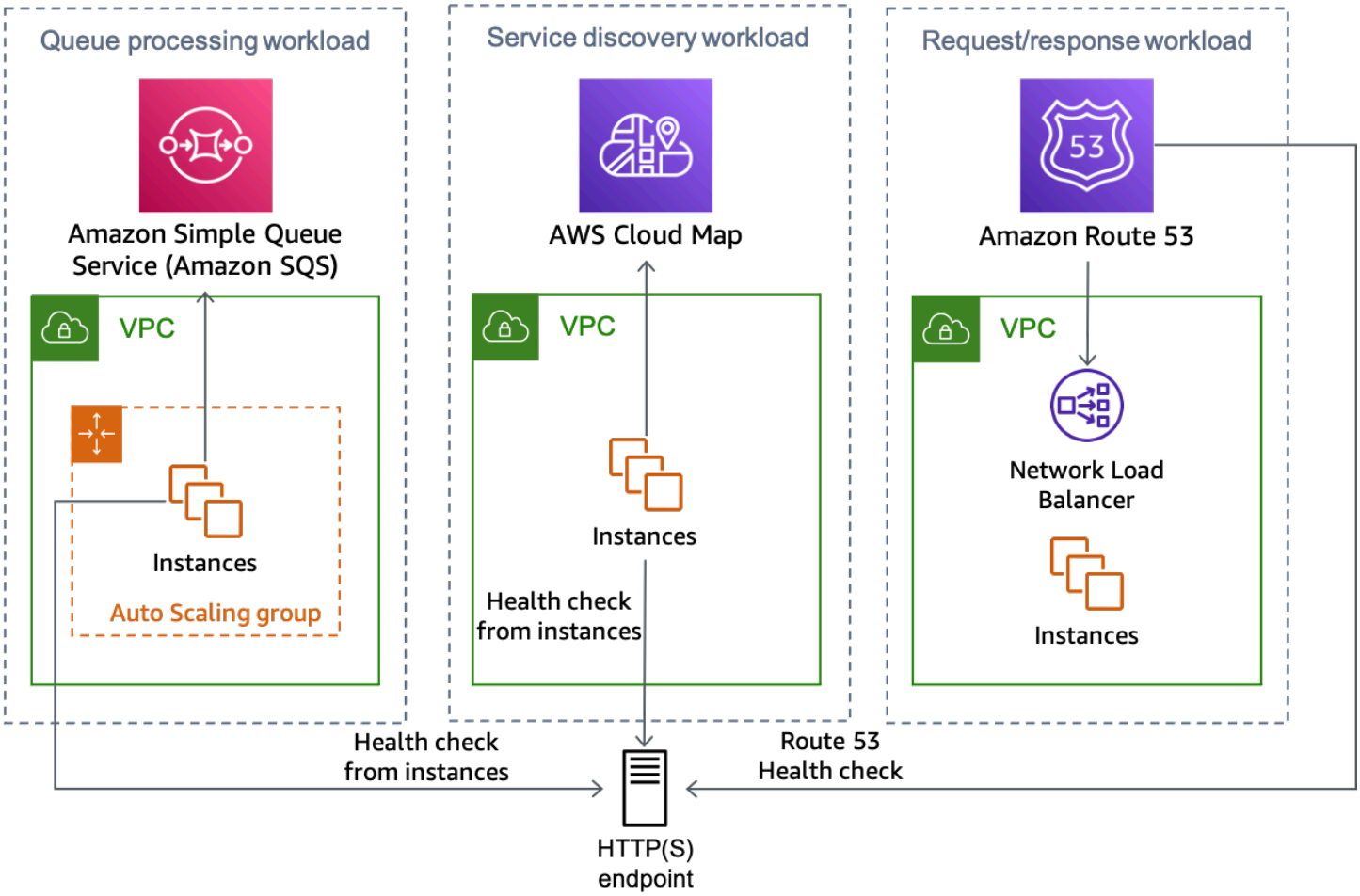
特定のアベイラビリティーゾーンのステータスを示す独自の HTTP エンドポイントを管理することで、このソリューションを実装することもできます。これにより、HTTP エンドポイントからのレスポンスに基づいて、アベイラビリティーゾーンに異常があるときを手動で指定できます。このソリューションは、Route 53 ARC を使用するよりもコストは低くなりますが、ゾーンシフトよりもコストがかかり、追加のインフラストラクチャを管理する必要があります。さまざまなシナリオに対してはるかに柔軟に対応できるという利点があります。

このパターンは NLB または ALB アーキテクチャと Route 53 ヘルスチェックで使用できます。また、ワーカーノードが独自のヘルスチェックを行うサービス検出システムやキュー処理システムなど、負荷分散されていないアーキテクチャでも使用できます。このようなシナリオでは、ホストはバックグラウンドスレッドを使用して、自分の AZ-ID を使用して HTTP エンドポイントに定期的にリクエストを送信し (これを見つける方法の詳細については 付録 A — アベイラビリティーゾーン ID の取得 を参照)、アベイラビリティーゾーンの状態に関するレスポンスを受け取ることができます。

アベイラビリティーゾーンに異常があると宣言された場合、対応方法には複数の選択肢があります。ELB や Route 53 などのソースからの外部ヘルスチェック、またはサービス検出アーキテクチャのカスタムヘルスチェックに失敗するように選択し、それらのサービスに対してホストに異常があるように見せることができます。また、リクエストを受け取った場合に即座にエラーで応答し、クライアントがバックオフして再試行できるようにすることもできます。イベント駆動型アーキテクチャでは、SQS メッセージを意図的にキューに返すなど、ノードでの作業の処理を意図的に失敗させることができます。中央のサービススケジュールが特定のホストで動作するワークルーターアーキテクチャでは、このパターンを使用することもできます。ルーターは、ワーカー、エンドポイント、また

はセルを選択する前に、アベイラビリティーゾーンのステータスを確認できます。AWS Cloud Map を使用するサービス検出アーキテクチャでは、AZ-ID などのフィルターをリクエストで指定することでエンドポイントを検出することができます。

次の図は、この方法を複数のタイプのワークロードにどのように使用できるかを示しています。



複数のワークロードのタイプすべてが HTTP エンドポイントソリューションを使用できる

HTTP エンドポイントアプローチを実装する方法は複数ありますが、次にそのうち 2 つについて概説します。

Amazon S3 の使用

このパターンは元々マルチリージョンのディザスタリカバリに関するこのブログ投稿で紹介されていました。アベイラビリティーゾーンの退避にも同じパターンを使用できます。

このシナリオでは、上記の Route 53 ARC シナリオおよび関連するヘルスチェックのように、ゾーン DNS レコードごとに Route 53 DNS リソースレコードセットを作成します。ただし、この実装で

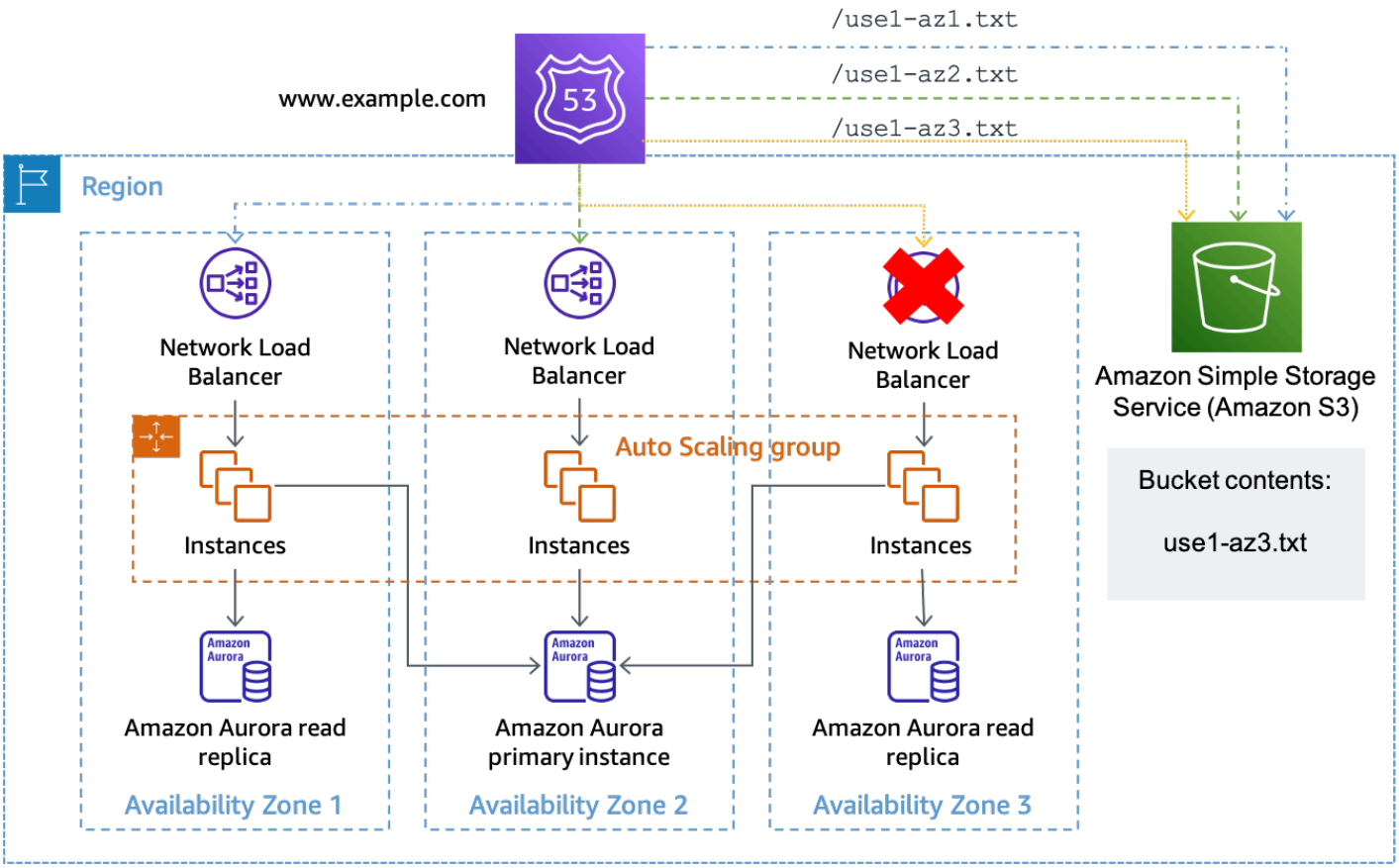
は、ヘルスチェックを Route 53 ARC ルーティング制御に関連付ける代わりに、HTTP エンドポイントを使用するように設定され、Amazon S3 の障害によって誤って退避がトリガーされるのを防ぐため、反転されています。ヘルスチェックは、オブジェクトが存在しないときは健全、オブジェクトが存在するときは異常があるとみなされます。このセットアップを次の表に示します。

表 4: アベイラビリティーゾーンごとの Route 53 ヘルスチェックを使用するための DNS レコード設定

ヘルスチェックタイプ: エンドポイントをモニタリングする プロトコル: HTTPS ID: dddd-4444 URL: https://*bucketname* .s3.us-east-1.amazonaws.com/use1-az1.txt	ヘルスチェックタイプ: エンドポイントをモニタリングする プロトコル: HTTPS ID: eeee-5555 URL: https://*bucketname* .s3.us-east-1.amazonaws.com/use1-az3.txt	ヘルスチェックタイプ: エンドポイントをモニタリングする プロトコル: HTTPS ID: ffff-6666 URL: https://*bucketname* .s3.us-east-1.amazonaws.com/use1-az2.txt	←	ヘルスチェック
↑	↑	↑		
ルーティングポリシー: 加重 名前: www.example.com タイプ:A (エイリアス)	ルーティングポリシー: 加重 名前: www.example.com タイプ:A (エイリアス)	ルーティングポリシー: 加重 名前: www.example.com タイプ:A (エイリアス)	←	NLB AZ 固有のエンドポイントを指定するトップレベルの均等に重み付けされたエイリアス A レコード

値: us-east-1b.load-balancer-name.elb.us-east-1.amazonaws.com	値: us-east-1a.load-balancer-name.elb.us-east-1.amazonaws.com	値: us-east-1c.load-balancer-name.elb.us-east-1.amazonaws.com
加重: 100	加重: 100	加重: 100
ターゲットの正常性の評価: true	ターゲットの正常性の評価: true	ターゲットの正常性の評価: true

アベイラビリティーゾーン us-east-1aをアベイラビリティーゾーンの退避を実行したいワークロードがあるアカウントの use1-az3 にマッピングすると仮定します。us-east-1a.load-balancer-name.elb.us-east-1.amazonaws.com に作成されたリソースレコードセットは、URL https://*bucket-name*.s3.us-east-1.amazonaws.com/use1-az3.txt をテストするヘルスチェックを関連付けます。use1-az3 に対してアベイラビリティーゾーンの退避を開始したい場合、use1-az3.txt という名前のファイルを CLI または API を使用してバケットにアップロードします。ファイルにコンテンツを含める必要はありませんが、Route 53 ヘルスチェックがファイルにアクセスできるように公開する必要があります。次の図は、この実装が use1-az3 の退避に使用されていることを示しています。



Route 53 ヘルスチェックのターゲットとして Amazon S3 を使用している

API Gateway と DynamoDB を使用する

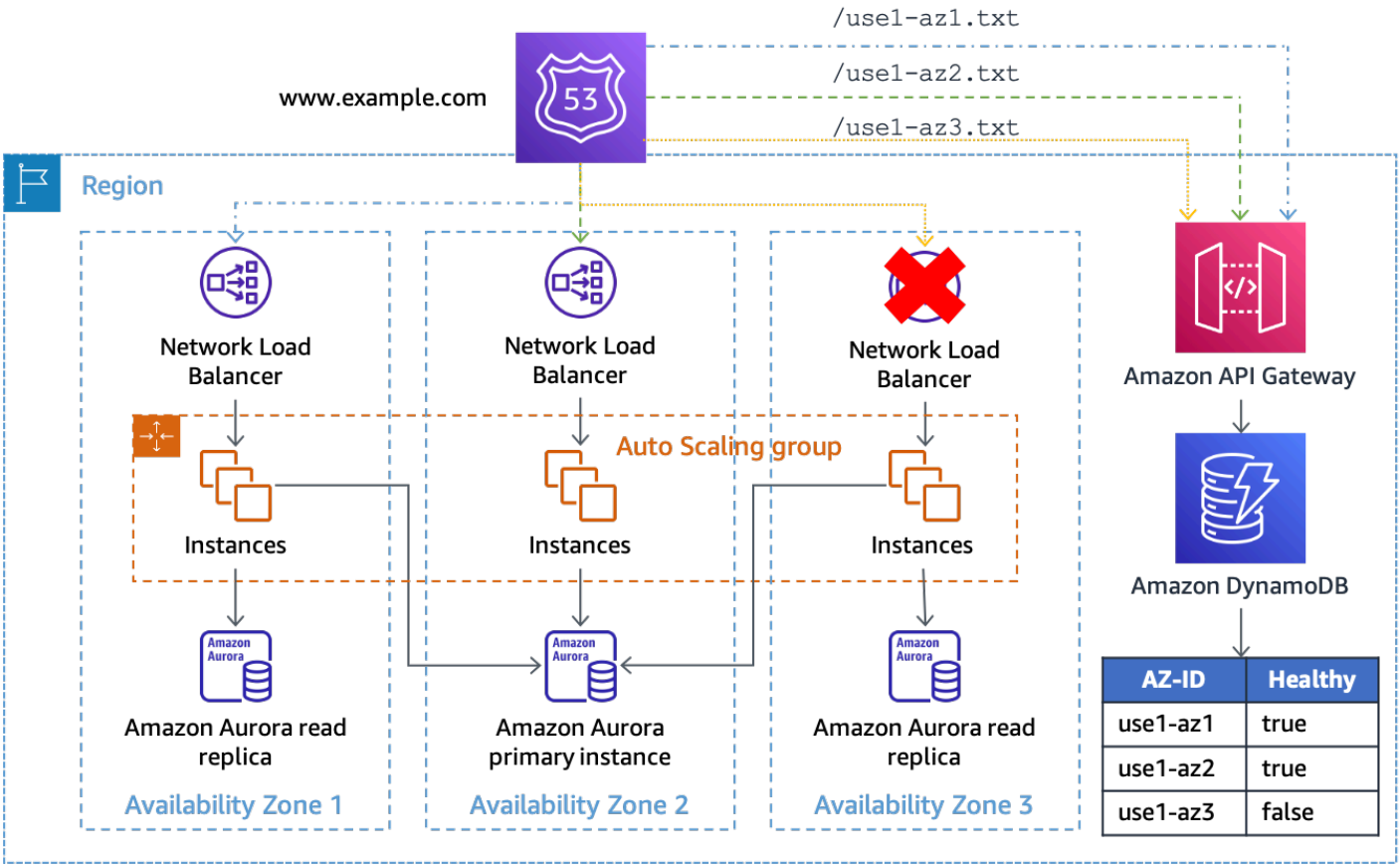
このパターンの 2 番目の実装では、Amazon API Gateway REST API を使用します。この API は、使用中の各アベイラビリティーゾーンのステータスを保存する Amazon DynamoDB へのサービス統合で設定します。この実装は Amazon S3 アプローチよりも柔軟ですが、より多くのインフラストラクチャを構築、運用、監視する必要があります。また、Route 53 のヘルスチェックと個々のホストによるヘルスチェックの両方で使用できます。

このソリューションを NLB または ALB アーキテクチャで使用している場合は、上記の Amazon S3 の例と同じ方法で DNS レコードを設定してください。ただし、API Gateway エンドポイントを使用するようにヘルスチェックパスを変更し、URL パスに `AZ-ID` を提供します。例えば、API Gateway がカスタムドメイン `az-status.example.com` で設定されているとします。`use1-az1` の完全なリクエストは `https://az-status.example.com/status/use1-az1` のようになります。アベイラビリティーゾーンの退避を開始する場合、CLI または API を使用して DynamoDB アイテムを作成または更新できます。このアイテムは `AZ-ID` をプライマリキーとして使用し、API Gateway がど

のように応答するかを示す Healthy と呼ばれる Boolean 属性があります。以下は、この判断を行うために API Gateway 設定で使用されるコード例です。

```
#set($inputRoot = $input.path('$'))
#if ($inputRoot.Item.Healthy['BOOL'] == (false))
    #set($context.responseOverride.status = 500)
#end
```

属性が true の場合 (または存在しない場合)、API Gateway はヘルスチェックに HTTP 200 で応答し、false の場合は HTTP 500 で応答します。この実装を次の図に示します。



Route 53 のヘルスチェックのターゲットとして API Gateway と DynamoDB を使用している

このソリューションでは、DynamoDB の前で API Gateway を使用する必要があります。これにより、エンドポイントをパブリックにアクセス可能にしたり、リクエスト URL を DynamoDB の GetItem リクエストに操作できるようになります。このソリューションでは、リクエストに追加データを含めたい場合にも柔軟に対応できます。例えば、アプリケーションごとなど、より詳細なステータスを作成したい場合は、パスまたはクエリ文字列にアプリケーション ID を指定するようにヘルスチェック URL を設定できます。これは DynamoDB アイテムとも一致します。

アベイラビリティーゾーンのステータスエンドポイントは一元的にデプロイできるため、AWS アカウント 全体にわたる複数のヘルスチェックリソースのすべてが、アベイラビリティーゾーンの状態の同じ一貫したビューを使用できるようになり (API Gateway REST API と DynamoDB テーブルが負荷を処理できるようにスケールされます)、Route 53 のヘルスチェックを共有する必要性がなくなります。

このソリューションは、Amazon DynamoDB グローバルテーブルと各リージョンで API Gateway REST API のコピーを使用して、複数の AWS リージョン にまたがってスケールすることもできます。これにより、このソリューションが単一のリージョンに依存することがなくなり、可用性が向上します。ソリューションを 3 つまたは 5 つのリージョンにデプロイし、各リージョンにアベイラビリティーゾーンの状態を問い合わせて、大多数のエンドポイントの結果を使用してクォーラムを確保できます。これにより、エンドポイントの応答を妨げる可能性のある障害が軽減されるだけでなく、最終的にグローバルテーブル全体で更新を一貫してレプリケートできます。例えば、5 つのリージョンを使用していて、3 つのエンドポイントから 1 つのアベイラビリティーゾーンが異常であると報告され、1 つのエンドポイントから同じアベイラビリティーゾーンが正常であると報告され、1 つのエンドポイントから応答がない場合は、このアベイラビリティーゾーンを異常として扱うことを選択します。また、m of n 計算を使用して Route 53 の計算済みヘルスチェックを作成し、このロジックを実行してアベイラビリティーゾーンのヘルスを判断することもできます。

個別のホストが AZ の状態を判断するメカニズムとして使用するソリューションを構築する場合は、代替方法として、ヘルスチェック用のプルメカニズムを提供する代わりに、プッシュ通知を使用できます。これを行う 1 つの方法は、コンシューマーがサブスクライブしている SNS トピックを使用することです。サーキットブレーカーをトリガーする場合は、どのアベイラビリティーゾーンに障害があるかを示すメッセージを SNS トピックに発行します。このアプローチでは、前者とのトレードオフが生じます。これにより、API Gateway インフラストラクチャを作成して運用したり、容量管理を実行したりする必要がなくなります。また、アベイラビリティーゾーンの状態をより迅速に収束できる可能性もあります。ただし、アドホッククエリを実行する機能は削除されます。また、各エンドポイントに通知を確実に届けるために SNS メッセージ配信の再試行ポリシーに依存します。また、各ワークロードやサービスが SNS 通知を受信してアクションを実行する方法を構築する必要もあります。

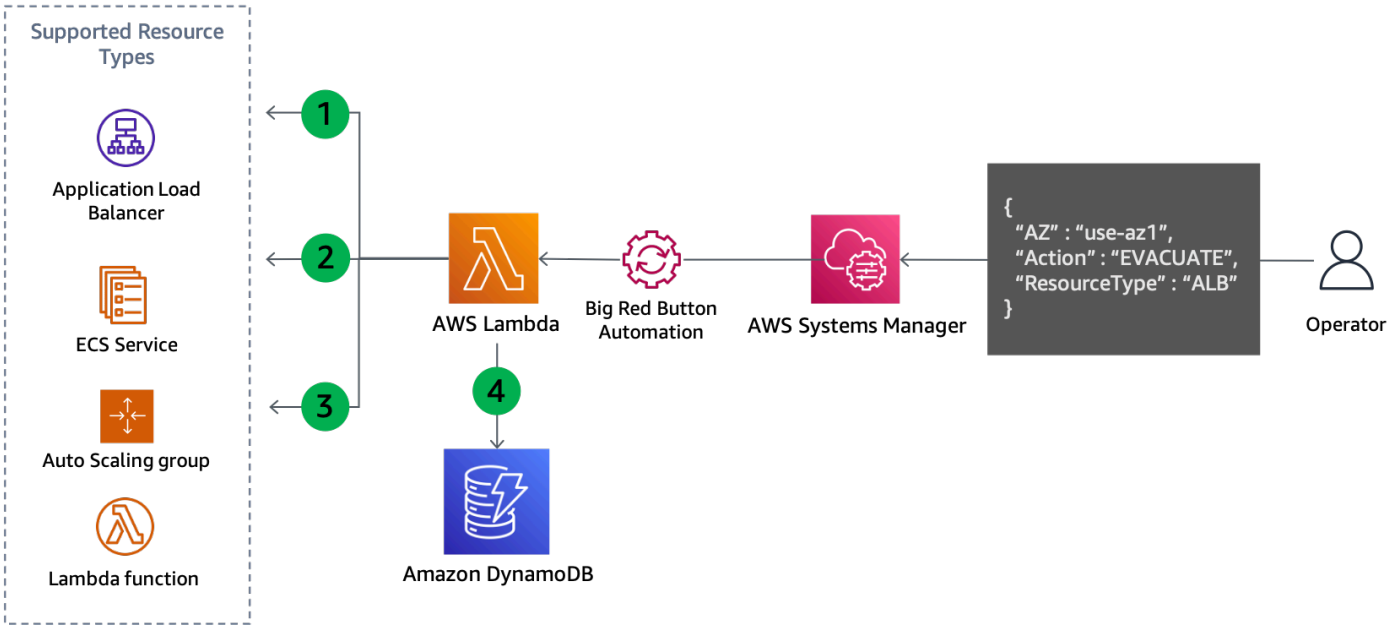
例えば、新しい EC2 インスタンスやコンテナを起動するたびに、ブートストラップ中に HTTP エンドポイントを使用してトピックをサブスクライブする必要があります。次に、各インスタンスは、このエンドポイントに配信される通知をリッスンするソフトウェアを実装する必要があります。さらに、インスタンスは、イベントの影響を受けた場合に、プッシュ通知を受け取らずに作業を続ける可能性があります。一方、プル通知を使用すると、インスタンスはプルリクエストが失敗したかどうかを知り、それに応じてどのアクションを実行するかを選択できます。

プッシュ通知を送信する 2 番目の方法は、存続期間の長い WebSocket 接続を使用することです。Amazon API Gateway を使用すると、コンシューマーが接続できる WebSocket API を指定でき、バックエンドから送信されたメッセージを受信できます。WebSocket を使用すると、インスタンスは定期的なプルを実行して接続が正常であることを確認するとともに、低レイテンシーのプッシュ通知を受け取ることができます。

コントロールプレーン制御による退避

最初のパターンでは、データプレーン操作を使用して、影響を受けているアベイラビリティーゾーンでの作業の実施を防止し、イベントの影響を軽減します。ただし、ロードバランサーを使用しないアーキテクチャや、ホストごとのヘルスチェックを設定できないアーキテクチャを使用している場合があります。あるいは、自動スケーリングや通常の作業スケジュール設定によって、影響を受けているアベイラビリティーゾーンに新しいキャパシティがデプロイされないようにしたい場合もあります。

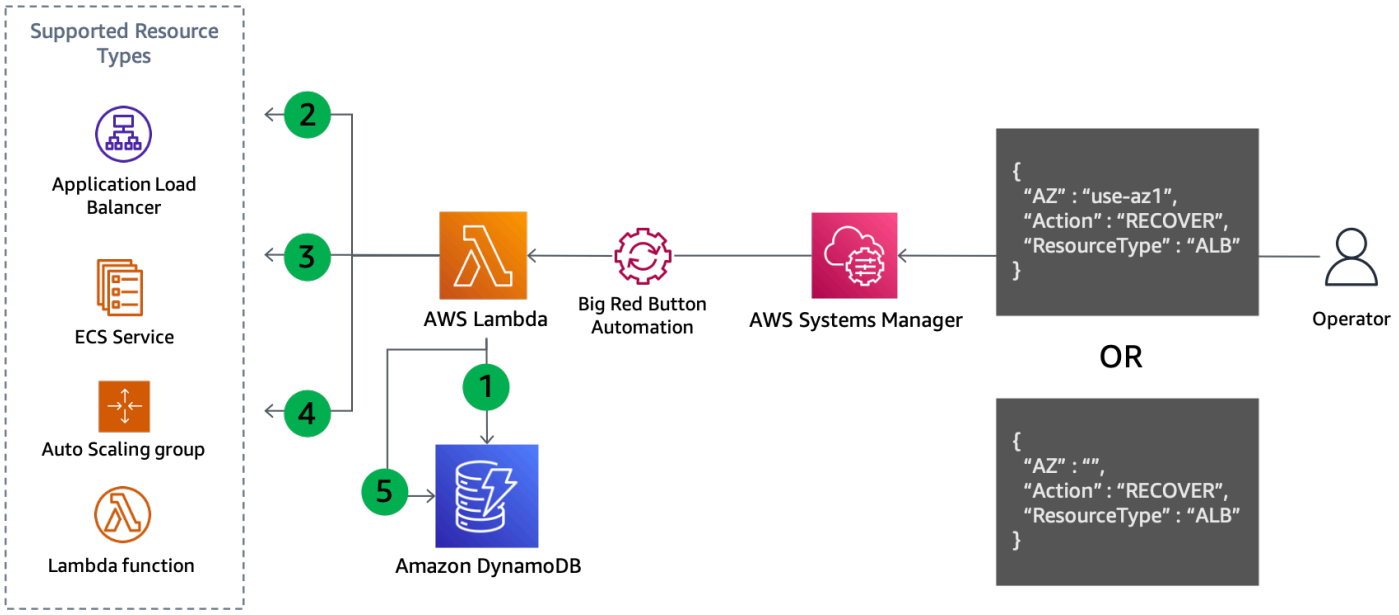
両方の状況に対処するには、リソースの設定を更新するためのコントロールプレーンのアクションが必要です。このパターンは、EC2 Auto Scaling、Amazon ECS、Lambda など、ネットワーク設定を更新できるすべてのサービスで機能します。サービスごとにコードを記述する必要がありますが、ビジネスロジックは標準的なパターンに従います。必要な依存関係を最小限に抑えるために、コードはイベントに応答するオペレーターがローカルで実行する必要があります。スクリプトロジックの基本的なフローを次の図に示します。



アベイラビリティーゾーンから退避するためのコントロールプレーンの更新

1. このスクリプトは、Auto Scaling グループ、ECS サービス、Lambda 関数など、指定されたタイプのすべてのリソースを一覧表示し、リソース情報からサブネットを取得します。サポートされるリソースは、スクリプトが何をサポートするように設定されているかによって異なります。
2. 各サブネットのアベイラビリティーゾーン名を、入力パラメータとして提供され、マップされたアベイラビリティーゾーン ID と比較することで、削除すべきサブネットを決定します。
3. 特定されたサブネットを削除するように、リソースのネットワーク設定を更新します。
4. 更新の詳細は DynamoDB テーブルに記録されます。アベイラビリティーゾーン ID はパーティションキーとして保存され、リソースの ARN または名前はソートキーとして保存されます。削除されたサブネットは、文字列配列として保存されます。最後に、リソースタイプも保存され、グローバルセカンダリインデックス (GSI) のハッシュキーとして使用されます。

ステップ 4 では、実行した更新が記録されるため、このアプローチは、次の図に示すように、復旧の準備ができたら簡単に元に戻すのにも役立ちます。



アベイラビリティーゾーンからの退避から復旧するためのコントロールプレーンの更新

回復手順:

1. GSI にクエリを実行して、指定したアベイラビリティーゾーン (指定していない場合はすべてのアベイラビリティーゾーン) 内の指定したタイプのリソースごとに削除済みのサブネットを取得します。
2. DynamoDB クエリで見つかった各リソースを記述して、現在のネットワーク設定を取得します。

3. 現在のネットワーク設定のサブネットと DynamoDB のクエリから取得したサブネットを組み合わせます。
4. リソースのネットワーク設定を新しいサブネットセットで更新します。
5. 更新が正常に完了したら、DynamoDB テーブルからレコードを削除します。

この一般化されたパターンにより、影響を受けたアベイラビリティーゾーンへの作業のルーティングと、そこでの新しい容量のデプロイの両方を阻止します。さまざまなサービスで、これを実現する方法の例を以下に示します。

- Lambda — 関数の VPC 設定を更新して、指定したアベイラビリティーゾーンのサブネットを削除します。
- Auto Scaling グループ — ASG 設定からサブネットを削除します。これにより、残りのアベイラビリティーゾーンでそのキャパシティを置き換えます。
- Amazon ECS — ECS サービスの VPC 設定を更新してサブネットを削除します。
- Amazon EKS — 影響を受けたアベイラビリティーゾーンのノードにテイントを適用して既存のポッドを削除し、そこに追加のポッドがスケジュールされないようにします。

サービスごとに、設定の更新に対する反応は異なります。例えば、Amazon ECS は、更新後にサービスのデプロイ設定に従い、新しいタスクのローリングデプロイやブルー/グリーンデプロイをトリガーします。

これらの更新により、一部のワークロードでは、作業が正常なアベイラビリティーゾーンにすぐに移行する可能性があります。障害に対して静的に安定するように設定されている一方 (影響を受けているアベイラビリティーゾーンの作業を処理するのに十分なキャパシティを残りのアベイラビリティーゾーンに事前にプロビジョニングしておく)、影響を受けているアベイラビリティーゾーンのキャパシティを徐々に段階的に廃止したい場合もあります。

> クロスゾーン負荷分散が無効になっているロードバランサーのターゲットグループである Auto Scaling グループのネットワーク設定を更新する予定がある場合、このガイダンスに従ってください。
>
> Auto Scaling はアベイラビリティーゾーンの再分散ロジックを使ってこの変更に対応します。希望するキャパシティに合わせて他のアベイラビリティーゾーンでインスタンスを起動し、削除したアベイラビリティーゾーンのインスタンスを終了します。ただし、ロードバランサーは、インスタンスが終了している間も、ASG から削除したアベイラビリティーゾーンを含め、各アベイラビリティーゾーンにトラフィックを均等に分散し続けます。これによ

り、すべてのインスタンスが正常に終了するまで、そのアベイラビリティーゾーンの残りのキャパシティが使い果たされる可能性があります。これは、クロスゾーン負荷分散が無効になっている場合のアベイラビリティーゾーンの不均衡に関する「アベイラビリティーゾーンの独立性」で説明されているのと同じ問題です。これを防ぐため、次のいずれかの方法を行うことができます。

- トラフィックが残りのアベイラビリティーゾーンにのみ分散されるように、必ずアベイラビリティーゾーンの退避を最初に実行します。
- そのアベイラビリティーゾーンに必要な最小ターゲット数と一致するように、DNS フェイルオーバー時の正常なターゲットの最小数を指定します。

これにより、インスタンスが終了を開始した後で削除したアベイラビリティーゾーンにトラフィックが送信されないようになります。

まとめ

次の表は、説明した退避パターンの長所と短所をまとめたものです。

表5：退避パターンの長所と短所

アプローチ	長所	短所
データプレーン制御の退避	データプレーンアクションにのみ依存する 影響を受けたアベイラビリティーゾーンでの作業の実行をすばやく阻止する アベイラビリティーゾーンのヘルスを一元的に確認できる柔軟なアプローチ	影響を受けたアベイラビリティーゾーンへの容量のデプロイを阻止しない すべてのワークロードタイプがこの方法を簡単に使用できるわけではありません。
コントロールプレーン制御による退避	影響を受けたアベイラビリティーゾーンへの新しい容量のデプロイを阻止する	各サービスのコントロールプレーンに依存する

アプローチ	長所	短所
	影響を受けたアベイラビリティーゾーンから既存の容量を削除する	サービスごとにコードを作成する必要がある サービスごとに完了する必要がある 更新中にキャパシティを使い果たさないように注意する必要がある

アベイラビリティーゾーンの退避計画の一部として、両方の方法を使用することになるでしょう。データプレーンで制御された退避アクションのうち、成功する可能性が高いものから始めて、影響を受けているアベイラビリティーゾーンでの処理作業を迅速に停止します。その後、初期の影響が緩和されたら、必要と思われる場合は、コントロールプレーン制御による退避アクションでフォローアップを行います。

結論

このホワイトペーパーでは、グレー障害とその発生方法の概要、発生時にこうした種類のイベントを軽減するためにオブザーバビリティおよび退避ツールを構築する必要がある理由について概説しました。次のセクションでは、マルチ AZ のオブザーバビリティと、1 つのアベイラビリティーゾーンの影響を検出するために実装できる 3 つのアプローチについて確認しました。最後のセクションで、このホワイトペーパーでは、アベイラビリティーゾーンの退避を行うための 2 つの一般的な方法について紹介しました。最初の方法では、データプレーンアクションを使用して影響を受けるアベイラビリティーゾーンに作業がルーティングされないようにします。一方、2 つ目の方法では、コントロールプレーンアクションを使用して、影響を受けるアベイラビリティーゾーンにキャパシティがプロビジョニングされないようにします。これら 2 つの方法を組み合わせると、アベイラビリティーゾーンの退避が意図する 2 つの成果が得られます。

このホワイトペーパーで説明している復旧パターンは、より大きなモニタリングおよび障害復旧ソリューションの一部となる可能性があります。この方法でシングルアベイラビリティーゾーンのグレー障害に対処するには、障害を検出するために必要な計測器と障害に対応するツールを構築するためのエンジニアリング作業が必要です。ただし、多くのワークロードの場合、このアプローチは、マルチリージョンアーキテクチャを構築するよりもシンプルでコストもかからない代替手段となる可能性があります。さらに、マルチリージョン DR と比較した場合、RPO と RTO を小さくする (ワークロードの可用性を高める) のに役立ちます。

付録 A — アベイラビリティーゾーン ID の取得

AWS .NET SDK (および JavaScript などの他の SDK) を使用しているか、Amazon EC2 インスタンスでシステム (Amazon ECS や EKS を含む) を実行している場合は、アベイラビリティーゾーン ID を直接取得できます。

- AWS .NET SDK

```
Amazon.Util.EC2InstanceMetadata.GetData(“/placement/availability-zone-id”)
```

- EC インスタンスメタデータサービス

```
curl http://169.254.169.254/latest/meta-data/placement/availability-zone-id
```

Lambda や Fargate などの他のプラットフォームでは、アベイラビリティーゾーン名を取得して、アベイラビリティーゾーン ID へのマッピングを見つける必要があります。アベイラビリティーゾーン名を使用すると、次のようなアベイラビリティーゾーン ID を見つけることができます。

```
aws ec2 describe-availability-zones --zone-names $AZ --output json
    --query ‘AvailabilityZones[0].ZoneId’
```

上の例で使用するアベイラビリティーゾーン名を検索する以下の例は、AWS CLI とパッケージ jq を使用して bash で記述しています。これらは、ワークロードで使用するプログラミング言語に変換する必要があります。

- Amazon ECS — インスタンスメタデータサービス (IMDS) がホストによってブロックされている場合は、代わりにコンテナメタデータファイルを使用できます。

```
AZ=$(cat $ECS_CONTAINER_METADATA_FILE | jq --raw-output
    .AvailabilityZone)
```

- Fargate (プラットフォームバージョン 1.4 以降)

```
AZ=$(curl $ECS_CONTAINER_METADATA_URI_V4/task | jq --raw-output
    .AvailabilityZone)
```

- Lambda — アベイラビリティーゾーンは関数に直接公開されません。これを見つけるには、いくつかのステップを完了する必要があります。そのために、リクエスタの IP アドレスを返すプライベート API ゲートウェイ REST エンドポイントを構築する必要があります。これにより、関数が使用している Elastic Network Interface に割り当てられたプライベート IP が識別されます。
  - Lambda `GetFunction` API を呼び出して、関数の VPC ID を見つけます。
  - API Gateway サービスを呼び出して、関数の IP を取得します。
  - IP と VPC ID を使用して、関連するネットワークインターフェイスを見つけ、アベイラビリティーゾーンを抽出します。

```
VPC_ID=$(aws lambda get-function --function-name $ AWS_LAMBDA_FUNCTION_NAME --
region $AWS_REGION --output json --query ‘Configuration.VpcConfig.VpcId’)

MY_IP=$(curl http://whats-my-private-ip.internal)

AZ=$(aws ec2 describe-network-interfaces --filters Name=private-ip-address,Values=
$MY_IP Name=vpc-id,Values=$VPC_ID --region $AWS_REGION --output json –query
 ‘NetworkInterfaces[0].AvailabilityZone’)
```

付録 B — カイ二乗計算の例

以下は、エラーメトリクスを収集し、データに対してカイ二乗検定を実行する例です。このコードは本番環境に対応しておらず、必要なエラー処理は行いませんが、ロジックの仕組みについての概念実証は提供されています。この例はニーズに合わせて更新する必要があります。

まず、Amazon EventBridge のスケジュールされたイベントによって 1 分おきに Lambda 関数が呼び出されます。イベントのコンテンツは、次のデータで構成されます。

```
{
  "timestamp": "2023-03-15T15:26:37.527Z",
  "namespace": "multi-az/frontend",
  "metricName": "5xx",
  "dimensions": [
    { "Name": "Region", "Value": "us-east-1" },
    { "Name": "Controller", "Value": "Home" },
    { "Name": "Action", "Value": "Index" }
  ],
  "period": 60,
  "stat": "Sum",
  "unit": "Count",
  "chiSquareMetricName": "multi-az/chi-squared",
  "azs": [ "use1-az2", "use1-az4", "use1-az6" ]
}
```

このデータを使用して、適切な CloudWatch メトリクス (名前空間、メトリックス名、ディメンションなど) の取得に必要な共通データを指定し、各アベイラビリティーゾーンのカイ二乗結果を公開します。Python 3.9 を使用した場合、Lambda 関数のコードは次のようになります。大まかな流れとしては、指定した CloudWatch メトリクスを過去 1 分間収集し、そのデータに対してカイ二乗検定を実行し、指定されたアベイラビリティーゾーンごとに検定結果に関する CloudWatch メトリクスをパブリッシュします。

```
import os
import boto3
import datetime
import copy
import json
from datetime import timedelta
from scipy.stats import chisquare
```

```
from aws_embedded_metrics import metric_scope

cw_client = boto3.client("cloudwatch", os.environ.get("AWS_REGION", "us-east-1"))

@metric_scope
def handler(event, context, metrics):
    metrics.set_property("Event", json.loads(json.dumps(event, default = str)))
    time = datetime.datetime.strptime(event["timestamp"], "%Y-%m-%dT%H:%M:%S.%fZ")

    # Round down to the previous minute
    end: datetime = roundTime(time)

    # Subtract a minute for the start
    start: datetime = end - timedelta(minutes = 1)

    # Get all the metrics that match the query
    results = get_all_metrics(event, start, end, metrics)
    metrics.set_property("MetricCounts", results)

    # Calculate the chi squared result
    chi_sq_result = chisquare(list(results.values()))
    expected = sum(list(results.values())) / len(results.values())
    metrics.set_property("ChiSquaredResult", chi_sq_result)

    # Put the chi square metrics into CloudWatch
    put_all_metrics(event, results, chi_sq_result[1], expected, start, metrics)

def get_all_metrics(detail: dict, start: datetime, end: datetime, metrics):
    """
    Gets all of the error metrics for each AZ specified
    """
    metric_query = {
        "MetricDataQueries": [
        ],
        "StartTime": start,
        "EndTime": end
    }

    for az in detail["azs"]:

        dim = copy.deepcopy(detail["dimensions"])
        dim.append({"Name": "AZ-ID", "Value": az})

        query = {
```

```
            "Id": az.replace("-", "_"),
            "MetricStat": {
                "Metric": {
                    "Namespace": detail["namespace"],
                    "MetricName": detail["metricName"],
                    "Dimensions": dim
                },
                "Period": int(detail["period"]),
                "Stat": detail["stat"],
                "Unit": detail["unit"]
            },
            "Label": az,
            "ReturnData": True
        }

        metric_query["MetricDataQueries"].append(query)

    metrics.set_property("GetMetricRequest", json.loads(json.dumps(metric_query,
default=str)))
    next_token: str = None
    results = {}

    while True:
        if next_token is not None:
            metric_query["NextToken"] = next_token

        data = cw_client.get_metric_data(**metric_query)

        if next_token is not None:
            metrics.set_property("GetMetricResult::" + next_token,
json.loads(json.dumps(data, default = str)))
        else:
            metrics.set_property("GetMetricResult", json.loads(json.dumps(data, default
= str)))

        for item in data["MetricDataResults"]:
            key = item["Id"].replace("_", "-")
            if key not in results:
              results[key] = 0

            results[key] += sum(item["Values"])

        if "NextToken" in data:
            next_token = data["NextToken"]
```

```
        if next_token is None:
            break

    return results

def put_all_metrics(detail: dict, results: dict, chi_sq_value: float, expected: float,
 timestamp: datetime, metrics):
    """
    Adds the chi squared metric for all AZs to CloudWatch
    """
    farthest_from_expected = None
    if len(results) > 0:
        keys = list(results.keys())
        farthest_from_expected = keys[0]

        for key in keys:
            if abs(results[key] - expected) > abs(results[farthest_from_expected] -
 expected):
               farthest_from_expected = key

    metric_query = {
        "Namespace": detail["namespace"],
        "MetricData": []
    }

    for az in detail["azs"]:
        dim = copy.deepcopy(detail["dimensions"])
        dim.append({"Name": "AZ-ID", "Value": az})

        query = {
            "MetricName": detail["chiSquareMetricName"],
            "Dimensions": dim,
            "Timestamp": timestamp,
        }

        if chi_sq_value <= 0.05 and az == farthest_from_expected:
            query["Value"] = 1
        else:
            query["Value"] = 0

        metric_query["MetricData"].append(query)
```

```
    metrics.set_property("PutMetricRequest", json.loads(json.dumps(metric_query,
 default = str)))

    cw_client.put_metric_data(**metric_query)

def roundTime(dt=None, roundTo=60):
   """Round a datetime object to any time lapse in seconds
   dt : datetime.datetime object, default now.
   roundTo : Closest number of seconds to round to, default 1 minute.
   """
   if dt == None : dt = datetime.datetime.now()
   seconds = (dt.replace(tzinfo=None) - dt.min).seconds
   rounding = (seconds+roundTo/2) // roundTo * roundTo
   return dt + datetime.timedelta(0,rounding-seconds,-dt.microsecond)
```

その後、AZ ごとにアラームを作成できます。以下は use1-az2 の例で、最大値が 1 に等しい 3 つの 1 分間のデータポイントを連続して警告します (1 は、カイ二乗検定で エラー率の統計的に有意な偏りが確認された場合に公開されるメトリクスです)。

```
{
    "Type": "AWS::CloudWatch::Alarm",
    "Properties": {
        "AlarmName": "use1-az2-chi-squared",
        "ActionsEnabled": true,
        "OKActions": [],
        "AlarmActions": [],
        "InsufficientDataActions": [],
        "MetricName": "multi-az/chi-squared",
        "Namespace": "multi-az/frontend",
        "Statistic": "Maximum",
        "Dimensions": [
            {
                "Name": "AZ-ID",
                "Value": "use1-az2"
            },
            {
                "Name": "Action",
                "Value": "Index"
            },
            {
                "Name": "Region",
```

```
                "Value": "us-east-1"
            },
            {
                "Name": "Controller",
                "Value": "Home"
            }
        ],
        "Period": 60,
        "EvaluationPeriods": 3,
        "DatapointsToAlarm": 3,
        "Threshold": 1,
        "ComparisonOperator": "GreaterThanOrEqualToThreshold",
        "TreatMissingData": "missing"
    }
}
```

また、m-of-n アラームを作成して、これら 2 つのアラームを複合アラームと組み合わせることもできます。また、各アベイラビリティーゾーンにあるコントローラー/アクションの組み合わせまたはマイクロサービスごとに同じアラームを作成する必要があります。最後に、外れ値検出による障害検出 に示すように、コントローラーとアクションの組み合わせごとに、カイ二乗複合アラームをアベイラビリティーゾーン固有のアラームに追加できます。

寄稿者

このドキュメントの寄稿者は次のとおりです。

- Amazon Web Services、Principal Solutions Architect、Michael Haken

ドキュメントの改訂

このホワイトペーパーの更新に関する通知を受け取るには、RSS フィードにサブスクライブしてください。

変更	説明	日付
ホワイトペーパーの更新	オブザーバビリティに関するガイダンスを追加し、新しいゾーンシフト機能を使用するように更新しました。	2023 年 7 月 11 日
初版発行	ホワイトペーパーの初回発行。	2022 年 3 月 2 日

> **Note**
>
> RSS の更新を購読するには、使用しているブラウザで RSS プラグインを有効にする必要があります。

注意

お客様は、本書に記載されている情報を独自に評価する責任を負うものとします。本書は、(a) 情報提供のみを目的とし、(b) AWS の現行製品と慣行について説明しており、これらは予告なしに変更されることがあり、(c) AWS およびその関連会社、サプライヤー、またはライセンサーからの契約上の義務や保証をもたらすものではありません。AWS の製品やサービスは、明示または黙示を問わず、一切の保証、表明、条件なしに「現状のまま」提供されます。お客様に対する AWS の責任は AWS 契約によって規定されます。本書は、AWS とお客様との間で締結されるいかなる契約の一部でもなく、その内容を修正するものでもありません。

© 2023 Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS 用語集

AWS の最新の用語については、「AWS の用語集リファレンス」の「AWS 用語集」を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。