AWS Well-Architected Framework

信頼性の柱

信頼性の柱: AWS Well-Architected Framework

Table of Contents

信頼性の柱 - AWS Well-Architected フレームワーク

公開日: 2024 年 6 月 27 日 (改訂履歴)

このペーパーが信頼性の柱として焦点を当てるのは AWS Well-Architected Framework.お客様がアマゾン ウェブ サービス (AWS) 環境の設計、配信、メンテナンスを行う際にベストプラクティスを適用するためのガイダンスを提供します。

はじめに

それらの AWS Well-Architected Framework は、AWS でワークロードを構築する際に行う決定のメリットとデメリットを理解するのに役立ちます。このフレームワークを使用すれば、信頼性、安全性、効率性、コスト効率に優れ、持続可能なワークロードを、クラウド内で設計および運用するためのアーキテクチャ上の最新のベストプラクティスを学ぶことができます。アーキテクチャをベストプラクティスに照らして評価し、改善すべき分野を特定する一貫した方法を提供します。AWS では、Well-Architected ワークロードを備えることによって、ビジネスが成功する可能性が大幅に高まると確信しています。

AWS Well-Architected フレームワークは 6 つの柱に基づきます。

- 運用上の優秀性
- セキュリティ
- 信頼性
- パフォーマンス効率
- コスト最適化
- サステナビリティ

このホワイトペーパーでは、信頼性の柱をお客様のソリューションに適用する方法について説明します。従来のオンプレミス環境では、単一障害点、自動化の欠如、伸縮性の欠如が原因で、信頼性の実現が困難な場合があります。このホワイトペーパーで解説するプラクティスを採用すれば、強固な基盤、一貫した変更管理、実証済みの障害復旧プロセスを持つ復元力のあるアーキテクチャを構築できます。

このホワイトペーパーの対象者は、最高技術責任者 (CTO)、アーキテクト、デベロッパー、オペレーションチームメンバーなどの技術担当者です。本稿を読むことで、信頼性の高いクラウドアーキテクチャを設計するための AWS のベストプラクティスと戦略を理解することができます。本ホワイ

トペーパーには、さらに詳しい実装情報、アーキテクチャパターン、その他リソースの参考資料が含まれています。

信頼性

信頼性の柱には、意図した機能を期待どおりに正しく一貫して実行するワークロードの能力が含まれます。これには、ワークロードのライフサイクル全体を通じてワークロードを運用およびテストする能力が含まれます。このホワイトペーパーでは、AWS で信頼性の高いワークロードを実装するための詳しいベストプラクティスガイダンスを提供します。

トピック

回復性に関する責任共有モデル

回復性については、AWS とお客様で責任を共有します。ディザスタリカバリ (DR) と可用性が回復性の一環として、この責任共有モデルでどのように運用されるのかを理解することが重要です。

AWS の責任 - クラウドの回復性

AWS は、AWS クラウド で提供されるすべてのサービスを実行するインフラストラクチャの回復性について責任を負います。このインフラストラクチャは、AWS クラウド サービスを実行するハードウェア、ソフトウェア、ネットワーク、設備で構成されます。AWS は、このような AWS クラウド サービスを利用可能にするうえで商業的に合理的な取り組みを行い、サービスの可用性が AWS サービスレベルアグリーメント (SLA) を満たすか、それ以上を提供することを確認します。

AWS グローバルクラウドインフラストラクチャは、お客様が回復力の高いワークロードアーキテクチャを構築できるように設計されています。各 AWS リージョン は完全に分離されており、物理的に分離されたインフラストラクチャのパーティションである複数のアベイラビリティーゾーンで構成されています。アベイラビリティーゾーンは、ワークロードの回復力に影響を及ぼす可能性のある障害を分離し、リージョン内のその他のゾーンへの影響を回避します。ただし同時に、AWS リージョン 内のすべてのゾーンは、高帯域幅、低レイテンシーのネットワークで相互接続されています。ゾーン間をつなぐのは、高スループット、低レイテンシーのネットワークを提供する、完全な冗長性を備えた専用メトロファイバーです。ゾーン間のすべてのトラフィックは暗号化されています。ゾーン間の同期レプリケーションを実行するうえで十分なネットワークパフォーマンスが提供されます。

アプリケーションを AZ 間でパーティショニングすると、企業は、停電、落雷、竜巻、台風などの問題から、よりよく隔離され保護されます。

お客様の責任 - クラウド内の回復性

お客様の責任は、選択した AWS クラウド サービスにより異なります。選択したサービスにより、お客様が回復性についての責任の一環として実行する必要がある設定作業の量が決まります。例えば、Amazon Elastic Compute Cloud (Amazon EC2) のようなサービスでは、お客様は必要となる回復性の設定と管理をすべて実行する必要があります。Amazon EC2 インスタンスをデプロイするお客様の場合は、Amazon EC2 インスタンスを複数のロケーション (AWS アベイラビリティーゾーンなど) にデプロイして、Auto Scaling などのサービスを使用して、自己修復を実装し、インスタンスにインストールしたアプリケーションに対して回復力のあるワークロードアーキテクチャのベストプラクティスを使用する責任があります。Amazon S3 と Amazon DynamoDB などのマネージドサービスの場合は、インフラストラクチャレイヤー、オペレーティングシステム、プラットフォームの運用を AWS が行い、お客様はエンドポイントにアクセスしてデータを保存、取得します。お客様は、バックアップ、バージョニング、レプリケーション戦略など、データの回復力を管理する責任があります。

AWS リージョン 内の複数のアベイラビリティーゾーンにワークロードをデプロイすることは、問題を単一のアベイラビリティーゾーンに分離することでワークロードを保護するように設計された高可用性戦略の一環です。これにより、その他のアベイラビリティーゾーンの冗長性を使用してリクエストの処理を継続できます。マルチ AZ アーキテクチャは、ワークロードをより適切に分離し、停電、落雷、竜巻、地震などの問題から保護するように設計された DR 戦略の一環でもあります。DR 戦略として、複数の AWS リージョン を利用することもできます。例えば、アクティブ/パッシブ設定では、アクティブなリージョンがリクエストを処理できなくなった場合、ワークロードのサービスはアクティブなリージョンから DR リージョンにフェイルオーバーします。

お客様と AWS のクラウド内とクラウドの回復力についての責任

お客様は AWS サービスを使用して、回復力の目標を達成できます。お客様は、システムの以下の側面を管理して、クラウドにおける回復力を実現する責任を担います。各サービスの具体的な詳細については、AWS のドキュメントを参照してください。

ネットワーク、クォータ、制約

- 責任共有モデルの分野のベストプラクティスについては、「基盤」セクションで詳しく説明されています。
- 必要に応じて予想される負荷リクエストの増加に基づいて、サービスクォータと、選択するサービスの制約を理解して、十分な余裕を持ってアーキテクチャプランニングを行ってください。
- ネットワークトポロジは、高可用性、冗長性、スケーラブルに設計します。

変更管理とオペレーショナルレジリエンス

- 変更管理には、環境に変更を導入して管理する方法が含まれます。変更を実装するには、ランブックを構築して最新の状態に維持し、アプリケーションとインフラストラクチャのデプロイ戦略が必要となります。
- ワークロードのリソースをモニタリングするための回復力戦略では、テクノロジーメトリクスとビジネスメトリクス、通知、自動化、分析など、すべてのコンポーネントを考慮します。

- クラウドのワークロードは、需要の変化に適応し、障害や使用量の変動に応じてスケーリングできる必要があります。

オブザーバビリティ管理と障害管理

- ワークロードがコンポーネントの障害に耐えられるように修復を自動化するには、モニタリングによる障害の観測が必要です。
- 障害管理には、データのバックアップ、ワークロードがコンポーネント障害に耐えられるためのベストプラクティスの適用、ディザスタリカバリプランニングが必要です。

ワークロードアーキテクチャ

- ワークロードアーキテクチャには、ビジネスドメインに関するサービスを設計する方法、SOA および分散システム設計を適用して障害を防止する方法、スロットリング、再試行、キュー管理、タイムアウト、緊急レバーなどの機能を組み込む方法が含まれます。
- 実証済みの AWS ソリューション、Amazon Builders Library、サーバーレスパターンを利用すると、ベストプラクティスに沿ってすぐに実装を開始できます。
- 継続的な改善を採用すると、システムを分散サービスに分解し、スケーリングとイノベーションを加速できます。AWS マイクロサービスガイダンスとマネージドサービスオプションを利用して、変化を導入して革新力を簡素化および加速します。

重要なインフラストラクチャの継続的テスト

- 信頼性テストとは、機能レベル、パフォーマンスレベル、カオスレベルでのテストを意味します。またこれは、インシデント分析とゲームデーのプラクティスを採用して、十分に理解されていない問題を解決するための専門知識を蓄積していくことでもあります。
- すべてをクラウドに移行した (オールイン) アプリケーションとハイブリッドアプリケーションの両方で、問題発生時やコンポーネントの停止時にアプリケーションがどのように動作するかを把握しておくと、停止から迅速かつ確実に復旧できます。
- 繰り返し可能な実験を作成して文書化して、期待どおりにいかない場合にシステムがどのように動作するかを理解しておきます。このようなテストは、全体的な回復力の有効性を証明するものであり、実際の障害シナリオに直面する前にオペレーション手順のフィードバックループを提供します。

設計原則

クラウドには、信頼性の向上に役立ついくつかの原則があります。ベストプラクティスについてこれから説明しますが、以下の原則を覚えておいてください。

- 障害から自動的に復旧する: 重要業績評価指標 (KPI) に関わるワークロードをモニタリングすることで、しきい値を超えた場合に自動操作をトリガーできます。この場合の KPI は、サービスの運用の技術的側面ではなく、ビジネス価値に関する指標である必要があります。これによって障害発生時の自動通知と追跡が可能になり、障害に対処する、または障害を修正するための復旧プロセスを自動化できます。より複雑な自動化を使用すると、障害が発生する前に修正を予期できます。
- 復旧手順をテストする: オンプレミス環境では、多くの場合、ワークロードが特定のシナリオで動作することを実証するために、テストを実施します。復旧戦略を検証するためにテストを実施することはあまりありません。クラウドでは、どのようにシステム障害が発生するかをテストでき、復旧の手順も検証できます。オートメーションにより、さまざまな障害のシミュレーションや過去の障害シナリオの再現を行うことができます。このアプローチでは、実際の障害シナリオが発生する前にテストおよび修正できる障害経路が公開されるため、リスクが軽減されます。
- 水平方向にスケールして集合的なワークロードの可用性を向上する: 単一の大規模なリソースを複数の小規模なリソースに置き換えることで、単一の障害がワークロード全体に及ぼす影響を軽減します。リクエストを複数の小規模なリソースに分散することで、一般的な障害点を共有しないようにします。
- キャパシティーを勘に頼らない: オンプレミスのワークロードにおける障害の一般的な原因はリソースの飽和状態で、ワークロードに対する需要がそのワークロードのキャパシティーを超えたときに発生します (これは頻繁にサービス妨害攻撃のターゲットとなる)。クラウドでは、需要とワークロード使用率をモニタリングし、リソースの追加と削除を自動化することで、プロビジョニングが過剰にも過小にもならない、需要を満たせる最適なレベルを維持できます。それでも制限はありますが、一部のクォータは制御でき、その他のクォータは管理可能です (「サービスクォータと制約を管理する) を指定する必要があります。
- オートメーションで変更を管理する: インフラストラクチャの変更はオートメーションを利用して行う必要があります。管理する必要がある変更には、自動化に対する変更が含まれており、それを追跡して確認することができます。

定義

このホワイトペーパーはクラウドの信頼性を対象としており、次の 4 つの領域のベストプラクティスについて説明します。

- 基盤
- ワークロードアーキテクチャ
- 変更管理
- 障害管理

信頼性を実現するためには、基盤、つまりワークロードに対して、サービスクオータとネットワークトポロジーを適応させた環境、これを作ることから始めなければなりません。分散システムにおけるワークロードのアーキテクチャは、障害を防止および軽減するように設計する必要があります。ワークロードは需要または要件の変化に対応する必要があり、障害を検出して自動的に復旧するように設計する必要があります。

トピック

- 回復力、および信頼性のコンポーネント
- 可用性
- ディザスタリカバリ (DR) 目標

回復力、および信頼性のコンポーネント

クラウド内のワークロードの信頼性はいくつかの要因に依存しますが、その主なものは回復力です。

- 回復力とは、インフラストラクチャやサービスの中断から復旧し、需要に適したコンピューティングリソースを動的に獲得し、設定ミスや一時的なネットワークの問題などの、中断の影響を緩和するワークロードの能力です。

ワークロードの信頼性に影響を与えるその他の要因には、次のものがあります。

- 運用上の優秀性。これには、変更の自動化、障害対応のためのプレイブックの利用、アプリケーションが本番運用の準備ができていることを確認するための運用準備状況レビュー (ORR) が含まれます。
- セキュリティ。これには、可用性に影響を与える、悪意のある行為によるデータまたはインフラストラクチャへの危害を防止することが含まれます。たとえば、データの安全性を確保するには、バックアップを暗号化します。
- パフォーマンス効率。これには、最大リクエストレートの設計とワークロードに対するレイテンシーの最小化が含まれます。

- コスト最適化。これには、静的な安定性を達成するために EC2 インスタンスにより多く費用をかけるか、より多くの容量が必要な場合に Auto Scaling に依存するかどうかといったトレードオフが含まれます。

回復性は、このホワイトペーパーにおける最大の焦点です。

その他の 4 つの側面も重要です。これらの側面については、AWS Well-Architected Framework のそれぞれの柱でカバーされています。ここに記載されているベストプラクティスの多くは、信頼性の面にも対応しますが、焦点は回復力です。

可用性

可用性 (サービス可用性とも呼ばれる) は、回復力を数量的に測定するためによく使用されるメトリクスであると同時に、的を絞った回復力目標でもあります。

- 可用性は、ワークロードが使用可能な時間の割合で示されます。

可用 (使用可能) とは、必要なときに取り決めた機能を実行できることを意味します。

この割合 (%) は、月、年、直近 3 年などの時間単位で計算します。可能な限り厳密に解釈すると、予定された中断や予定外の中断を含め、アプリケーションが正常に動作しないときは可用性が下がることになります。可用性は以下のとおり定義されます。

$$\text{可用性} = \frac{\text{使用可能時間}}{\text{合計時間}}$$

- 可用性は、一定期間 (通常は 1 か月または 1 年) の稼働時間の割合 (例: 99.9%) です。
- 一般的には「9 の個数」で省略して表現され、例えば、「ファイブナイン」は 99.999% の可用性という意味になります。
- お客様によっては、定義の式にある合計時間から、予定されたサービスダウンタイム (定期メンテナンスなど) を除外する場合もあります。ただし、このような時間にユーザーがサービスを使用したい可能性があるため、この方法はお勧めしません。

アプリケーションの可用性における一般的な設計目標と、この目標を達成しながら 1 年以内に中断が発生する可能性のある最大時間を以下の表に示します。この表には、可用性レベルごとによく知ら

れているアプリケーションタイプの例が示されています。このドキュメント全体を通して、これらの可用性の値を参考にします。

可用性	最大利用不可能時間 (年間)	アプリケーションのカテゴリ
99%	3 日と 15 時間	バッチ処理、データの抽出、転送、ロードのジョブ
99.9%	8 時間 45 分	ナレッジ管理、プロジェクト追跡などの社内ツール
99.95%	4 時間 22 分	オンラインコマース、POS
99.99%	52 分間	動画配信、ブロードキャストのワークロード
99.999%	5 分間	ATM トランザクション、通信のワークロード

リクエストに基づく可用性の測定。 サービスの場合、「使用可能な時間」の代わりに、成功したリクエスト数と失敗したリクエスト数をカウントする方が容易かもしれません。この場合、次の計算を使用できます。

$$\text{可用性} = \frac{\text{成功応答}}{\text{有効な要求}}$$

これは、多くの場合、1 分間または 5 分間で測定されます。次に、これらの期間の平均から、1 か月のアップタイム率 (時間ベースの可用性測定) を計算できます。特定の期間に受信したリクエストがなかった場合、その時間の可用性は 100% になります。

ハードな依存関係を持つ可用性を計算する 多くのシステムは他のシステムとハードな依存関係にあり、依存するシステムでサービス停止が起こると、それを呼び出す側のシステムにも影響します。この反対はソフトな依存関係で、依存関係にあるシステムに障害が起こると、アプリケーションがそれを補完します。ハードな依存関係が存在すると、呼び出す側のシステムにおける可用性は、依存するシステムの製品の可用性であるということになります。たとえば、可用性 99.99% を実現するように

設計されたシステムが、同様に可用性が 99.99% である他の 2 つのシステムに依存する場合、このワークロードの可用性は理論的には 99.97% になります。

$Avail_{invok} \times Avail_{dep1} \times Avail_{dep2} = Avail_{ワークロード}$

99.99% × 99.99% × 99.99% = 99.97%

したがって、お客様自身で可用性を計算するにあたって、このシステムとの依存関係と、それらの可用性の設計目標を理解することが重要です。

冗長コンポーネントの可用性を計算する システムが、独立し冗長化されたコンポーネント (複数のアベイラビリティゾーンの冗長リソースなど) を使用する場合、理論的な可用性は、100% からそのコンポーネントの障害率の積を引いたものになります。例えば、あるシステムが 2 つの独立したコンポーネントを利用し、それぞれ 99.9% の可用性を持つ場合、この依存関係の実効可用性は 99.999% になります。

$Avail_{費用対効果} = Avail_{MAX} - ((100\% - Avail_{dependency}) \times (100\% - Avail_{dependency}))$

99.9999% = 100% − (0.1%×0.1%)

簡単な計算方法: 計算内のすべてのコンポーネントの可用性が 9 のみで構成される場合、9 の桁の数を合計するだけで答えが得られます。上記の例では、2 つの冗長な独立したコンポーネントは 9 が 3 つの可用性を持つため、結果は 9 が 6 つになります。

依存するシステムの可用性を計算する 依存関係によっては、多くの AWS のサービスの可用性設計目標など、可用性に関するガイダンスを提供するものもあります を指定する必要があります。しかしここに情報がない場合 (例えば、メーカーが可用性に関する情報を開示していない場合) は、平均故障間隔 (MTBF) および平均復旧時間 (MTTR) を計算して推測値を出す方法があります。可用性の推測値は、次の方法で計算できます。

$$Avail_{EST} = \frac{MTBF}{MTBF + MTTR}$$

例えば、MTBF が 150 日、MTTR が 1 時間なら、推測値は 99.97% です。

詳細については、「Availability and Beyond: Understanding and improving the resilience of distributed systems on AWS」(可用性とその先へ: AWS での分散システムの回復力の把握と改善) を確認すると、可用性の計算の役に立ちます。

可用性のコスト 通常はアプリケーションの可用性レベルを高く設計すればコストは増大するため、設計に着手する前に、可用性の正確なニーズを特定することが重要です。可用性レベルが高いと、網羅的な障害シナリオのもとで厳しいテストおよび検証条件が課されることになります。このような場合、全障害パターンからの自動復旧、全システム動作が同一基準に基づいて構築およびテストされることが求められます。例えば、キャパシティーの追加や削除、更新されたソフトウェアや設定変更のデプロイメントおよびロールバック、システムデータの移行などが、可用性の設計目標を満たすように実施される必要があります。非常に高いレベルの可用性を前提にソフトウェア開発のコストを積み上げると、システムのデプロイメント速度は遅くなるため、イノベーションは難しくなります。したがってこれに対するガイダンスは、基準を適用しながら、システム稼働におけるライフサイクル全体にわたって適切な可用性の目標を検討することです。

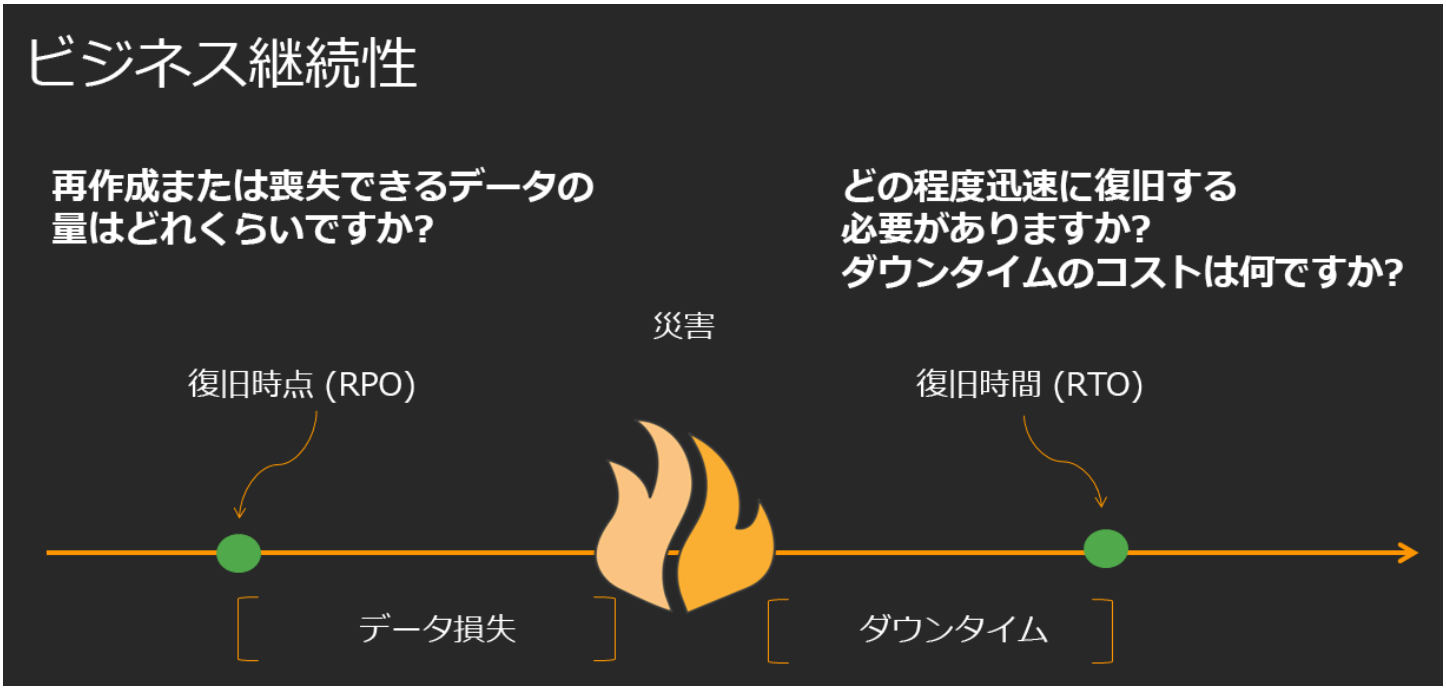
可用性の設計目標が高いシステムにおけるもう一つのコスト増大の原因は、依存関係にあるシステムの選択にあります。目標レベルが高ければ、依存関係を持つソフトウェアまたはサービスの選択肢が狭くなり、前述したようにそれに基づくコストも増大していきます。可用性の設計目標が高くなるほど、リレーショナルデータベースのような多目的のサービスは少なくなり、目的に特化したサービスが多くなります。この理由は、後者のサービスの評価、テスト、自動化は簡単で、使用されていない機能で予期しない動作が起こる可能性が低くなるからです。

ディザスタリカバリ (DR) 目標

可用性目標のほかに、回復力戦略には、災害イベント時にワークロードを復旧する戦略に基づいたディザスタリカバリ (DR) 目標も含める必要があります。ディザスタリカバリは、自然災害、大規模な技術的障害、または攻撃や エラーなどの人的脅威に対応する 1 回限りの復旧目標に焦点を当てています。これは、コンポーネントの障害、負荷の急増、またはソフトウェアのバグに対応して、一定期間の平均回復力を測定する可用性とは異なります。

目標復旧時間 (RTO) 組織が定義します。RTO は、サービスの中断からサービスの復元までの最大許容遅延です。これにより、サービスが利用できないときに許容可能と見なされる時間枠が決まります。

目標復旧時点 (RPO) 組織が定義します。RPO は、最後のデータ復旧ポイントからの最大許容時間です。これにより、最後の復旧ポイントからサービスの中断までの間に許容可能と見なされるデータ損失が決まります。



RPO (目標復旧時点) 、RTO (目標復旧時間) 、災害イベントの関係。

RTO は、停止の開始からワークロード復旧までの時間を測定する点で、MTTR (平均復旧時間) と似ています。ただし、MTTR は、一定の期間にわたって複数の可用性に影響を与えるイベントに対して取られた平均値であり、RTO は、可用性に影響を与える 1 つのイベントに対するターゲット、または許容できる最大値です。

可用性ニーズの理解

アプリケーションの可用性を、初めにアプリケーション全体の単一ターゲットとして考えてしまうケースがよくあります。しかし詳しく分析してみれば、アプリケーションやサービスは、特定の側面によって求められる可用性がさまざまであることに気付くこともしばしばです。たとえば、システムによっては、既存データを取得するよりも、新しいデータを受信して保存する機能を優先させる場合があります。また、システムの構成や環境を変更するオペレーションよりも、リアルタイムオペレーションを優先させるシステムもあります。1 日のうち特定の時間帯に非常に高い可用性が要求される

サービスでも、その時間帯以外は長時間の中断ができる可能性もあります。これらは、1 つのアプリケーションを分解し、それぞれの構成要素の可用性要件を評価することのほんの一例です。この考え方のメリットは、システム全体を最も厳格な要件に合わせて設計するのではなく、特定のニーズに応じた可用性に労力 (および費用) をかけられることです。

推奨

各アプリケーションに固有の局面を厳密に評価し、必要な場合は、ビジネスのニーズを反映して、可用性とディザスタリカバリの設計目標を差別化してください。

AWS では、通常、サービスを「データプレーン」と「コントロールプレーン」に分けて考えます。コントロールプレーンで環境を設定しつつ、データプレーンではリアルタイムのサービスを提供します。例えば、Amazon EC2 インスタンス、Amazon RDS データベース、Amazon DynamoDB テーブルの読み取り / 書き込みオペレーションはすべてデータプレーンによる操作です。逆に、EC2 インスタンスや RDS データベースの起動、DynamoDB のテーブルメタデータの追加や変更などは、すべてコントロールプレーンによる操作です。これらすべての機能には高レベルの可用性が重要となりますが、一般的にデータプレーンの可用性設計の目標は、コントロールプレーンより高くなります。したがって、高い可用性要件を持つワークロードでは、コントロールプレーンの操作に対するランタイム依存を避ける必要があります。

AWS のお客様の多くは、類似のアプローチを採用して、アプリケーションを厳密に評価し、さまざまな可用性ニーズを持つサブコンポーネントを特定しています。次に、さまざまな側面に応じた可用性設計目標の調整を行い、システムを設計するための適切な作業が行われます。AWS には、99.999% 以上の可用性を持つサービスを含め、広い範囲の可用性設計目標を持つアプリケーションを開発する豊富な経験があります。AWS のソリューションアーキテクト (SA) は、お客様の可用性目標に対する適切な設計を支援します。設計プロセスの初期段階から AWS を導入していただくことで、私たちが可用性目標の達成を支援する能力も向上します。可用性の計画は、実際のワークロードが始動する直前にだけ立てるものではありません。また、運用上の経験を積み、現実世界の出来事から学び、さまざまなタイプの障害に耐えながら、設計を改良し続けることも行います。そうすることで、適切な作業を行って実際の運用を改善できます。

ワークロードに求められる可用性のニーズは、ビジネスのニーズと重要度に合わせる必要があります。まず、RTO、RPO、および可用性を明確にしてビジネスにとって何が重要なのかのフレームワークを明確にすることで、各ワークロードを評価できます。このようなアプローチでは、ワークロードを実際に運用する人がフレームワークに精通し、そのワークロードがビジネスニーズに与える影響を理解している必要があります。

基盤

基盤となる要件は、単一のワークロードまたはプロジェクトの範囲を超える要件です。システムを設計する前に、信頼性に影響を与える基本的な要件を満たしておく必要があります。例えば、データセンターへの十分なネットワーク帯域幅が必要です。

オンプレミス環境では、依存関係により、このような要件が長いリードタイムの原因となる可能性があるため、初期計画に組み込んでおく必要があります。ただし、AWS では、このような基本的な要件のほとんどが既に組み込まれており、必要に応じて変更できます。クラウドは、ほぼ制限を持たないように設計されています。つまり、十分なネットワーク性能とコンピューティング性能の要件を満たすのは AWS の責任であり、お客様はリソースのサイズと割り当てを需要に応じて自由に変更できます。

以下のセクションでは、このような信頼性について考慮すべき点に焦点を当てたベストプラクティスについて説明します。

トピック

- サービスクォータと制約を管理する
- ネットワークトポロジを計画する

サービスクォータと制約を管理する

クラウドベースのワークロードアーキテクチャには、サービスクォータ (サービスの制限とも呼ばれます) というものがあります。このようなクォータは、誤って必要以上のリソースをプロビジョニングするのを防ぎ、サービスを不正使用から保護することを目的として API 操作のリクエスト頻度を制限するために存在します。リソースにも制約があります。たとえば、光ファイバーケーブルのビットレートや、物理ディスクの記憶容量などです。

AWS Marketplace のアプリケーションを使用している場合、アプリケーションの制限を理解する必要があります。サードパーティのウェブサービスや SaaS を使用している場合は、これらの制限も認識しておく必要があります。

ベストプラクティス

- REL01-BP01 サービスクォータと制約を認識する
- REL01-BP02 アカウントおよびリージョンをまたいでサービスクォータを管理する
- REL01-BP03 アーキテクチャを通じて、固定サービスクォータと制約に対応する

- REL01-BP04 クォータをモニタリングおよび管理する
- REL01-BP05 クォータ管理を自動化する
- REL01-BP06 フェイルオーバーに対応するために、現在のクォータと最大使用量の間に十分なギャップがあることを確認する

REL01-BP01 サービスクォータと制約を認識する

デフォルトのクォータに注意して、ワークロードアーキテクチャに対するクォータ引き上げリクエストを管理しましょう。ディスクやネットワークなど、どのクラウドリソースの制約が潜在的に大きな影響を与えるかを知っておきましょう。

期待される成果: 主要メトリクスのモニタリング、インフラストラクチャのレビュー、自動修復手順に適切なガイドラインを設けて、サービスの機能低下や停止につながるサービスのクォーターや制約に達していないことを検証することで、AWS アカウントのサービスの機能低下や停止を防止できます。

一般的なアンチパターン:

- 使用しているサービスのハードまたはソフト上のクォータや制限を理解せずにワークロードをデプロイする。
- 必要なクォータの分析と再設定、またはサポートへの事前連絡をせずに、代替ワークロードをデプロイする。
- クラウドサービスには制限がなく、料金、制限、回数、数量を気にせずにサービスを使用できると考えている。
- クォータは自動的に増加すると考えている。
- クォータリクエストのプロセスやスケジュールを知らない。
- クラウドサービスのデフォルトのクォータが、リージョン間で比較する各サービスで同一だと考えている。
- サービスの制約は破ることが可能で、システムによりリソースの制約を超えて自動スケールまたは制限の増加が追加されると考えている。
- リソースの使用率にストレスをかけるためにピーク時トラフィックでアプリケーションをテストしていない。
- 必要なリソースサイズを分析せずにリソースをプロビジョニングする。
- 実際に必要な分または予想されるピークを遥かに超えるリソースタイプを選択することでキャパシティを過剰プロビジョニングする。

- 新規顧客イベントや新技術のデプロイに先駆けて、新しいレベルのトラフィックのキャパシティ要件を評価していない。

このベストプラクティスを活用するメリット: サービスクォータとリソースの制約をモニタリングおよび自動管理すると、エラーを予防できます。ベストプラクティスに従っていないと、顧客のサービスにおけるトラフィックパターンの変化により、停止または機能低下が起こる可能性があります。これらの値をすべてのリージョンとすべてのアカウントでモニタリングし管理することで、有害なイベントや計画外のイベントにおける、アプリケーションの回復力が向上します。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

Service Quotas は、250 を超える AWS のサービスのクォータを一元的に管理するのに役立つ AWS のサービスです。クォータ値の検索に加えて、Service Quotas コンソールから、または AWS SDK を使用してクォータ増加をリクエスト、追跡することもできます。AWS Trusted Advisor には、あるサービスの一部の要素に関する使用状況とクォータを表示するサービスクォータチェックが用意されています。サービスごとのデフォルトのサービスクォータは、それぞれのサービスの AWS ドキュメントにも記載されています (例えば、Amazon VPC クォータを参照してください)。

スロットルされた API のレート制限など、一部のサービス上の制限は、Amazon API Gateway 内で使用量プランを変更することで設定できます。それぞれのサービス上の構成として設定される一部の制限には、プロビジョンド IOPS、割り当てられた Amazon RDS ストレージ、Amazon EBS ボリューム割り当てなどがあります。Amazon Elastic Compute Cloud には、インスタンス、Amazon Elastic Block Store、および Elastic IP アドレスの制限を管理するのに役立つ独自のサービスの制限ダッシュボードがあります。サービスクォータがアプリケーションのパフォーマンスに影響を及ぼし、ニーズに合わせて調整できないような事例が発生した場合は、AWS Support に連絡し、緩和策の有無についてお問い合わせください。

サービスクォータはその性質上、リージョン固有である場合も、グローバルである場合もあります。クォータに達している AWS サービスを使用すると、通常の使用で予想どおりに動作しないことや、サービスの停止や機能低下を招くことがあります。例えば、あるサービスクォータではリージョンで使用する DL Amazon EC2 数を制限しており、Auto Scaling グループ (ASG) を使用したトラフィックスケーリングイベント中にその制限に達する可能性があります。

各アカウントのサービスクォータについて、使用量を定期的に評価し、そのアカウントにおける適切なサービス制限を判断する必要があります。このようなサービスクォータは、意図せず必要以上のリソースをプロビジョニングすることを防止する運用上のガードレールとして存在しています。

また、API オペレーションにおけるリクエスト率を制限し、不正使用からサービスを保護する役目も持っています。

サービスの制約は、サービスクォータとは異なります。サービスの制約は、リソースタイプごとに決まっている特定のリソースの制限を表します。ストレージキャパシティ (例: gp2 のサイズ制限は 1 GB ~ 16 TB) だったりディスクスループット (10,0000 iops) だったりします。制限に達する可能性のある使用量について、リソースタイプの制約を監督し定期的に評価することが不可欠です。予期せず制約に達した場合、アカウントのアプリケーションまたはサービスが機能低下または停止する恐れがあります。

サービスクォータがアプリケーションのパフォーマンスに影響を及ぼし、ニーズに合わせて調整できないような事例がある場合は、AWS Support に連絡し、緩和策の有無についてお問い合わせください。修正されたクォータの調整について詳しくは、REL01-BP03 アーキテクチャを通じて、固定サービスクォータと制約に対応する を参照してください。

Service Quotas をモニタリングおよび管理できる AWS のサービスやツールが多数用意されています。これらのサービスやツールは、クォータレベルの自動または手動チェックの提供に活用するものです。

- AWS Trusted Advisor は、いくつかのサービスの一部の側面に対する利用状況とクォータを表示する、サービスクォータチェックを提供しています。クォータに迫っているサービスの特定に役立ちます。
- AWS Management Consoleでは、サービスのクォータ値の表示、管理、新しいクォータのリクエスト、クォータリクエストのステータスのモニタリング、クォータの履歴の表示を行う手段を提供しています。
- AWS CLI および CDK は、サービスクォータのレベルと使用量を自動で管理およびモニタリングする、プログラムによる手段を提供します。

実装手順

Service Quotas の場合:

- AWS Service Quotas をレビューします。
- 既存のサービスクォータを把握し、使用されているサービス (IAM Access Analyzer など) を判断します。サービスクォータで制御されている AWS のサービスは約 250 あります。次に、各アカウントとリージョンで使用されている可能性のある特定のサービスクォータ名を判断します。リージョンごとに約 3,000 のサービスクォータ名があります。

- このクォータ分析を AWS Config で強化して、AWS アカウントで使用されているすべての AWS リソースを見つけます。
- AWS CloudFormation データを使用して、使用されている AWS リソースを判断します。AWS Management Consoleまたは `list-stack-resources` AWS CLI コマンドを使用して作成されたリソースを見つけます。テンプレート自体にデプロイされるように設定されたリソースも確認できます。
- デプロイコードを見て、ワークロードに必要なすべてのサービスを決定します。
- 適用されるサービスクォータを決定します。Trusted Advisor および Service Quotas を使用してプログラムでアクセスできる情報を使用します。
- サービスクォータが制限に近づいたか達した場合にアラートを発して知らせる自動モニタリング手段を作成します (REL01-BP02 アカウントおよびリージョンをまたいでサービスクォータを管理する および REL01-BP04 クォータをモニタリングおよび管理する を参照)。
- 同一アカウント内のあるリージョンでサービスクォータが変更されたが他のリージョンでは変更されていない場合を確認する、自動またはプログラムによる手段を作成します (REL01-BP02 アカウントおよびリージョンをまたいでサービスクォータを管理する および REL01-BP04 クォータをモニタリングおよび管理する を参照)。
- アプリケーションログやメトリクスのスキャンを自動化して、クォータまたはサービス制約のエラーが発生しているか判断します。エラーが発生している場合は、モニタリングシステムにアラートを送信します。
- 特定のサービスでクォータの引き上げが必要であると判断された場合に、クォータで必要な変更を計算するエンジニアリング手順を作成します (REL01-BP05 クォータ管理を自動化する を参照)。
- サービスクォータの変更をリクエストするプロビジョニングおよび承認ワークフローを作成します。これには、リクエストが拒否または一部承認された場合の例外ワークフローを含めます。
- 本稼働環境またはロード済み環境にロールアウトする前に、新しい AWS サービスをプロビジョニングして使用するにあたって、サービスクォータをレビューするエンジニアリング手段を作成します (例: ロードテストアカウント)。

サービス制約の場合:

- 読み取りがリソースの制約に近づいているリソースについてアラートを発報するモニタリングおよびメトリクス手段を作成します。必要に応じて CloudWatch を活用して、メトリクスまたはログをモニタリングします。
- 制約のある各リソースについて、アプリケーションまたはシステムにとって有意なアラートのしきい値を設定します。

- 制約が使用量に近い場合、リソースタイプを変更するワークフローまたはインフラストラクチャ管理手順を作成します。このワークフローには、ベストプラクティスとして負荷テストを含め、新しいタイプが新しい制約のある適切なリソースタイプであることを検証します。
- 既存の手順やプロセスを使用して、特定したリソースを推奨の新しいリソースタイプに移行します。

リソース

関連するベストプラクティス:

- REL01-BP02 アカウントおよびリージョンをまたいでサービスクォータを管理する
- REL01-BP03 アーキテクチャを通じて、固定サービスクォータと制約に対応する
- REL01-BP04 クォータをモニタリングおよび管理する
- REL01-BP05 クォータ管理を自動化する
- REL01-BP06 フェイルオーバーに対応するために、現在のクォータと最大使用量の間に十分なギャップがあることを確認する
- REL03-BP01 ワークロードをセグメント化する方法を選択する
- REL10-BP01 複数の場所にワークロードをデプロイする
- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する
- REL11-BP03 すべてのレイヤーの修復を自動化する
- REL12-BP05 カオスエンジニアリングを使用して回復力をテストする

関連するドキュメント:

- AWS Well Architected フレームワークの信頼性の柱: 可用性
- AWS Service Quotas (旧称は「サービスの制限」)
- AWS Trusted Advisor ベストプラクティスチェックリスト (「サービスの制限」セクションを参照)
- AWS Answers の AWS Limit Monitor
- Amazon EC2 サービスの制限
- Service Quotas とは
- クォータの引き上げのリクエスト方法
- サービスエンドポイントとクォータ

- Service Quotas ユーザーガイド
- AWS のクォータモニター
- AWS Fault Isolation Boundaries (AWS の障害分離境界)
- Availability with redundancy (冗長性を備えた可用性)
- データのための AWS
- 継続的インテグレーションとは
- 継続的デリバリーとは
- APN パートナー: 設定管理を支援できるパートナー
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS(AWS のテナント別アカウント SaaS 環境でアカウントのライフサイクルを管理する)
- Managing and monitoring API throttling in your workloads (ワークロードの API スロットリングの管理とモニタリング)
- View AWS Trusted Advisor recommendations at scale with AWS Organizations(AWS Organizations で AWS Trusted Advisor の推奨事項を大規模に表示する)
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower(AWS Control Tower でサービス制限の緩和とエンタープライズサポートを自動化する)

関連動画:

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas (Service Quotas を使用して AWS のサービスのクォータを表示および管理する)
- AWS IAM Quotas Demo (AWS IAM のクォータのデモ)

関連ツール:

- Amazon CodeGuru Reviewer
- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru

- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP02 アカウントおよびリージョンをまたいでサービスクォータを管理する

複数のアカウントまたはリージョンをご利用の場合は、本番ワークロードを実行するすべての環境で適切なクォータをリクエストしてください。

期待される成果: 複数のアカウントまたはリージョンにまたがる設定、またはゾーン、リージョン、アカウントのフェイルオーバーを使用したレジリエンス設計になっている設定において、サービスクォータの枯渇によってサービスやアプリケーションが影響を受けないようにします。

一般的なアンチパターン:

- 1 つの分離リージョンでのリソース使用量の増加を許可するが、他のリージョンではキャパシティーを維持するメカニズムがない。
- 分離リージョン内のすべてのクォータを手動で設定する。
- 主要ではないリージョンの能力が低下している際に、将来のクォータの必要性について、回復力のあるアーキテクチャ (アクティブやパッシブなど) の影響を考慮していない。
- ワークロードが実行されているすべてのリージョンやアカウントにおいて、クォータを定期的に評価し、必要な変更を行っていない。
- 複数のリージョンやアカウントにまたがって緩和をリクエストする際に、クォータリクエストテンプレートを活用していない。
- クォータの緩和は、コンピューティング予約リクエストのように、コストに影響があるという誤った考えのもと、サービスクォータを更新していない。

このベストプラクティスを活用するメリット: リージョンでのサービスが利用不可になった際に、セカンダリリージョンやアカウントで現在の負荷を処理できることを検証します。これにより、リージョンを利用できない場合に発生するエラー数や機能低下のレベルを削減できます。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

サービスクォータの追跡はアカウントごとに行います。特に明記されていない限り、各クォータは AWS リージョン 固有です。テストと開発が妨げられないように、本番環境に加えて、該当するすべての非本番環境でもクォータを管理します。高レベルの回復性を維持するには、サービスクォータを継続的に評価する必要があります (自動または手動で)。

アクティブ/アクティブ、アクティブ/パッシブ - ホット、アクティブ/パッシブ - コールド、アクティブ/パッシブ - パイロットライトの各アプローチを使用した設計の実装により、複数のリージョンにまたがるワークロードが増えると、すべてのリージョンおよびアカウントのクォータレベルを把握することが不可欠になってきます。サービスクォータが正しく設定されている場合、過去のトラフィックパターンが常に適した指標になるとは限りません。

同じくらい重要なのが、サービスクォータ名の制限が各リージョンで常に同じとは限らないことです。あるリージョンでは値が 5 であり、別のリージョンでは値が 10 であることもありえます。負荷時の一貫した回復力を提要するために、このようなクォータの管理は、すべての同じサービス、アカウント、リージョンを網羅する必要があります。

異なるリージョン (アクティブリージョンまたはパッシブリージョン) 間の全てのサービスクォータの差異を調整し、これらの差異を継続的に調整するプロセスを作成します。パッシブリージョンのフェイルオーバーのテスト計画は、ピーク時のアクティブキャパシティまでスケールすることはめったにありません。つまり、ゲームデーや机上演習では、リージョン間のサービスクォータの差異を見つけられない場合があり、したがって適切な制限を維持できない場合があります。

サービスクォータのドリフトとは、特定の名前のクォータにおけるサービスクォータの制限が、すべてのリージョンではなく、1 つのリージョンで変更される状況であり、追跡と評価が非常に重要になります。トラフィックを伴う、またはトラフィックを伴う可能性があるリージョンでのクォータの変更を、考慮する必要があります。

- サービスの要件、レイテンシー、およびディザスタリカバリ (DR) 要件に基づいて、関連するアカウントとリージョンを選択します。
- 関連するすべてのアカウント、リージョン、アベイラビリティーゾーン全体のサービスクォータを特定します。制限の対象範囲はアカウントとリージョンです。これらの値を比較して差異を把握する必要があります。

実装手順

- 使用量のリスクレベルを超えている可能性がある Service Quotas の値をレビューします。AWS Trusted Advisor は 80% および 90% のしきい値で違反を警告します。

- パッシブリージョンのサービスクォータの値をレビューします (アクティブ/パッシブ設計の場合)。プライマリリージョンで障害があった場合に、負荷が正常にセカンダリリージョンで実行されることを検証します。
- サービスクォータのドリフトが同一アカウント内のリージョン間で発生し、それに伴って制限が変更された場合の評価を自動化します。
- お客様の組織単位 (OU) がサポートされている方法で構成されている場合、サービスクォータテンプレートを更新して、クォータの変更を反映し、複数のリージョンやアカウントに適用する必要があります。
  - テンプレートを作成して、リージョンをクォータの変更に関連付けます。
  - 既存のすべてのサービスクォータテンプレートをレビューして、変更が必要かどうかを確認します (リージョン、制限、アカウント)。

リソース

関連するベストプラクティス:

- REL01-BP01 サービスクォータと制約を認識する
- REL01-BP03 アーキテクチャを通じて、固定サービスクォータと制約に対応する
- REL01-BP04 クォータをモニタリングおよび管理する
- REL01-BP05 クォータ管理を自動化する
- REL01-BP06 フェイルオーバーに対応するために、現在のクォータと最大使用量の間に十分なギャップがあることを確認する
- REL03-BP01 ワークロードをセグメント化する方法を選択する
- REL10-BP01 複数の場所にワークロードをデプロイする
- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する
- REL11-BP03 すべてのレイヤーの修復を自動化する
- REL12-BP05 カオスエンジニアリングを使用して回復力をテストする

関連するドキュメント:

- AWS Well Architected フレームワークの信頼性の柱: 可用性
- AWS Service Quotas (旧称は「サービスの制限」)
- AWS Trusted Advisor ベストプラクティスチェックリスト (「サービスの制限」セクションを参照)

- AWS Answers の AWS Limit Monitor
- Amazon EC2 サービスの制限
- Service Quotas とは
- クォータの引き上げのリクエスト方法
- サービスエンドポイントとクォータ
- Service Quotas ユーザーガイド
- AWS のクォータモニター
- AWS Fault Isolation Boundaries (AWS の障害分離境界)
- Availability with redundancy (冗長性を備えた可用性)
- データのための AWS
- 継続的インテグレーションとは
- 継続的デリバリーとは
- APN パートナー: 設定管理を支援できるパートナー
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS(AWS のテナント別アカウント SaaS 環境でアカウントのライフサイクルを管理する)
- Managing and monitoring API throttling in your workloads (ワークロードの API スロットリングの管理とモニタリング)
- View AWS Trusted Advisor recommendations at scale with AWS Organizations(AWS Organizations で AWS Trusted Advisor の推奨事項を大規模に表示する)
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower(AWS Control Tower でサービス制限の緩和とエンタープライズサポートを自動化する)

関連動画:

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas (Service Quotas を使用して AWS のサービスのクォータを表示および管理する)
- AWS IAM Quotas Demo (AWS IAM のクォータのデモ)

関連サービス:

- Amazon CodeGuru Reviewer

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP03 アーキテクチャを通じて、固定サービスクォータと制約に対応する

サービスクォータ、サービスの制約、物理リソースの制限には変更できないものもあることに注意します。このような制限が信頼性に影響を及ぼさないように、アプリケーションとサービスのアーキテクチャを設計します。

例えば、ネットワーク帯域幅、サーバーレス関数呼び出しのペイロードサイズ、API Gateway のスロットルバーストレート、データベースへの同時ユーザー接続数などは変更できません。

期待される成果: アプリケーションやサービスは、通常のトラフィック状況および高トラフィック状況で期待されるとおりに動作します。アプリケーションやサービスは、リソースの固定制約またはサービスクォータの制限内で機能するように設計されています。

一般的なアンチパターン:

- 単一のサービスリソースを使用する設計を選択し、スケーリング時に障害が発生する原因となる設計上の制約があることを認識していない。
- テスト中にサービスの固定クォータに到達してしまう非現実的なベンチマークを採用している。(例: バースト制限を利用しながら長時間テストを実行する)
- 固定サービスクォータを超えてもスケーリングしたり変更したりできない設計を選択する。(例: 256KB のペイロードサイズの SQS)
- 高トラフィックイベント中に危険にさらされる可能性があるサービスクォータのしきい値をモニタリングしたり警告したりするために、オブザーバビリティが設計および実装されていない。

このベストプラクティスを活用するメリット: 予測されるあらゆるサービス負荷レベルで、アプリケーションが中断や低下することなく実行されることを確認できます。

このベストプラクティスを確立しない場合のリスクレベル: 中

実装のガイダンス

ソフトサービスクォータ、つまりより高い容量ユニットに置き換えられるリソースとは異なり、AWS サービスの固定クォータは変更できません。つまり、アプリケーションの設計で使用する場合、このようなタイプのすべての AWS サービスの容量のハード制限を評価する必要があります。

ハード制限は Service Quotas コンソールに表示されます。列に `ADJUSTABLE = No`と表示されていれば、そのサービスにはハード制限があります。ハード制限は、一部のリソース設定ページにも表示されます。例えば、Lambda には調整できない特定のハード制限があります。

例としては、Lambda 関数で実行する Python アプリケーションを設計する場合、Lambda が 15 分以上実行される可能性があるかを判断するためにアプリケーションを評価する必要があります。コードがこのサービスクォータ制限を超過して実行される可能性がある場合は、代替テクノロジーや設計を検討する必要があります。本番環境へのデプロイ後にこの制限に達すると、修正されるまでアプリケーションの機能が低下し、中断することになります。ソフトクォータとは異なり、このような制限は、重大度 1 の緊急事態が発生した場合でも、変更できません。

アプリケーションがテスト環境にデプロイされたら、ハード制限に到達できるかどうかを確認する戦略を行使する必要があります。ストレステスト、負荷テスト、カオステストは、導入テスト計画の一環とする必要があります。

実装手順

- 使用できる AWS サービスの完全なリストをアプリケーションの設計段階で確認します。
- これらすべてのサービスについて、ソフトクォータ制限とハードクォータ制限を確認します。すべての制限が Service Quotas コンソールに表示されているとは限りません。サービスによっては、このような制限について、別の場所で説明されています。
- アプリケーションを設計する際は、ビジネス上の成果、ユースケース、依存するシステム、可用性目標、ディザスタリカバリオブジェクトなど、ワークロードのビジネスとテクノロジーの推進要因を確認します。ビジネスとテクノロジーの推進要因に従って、ワークロードに適した分散システムを特定するプロセスを進めます。
- リージョン全体とアカウント全体にわたるサービス負荷を分析します。ハード制限の多くは、サービスのリージョンに基づいていますが、アカウントに基づく制限もあります。

- ゾーン障害時とリージョン障害時のリソース使用状況について、回復力あるアーキテクチャを分析します。「アクティブ/アクティブ」、「アクティブ/パッシブ – ホット」、「アクティブ/パッシブ – コールド」、「アクティブ/パッシブ – パイロットライト」のアプローチを使用するマルチリージョン設計を進めると、このような障害ケースはより高い使用状況につながり、ハード制限に達するユースケースを生み出す可能性が出てきます。

リソース

関連するベストプラクティス:

- REL01-BP01 サービスクォータと制約を認識する
- REL01-BP02 アカウントおよびリージョンをまたいでサービスクォータを管理する
- REL01-BP04 クォータをモニタリングおよび管理する
- REL01-BP05 クォータ管理を自動化する
- REL01-BP06 フェイルオーバーに対応するために、現在のクォータと最大使用量の間に十分なギャップがあることを確認する
- REL03-BP01 ワークロードをセグメント化する方法を選択する
- REL10-BP01 複数の場所にワークロードをデプロイする
- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する
- REL11-BP03 すべてのレイヤーの修復を自動化する
- REL12-BP05 カオスエンジニアリングを使用して回復力をテストする

関連するドキュメント:

- AWS Well Architected フレームワークの信頼性の柱: 可用性
- AWS Service Quotas (旧称は「サービスの制限」)
- AWS Trusted Advisor ベストプラクティスチェックリスト (「サービスの制限」セクションを参照)
- AWS Answers の AWS Limit Monitor
- Amazon EC2 サービスの制限
- Service Quotas とは
- クォータの引き上げのリクエスト方法
- サービスエンドポイントとクォータ

- Service Quotas ユーザーガイド
- AWS のクォータモニター
- AWS Fault Isolation Boundaries (AWS の障害分離境界)
- Availability with redundancy (冗長性を備えた可用性)
- データのための AWS
- 継続的インテグレーションとは
- 継続的デリバリーとは
- APN パートナー: 設定管理を支援できるパートナー
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS(AWS のテナント別アカウント SaaS 環境でアカウントのライフサイクルを管理する)
- Managing and monitoring API throttling in your workloads (ワークロードの API スロットリングの管理とモニタリング)
- View AWS Trusted Advisor recommendations at scale with AWS Organizations(AWS Organizations で AWS Trusted Advisor の推奨事項を大規模に表示する)
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower(AWS Control Tower でサービス制限の緩和とエンタープライズサポートを自動化する)
- Actions, Resources, and Condition Keys for Service Quotas Services (AWS サービスのアクション、リソース、および条件キー)

関連動画:

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas (Service Quotas を使用して AWS のサービスのクォータを表示および管理する)
- AWS IAM Quotas Demo (AWS IAM のクォータのデモ)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small (AWS re:Invent 2018: ループをクローズして柔軟に対応: サイズを問わず、システムを制御する方法)

関連ツール:

- AWS CodeDeploy
- AWS CloudTrail

- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP04 クォータをモニタリングおよび管理する

予想される使用量を評価し、クォータを必要に応じて引き上げて、使用量を予定通り増やせるようにします。

期待される成果: 管理およびモニタリングするアクティブで自動化されたシステムが導入されます。このような運用ソリューションを採用すると、使用量がクォータのしきい値に近づいているかどうかを確認することができます。クォータの変更が要請されると、これらは積極的に修正されます。

一般的なアンチパターン:

- サービスクォータのしきい値をチェックするようにモニタリングを設定していない。
- ハード制限の値は変更できないにもかかわらず、モニタリングを設定していない。
- ソフトクォータの変更を要請して確保するのに必要な時間は即時または短期間であると想定している。
- サービスクォータに近づいた際のアラームは設定していても、アラートの対応方法に関するプロセスがない。
- AWS Service Quotas でサポートされているサービスのアラームのみを設定し、その他の AWS サービスのモニタリングを行っていない。
- 「アクティブ/アクティブ」、「アクティブ/パッシブ – ホット」、「アクティブ/パッシブ – コールド」、「アクティブ/パッシブ – パイロットライト」のアプローチなど、複数リージョンの回復力ある設計のクォータ管理を考慮していない。
- リージョン間のクォータの違いを評価していない。
- 特定のクォータ引き上げ要請について、すべてのリージョンのニーズを評価していない。

- マルチリージョンクォータ管理向けのテンプレートを活用していない。

このベストプラクティスを活用するメリット: AWS Service Quotas を自動的に追跡し、これらのクォータに対する使用状況をモニタリングすることで、クォータ制限に近づいていることを確認できます。このモニタリングデータを使用して、クォータの枯渇による低下を制限することもできます。

このベストプラクティスを確立しない場合のリスクレベル: 中

実装のガイダンス

サポートされているサービスについては、評価してアラートまたはアラームを送信できるさまざまなサービスを設定することで、クォータをモニタリングできます。これにより、使用状況のモニタリングがサポートされ、クォータに近づいていることのアラートが送信されます。このアラームは、AWS Config、Lambda 関数、Amazon CloudWatch、または AWS Trusted Advisor からトリガーできます。また、CloudWatch Logs のメトリクスフィルターを使用して、ログのパターンを検索して抽出し、使用量がクォータのしきい値に近づいているかどうかを判断することもできます。

実装手順

モニタリング:

- 現在のリソース消費 (バケットやインスタンスなど) を把握します。Amazon EC2 `DescribeInstances` などの API オペレーションを使用して、現在のリソース消費の情報を収集します。
- 以下を使用して、サービスに不可欠で適用できる現在のクォータをキャプチャします。
  - AWS Service Quotas
  - AWS Trusted Advisor
  - AWS ドキュメントを
  - AWS サービス固有のページ
  - AWS Command Line Interface (AWS CLI)
  - AWS Cloud Development Kit (AWS CDK)
- AWS Service Quotas を使用します。これは、250 を超える AWS のサービスのクォータを一元的に管理するのに役立つ AWS のサービスです。
- さまざまなしきい値で現在のサービス制限をモニタリングするには、Trusted Advisor サービス制限を使用します。
- リージョンの使用状況の増加をチェックするには、サービスクォータ履歴 (コンソールまたは AWS CLI) を使用します。

- 各リージョンと各アカウントのサービスクォータの変化を比較して、必要に応じて同等性を確立します。

管理:

- 自動化: AWS Config カスタムルールを設定し、リージョン間でサービスクォータをスキャンして、違いを比較します。
- 自動化: リージョン全体にわたるサービスクォータをスキャンするスケジュールされた Lambda 関数を設定して、違いを比較します。
- 手動: リージョン全体にわたるサービスクォータをスキャンするために、AWS CLI、API、または AWS コンソールを介してサービスクォータをスキャンして、違いを比較します。違いについてのレポートを作成します。
- リージョン間でクォータの違いが確認された場合は、必要に応じてクォータの変更を要請します。
- すべての変更の結果を確認します。

リソース

関連するベストプラクティス:

- REL01-BP01 サービスクォータと制約を認識する
- REL01-BP02 アカウントおよびリージョンをまたいでサービスクォータを管理する
- REL01-BP03 アーキテクチャを通じて、固定サービスクォータと制約に対応する
- REL01-BP05 クォータ管理を自動化する
- REL01-BP06 フェイルオーバーに対応するために、現在のクォータと最大使用量の間に十分なギャップがあることを確認する
- REL03-BP01 ワークロードをセグメント化する方法を選択する
- REL10-BP01 複数の場所にワークロードをデプロイする
- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する
- REL11-BP03 すべてのレイヤーの修復を自動化する
- REL12-BP05 カオスエンジニアリングを使用して回復力をテストする

関連するドキュメント:

- AWS Well Architected フレームワークの信頼性の柱: 可用性

- AWS Service Quotas (旧称は「サービスの制限」)
- AWS Trusted Advisor ベストプラクティスチェックリスト (「サービスの制限」セクションを参照)
- AWS Answers の AWS Limit Monitor
- Amazon EC2 サービスの制限
- Service Quotas とは
- クォータの引き上げのリクエスト方法
- サービスエンドポイントとクォータ
- Service Quotas ユーザーガイド
- AWS のクォータモニター
- AWS Fault Isolation Boundaries (AWS の障害分離境界)
- Availability with redundancy (冗長性を備えた可用性)
- データのための AWS
- 継続的インテグレーションとは
- 継続的デリバリーとは
- APN パートナー: 設定管理を支援できるパートナー
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS(AWS のテナント別アカウント SaaS 環境でアカウントのライフサイクルを管理する)
- Managing and monitoring API throttling in your workloads (ワークロードの API スロットリングの管理とモニタリング)
- View AWS Trusted Advisor recommendations at scale with AWS Organizations(AWS Organizations で AWS Trusted Advisor の推奨事項を大規模に表示する)
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower(AWS Control Tower でサービス制限の緩和とエンタープライズサポートを自動化する)
- Actions, Resources, and Condition Keys for Service Quotas Services (AWS サービスのアクション、リソース、および条件キー)

関連動画:

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas (Service Quotas を使用して AWS のサービスのクォータを表示および管理する)
- AWS IAM Quotas Demo (AWS IAM のクォータのデモ)

- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small (AWS re:Invent 2018: ループをクローズして柔軟に対応: サイズを問わず、システムを制御する方法)

関連ツール:

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP05 クォータ管理を自動化する

しきい値に近づいたときに警告するツールを実装します。AWS Service Quotas API を使用すると、クォータの引き上げリクエストを自動化できます。

お使いの Configuration Management Database (CMDB) またはチケット発行システムを Service Quotas と統合すると、クォータの引き上げリクエストと現在のクォータに関する情報のトラッキングを自動化できます。AWS SDK のほかに、Service Quotas も AWS Command Line Interface (AWS CLI) を使用した自動化を提供しています。

一般的なアンチパターン:

- スプレッドシートでクォータと使用状況を追跡する。
- 毎日、毎週、または毎月の使用状況に関するレポートを実行し、使用量とクォータを比較する。

このベストプラクティスを活用するメリット: AWS のサービスクォータの自動追跡と、そのクォータに対する使用量のモニタリングにより、クォータに近づいていることを確認できます。必要に応じてクォータの引き上げをリクエストできるように、オートメーションを設定できます。使用量の傾向

が反対の方向にある場合は、(認証情報が漏えいした場合の) リスクの低下とコスト削減のメリットを実現するために、クォータの削減を検討することをお勧めします。

このベストプラクティスを活用しない場合のリスクレベル: ミディアム

実装のガイダンス

- 自動モニタリングをセットアップする しきい値に近づいたときにアラートを発行するため、SDK を使用するツールを実装します。
  - Service Quotas を使用し、AWS Limit Monitor や AWS Marketplace からのサービスなど、自動クォータモニタリングソリューションでサービスを補強します。
    - Service Quotas とは何ですか?
    - AWS でのクォータモニタリング - AWS ソリューション
  - Amazon SNS および AWS Service Quotas API を使用して、クォータしきい値に基づいてトリガーされるレスポンスをセットアップします。
  - 自動化をテストします。
    - 制限のしきい値を設定します。
    - AWS Config、デプロイパイプライン、Amazon EventBridge、またはサードパーティーからの変更イベントと統合します。
    - 応答をテストするために、人為的に低いクォータしきい値を設定します。
    - 通知に対して適切なアクションを取り、必要に応じて AWS Support に問い合わせるトリガーをセットアップします。
    - 変更イベントを手動でトリガーします。
    - ゲームデーを実行して、クォータ引き上げの変更プロセスをテストします。

リソース

関連するドキュメント:

- APN パートナー: 設定管理を支援できるパートナー
- AWS Marketplace: 制限の追跡に役立つ CMDB 製品
- AWS Service Quotas (旧称は「サービスの制限」)
- AWS Trusted Advisor ベストプラクティスのチェック (「サービスの制限」セクションを参照)
- AWS でのクォータモニタリング - AWS ソリューション
- Amazon EC2 サービスの制限

- Service Quotas とは何ですか?

関連動画:

- AWS Live re:Inforce 2019 - Service Quotas

REL01-BP06 フェイルオーバーに対応するために、現在のクォータと最大使用量の間に十分なギャップがあることを確認する

リソースに障害が発生するか、アクセスできない場合、リソースが正常に終了されるまで、クォータにカウントされることがあります。障害が発生したリソースまたはアクセスできないリソースと、代替のリソースの合計リソース数がクォータ内に収まることを確認します。このギャップを算出する際は、ネットワーク障害、アベイラビリティーゾーンの不具合、またはリージョン規模の不具合などのユースケースを考慮する必要があります。

期待される成果: 現在のサービスのしきい値を使用して、規模を問わず、リソースやリソースへのアクセスで障害に対応できます。リソースプランニングでは、ゾーン障害、ネットワーク障害、さらにはリージョン規模の不具合が考慮されています。

一般的なアンチパターン:

- フェイルオーバーシナリオを考慮せずに、現在のニーズに基づいてサービスクォータを設定する。
- ピーク時のサービスクォータの計算時に静的安定性の原則を考慮しない。
- 各リージョンに求められる合計クォータを計算する際に、アクセスできないリソースの可能性を考慮しない。
- AWS サービス障害分離境界と、予測される異常な使用パターンを考慮していないサービスがある。

このベストプラクティスを活用するメリット: サービス中断イベントがアプリケーションの可用性に影響を与えるような場合、クラウドを使用すると、このようなイベントを軽減または復旧するための戦略を実装できます。そのような戦略には、多くの場合、障害が発生したリソース、またはアクセスできないリソースに置き換わる追加リソースの作成が含まれます。このようなフェイルオーバーの状況に対応し、サービス制限の枯渇による追加の低下を発生させないようなクォータ戦略を策定します。

このベストプラクティスを確立しない場合のリスクレベル: 中

実装のガイダンス

クォータ制限を評価する際は、何らかの劣化が原因で発生する可能性のあるフェイルオーバーのケースを検討します。以下のタイプのフェイルオーバーのケースを検討する必要があります。

- 中断したり、アクセスできない VPC。
- アクセスできないサブネット。
- 多くのリソースへのアクセスに影響を及ぼすレベルまで低下したアベイラビリティーゾーン。
- まざまなネットワークルートまたは受信ポイントと送信ポイントがブロックされたり、変更されたりしている。
- 多くのリソースへのアクセスに影響を及ぼすレベルまで低下したリージョン。
- 複数のリソースがあるが、すべてのリソースがリージョンやアベイラビリティーゾーンの不具合の影響を受けているわけではない。

上記のリストのような障害は、フェイルオーバーイベントを開始するトリガーになる可能性があります。ビジネスへの影響は大きく異なる可能性があり、フェイルオーバーの決定は状況や顧客ごとに異なります。ただし、アプリケーションまたはサービスのフェイルオーバーという運用上の決定を下す場合、イベントが発生する前に、フェイルオーバーの口ケーションでのリソースのキャパシティプランニングと、それに関連するクォータについて対処する必要があります。

発生する可能性がある通常のピーク時よりも高いピークを考慮に入れて、各サービスのクォータを確認します。このようなピークは、ネットワークまたはアクセス許可のために到達でき、まだアクティブなリソースに関連している可能性があります。終了していないアクティブなリソースは、サービスクォータ制限に対して引き続きカウントされます。

実装手順

- フェイルオーバーまたはアクセスできなくなった場合に対応するため、サービスクォータと最大使用量の間に十分なギャップがあることを確認します。
- デプロイのパターン、可用性の要件、消費の増加を考慮して、サービスクォータを決定します。
- 必要に応じてクォータの引き上げをリクエストします。クォータの引き上げリクエストの実行に必要な時間を計画します。
- 信頼性の要件 (「9 の数」としても知られる) を決定します。
- 障害シナリオ (コンポーネント、アベイラビリティーゾーン、リージョンの損失など) を確立します。

- デプロイ手法 (例えば、Canary、ブルー/グリーン、レッド/ブラック、ローリングなど) を確立します。
- 現在の制限に適切なバッファ (例えば、15%) を含めます。
- 必要に応じて、静的安定性 (ゾーンおよびリージョン) の計算を含めます。
- 消費の増加を計画します (例えば、消費の傾向をモニタリングする)。
- 最も重要なワークロードへの静的安定性の影響を考慮します。すべてのリージョンとアベイラビリティーゾーンで静的に安定性のあるシステムに相当するリソースを評価します。
- オンデマンドキャパシティ予約を使用して、フェイルオーバーが発生する前に容量をスケジュールすることを検討します。これは、フェイルオーバー発生時、最も重要なビジネススケジュール中に、適切な量および種類のリソースを取得するうえで予測できるリスクの軽減に役立つ戦略です。

リソース

関連するベストプラクティス:

- REL01-BP01 サービスクォータと制約を認識する
- REL01-BP02 アカウントおよびリージョンをまたいでサービスクォータを管理する
- REL01-BP03 アーキテクチャを通じて、固定サービスクォータと制約に対応する
- REL01-BP04 クォータをモニタリングおよび管理する
- REL01-BP05 クォータ管理を自動化する
- REL03-BP01 ワークロードをセグメント化する方法を選択する
- REL10-BP01 複数の場所にワークロードをデプロイする
- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する
- REL11-BP03 すべてのレイヤーの修復を自動化する
- REL12-BP05 カオスエンジニアリングを使用して回復力をテストする

関連するドキュメント:

- AWS Well Architected フレームワークの信頼性の柱: 可用性
- AWS Service Quotas (旧称は「サービスの制限」)
- AWS Trusted Advisor ベストプラクティスチェックリスト (「サービスの制限」セクションを参照)
- AWS Answers の AWS Limit Monitor
- Amazon EC2 サービスの制限

- Service Quotas とは
- クォータの引き上げのリクエスト方法
- サービスエンドポイントとクォータ
- Service Quotas ユーザーガイド
- AWS のクォータモニター
- AWS Fault Isolation Boundaries (AWS の障害分離境界)
- Availability with redundancy (冗長性を備えた可用性)
- データのための AWS
- 継続的インテグレーションとは
- 継続的デリバリーとは
- APN パートナー: 設定管理を支援できるパートナー
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS(AWS のテナント別アカウント SaaS 環境でアカウントのライフサイクルを管理する)
- Managing and monitoring API throttling in your workloads (ワークロードの API スロットリングの管理とモニタリング)
- View AWS Trusted Advisor recommendations at scale with AWS Organizations(AWS Organizations で AWS Trusted Advisor の推奨事項を大規模に表示する)
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower(AWS Control Tower でサービス制限の緩和とエンタープライズサポートを自動化する)
- Actions, Resources, and Condition Keys for Service Quotas Services (AWS サービスのアクション、リソース、および条件キー)

関連動画:

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas (Service Quotas を使用して AWS のサービスのクォータを表示および管理する)
- AWS IAM Quotas Demo (AWS IAM のクォータのデモ)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small (AWS re:Invent 2018: ループをクローズして柔軟に対応: サイズを問わず、システムを制御する方法)

関連ツール:

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

ネットワークトポロジを計画する

多くの場合、ワークロードは複数の環境に存在します。このような環境には、複数のクラウド環境 (パブリックにアクセス可能なクラウド環境とプライベートの両方) と既存のデータセンターインフラストラクチャなどがあります。計画する際は、システム内およびシステム間の接続、パブリック IP アドレスの管理、プライベート IP アドレスの管理、ドメイン名解決といったネットワークに関する項目も考慮に含めなければなりません。

IP アドレスベースのネットワークを使用するシステムを構築する際は、予想される障害を見越してネットワークトポロジとアドレス指定を考慮し、さらに将来の成長と他のシステムやネットワークと統合できる余地を残しておくように計画する必要があります。

Amazon Virtual Private Cloud (Amazon VPC) では、AWS クラウドのプライベートで隔離されたセクションをプロビジョニングでき、仮想ネットワークで AWS リソースを起動できます。

ベストプラクティス

- REL02-BP01 ワークロードのパブリックエンドポイントに高可用性ネットワーク接続を使用する
- REL02-BP02 クラウド環境とオンプレミス環境のプライベートネットワーク間の冗長接続をプロビジョニングする
- REL02-BP03 拡張性と可用性を考慮した IP サブネットの割り当てを確実に行う
- REL02-BP04 多対多メッシュよりもハブアンドスポークトポロジを優先する
- REL02-BP05 接続されているすべてのプライベートアドレス空間において、重複しないプライベート IP アドレス範囲を指定する

REL02-BP01 ワークロードのパブリックエンドポイントに高可用性ネットワーク接続を使用する

ワークロードのパブリックエンドポイントに高可用性ネットワーク接続を構築すると、接続の喪失によるダウンタイムを低減し、ワークロードの可用性と SLA を向上できます。これを実現するには、可用性の高い DNS、コンテンツ配信ネットワーク、API ゲートウェイ、負荷分散、またはリバースプロキシを使用します。

期待される成果: パブリックエンドポイントの高可用性ネットワーク接続を計画、構築、運用化することが重要です。接続が失われたためにワークロードにアクセスできなくなった場合、ワークロードが実行中で利用可能であっても、顧客はシステムがダウンしていると見なします。ワークロードのパブリックエンドポイントに対する可用性と回復力に優れたネットワーク接続と、ワークロード自体の回復力のあるアーキテクチャを組み合わせることで、可能な限り最高レベルの可用性とサービスレベルを顧客に提供できます。

AWS Global Accelerator、Amazon CloudFront、Amazon API Gateway、AWS Lambda 関数 URL、AWS AppSync API、Elastic Load Balancing (ELB) はすべて、高可用性のパブリックエンドポイントを提供します。Amazon Route 53 は、ドメイン名解決のための高可用性 DNS サービスを提供し、パブリックエンドポイントアドレスを解決できることを確認できます。

また、ロードバランシングとプロキシ処理のために、AWS Marketplace のソフトウェアアプライアンスを評価することもできます。

一般的なアンチパターン:

- 高可用性に向けて DNS とネットワーク接続を計画せず、高可用性ワークロードを設計する。
- 個別のインスタンスまたはコンテナでパブリックインターネットアドレスを使用し、DNS 経由でそれらのアドレスへの接続を管理する。
- サービスの検索に、ドメイン名ではなく、IP アドレスを使用する。
- パブリックエンドポイントへの接続が失われるシナリオをテストしていない。
- ネットワークスループットのニーズと分散パターンを分析していない。
- ワークロードのパブリックエンドポイントへのインターネットにおけるネットワーク接続が中断される可能性を考慮したシナリオをテストおよび計画していない。
- 大規模な地理的領域にコンテンツ (ウェブページ、静的アセット、またはメディアファイル) を提供し、コンテンツ配信ネットワークを使用しない。
- 分散サービス拒否 (DDoS) 攻撃に備えた計画をしていない。DDoS 攻撃は、正当なトラフィックを遮断し、ユーザーの可用性を低下させるリスクがあります。

このベストプラクティスを活用するメリット: 可用性と回復力に優れたネットワーク接続を設計することで、ユーザーはワークロードにアクセスして利用できます。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

パブリックエンドポイントへの高可用性ネットワーク接続を構築する際の中核となるのが、トラフィックのルーティングです。トラフィックがエンドポイントに到達できることを確認するには、ドメイン名を DNS が対応する IP アドレスに解決することができる必要があります。ドメインの DNS レコードを管理するには、Amazon Route 53 などの高可用性かつスケーラブルなドメインネームシステム (DNS) を使用します。Amazon Route 53 が提供するヘルスチェックも利用できます。ヘルスチェックは、アプリケーションが到達可能かつ利用可能で、機能していることを確認します。ヘルスチェックは、ウェブページや特定の URL を要求するなどのユーザーの動作を模倣するように設定できます。障害が発生した場合、Amazon Route 53 は DNS 解決リクエストに応答し、トラフィックを正常なエンドポイントのみに転送します。Amazon Route 53 が提供する位置情報 DNS およびレイテンシーベースルーティング機能の使用を検討することもできます。

ワークロード自体の可用性が高いことを確認するには、Elastic Load Balancing (ELB) を使用します。Amazon Route 53 は、トラフィックを ELB にターゲットとするために使用できます。ELB は、このトラフィックを、ターゲットのコンピューティングインスタンスに分散します。サーバーレスソリューションには、Amazon API Gateway と AWS Lambda を組み合わせて使用できます。また、複数の AWS リージョン でワークロードを実行することもできます。マルチサイトのアクティブ/アクティブパターンを使用すると、ワークロードは複数のリージョンからのトラフィックを処理できます。マルチサイトのアクティブ/パッシブパターンの場合、ワークロードはアクティブリージョンからのトラフィックを処理し、データはセカンダリリージョンにレプリケートされ、プライマリリージョンで障害が発生した場合にアクティブになります。その後、Route 53 のヘルスチェックを使用して、プライマリリージョンの任意のエンドポイントからセカンダリリージョンのエンドポイントへの DNS フェイルオーバーを制御して、ワークロードが到達可能であり、ユーザーが利用できることを確認することができます。

Amazon CloudFront は、世界中のエッジロケーションのネットワークを使用してリクエストを処理することにより、低レイテンシーかつ高データ転送速度でコンテンツを配信する、シンプルな API を提供します。コンテンツ配信ネットワークは、ユーザーの所在地に近いロケーションにあるか、近いロケーションでキャッシュされているコンテンツを提供することで、顧客にサービスを提供します。これにより、コンテンツの負荷がサーバーから CloudFront のエッジロケーションへと移動するため、アプリケーションの可用性も向上します。エッジロケーションとリージョンのエッジキャッシュは、視聴者の近くにコンテンツのキャッシュコピーを保持するため、ワークロードの取得が迅速化し、到達可能性と可用性が向上します。

ユーザーが地理的に分散しているワークロードの場合、AWS Global Accelerator を使用すると、アプリケーションの可用性とパフォーマンスを向上できます。AWS Global Accelerator は、1 つまたは複数の AWS リージョン でホストされているアプリケーションへの固定エントリポイントとして機能するエニーキャスト静的 IP アドレスを提供します。これにより、トラフィックがユーザーにできるだけ近い AWS グローバルネットワークに入ることができ、ワークロードの到達可能性と可用性が向上します。AWS Global Accelerator を使用すると、TCP、HTTP、および HTTPS ヘルスチェックを使用して、アプリケーションエンドポイントの健全性もモニタリングできます。エンドポイントの正常性または設定に変更が生じると、正常なエンドポイントへのユーザートラフィックのリダイレクトがトリガーされ、最高のパフォーマンスと可用性がユーザーに提供されます。さらに、AWS Global Accelerator は障害を分離するように設計されており、独立したネットワークゾーンによって提供される 2 つの静的 IPv4 アドレスを使用して、アプリケーションの可用性を向上します。

DDoS 攻撃からの保護として、AWS は、AWS Shield Standard を提供しています。Shield Standard は自動的に有効にされており、SYN/UDP フラッド攻撃やリフレクション攻撃などの一般的なインフラストラクチャ (レイヤー 3 および 4) 攻撃から保護し、AWS 上のアプリケーションの高可用性をサポートします。より高度で大規模な攻撃 (UDP フラッド攻撃など)、State-Exhaustion 攻撃 (TCP SYN フラッドなど) に対する追加の保護、および Amazon Elastic Compute Cloud (Amazon EC2)、Elastic Load Balancing (ELB)、Amazon CloudFront、AWS Global Accelerator、Route 53 上で実行されるアプリケーションの保護には、AWS Shield Advanced の使用を検討できます。HTTP POST や GET フラッド攻撃などのアプリケーションレイヤー攻撃からの保護には、AWS WAF を使用します。AWS WAF を使用すると、IP アドレス、HTTP ヘッダー、HTTP ボディ、URI 文字列、SQL インジェクション、クロスサイトスクリプティング条件を使用して、リクエストをブロックするか許可するかを決定できます。

実装手順

1. 高可用性の DNS の設定: Amazon Route 53 は、高可用性かつスケーラブルなドメインネームシステム (DNS) ウェブサービスです。Route 53 は、AWS またはオンプレミスで実行されるインターネットアプリケーションにユーザーリクエストを接続します。詳細については、「DNS サービスとしての Amazon Route 53 の設定」を参照してください。
2. ヘルスチェックの設定: Route 53 を使用する場合、正常なターゲットのみが解決可能であることを確認します。Route 53 ヘルスチェックの作成と DNS フェイルオーバーの設定から始めます。ヘルスチェックを設定する際には、以下を考慮することが重要です。
   a. Amazon Route 53 でヘルスチェックの正常性を判断する方法
   b. ヘルスチェックの作成、更新、削除
   c. ヘルスチェックステータスのモニタリングと通知の受信
   d. Amazon Route 53 DNS のベストプラクティス

3. DNS サービスをエンドポイントに接続します。
    a. Elastic Load Balancing をトラフィックのターゲットとして使用する場合、Amazon Route 53 を使用してロードバランサーのリージョンエンドポイントを指すエイリアスレコードを作成します。エイリアスレコードの作成中に、[Evaluate Target Health (ターゲットの正常性の評価)] オプションを [Yes (あり)] に設定します。
    b. サーバーレスワークロードまたはプライベート API については、API Gateway を使用する場合、Route 53 を使用してトラフィックを API Gateway にルーティングします。
4. コンテンツ配信ネットワークを決定します。
    a. ユーザーに近いエッジロケーションを使用してコンテンツを配信するには、CloudFront がコンテンツを配信する方法を理解することから始めます。
    b. 簡単な CloudFront ディストリビューションの使用から始めます。これにより、CloudFront は、コンテンツの配信元と、コンテンツ配信の追跡および管理方法に関する詳細を認識できます。CloudFront のディストリビューションを設定する際には、以下の側面を理解して、考慮することが重要です。
        i. CloudFront エッジロケーションでのキャッシュの仕組み
        ii. CloudFront キャッシュから直接提供されるリクエストの比率 (キャッシュヒット率) の向上
        iii. Amazon CloudFront Origin Shield の使用
        iv. CloudFront オリジンフェイルオーバーによる高可用性の最適化
5. アプリケーションレイヤー保護の設定: AWS WAF は、可用性低下、セキュリティの侵害、リソースの過剰消費といった、一般的なウェブのエクスプロイトやボットから保護します。詳細を理解するには、「AWS WAF の仕組み」を確認し、アプリケーションレイヤーの HTTP POST および GET フラッド攻撃からの保護を実装する準備が整ったら、「AWS WAF の開始方法」を確認してください。AWS WAF は CloudFront と組み合わせて使用することもできます。ドキュメントについては、「AWS WAF と Amazon CloudFront の機能との連携」を参照してください。
6. 追加の DDoS 保護の設定: デフォルトでは、すべての AWS のお客様が追加料金なしで、ウェブサイトやアプリケーションをターゲットする一般的で最も頻繁に発生するネットワーク層およびトランスポート層の DDoS 攻撃に対する AWS Shield Standard を使用した保護を受けています。Amazon EC2、Elastic Load Balancing、Amazon CloudFront、AWS Global Accelerator、Amazon Route 53 で実行されているインターネット接続アプリケーションの保護を強化するには、AWS Shield Advanced を検討し、「DDoS に対する回復力が高いアーキテクチャの例」を確認してください。ワークロードとパブリックエンドポイントを DDoS 攻撃から保護するには、「AWS Shield Advanced の開始方法」を確認してください。

リソース

関連するベストプラクティス:

- REL10-BP01 複数の場所にワークロードをデプロイする
- REL10-BP02 マルチロケーションデプロイ用の適切な場所の選択
- REL11-BP04 復旧中はコントロールプレーンではなくデータプレーンを利用する
- REL11-BP06 イベントが可用性に影響する場合に通知を送信する

関連するドキュメント:

- APN パートナー: ネットワークの計画を支援できるパートナー
- AWS Marketplace for Network Infrastructure (ネットワークインフラストラクチャ向け AWS Marketplace)
- What Is AWS Global Accelerator? (AWS Global Accelerator とは)
- Amazon CloudFront とは何ですか?
- Amazon Route 53 とは ?
- Elastic Load Balancing とは?
- Network Connectivity capability - Establishing Your Cloud Foundations (ネットワーク接続機能 - クラウド基盤の確立)
- Amazon API Gateway とは何ですか?
- AWS WAF、AWS Shield、AWS Firewall Manager とは
- What is Amazon Route 53 Application Recovery Controller? (Amazon Route 53 Application Recovery Controller とは)
- Amazon Route 53 ヘルスチェックの作成と DNS フェイルオーバーの設定

関連動画:

- AWS re:Invent 2022 - Improve performance and availability with AWS Global Accelerator(AWS re:Invent 2022 - AWS Global Accelerator を使用してパフォーマンスと可用性を向上する)
- AWS re:Invent 2020: Global traffic management with Amazon Route 53 (AWS re:Invent 2020: Amazon Route 53 を使用したグローバルトラフィック管理)
- AWS re:Invent 2022 - Operating highly available Multi-AZ applications (AWS re:Invent 2022 - 高可用性のマルチ AZ アプリケーションの運用)

- AWS re:Invent 2022 - Dive deep on AWS networking infrastructure (AWS re:Invent 2022 - AWS ネットワークインフラストラクチャの詳細)
- AWS re:Invent 2022 - Building resilient networks (AWS re:Invent 2022 - 回復力のあるネットワークの構築)

関連する例:

- Disaster Recovery with Amazon Route 53 Application Recovery Controller (ARC) (Amazon Route 53 Application Recovery Controller (ARC) を使用したディザスタリカバリ)
- Reliability Workshops (信頼性ワークショップ)
- AWS Global Accelerator ワークショップ

REL02-BP02 クラウド環境とオンプレミス環境のプライベートネットワーク間の冗長接続をプロビジョニングする

クラウド環境とオンプレミス環境のプライベートネットワーク間の接続に冗長性を実装して、接続の耐障害性を実現します。これは、ネットワーク障害が発生した場合でも接続を維持できるように、2 つ以上のリンクとトラフィックパスをデプロイすることで可能になります。

一般的なアンチパターン:

- 単一のネットワーク接続に依存することで、単一障害点が発生する。
- 1 つの VPN トンネルのみを使用するか、同じアベイラビリティーゾーンで終端する複数のトンネルを使用する。
- 1 社の ISP に VPN 接続を依存することで、ISP にアウテージが発生すると完全なネットワーク障害につながる可能性がある。
- ネットワーク中断時のトラフィック再ルーティングに不可欠な BGP などの動的ルーティングプロトコルを実装していない。
- VPN トンネルの帯域幅制限を無視し、バックアップ機能を過大評価する。

このベストプラクティスを確立するメリット: クラウド環境と企業/オンプレミス環境の間に冗長接続を実装することで、2 つの環境間で依存するサービスの安定した通信が可能になります。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

AWS Direct Connect を使用してオンプレミスネットワークを AWS に接続する場合、複数のオンプレミスのロケーションと複数の AWS Direct Connect ロケーションにある個別のデバイスで終端する個別の接続を使用することで、ネットワークの耐障害性を最大限に高める (99.99% の SLA) ことができます。このトポロジーは、デバイス障害、接続問題、およびロケーション全体での障害に対する耐障害性を提供します。または、複数のロケーション (各オンプレミスのロケーションを 1 つの Direct Connect ロケーションに接続) への 2 つの個別の接続を使用することで、高い耐障害性 (99.9% の SLA) を実現できます。このアプローチにより、ファイバーの切断やデバイスの障害による接続の中断を防ぎ、ロケーション全体での障害を軽減できます。AWS Direct Connect Resiliency Toolkit は、AWS Direct Connect トポロジーの設計に役立ちます。

また、プライマリ AWS Direct Connect 接続に対する費用対効果の高いバックアップとして、AWS Transit Gateway で終端する AWS Site-to-Site VPN を検討することもできます。この設定により、複数の VPN トンネルでの等コストマルチパス (ECMP) ルーティングが可能になり、各 VPN トンネルの上限が 1.25 Gbps であっても、最大 50 Gbps のスループットが可能になります。ただし、ネットワークの中断を最小限に抑え、安定した接続を提供するためには、AWS Direct Connect が依然として最も効果的な選択肢であることに注意してください。

インターネット経由で VPN を使用してクラウド環境をオンプレミスのデータセンターに接続する場合は、単一のサイト間 VPN 接続の一部として 2 つの VPN トンネルを設定します。高可用性を実現するため、各トンネルが異なるアベイラビリティーゾーンで終端するようにし、オンプレミスデバイスの障害を防ぐために冗長ハードウェアを使用する必要があります。さらに、1 社のインターネットサービスプロバイダー (ISP) の停止によって VPN 接続が完全に中断してしまうことを防ぐため、オンプレミスのロケーションで異なる ISP による複数のインターネット接続の利用を検討してください。ルーティングとインフラストラクチャが多様な ISP、特に AWS エンドポイントに複数の物理パスがある ISP を選択すると、高い接続可用性が得られます。

複数の AWS Direct Connect 接続と複数の VPN トンネル (または両方の組み合わせ) による物理的な冗長性に加え、ボーダーゲートウェイプロトコル (BGP) の動的ルーティングを実装することも重要です。動的 BGP は、リアルタイムのネットワーク状態と設定されているポリシーに基づいて、1 つのパスから別のパスにトラフィックを自動的に再ルーティングします。この動的な動作は、リンクまたはネットワーク障害の発生時に、ネットワークの可用性とサービスの継続性を維持するうえで特に役立ちます。代替パスが瞬時に選択されるため、ネットワークの耐障害性と信頼性が高まります。

実装手順

- AWS とオンプレミス環境間の可用性の高い接続を確保します。

    - 個別にデプロイされたプライベートネットワーク間で複数の AWS Direct Connect 接続または VPN トンネルを使用します。
    - 複数の AWS Direct Connect ロケーションを使用して、高可用性を確保します。
    - 複数の AWS リージョン を使用している場合は、少なくとも 2 つのリージョンで冗長性を確保します。
- 可能な場合は AWS Transit Gateway を使用して、VPN 接続を終端します。
- VPN を終端する、または SD-WAN を AWS に拡張するための AWS Marketplace アプライアンスを評価します。AWS Marketplace アプライアンスを使用する場合は、さまざまなアベイラビリティーゾーンで高可用性を実現するために冗長インスタンスをデプロイします。
- オンプレミス環境への冗長接続を確保します。
    - 可用性のニーズを満たすため、複数の AWS リージョン への冗長接続が必要な場合があります。
    - AWS Direct Connect Resiliency Toolkit を使用して開始します。

リソース

関連するドキュメント:

- AWS Direct Connect の回復性に関する推奨事項
- Using Redundant Site-to-Site VPN Connections to Provide Failover
- ルーティングポリシーと BGP コミュニティ
- Active/Active and Active/Passive Configurations in AWS Direct Connect
- APN パートナー: ネットワークの計画を支援できるパートナー
- AWS Marketplace for Network Infrastructure (ネットワークインフラストラクチャ向け AWS Marketplace)
- Amazon Virtual Private Cloud Connectivity Options Whitepaper
- Building a Scalable and Secure Multi-VPC AWS Network Infrastructure (スケーラブルで安全なマルチ VPC AWS ネットワークインフラストラクチャの構築)
- Using redundant Site-to-Site VPN connections to provide failover
- Using the AWS Direct Connect Resiliency Toolkit to get started
- VPC Endpoints and VPC Endpoint Services (AWS PrivateLink)
- What Is Amazon VPC?

- What is a transit gateway?
- What is AWS Site-to-Site VPN?
- Working with Direct Connect gateways

関連動画:

- AWS re:Invent 2018: Advanced VPC Design and New Capabilities for Amazon VPC
- AWS re:Invent 2019: AWS Transit Gateway reference architectures for many VPCs

REL02-BP03 拡張性と可用性を考慮した IP サブネットの割り当てを確実に行う

Amazon VPC IP アドレスの範囲は、将来の拡張や アベイラビリティゾーンをまたがるサブネットへの IP アドレスの割り当てを考慮し て、ワークロードの要件に対応するのに十分な大きさでなければなりません。これには、ロードバランサー、EC2 インスタンス、コンテナーベースのアプリケーションが含まれます。

ネットワークトポロジの計画は、IP アドレス空間の定義から始めます。プライベート IP アドレス範囲 (RFC 1918 ガイドラインに準拠) は、VPC ごとに割り当てる必要があります。このプロセスの一環として、次の要件を満たすようにします。

- リージョンごとに複数の VPC 用の IP アドレス空間を割り当てます。
- VPC 内で、複数のアベイラビリティーゾーンを網羅できるように、複数のサブネット用のスペースを確保します。
- 将来の拡張のために、未使用の CIDR ブロック空間を VPC 内に残しておくことを検討します。
- 機械学習用のスポットフリート、Amazon EMR クラスター、Amazon Redshift クラスターなど、使用する可能性のある Amazon EC2 インスタンスの一時的なフリートのニーズを満たす IP アドレス空間を確保します。各 Kubernetes ポッドにはデフォルトで VPC CIDR ブロックからルーティング可能なアドレスが割り当てられるため、Amazon Elastic Kubernetes Service (Amazon EKS) などの Kubernetes クラスターについても同様の考慮が必要です。
- 各サブネット CIDR ブロックの最初の 4 つの IP アドレスと最後の IP アドレスはリザーブドのため、お客様はご使用いただけません。
- VPC に割り当てられた最初の VPC CIDR ブロックは変更または削除することはできませんが、重複していない CIDR ブロックを VPC に追加することはできます。サブネット IPv4 CIDR は変更できませんが、IPv6 CIDR は変更できます。

- 利用可能な VPC CIDR ブロックの最大サイズは /16、最小サイズは /28 です。
- 他の接続ネットワーク (VPC、オンプレミス、その他のクラウドプロバイダー) を検討し、IP アドレス空間が重複しないようにしてください。詳細については、以下をご覧ください。 REL02-BP05 接続されているすべてのプライベートアドレス空間において、重複しないプライベート IP アドレス範囲を指定する

期待される成果: IP サブネットを拡張できるため、将来の成長に対応し、無駄を回避できます。

一般的なアンチパターン:

- 将来の成長が考慮されていないため、CIDR ブロックが小さすぎて再構成が必要になり、ダウンタイムが発生する可能性がある。
- Elastic Load Balancing が使用できる IP アドレスの数を不正確に見積もる。
- 多数の高トラフィックロードバランサーを同じサブネットにデプロイする。
- IP アドレスの消費をモニタリングできない状態で、自動スケーリングメカニズムを使用する。
- 将来の成長予測をはるかに超える過剰に大きな CIDR 範囲を定義したせいで、アドレス範囲が重複する他のネットワークとのピアリングが困難になる可能性がある。

このベストプラクティスを活用するメリット: これにより、ワークロードの増大に対応し、スケールアップ時に可用性を引き続き提供できます。

このベストプラクティスが確立されていない場合のリスクレベル: 中

実装のガイダンス

拡張、コンプライアンス、他のネットワークとの統合に対応できるようにネットワークを計画します。適切に計画しないと、拡張の見積もりが甘くなったり、規制コンプライアンスが変わったり、取得やプライベートネットワーク接続の設定が難しくなったりする場合があります。

- サービス要件、レイテンシー、規制、およびディザスタリカバリ (DR) 要件に基づいて、関連する AWS アカウント とリージョンを選択します。
- リージョン別 VPC デプロイのニーズを明確にします。
- VPC のサイズを明確にします。
  - マルチ VPC 接続をデプロイするかどうかを判断します。
    - Transit Gateway とは?
    - 単一リージョンの複数 VPC 接続

- 規制要件のためネットワークの分離が必要かどうかを判断します。
- 現在および将来のニーズに合わせて、適切なサイズの CIDR ブロックを持つ VPC を作成します。
  - 成長予測が不確かな場合は、将来の再構成のリスクを軽減するため、大きめの CIDR ブロックを選択しておいた方がよいでしょう。
- デュアルスタック VPC の一部として、サブネットに IPv6 アドレス指定 を使用することを検討してください。IPv6 は、IPv4 アドレスであれば大量に必要となるであろう、一時的なインスタンスやコンテナのフリートを含むプライベートサブネットでの使用に適しています。

リソース

関連する Well-Architected のベストプラクティス:

- REL02-BP05 接続されているすべてのプライベートアドレス空間において、重複しないプライベート IP アドレス範囲を指定する

関連するドキュメント:

- APN パートナー: ネットワークの計画を支援できるパートナー
- ネットワークインフラストラクチャ向け AWS Marketplace
- Amazon Virtual Private Cloud の接続オプションホワイトペーパー
- Multiple data center HA network connectivity
- 単一リージョンの複数 VPC 接続
- Amazon VPC とは?
- AWS の IPv6
- リファレンスアーキテクチャの IPv6
- Amazon Elastic Kubernetes Service launches IPv6 support

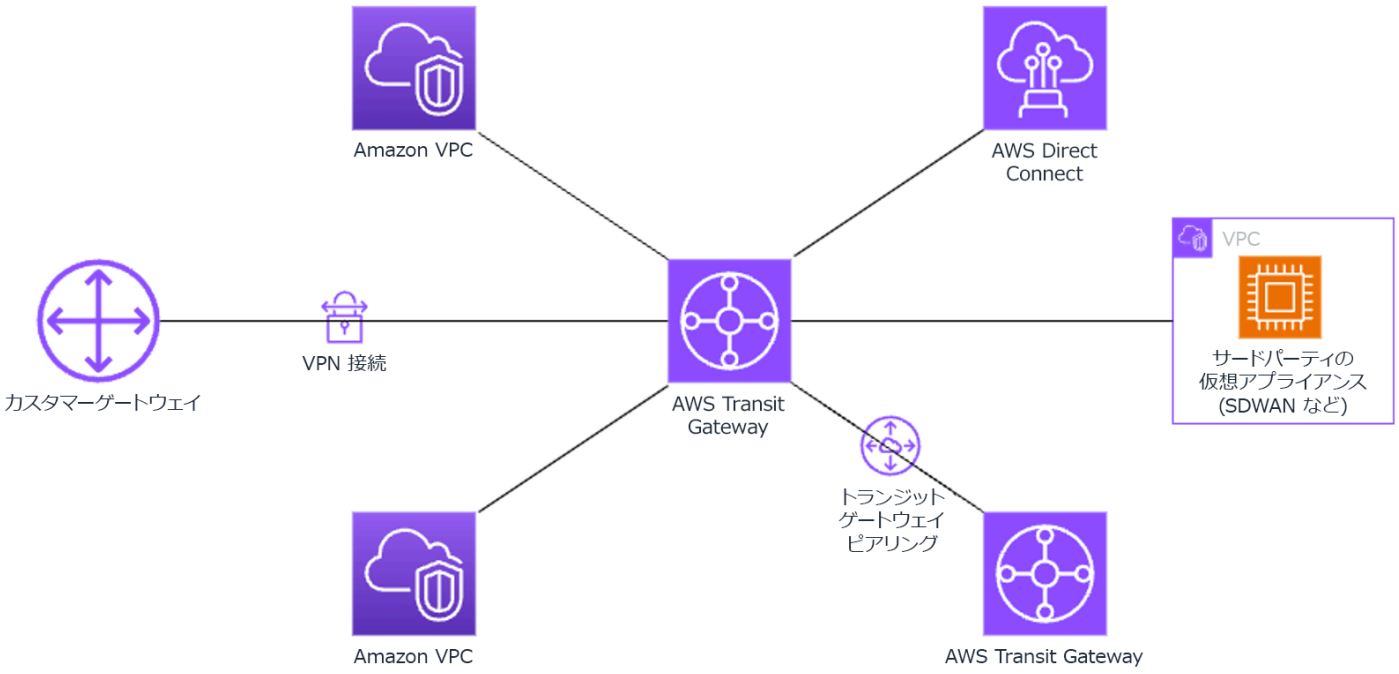
関連動画:

- AWS re:Invent 2018: Advanced VPC Design and New Capabilities for Amazon VPC (NET303)
- AWS re:Invent 2019: AWS Transit Gateway reference architectures for many VPCs (NET406-R1)
- AWS re:Invent 2023: AWS Ready for what's next? Designing networks for growth and flexibility (NET310)

REL02-BP04 多対多メッシュよりもハブアンドスポークトポロジを優先する

仮想プライベートクラウド (VPC) やオンプレミスネットワークなど、複数のプライベートネットワークを接続する場合は、メッシュトポロジではなくハブアンドスポークトポロジを選択します。各ネットワークが互いに直接接続され、複雑さと管理オーバーヘッドが増加するメッシュトポロジとは異なり、ハブアンドスポークアーキテクチャでは、接続が 1 つのハブを介して一元化されます。この一元化により、ネットワーク構造が簡素化され、運用性、スケーラビリティ、およびコントロールが強化されます。

AWS Transit Gateway は、AWS でハブアンドスポークネットワークを構築するために設計されたスケーラブルで可用性の高いマネージドサービスです。これは、ネットワークのセントラルハブとして機能し、ネットワークのセグメンテーション、一元化されたルーティング、およびクラウド環境とオンプレミス環境の両方へのシンプルな接続を提供します。次の図は、AWS Transit Gateway を使用してハブアンドスポークトポロジを構築する方法を示しています。



一般的なアンチパターン:

- ハブアンドスポークアーキテクチャではルーティングポリシーが複雑になりすぎると、ネットワークの効率が低下し、トラブルシューティングと事前管理の両方が複雑になります。

- ハブ内のルーティングベースのセグメンテーションが不十分だと、脆弱性が発生し、ネットワークが不正アクセスにさらされる可能性があります。
- 特にアベイラビリティーゾーンやリージョンを横断するトラフィックの場合、ハブを経由するトラフィックのデータ転送コストが高くなる可能性があるため、慎重な最適化が必要です。コストを抑えるには、効果的なトラフィック管理戦略が不可欠です。

このベストプラクティスを活用するメリット: 接続されているネットワークの数が増加するにつれて、メッシュ接続の管理と拡張はますます困難になります。AWS Transit Gateway は、ハブアンドスポークトポロジの構築と運用のためのスケーラブルで信頼性の高いハブを提供します。AWS Transit Gateway を使用すると、接続を確立し、複数のネットワークにわたるトラフィックルーティングを一元化できます。

このベストプラクティスが確立されていない場合のリスクレベル: 中

実装のガイダンス

- ネットワークを計画します。
- AWS Transit Gateway を作成します。
- VPC をアタッチします。
- 必要に応じて、VPN 接続または Direct Connect ゲートウェイを作成し、それらを Transit Gateway に関連付けます。
- Transit Gateway のルートテーブルを設定して、接続されている VPC と他の接続間のトラフィックのルーティング方法を定義します。
- パフォーマンスとコストの最適化のため、Amazon CloudWatch を使用して、必要に応じて構成のモニタリングと調整を行います。

リソース

関連するドキュメント:

- Transit Gateway とは
- スケーラブルで安全なマルチ VPC の AWS ネットワークインフラストラクチャを構築する
- Building a global network using AWS Transit Gateway Inter-Region peering
- Amazon Virtual Private Cloud Connectivity Options
- APN パートナー: ネットワークの計画を支援できるパートナー

- AWS Marketplace for Network Infrastructure

関連動画:

- AWS re:Invent 2023 - AWS networking foundations
- AWS re:Invent 2023 - Advanced VPC designs and new capabilities

REL02-BP05 接続されているすべてのプライベートアドレス空間において、重複しないプライベート IP アドレス範囲を指定する

各 VPC の IP アドレス範囲が、ピア接続時、Transit Gateway 経由での接続時、または VPN 経由での接続時に重複しないようにする必要があります。VPC とオンプレミス環境の間、または使用する他のクラウドプロバイダーとの IP アドレスの競合を回避してください。また、必要に応じてプライベート IP アドレス範囲を割り当てる方法を用意する必要もあります。これを自動化するには、IP アドレス管理 (IPAM) システムが役立ちます。

期待される成果:

- VPC、オンプレミス環境、または他のクラウドプロバイダー間で IP アドレス範囲が競合しないこと。
- 適切に IP アドレスを管理することにより、ネットワークインフラストラクチャを容易にスケールし、規模の拡大やネットワーク要件の変更に対応できること。

一般的なアンチパターン:

- オンプレミス、社内ネットワーク、または他のクラウドプロバイダーにあるものと同じ IP 範囲を VPC で使用する。
- ワークロードのデプロイに使用されている VPC の IP 範囲を追跡しない。
- スプレッドシートなど、手動の IP アドレス管理プロセスを使用する。
- CIDR ブロックサイズの過剰/過小なサイズ設定によって、無駄な IP アドレスが発生する、またはワークロード用のアドレススペースに不足が発生する。

このベストプラクティスを確立するメリット: ネットワークを積極的に計画することで、相互接続されたネットワークで同じ IP アドレスが複数出現しないようにできます。これにより、異なるアプリ

ケーションを使用しているワークロードの一部でルーティングの問題が発生するのを防ぐことができます。

このベストプラクティスが確立されていない場合のリスクレベル: 中

実装のガイダンス

Amazon VPC IP Address Manager などの IPAM を使用して、CIDR の使用状況をモニタリングおよび管理します。AWS Marketplace で複数の IPAM が提供されています。AWS での予想される使用量を評価して、CIDR の範囲を既存の VPC に追加し、計画的な使用量の増加を可能にする VPC を作成します。

実装手順

- 現在の CIDR 消費量 (VPC、サブネットなど) を把握します。
  - サービス API オペレーションを使用して、現在の CIDR 消費量を収集します。
  - Amazon VPC IP Address Manager を使用してリソースを検出します。
- 現在のサブネットの使用量を把握します。
  - サービス API オペレーションを使用して、各リージョンでの VPC あたりのサブネット数を収集します。
  - Amazon VPC IP Address Manager を使用してリソースを検出します。
- 現在の使用量を記録します。
- 重複している IP 範囲を作成したかどうか確認します。
- 予備容量を計算します。
- 重複している IP 範囲を特定します。新しいアドレス範囲に移行するか、プライベート NAT ゲートウェイや AWS PrivateLink などの手法を使用することを検討してください。

リソース

関連するベストプラクティス:

- ネットワークの保護

関連するドキュメント:

- APN パートナー: ネットワークの計画を支援できるパートナー

- AWS Marketplace for Network Infrastructure (ネットワークインフラストラクチャ向け AWS Marketplace)
- Amazon Virtual Private Cloud Connectivity Options Whitepaper
- 複数データセンターの HA ネットワーク接続
- Connecting Networks with Overlapping IP Ranges
- What Is Amazon VPC?
- IPAM とは

関連動画:

- AWS re:Invent 2023 - Advanced VPC designs and new capabilities
- AWS re:Invent 2019: AWS Transit Gateway reference architectures for many VPCs
- AWS re:Invent 2023 - Ready for what's next? Designing networks for growth and flexibility
- AWS re:Invent 2021 - {New Launch} Manage your IP addresses at scale on AWS

ワークロードアーキテクチャ

信頼性の高いワークロードの実現は、ソフトウェアとインフラストラクチャの両方について事前に設計を決定することから始まります。アーキテクチャの選択は、Well-Architected の 6 つの柱のすべてにおいて、ワークロードの動作に影響を与えます。高い信頼性を保つには、特定のパターンに従う必要があります。

次のセクションでは、高い信頼性を保つためにこのようなパターンで使用するベストプラクティスについて説明します。

トピック

- ワークロードサービスアーキテクチャを設計する
- 障害を防ぐために分散システムでの操作を設計する
- 障害を軽減または障害に耐えるために分散システムでの操作を設計する

ワークロードサービスアーキテクチャを設計する

サービス指向アーキテクチャ (SOA) またはマイクロサービスアーキテクチャを使用して、拡張性と信頼性の高いワークロードを構築します。サービス指向アーキテクチャ (SOA) は、サービスインターフェイスを介してソフトウェアコンポーネントを再利用できるようにする方法です。マイクロサービスアーキテクチャは、その一歩先を行き、コンポーネントをさらに小さくシンプルにしています。

サービス指向アーキテクチャ (SOA) インターフェイスは一般的な通信標準を使用しているため、新しいワークロードに迅速に組み込むことができます。SOA は、相互に依存して分割不可能なユニットで設定されるモノリスアーキテクチャを構築するプラクティスに取って代わりました。

AWS では長く SOA を使用してきましたが、現在はマイクロサービスを使用してシステムを構築することを受け入れています。マイクロサービスには多くの魅力がありますが、可用性という観点から重要なのは、マイクロサービスが小さくてシンプルであることです。マイクロサービスでは、さまざまなサービスに求められる可用性を区別して、最も高い可用性ニーズを持つマイクロサービスに特化して投資を行うことができます。たとえば、Amazon.com で製品情報ページ (詳細ページ) を配信するには、ページの個別の部分を作成するために何百ものマイクロサービスが呼び出されます。製品と料金の詳細を表示するために不可欠なサービスはいくつかありますが、そのサービスが利用できないときは、ページ上のコンテンツの大部分を単純に削除できます。写真やレビューなどでも、顧客が製品を購入できる場合にエクスペリエンスを提供する必要はありません。

ベストプラクティス

- REL03-BP01 ワークロードをセグメント化する方法を選択する
- REL03-BP02 特定のビジネスドメインと機能に重点を置いたサービスを構築する
- REL03-BP03 API ごとにサービス契約を提供する

REL03-BP01 ワークロードをセグメント化する方法を選択する

アプリケーションの回復力要件を決定する際に、ワークロードのセグメント化は重要です。モノリシックアーキテクチャはできるだけ避ける必要があります。代わりに、どのアプリケーションコンポーネントをマイクロサービスに分けられるかを注意深く考慮します。アプリケーションの要件によっては、最終的にサービス指向アーキテクチャ (SOA) とマイクロサービスの組み合わせになることもあります。ステートレス化が可能なワークロードは、マイクロサービスとしてデプロイすることができます。

期待される成果: ワークロードは、サポート可能で、スケーラブルであり、可能な限り疎結合であるべきです。

ワークロードのセグメント化について選択を行う場合は、複雑さに対してどれだけメリットがあるかを考えてください。新製品のローンチ時に適しているものは、初期からスケーリングのことを考えて構築したワークロードとは異なります。既存のモノリスをリファクタリングする場合、アプリケーションがステートレスへの分解をどの程度サポートできるかを検討する必要があります。サービスを小さく分割することで、小規模で明確なチームが開発、管理することができます。しかし、サービスの規模が小さくなると、レイテンシーの増加、デバッグの複雑化、運用負荷の増大など、複雑な問題が発生する可能性があります。

一般的なアンチパターン:

- 「マイクロサービス Death Star」とは、アトミックコンポーネントが強く依存しあっているために、1 つの失敗がより大きな失敗となり、コンポーネントがモノリスのように柔軟性が低く、壊れやすくなっている状態のことです。

このプラクティスを活用するメリット:

- より特化したセグメントは、高い俊敏性、組織の柔軟性、およびスケーラビリティにつながる。
- サービス中断の影響が小さくなる。
- アプリケーションコンポーネントには異なる可用性要件があり、より特化したセグメント化によってサポートされることがある。

- ワークロードをサポートするチームの責任が明確に定義される。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

ワークロードをセグメント化する方法に基づいてアーキテクチャタイプを選択します。SOA またはマイクロサービスアーキテクチャ (場合によってはモノリシックアーキテクチャ) を選択します。モノリスアーキテクチャから開始する場合でも、それがモジュラー型で、ユーザーの導入に合わせて製品がスケールされるにつれて最終的に SOA またはマイクロサービスに進化できることを確認する必要があります。SOA とマイクロサービスは、それぞれより小さなセグメントを提供し、最新のスケーラブルで信頼性の高いアーキテクチャとして好まれていますが、特にマイクロサービスアーキテクチャを展開する際には、トレードオフを考慮しなければなりません。

主なトレードオフとしては、分散コンピューティングアーキテクチャを採用することになり、ユーザーのレイテンシー要件を達成するのが難しくなることと、ユーザーインタラクションのデバッグとトレースにさらなる複雑さが生じることが挙げられます。AWS X-Ray を使ってこの問題の解決に役立てることができます。また、管理するアプリケーションの数が増え、複数の独立したコンポーネントを配置する必要があるため、運用が複雑になることも考慮しなければなりません。

モノリシック、サービス指向、マイクロサービスアーキテクチャ

実装手順

- アプリケーションのリファクタリングやビルドに適したアーキテクチャを決定します。SOA とマイクロサービスは、それぞれより小さなセグメンテーションを提供し、最新のスケーラブルで信

頼性の高いアーキテクチャとして好まれます。SOA は、マイクロサービスの複雑さを回避しながら、より小さなセグメント化を達成するための優れた折衷案となり得ます。詳細については、マイクロサービスのトレードオフ.

- ワークロードが適していて、組織がサポートできる場合は、最高の俊敏性と信頼性を実現するために、マイクロサービスアーキテクチャを使用すべきです。詳細については、AWS でのマイクロサービスの実装
- モノリスを Strangler Fig パターン に従って、より小さいコンポーネントにリファクタリングします。これには、特定のアプリケーションコンポーネントを新しいアプリケーションとサービスに徐々に置き換えることが含まれます。AWS Migration Hub Refactor Spaces は、増分リファクタリングの開始点として機能します。詳細については、「オンプレミスのレガシーワークロードをシームレスに移行する」を参照してください。
- マイクロサービスを実装する場合、これらの分散したサービスが互いに通信できるようにするためのサービス検出メカニズムが必要になる場合があります。AWS App Mesh をサービス指向アーキテクチャとともに使用することで、高い信頼性をもってサービスを検出し、サービスにアクセスできます。 AWS Cloud Map は、動的 DNS ベースのサービス検出にも使用できます。
- モノリスから SOA へ移行する場合、Amazon MQ は、レガシーアプリケーションをクラウドで再設計する際に、サービスバスとしてギャップを埋めるのに役立ちます。
- 単一の共有されたデータベースがある既存のモノリスには、データを再編成して小さなセグメントにする方法を選択します。これは、ビジネスユニット、アクセスパターン、またはデータ構造によって行うことができます。リファクタリングプロセスのこの時点では、リレーショナルまたは非リレーショナル (NoSQL) タイプのデータベースを選択して進めていく必要があります。詳細については、「SQL から NoSQL へ」を参照してください。

実装計画に必要な工数レベル: 高

リソース

関連するベストプラクティス:

- REL03-BP02 特定のビジネスドメインと機能に重点を置いたサービスを構築する

関連するドキュメント:

- Amazon API Gateway: OpenAPI を使用した REST API の設定
- サービス指向アーキテクチャとは
- 境界付けられたコンテキスト (ドメイン駆動設計の中心的なパターン)

- AWS でのマイクロサービスの実装
- マイクロサービスのトレードオフ
- Microservices - a definition of this new architectural term
- AWS でのマイクロサービス
- AWS App Mesh とは

関連する例:

- Iterative App Modernization Workshop

関連動画:

- Delivering Excellence with Microservices on AWS

REL03-BP02 特定のビジネスドメインと機能に重点を置いたサービスを構築する

サービス指向アーキテクチャ (SOA) は、ビジネスニーズに合わせて明確に定義された機能を備えたサービスを定義します。マイクロサービスは、ドメインモデルと制限付きコンテキストを使用して、ビジネスコンテキストの境界に沿ってサービスの境界を描きます。ビジネスドメインと機能に重点を置くことで、チームがサービスの独立した信頼性要件を定義しやすくなります。コンテキストに制限があると、ビジネスロジックが分離されてカプセル化されるため、チームは障害の処理方法についてより的確に判断できるようになります。

期待される成果: エンジニアとビジネス関係者は共同で境界のあるコンテキストを定義し、それを使用して特定のビジネス機能を果たすサービスとしてシステムを設計します。これらのチームは、イベントストーミングなどの確立された手法を使用して要件を定義します。新しいアプリケーションは、境界を明確に定義し、ゆるく結合するサービスとして設計されています。既存のモノリスは境界コンテキストに分解され、システム設計は SOA またはマイクロサービスアーキテクチャに移行します。モノリスをリファクタリングする際には、バブルコンテキストやモノリスの分解パターンなどの確立されたアプローチが適用されます。

ドメイン指向のサービスは、状態を共有しない 1 つ以上のプロセスとして実行されます。需要の変動に独自に対応し、ドメイン固有の要件に照らして障害シナリオを処理します。

一般的なアンチパターン:

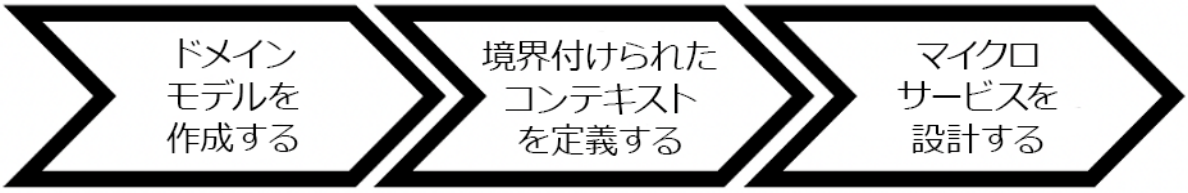
- チームは、特定のビジネスドメインではなく、UI や UX、ミドルウェア、データベースなどの特定の技術ドメインを中心に形成されます。
- アプリケーションはドメインの担当範囲にまたがります。限定されたコンテキストにまたがるサービスは、メンテナンスが難しく、大規模なテスト作業が必要になり、複数のドメインチームがソフトウェアアップデートに参加する必要がある場合があります。
- ドメインエンティティライブラリと同様に、ドメイン依存関係はサービス間で共有されるため、あるサービスドメインを変更すると、他のサービスドメインも変更する必要があります。
- サービス契約とビジネスロジックは、エンティティを共通で一貫性のあるドメイン言語で表現しないため、翻訳層が複雑になり、デバッグ作業が増えます。

このベストプラクティスを活用するメリット: アプリケーションは、ビジネスドメインによって区切られた独立したサービスとして設計され、共通のビジネス言語を使用します。サービスは独立してテストおよびデプロイできます。サービスは、実装されたドメインのドメイン固有の耐障害性要件を満たします。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

ドメイン主導型意思決定 (DDD) は、ビジネスドメインを中心にソフトウェアを設計および構築するための基本的なアプローチです。ビジネスドメインに焦点を当てたサービスを構築する際には、既存のフレームワークを使用すると便利です。既存のモノリシックアプリケーションを扱う場合は、確立された手法を提供する分解パターンを利用して、アプリケーションをモダナイズしてサービスにすることができます。



ドメイン主導の意思決定

実装手順

- チームは イベントストーミング ワークショップを開催して、軽量の付箋形式でイベント、コマンド、集計、ドメインをすばやく特定できます。

- ドメインのエンティティと機能をドメインコンテキストで形成したら、以下のようにドメインをサービスに分割できます。境界コンテキスト、そこで類似した機能と属性を共有するエンティティがグループ化されます。モデルをコンテキストに分割したら、マイクロサービスを境界で区切る方法のテンプレートが現れます。
  - 例えば、Amazon.com ウェブサイトエンティティには、パッケージ、配送、スケジュール、料金、割引、通貨などがあります。
  - パッケージ、配送、スケジュールは出荷コンテキストにグループ化され、料金、割引、通貨は料金設定コンテキストにグループ化されます。
- モノリスのマイクロサービスへの分解 マイクロサービスをリファクタリングするためのパターンを概説します。ビジネス機能、サブドメイン、またはトランザクション別に分解するパターンを使用することは、ドメイン主導のアプローチによく合います。
- バブルコンテキストなどの 戦術的なテクニックを使用すると、 事前に書き直したり、DDD に全面的にコミットしたりすることなく、既存またはレガシーアプリケーションに DDD を導入できます。バブルコンテキストアプローチでは、サービスマッピングと調整、または 破損対策レイヤーを使用して小さな境界コンテキストが確立され、新しく定義されたドメインモデルを外部の影響から保護します。

チームがドメイン分析を行い、エンティティとサービス契約を定義したら、AWS のサービスを活用してドメイン主導型の設計をクラウドベースのサービスとして実装できます。

- ドメインのビジネスルールを実践するテストを定義することから開発を始めましょう。テスト駆動開発 (TDD) と行動主導開発 (BDD) は、チームがサービスをビジネス上の問題の解決に集中させるのに役立ちます。
- ビジネスドメインの要件とマイクロサービスアーキテクチャに最適な AWS のサービスは を 選択する:
  - AWS サーバーレス を使用すると、チームはサーバーやインフラストラクチャを管理するのではなく、特定のドメインロジックに集中できます。
  - AWS でのコンテナ インフラストラクチャの管理が簡素化されるため、ドメイン要件に集中できます。
  - 目的別データベース は、ドメイン要件を最適なデータベースタイプに一致させるのに役立ちます。
- AWS 上でヘキサゴナルアーキテクチャを構築すると、 ビジネスドメインから逆算してサービスにビジネスロジックを構築して機能要件を満たし、統合アダプターを接続するフレームワークの概要が示されます。インターフェイスの詳細と AWS のサービスのビジネスロジックを分離するパターンは、チームがドメインの機能に集中し、ソフトウェアの品質を向上させるのに役立ちます。

リソース

関連するベストプラクティス:

- REL03-BP01 ワークロードをセグメント化する方法を選択する
- REL03-BP03 API ごとにサービス契約を提供する

関連するドキュメント:

- AWS マイクロサービス
- AWS でのマイクロサービスの実装
- モノリスをマイクロサービスに分割する方法
- レガシーシステムに囲まれているときの DDD の使用開始
- “Domain-Driven Design: Tackling Complexity in the Heart of Software”
- AWS 上でヘキサゴナルアーキテクチャを構築すると、
- モノリスのマイクロサービスへの分解
- イベントストーミング
- 制限されたコンテキスト間のメッセージ
- マイクロサービス
- テスト駆動型開発
- 行動主導型開発

関連サンプル:

- エンタープライズクラウドネイティブワークショップ
- AWS でのクラウドネイティブマイクロサービスの設計 (DDD/EventStormingWorkshop より)

関連ツール:

- AWS クラウド データベース
- AWS でのサーバーレス
- AWS でのコンテナ

REL03-BP03 API ごとにサービス契約を提供する

サービス契約とは、機械が読み取れる API 定義で定義された、API プロデューサーとコンシューマー間の文書化された契約です。契約バージョニング戦略により、コンシューマーは既存の API を引き続き使用し、準備ができたらアプリケーションを新しい API に移行できます。プロデューサーのデプロイは、契約がある限りいつでも行うことができます。サービスチームは、選択した技術スタックを使用して、API 契約の条件を満たすことができます。

期待される成果:

一般的なアンチパターン: サービス指向またはマイクロサービスアーキテクチャで構築されたアプリケーションは、ランタイム依存関係を統合しながら独立して動作できます。API コンシューマーまたはプロデューサーに変更をデプロイしても、双方が共通の API 契約に従っていれば、システム全体の安定性が損なわれることはありません。サービス API を介して通信するコンポーネントは、相互にほとんどまたはまったく影響を与えずに、独立した機能リリース、ランタイム依存関係のアップグレード、またはディザスタリカバリ (DR) サイトへのフェイルオーバーを実行できます。さらに、ディスクリートサービスでは、他のサービスを一斉にスケーリングしなくても、吸収するリソース需要を個別にスケーリングできます。

- 厳密に型指定されたスキーマを使用しないサービス API を作成します。その結果、API バインディングの生成に使用できない API や、プログラムで検証できないペイロードが生成されます。
- API の利用者に更新とリリースを強いるようなバージョニング戦略を採用していない。そうしないと、サービス契約が発展したときに失敗する。
- ドメインコンテキストや言語での統合の失敗を説明するのではなく、基盤となるサービス実装の詳細を漏らすエラーメッセージ。
- サービスコンポーネントを個別にテストできるようにするために、API 契約を使用せずにテストケースを開発したりモック API 実装を使用したりします。

このベストプラクティスを活用するメリット: API サービス契約を介して通信するコンポーネントで構成される分散型システムでは、信頼性を向上させることができます。デベロッパーは、コンパイル中にタイプチェックを行って、リクエストとレスポンスが API 契約に従っていることと、必須フィールドが存在することを確認することで、開発プロセスの早い段階で潜在的な問題を発見できます。API 契約は、API 用のわかりやすい自己文書化インターフェイスを提供し、さまざまなシステムやプログラミング言語間の相互運用性を向上させます。

このベストプラクティスを活用しない場合のリスクレベル: 中

実装のガイダンス

ビジネスドメインを特定し、ワークロードのセグメント化を決定したら、サービス API を開発できます。まず、機械が読み取れる API のサービス契約を定義し、次に API バージョニング戦略を実装します。REST、GraphQL、非同期イベントなどの一般的なプロトコルでサービスを統合する準備ができたら、AWSのサービスをアーキテクチャに組み込んで、コンポーネントを厳密に型指定された API 契約と統合できます。

サービス API 契約の AWS のサービス

以下を含む AWS のサービスを Amazon API Gateway、AWS AppSync、 Amazon EventBridge アプリケーションで API サービス契約を使用するようにアーキテクチャに組み込むことができます。Amazon API Gateway は、ネイティブ AWS サービスや他の Web サービスと直接統合するのに役立ちます。API Gateway は、 OpenAPI 仕様 とバージョニングをサポートしています。AWS AppSync は、クエリ、ミューテーション、およびサブスクリプションのサービスインターフェイスを定義する GraphQL スキーマを定義することによって構成するマネージド GraphQL エンドポイントですAmazon EventBridge はイベントスキーマを使用してイベントを定義し、イベントのコードバインディングを生成します。

実装手順

- まず、API の契約を定義します。契約では、API の機能を説明するだけでなく、API の入出力用に厳密に型指定されたデータオブジェクトとフィールドを定義します。
- API Gateway で API を設定すると、エンドポイントの OpenAPI 仕様をインポートおよびエクスポートできます。
  - OpenAPI 定義をインポートする API の作成が簡単になり、次のようなコードツールとして AWS インフラストラクチャと統合できます AWS Serverless Application Model および AWS Cloud Development Kit (AWS CDK).
  - API 定義をエクスポートすると、 API テストツールとの統合が簡素化され、サービス利用者に統合仕様が提供されます。
- GraphQL API は次の方法で AWS AppSync を使用して定義および管理できます。 GraphQL スキーマファイルを定義して 契約インターフェイスを生成し、複雑な REST モデル、複数のデータベーステーブル、またはレガシーサービスとのやり取りを簡素化します。
- AWS Amplify は、AWS AppSync と統合されたプロジェクトで、アプリケーションで使用するための厳密に型指定された JavaScript クエリファイルと、AWS AppSync GraphQL クライアントライブラリを Amazon DynamoDB テーブル用に生成します。

- Amazon EventBridge からサービスイベントを利用する場合、イベントはスキーマレジストリに既に存在するスキーマや OpenAPI 仕様で定義したスキーマに従います。レジストリでスキーマを定義すると、スキーマ契約からクライアントバインディングを生成して、コードをイベントと統合することもできます。
- API の拡張またはバージョニング。オプションフィールドまたは必須フィールドのデフォルト値で構成できるフィールドを追加する場合、API を拡張する方が簡単なオプションです。
  - REST や GraphQL などのプロトコルの JSON ベースの契約は、契約の拡張に適しています。
  - SOAP のようなプロトコルの XML ベースの契約をサービスコンシューマーとテストして、契約延長の可能性を判断する必要があります。
- API をバージョニングするときは、ロジックを単一のコードベースで管理できるように、ファサードを使用してバージョンをサポートするプロキシバージョニングの実装を検討してください。
  - API Gatewayを使用すると、リクエストとレスポンスのマッピングを使用して、新しいフィールドにデフォルト値を提供したり、リクエストまたはレスポンスから削除されたフィールドを削除するファサードを確立したりすることで、契約の変更の吸収を簡素化できます。この方法では、基盤となるサービスは単一のコードベースを維持できます。

リソース

関連するベストプラクティス:

- REL03-BP01 ワークロードをセグメント化する方法を選択する
- REL03-BP02 特定のビジネスドメインと機能に重点を置いたサービスを構築する
- REL04-BP02 疎結合の依存関係を実装する
- REL05-BP03 再試行呼び出しを制御および制限する
- REL05-BP05 クライアントタイムアウトを設定する

関連するドキュメント:

- API (アプリケーションプログラミングインターフェイス) とは。
- AWS でのマイクロサービスの実装
- マイクロサービスのトレードオフ
- Microservices - a definition of this new architectural term
- AWS でのマイクロサービス
- OpenAPI の API Gateway 拡張機能の使用

- OpenAPI 仕様
- GraphQL: スキーマとタイプ
- Amazon EventBridge コードバインディング

関連サンプル:

- Amazon API Gateway: OpenAPI を使用した REST API の設定
- OpenAPI を使用した Amazon API Gateway から Amazon DynamoDB への CRUD アプリケーション
- サーバーレス時代のモダンアプリケーション統合パターン: API Gateway サービス統合
- Amazon CloudFront を使用したヘッダーベースの API Gateway バージョニングの実装
- AWS AppSync: クライアントアプリケーションの構築

関連動画:

- AWS SAM で OpenAPI を使用して API Gateway を管理する

関連ツール:

- Amazon API Gateway
- AWS AppSync
- Amazon EventBridge

障害を防ぐために分散システムでの操作を設計する

分散システムは、サーバーやサービスなどのコンポーネントを相互接続するために通信ネットワークを利用しています。このネットワークでデータの損失やレイテンシーがあっても、ワークロードは確実に動作する必要があります。分散システムのコンポーネントは、他のコンポーネントやワークロードに悪影響を与えない方法で動作する必要があります。これらのベストプラクティスは、故障を防ぎ、平均故障間隔 (MTBF) を改善します。

ベストプラクティス

- REL04-BP01 依存している分散システムの種類を特定する
- REL04-BP02 疎結合の依存関係を実装する

- REL04-BP03 継続動作を行う
- REL04-BP04 すべての応答に冪等性を持たせる

REL04-BP01 依存している分散システムの種類を特定する

分散システムには、同期、非同期、またはバッチの 3 つの種類があります。同期システムは、リクエストを可能な限り迅速に処理し、HTTP/S、REST、またはリモートプロシージャコール (RPC) プロトコルを使用して同期リクエスト呼び出しと応答呼び出しを行うことで相互に通信する必要があります。非同期システムは、個々のシステムを結合することなく、中間サービスを介して非同期的にデータを交換することによって相互に通信します。バッチシステムは大量の入力データを受け取り、人の介入なしに自動データプロセスを実行し、出力データを生成します。

期待される成果: 同期、非同期、およびバッチの依存関係と効果的に相互作用するワークロードを設計します。

一般的なアンチパターン:

- ワークロードは依存関係からの応答を無期限に待つため、リクエストが受信されたかどうかが不明なまま、ワークロードクライアントがタイムアウトすることがあります。
- ワークロードは、相互に同期呼び出しを行う一連の依存システムを使用しています。これには、チェーン全体で正常に処理が行われる前に、各システムが利用可能な状態にあり、システムでリクエストを正常に処理する必要があるため、動作が脆弱になり、全体的な可用性が損なわれる可能性があります。
- ワークロードは依存関係と非同期で通信し、重複したメッセージを受信する場合が多いのにもかかわらず、1 回のみのメッセージ処理が保証されるという概念に基づいています。
- 適切なバッチスケジューリングツールを使用していないため、ワークロードは同じバッチジョブを同時に実行します。

このベストプラクティスを活用するメリット: ワークロードでは、同期、非同期、バッチの 1 つまたは複数の通信スタイルを実装するのが一般的です。このベストプラクティスは、それぞれの通信スタイルに関連するさまざまなトレードオフを特定し、ワークロードが依存関係の中断に耐えられるようにするのに役立ちます。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

以下のセクションでは、各種類の依存関係に関する一般的な実装ガイダンスと固有の実装ガイダンスの両方について説明します。

一般的ガイドライン:

- 依存関係が提供するパフォーマンスと信頼性のサービスレベル目標 (SLO) が、ワークロードのパフォーマンスと信頼性の要件を満たしていることを確認します。
- AWS オブザーバビリティサービスを使用して応答時間とエラー率をモニタリングし、依存関係がワークロードに必要なレベルでサービスを提供していることを確認します。
- 依存関係と通信する際に、ワークロードが直面する可能性のある課題を特定します。分散システムには、アーキテクチャの複雑さ、運用上の負担、コストを増大させる可能性のあるさまざまな課題があります。一般的な課題には、レイテンシー、ネットワークの中断、データ損失、スケーリング、データ複製の遅延などがあります。
- 堅牢なエラー処理とロギングを実装して、依存関係で問題が発生した場合のトラブルシューティングに役立てます。

同期依存関係

同期通信では、ワークロードは依存関係にリクエストを送信し、応答を待っている操作をブロックします。依存関係がリクエストを受け取ると、すぐに処理を試み、応答をワークロードに送り返します。同期通信の大きな課題は、一時的な結合が発生するため、ワークロードとその依存関係を同時に利用できるようにする必要があることです。ワークロードで依存関係との同期通信が必要な場合は、以下のガイダンスを検討します。

- ワークロードが 1 つの機能を実行するために複数の同期依存関係に依存するべきではありません。リクエストを正常に完了させるためにパス内のすべての依存関係が利用可能である必要があるため、この依存関係の連鎖は全体的な脆弱性を高めます。
- 依存関係が正常でない場合や利用できない場合、エラー処理と再試行戦略を決定します。バイモーダル動作の使用は避けてください。バイモーダル動作とは、通常モードと障害モードでワークロードが異なる動作を示す場合をいいます。バイモーダル動作の詳細については、「REL11-BP05 静的安定性を使用してバイモーダル動作を防止する」を参照してください。
- ワークロードを待機させるより、フェイルファストの方が良いことを覚えておいてください。例えば、AWS Lambda デベロッパーガイドには、Lambda 関数を呼び出す際の再試行と失敗の処理方法が記載されています。

- ワークロードが依存関係を呼び出す際のタイムアウトを設定します。これにより、応答を待つ時間が長すぎたり、無期限に待ったりすることを回避できます。この問題に関する説明については、「レイテンシーを考慮した Amazon DynamoDB アプリケーションのための AWS Java SDK HTTP リクエスト設定の調整」を参照してください。
- 1 つのリクエストを処理するためのワークロードから依存関係への呼び出し回数を最小限に抑えます。呼び出し回数の多さは、結合とレイテンシーの増加につながります。

非同期依存関係

ワークロードを依存関係から一時的に切り離すには、非同期で通信する必要があります。非同期アプローチを使用すると、ワークロードの依存関係や一連の依存関係からの応答の送信を待つことなく、他の処理を実行できます。

ワークロードで依存関係との非同期通信が必要な場合は、以下のガイダンスを検討します。

- ユースケースと要件に基づいて、メッセージングとイベントストリーミングのどちらを使用するかを決定します。メッセージングでは、メッセージブローカーを介してメッセージを送受信することで、ワークロードが依存関係と通信できるようになります。イベントストリーミングでは、ワークロードとその依存関係がストリーミングサービスを使用して連続したデータストリームとして配信され、できるだけ早く処理する必要のあるイベントのパブリッシュとサブスクライブを行えます。

- メッセージングとイベントストリーミングではメッセージの処理方法が異なるため、以下に基づいてトレードオフを決定する必要があります。
  - メッセージの優先度: メッセージブローカーは、優先度の高いメッセージを通常のメッセージよりも先に処理できます。イベントストリーミングでは、すべてのメッセージの優先度が同じになります。
  - メッセージ消費: メッセージブローカーは、コンシューマーがメッセージを受信することを保証します。イベントストリーミングのコンシューマーは、最後に読んだメッセージを常に把握しておく必要があります。
  - メッセージの順序付け: 先入れ先出し (FIFO) 方式を使用しない限り、メッセージングでは送信された順序でメッセージが受信されることは保証されません。イベントストリーミングでは、データが生成された順序が常に保持されます。
  - メッセージの削除: メッセージングでは、コンシューマーは処理後にメッセージを削除する必要があります。イベントストリーミングサービスはメッセージをストリームに追加し、メッセージの保存期間が終了するまでストリームに残ります。この削除ポリシーにより、イベントストリーミングはメッセージの再生に適したものになります。

- 依存関係がいつ作業を完了したかをワークロードがどのように認識するかを定義します。例えば、ワークロードが Lambda 関数を非同期で呼び出すと、Lambda によってイベントはキューに入れられ、追加情報なしで成功応答が返されます。処理が完了すると、Lambda 関数は成功または失敗に基づいて設定可能な宛先に結果を送信できます。
- べき等性を活用して、重複メッセージを処理するワークロードを構築します。べき等性とは、同じメッセージに対してワークロードが複数回生成されても、ワークロードの結果は変化しないことを指します。メッセージングサービスまたはストリーミングサービスは、ネットワーク障害が発生したり、受信確認を受け取らなかったりした場合に、メッセージの再送信を行います。
- ワークロードが依存関係から応答を受け取らない場合は、ワークロードはリクエストを再送信します。リトライ回数を制限して、ワークロードの CPU、メモリ、ネットワークリソースの消費を抑え、他のリクエストを処理できるようにすることを検討してください。AWS Lambda のドキュメントには、非同期呼び出しのエラーの処理方法が記載されています。
- 適切なオブザーバビリティ、デバッグ、トレースツールを活用して、ワークロードの非同期通信とその依存関係を管理し運用します。Amazon CloudWatch を使用して、メッセージングサービスおよびイベントストリーミングサービスをモニタリングできます。また、ワークロードを AWS X-Ray でインストルメント化して、問題のトラブルシューティングに役立つインサイトをすばやく得ることもできます。

バッチ依存

バッチシステムは、手動操作なしで、入力データを受け取り、処理するための一連のジョブを開始し、いくつかの出力データを生成します。データサイズにもよりますが、ジョブは数分から、場合によっては数日かかることもあります。ワークロードで依存関係とのバッチ通信を行う場合は、以下のガイダンスを検討します。

- ワークロードでバッチジョブを実行する時間枠を定義します。ワークロードでは、例えば 1 時間ごとまたは月末に、バッチシステムを呼び出す繰り返しパターンを設定できます。
- データ入力と処理済みデータ出力の場所を定義します。Amazon Simple Storage Service (Amazon S3)、Amazon Elastic File System (Amazon EFS)、Amazon FSx for Lustre など、ワークロードでスケーラブルなファイルの読み書きを行えるストレージサービスを選択します。
- ワークロードで複数のバッチジョブを呼び出す必要がある場合は、AWS Step Functions を活用して、AWS またはオンプレミスで実行されるバッチジョブのオーケストレーションを簡素化できます。このサンプルプロジェクトは、Step Functions、AWS Batch、Lambda を使用したバッチジョブのオーケストレーションを示しています。
- バッチジョブをモニタリングして、ジョブの完了に本来よりも時間がかかっているなどの異常がないかを確認します。CloudWatch Container Insights のようなツールを使用して、AWS Batch 環境

やジョブをモニタリングできます。この場合、ワークロードによって次のジョブの開始が停止し、関連するスタッフに例外が通知されます。

リソース

関連するドキュメント:

- AWS クラウド オペレーション: モニタリングとオブザーバビリティ
- The Amazon Builder's Library: 分散システムの課題
- REL11-BP05 静的安定性を使用してバイモーダル動作を防止する
- AWS Lambda デベロッパーガイド: AWS Lambda でのエラー処理と自動再試行
- レイテンシーを考慮した Amazon DynamoDB アプリケーションのための AWS Java SDK HTTP リクエスト設定の調整
- AWS Messaging
- ストリーミングデータとは
- AWS Lambda デベロッパーガイド: 非同期呼び出し
- Amazon Simple Queue Service に関するよくある質問
- Amazon Kinesis Data Streams 開発者ガイド: Handling Duplicate Records
- Amazon Simple Queue Service 開発者ガイド: Available CloudWatch metrics for Amazon SQS
- Amazon Kinesis Data Streams 開発者ガイド: Monitoring the Amazon Kinesis Data Streams Service with Amazon CloudWatch
- AWS X-Ray 開発者ガイド: AWS X-Ray concepts
- AWS Samples on GitHub: AWS Step functions Complex Orchestrator App
- AWS Batch ユーザーガイド: AWS Batch CloudWatch Container Insights

関連動画:

- AWS Summit SF 2022 - Full-stack observability and application monitoring with AWS (COP310)

関連ツール:

- Amazon CloudWatch
- Amazon CloudWatch Logs

- AWS X-Ray
- Amazon Simple Storage Service (Amazon S3)
- Amazon Elastic File System (Amazon EFS)
- Amazon FSx for Lustre
- AWS Step Functions
- AWS Batch

REL04-BP02 疎結合の依存関係を実装する

キューイングシステム、ストリーミングシステム、ワークフロー、ロードバランサーなどの依存関係は、緩やかに結合しています。疎結合は、コンポーネントの動作をそれに依存する他のコンポーネントから分離するのに役立ち、弾力性と俊敏性を高めます。

密結合のシステムでは、あるコンポーネントを変更すると、そのコンポーネントに依存する他のコンポーネントも変更しなければならなくなり、結果として、すべてのコンポーネントのパフォーマンスが低下する可能性があります。疎結合はこの依存関係を壊すため、依存コンポーネントが知る必要があるのは、バージョン管理されて公開されたインターフェイスのみです。依存関係があるコンポーネント間に疎結合を実装すると、あるコンポーネントの障害が別のコンポーネントに影響を及ぼさないようにすることができます。

疎結合では、コードの変更やコンポーネントへの機能の追加を自由にできる一方で、そのコンポーネントに依存する他のコンポーネントへのリスクを最小限に抑えることができます。また、回復力を細分化でき、コンポーネントレベルでスケールアウトしたり、依存関係の根本的な実装さえも変更したりできます。

疎結合によって弾力性をさらに向上させるには、可能な場合はコンポーネント間のやりとりを非同期にします。このモデルは、即時応答を必要とせず、リクエストが登録されていることの確認で十分な状況では、どのような対話にも最適です。イベントを生成するコンポーネントと、イベントを消費するコンポーネントがあります。2 つのコンポーネントは、直接的なポイントツーポイントのやりとりではなく、通常、Amazon SQS キューのような中間的な耐久性の高いストレージレイヤーや Amazon Kinesis のようなストリーミングデータプラットフォーム、または AWS Step Functions を介して統合されます。

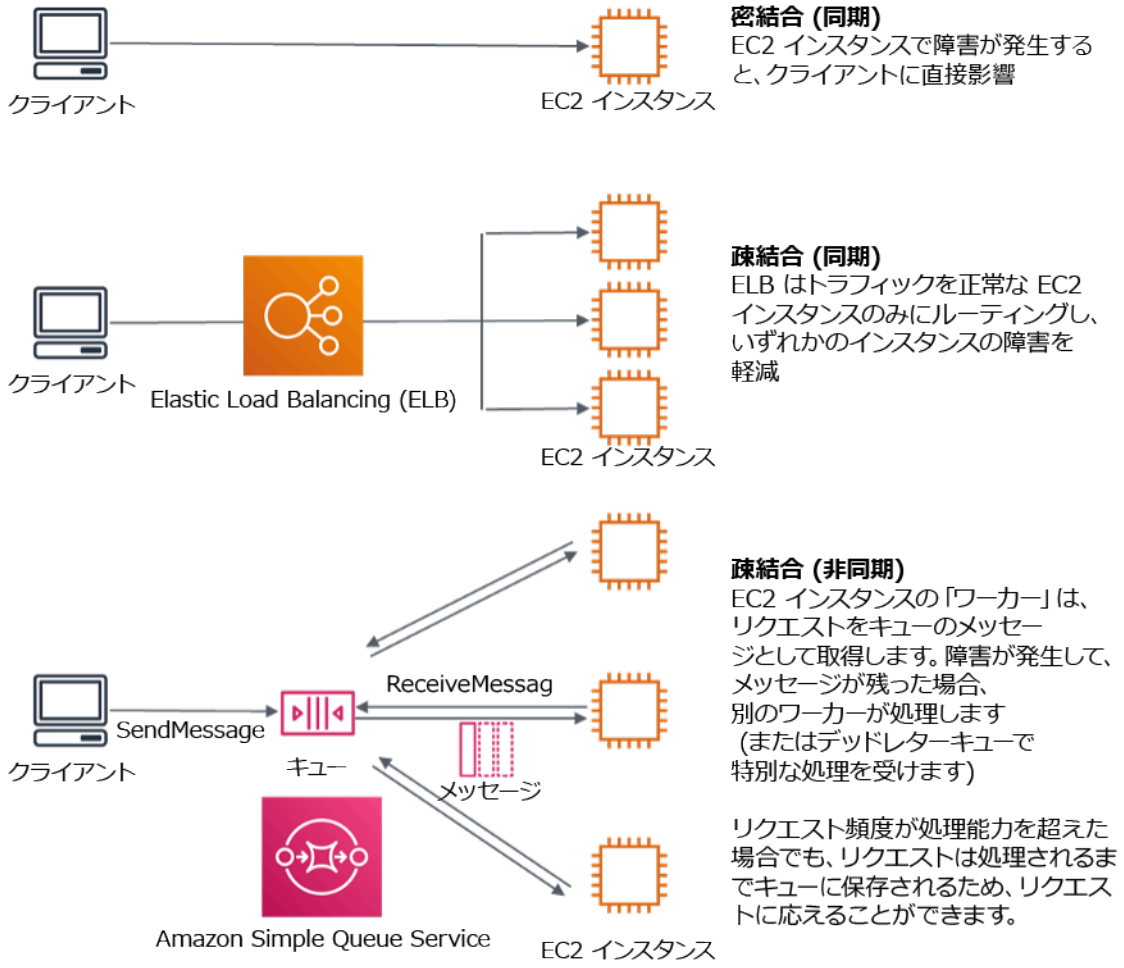


図 4: キューイングシステムやロードバランサーなどの依存関係が疎結合されています。

Amazon SQS キューと Elastic Load Balancing は、疎結合の中間レイヤーを追加する方法のうちの 2 つにすぎません。イベント駆動型アーキテクチャは、Amazon EventBridge を使用して AWS クラウド に構築することもできます。これにより、依存しているサービス (イベントコンシューマー) からクライアント (イベントプロデューサー) を抽出することができます。Amazon Simple Notification Service (Amazon SNS) は、高スループット、プッシュベースの多対多メッセージングが必要な場合に効果的なソリューションです。Amazon SNS トピックを使用すると、パブリッシャーシステムは、メッセージを多数のサブスクライバーエンドポイントにファンアウトして、並列処理できます。

キューにはいくつかの利点がありますが、ほとんどのハードリアルタイムシステムでは、しきい値の時間 (多くの場合、数秒) よりも長時間かかっているリクエストは古くなっていると見なされ (クライアントが停止し、応答を待機しなくなる)、処理されません。このように、古くなったリクエストの代わりに、より新しい (そしておそらくまだ有効な) リクエストを処理することができます。

期待される成果: 疎結合の依存関係を実装することで、障害の影響が及ぶ範囲をコンポーネントレベルまで最小限に抑えることができ、問題の診断と解決に役立ちます。また、開発サイクルが簡素化され、チームはモジュールレベルで変更を実装できるようになり、その部分に依存する他のコンポーネントのパフォーマンスに影響は及びません。このアプローチでは、リソースのニーズに基づいてコンポーネントレベルでスケールアウトできるだけでなく、コスト効率良くコンポーネントを活用できるようになります。

一般的なアンチパターン:

- モノリシックワークロードのデプロイ。
- リクエストのフェイルオーバーや非同期処理を行うことはできない状態で、ワークロード層間で直接 API を呼び出す。
- 共有データを使用した密結合。疎結合のシステムでは、共有データベースや他の形で密結合されたデータストレージを介したデータの共有を避ける必要があります。そうしたデータ共有が密結合を持ち込み、スケーラビリティを妨げる可能性があります。
- バックプレッシャーを無視する。ワークロードには、コンポーネントが同じ速度でデータを処理できない場合に、データの受信を遅らせたり停止したりする機能が必要です。

このベストプラクティスを確立するメリット: 疎結合は、コンポーネントの動作をそれに依存する他のコンポーネントから分離するのに役立ち、弾力性と俊敏性を高めます。1 つのコンポーネントの障害は他のコンポーネントから分離されます。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

疎結合の依存関係を実装します。疎結合のアプリケーションを構築するためのさまざまなソリューションがあります。フルマネージド型のキュー、自動化されたワークフロー、イベントへの対応、API などを実装するサービスがこれに当たりますが、いずれも、コンポーネントの動作を他のコンポーネントから分離するのに役立ち、弾力性と俊敏性を高めます。

- イベント駆動型アーキテクチャを構築する: Amazon EventBridge では、疎結合かつ分散型のイベント駆動型アーキテクチャを構築できます。
- 分散型システムにキューを実装する: Amazon Simple Queue Service (Amazon SQS) を使用して、分散型システムを統合および分離できます。
- コンポーネントをマイクロサービスとしてコンテナ化する: チームはマイクロサービスを利用して、小さな独立したコンポーネントで構成されるアプリケーションを構築でき、それらのコンポーネントは、明確に定義された API を介して通信します。Amazon Elastic Container Service

(Amazon ECS) や Amazon Elastic Kubernetes Service (Amazon EKS) を使用して、コンテナの運用をすぐに始めることができます。

- Step Functions でワークフローを管理する: Step Functions では、複数の AWS サービスを調整し、柔軟なワークフローにまとめることができます。
- パブリッシュ/サブスクライブ (pub/sub) メッセージングアーキテクチャを活用する: Amazon Simple Notification Service(Amazon SNS) は、パブリッシャーからサブスクライバー (プロデューサーとコンシューマーと呼ぶ場合もある) へのメッセージ配信を行います。

実装手順

- イベント駆動型アーキテクチャのコンポーネントは、イベントによって開始されます。イベントは、ユーザーがカートに商品を追加するなど、システム内で発生するアクションです。アクションが正常に実行されると、システムの次のコンポーネントを起動するイベントが生成されます。
  - Building Event-driven Applications with Amazon EventBridge
  - AWS re:Invent 2022 - Designing Event-Driven Integrations using Amazon EventBridge
- 分散型メッセージングシステムには、キューベースのアーキテクチャを確立するために主に 3 つの部分を実装する必要があります。これには、分散型システムのコンポーネント、分離のために使用するキュー (Amazon SQS サーバー上で分散される)、キュー内のメッセージが該当します。一般的なシステムには、キューへのメッセージ配信を開始するプロデューサーと、キューからメッセージを受信するコンシューマーがあります。キューは、冗長性を確保するために、複数の Amazon SQS サーバーにメッセージを格納します。
  - Basic Amazon SQS architecture
  - Send Messages Between Distributed Applications with Amazon Simple Queue Service
- マイクロサービスをうまく利用すれば、疎結合のコンポーネントが独立したチームによって管理されるため、保守性とスケーラビリティが向上します。また、変更が必要になっても、動作を分離し、単一のコンポーネントに限定できます。
  - AWS でのマイクロサービスの実装
  - Let's Architect! Architecting microservices with containers
- AWS Step Functions では、分散型アプリケーションの構築、プロセスの自動化、マイクロサービスのオーケストレーションなどを行うことができます。複数のコンポーネントをオーケストレーションして 1 つのワークフローとしてまとめ、自動化することで、アプリケーション内の依存関係を分離できます。
  - Create a Serverless Workflow with AWS Step Functions and AWS Lambda
  - AWS Step Functions の開始方法

リソース

関連するドキュメント:

- Amazon EC2: Ensuring Idempotency
- The Amazon Builders' Library: 分散システムの課題
- The Amazon Builders' Library: 信頼性、動作の継続、一杯のコーヒー
- Amazon EventBridge とは
- What Is Amazon Simple Queue Service?
- Break up with your monolith
- Orchestrate Queue-based Microservices with AWS Step Functions and Amazon SQS
- Basic Amazon SQS architecture
- キューベースのアーキテクチャ

関連動画:

- AWS New York Summit 2019: Intro to Event-driven Architectures and Amazon EventBridge (MAD205)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (includes loose coupling, constant work, static stability)
- AWS re:Invent 2019: Moving to event-driven architectures (SVS308)
- AWS re:Invent 2019: Scalable serverless event-driven applications using Amazon SQS and Lambda (API304)
- AWS re:Invent 2019: Scalable serverless event-driven applications using Amazon SQS and Lambda
- AWS re:Invent 2022 - Designing event-driven integrations using Amazon EventBridge
- AWS re:Invent 2017: Elastic Load Balancing Deep Dive and Best Practices

REL04-BP03 継続動作を行う

負荷が急激に大きく変化すると、システム障害が発生することがあります。例えば、ワークロードで何千台ものサーバーのヘルスをモニタリングするヘルスチェックを実行する場合、毎回同じサイズのペイロード (現在の状態の完全なスナップショット) を送信しています。障害が発生しているサー

バーがなくても、またはそのすべてに障害が発生していても、ヘルスチェックシステムは、大規模で急激な変更なしに常に作業を行っています。

たとえば、ヘルスチェックシステムが 100,000 台のサーバーをモニタリングしている場合、通常のサーバー障害率が軽いときは、その負荷はわずかです。ただし、重大なイベントによってこれらのサーバーの半分が異常な状態になると、ヘルスチェックシステムは、通知システムを更新し、クライアントに状態を通知しようとして過負荷になるでしょう。したがって、ヘルスチェックシステムは現在の状態の完全なスナップショットを毎回送信する必要があります。サーバー 100,000 台 のヘルス状態 (それぞれ 1 ビットで表される) のペイロードは、12.5 KB にすぎません。サーバーに障害が発生していないか、またはすべてに発生しているかにかかわらず、ヘルスチェックシステムは定期的に作業を行っているため、大規模の急激な変化はシステムの安定性を脅かすものではありません。これは実際に Amazon Route 53 がエンドポイントのヘルスチェック (IP アドレスなど) によってエンドユーザーがどのようにルーティングされているかを調べる際の方法です。

このベストプラクティスが確立されていない場合のリスクレベル: 低

実装のガイダンス

- 負荷が急激に変化してシステム障害が発生しないように、継続動作を行います。
- 疎結合の依存関係を実装します。キューイングシステム、ストリーミングシステム、ワークフロー、ロードバランサーなどの依存関係は、緩やかに結合しています。疎結合は、コンポーネントの動作をそれに依存する他のコンポーネントから分離するのに役立ち、弾力性と俊敏性を高めます。
  - The Amazon Builders' Library: 信頼性、動作の継続、一杯のコーヒー
  - AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (ループを閉じ、発想を開く: 大小さまざまなシステムをコントロールする方法) ARC337 (継続動作を含む)
    - 100,000 台のサーバーをモニタリングする健康診断システムの例の場合、成功または失敗の数に関係なく、ペイロードサイズが一定になるように、ワークロードを設計します。

リソース

関連するドキュメント:

- Amazon EC2: べき等性の確保
- The Amazon Builders' Library: 分散システムの課題
- The Amazon Builders' Library: 信頼性、動作の継続、一杯のコーヒー

関連動画:

- AWS New York Summit 2019: Intro to Event-driven Architectures and Amazon EventBridge (MAD205) (イベント駆動型アーキテクチャーと Amazon EventBridge 入門)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (ループを閉じ、発想を開く: 大小さまざまなシステムをコントロールする方法) ARC337 (継続動作を含む)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (ループを閉じ、発想を開く: 大小さまざまなシステムをコントロールする方法) ARC337 (疎結合、継続動作、静的安定性を含む)
- AWS re:Invent 2019: Moving to event-driven architectures (SVS308) (イベント駆動型アーキテクチャへの移行)

REL04-BP04 すべての応答に冪等性を持たせる

べき等のサービスは、各リクエストが 1 回だけで完了することを約束します。そのため、同一のリクエストを複数回行っても、リクエストを 1 回行ったのと同じ効果しかありません。べき等サービスを使用すると、リクエストが誤って複数回処理されることを恐れる必要がなくなるため、クライアントが再試行を行いやすくなります。このために、クライアントはべき等性トークンを使用して API リクエストを発行できます。リクエストが繰り返される場合は常に同じトークンが使われます。べき等サービス API はトークンを使用して、リクエストが最初に完了したときに返された応答と同じ応答を返します。

分散システムでは、アクションを最大で 1 回 (クライアントがリクエストを 1 回だけ行う)、または少なくとも 1 回 (クライアントが成功を確認するまでリクエストを続ける) 実行するのは簡単です。ただし、アクションがべき等であることを保証することは困難です。つまり、 正確に 1 回だけ実行し、同一のリクエストを複数回行っても、リクエストを 1 回行うのと同じ効果を得るようにするということです。API でべき等性トークンを使用すると、サービスは、重複レコードや副作用を生むことなく、変更リクエストを 1 回または複数回受け取ることができます。

このベストプラクティスを活用しない場合のリスクレベル: ミディアム

実装のガイダンス

- すべての応答に冪等性を持たせます。べき等のサービスは、各リクエストが 1 回だけで完了することを約束します。そのため、同一のリクエストを複数回行っても、リクエストを 1 回行ったのと同じ効果しかありません。

- クライアントはべき等性トークンを使用して API リクエストを発行できます。リクエストが繰り返される場合は常に同じトークンが使われます。べき等サービス API はトークンを使用して、リクエストが最初に完了したときに返された応答と同じ応答を返します。
  - Amazon EC2: 冪等性の確保

リソース

関連するドキュメント:

- Amazon EC2: Ensuring Idempotency
- The Amazon Builders' Library: 分散システムの課題
- The Amazon Builders' Library: 信頼性、動作の継続、一杯のコーヒー

関連動画:

- AWS New York Summit 2019: Intro to Event-driven Architectures and Amazon EventBridge (MAD205)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (疎結合、継続動作、静的安定性を含む)
- AWS re:Invent 2019: Moving to event-driven architectures (SVS308)

障害を軽減または障害に耐えるために分散システムでの操作を設計する

分散システムは、サーバーやサービスなどのコンポーネントを相互接続するために通信ネットワークを利用しています。このネットワークでデータの損失やレイテンシーがあっても、ワークロードは確実に動作する必要があります。分散システムのコンポーネントは、他のコンポーネントやワークロードに悪影響を与えない方法で動作する必要があります。これらのベストプラクティスに従うことで、ワークロードはストレスや障害に耐え、より迅速に復旧し、そのような障害の影響を軽減できます。その結果、平均復旧時間 (MTTR) が向上します。

これらのベストプラクティスは、故障を防ぎ、平均故障間隔 (MTBF) を改善します。

ベストプラクティス

REL05-BP01 該当するハードな依存関係をソフトな依存関係に変換するため、グレイスフルデグラデーションを実装する

アプリケーションコンポーネントは、依存関係が使用できなくなっても、引き続きコア機能を実行する必要があります。少し古いデータ、代替データ、またはまったくデータを提供していない可能性があります。これにより、局所的な障害によるシステム全体の機能への影響を最小限に抑えながら、中心的なビジネス価値を実現できます。

期待される成果: コンポーネントの依存関係が異常な場合でも、コンポーネント自体は機能しますが、パフォーマンスが低下します。コンポーネントの故障モードは通常の動作と見なしてください。ワークフローは、このような障害が完全な障害につながらないように、あるいは少なくとも予測可能で回復可能な状態になるように設計する必要があります。

一般的なアンチパターン:

- 必要な中核的なビジネス機能が特定されていない。依存関係に障害が発生してもコンポーネントが機能することをテストしていません。
- エラーに関するデータを提供しない場合や、複数の依存関係のうち 1 つしか使用できず、結果の一部が返される場合もあります。
- トランザクションが部分的に失敗すると、一貫性のない状態になる。
- 中央パラメータストアにアクセスする代替手段がない。
- 更新に失敗した結果、その結果を考慮せずにローカルステートを無効化または空にする。

このベストプラクティスを活用するメリット: グレースフルデグラデーションを行うと、システム全体の可用性が向上し、障害が発生しても最も重要な機能の機能が維持されます。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

グレースフルデグラデーションを実装することで、依存関係の障害がコンポーネントの機能に与える影響を最小限に抑えることができます。コンポーネントが依存関係の障害を検出し、他のコンポーネントやカスタマーへの影響を最小限に抑える方法で回避するのが理想的です。

グレースフルデグラデーションを考慮した設計とは、依存関係の設計時に潜在的な障害モードを考慮することを意味します。障害モードごとに、コンポーネントのほとんどの機能、または少なくとも最も重要な機能を発信者または顧客に提供する方法を用意してください。これらの考慮事項は、テストや検証が必要な追加要件になる可能性があります。理想的には、1 つまたは複数の依存関係に障害が発生した場合でも、コンポーネントがコア機能を許容範囲内で実行できることが理想的です。

これは技術的な議論であると同時にビジネス上の議論でもあります。すべてのビジネス要件は重要であり、可能であれば満たす必要があります。ただし、すべてが満たされない場合に何が起こるかをたずねることは依然として理にかなっています。システムは可用性と一貫性を保つように設計できますが、1 つの要件を削除しなければならない状況では、どちらの要件がより重要でしょうか。 支払い処理については、一貫性があるかもしれません。リアルタイムアプリケーションの場合、可用性が高くなる可能性があります。カスタマー向けウェブサイトの場合、答えはカスタマーの期待するものによって異なる場合があります。

これが何を意味するかは、コンポーネントの要件と、そのコア機能と見なすべき内容によって異なります。以下はその例です。

- e コマースウェブサイトでは、パーソナライズされたレコメンデーション、最高ランクの商品、カスタマーの注文状況など、複数の異なるシステムからのデータがランディングページに表示される場合があります。上流システムの 1 つに障害が発生した場合でも、エラーページをカスタマーに表示するのではなく、他のシステムすべてを表示する方が理にかなっています。
- バッチ書き込みを実行するコンポーネントは、個々の操作のいずれかが失敗した場合でも、バッチの処理を続行できます。再試行メカニズムを実装するのは簡単なはずです。これは、どの操作が成功し、どの操作が失敗したか、なぜ失敗したかについての情報を呼び出し元に返すか、失敗したリクエストをデッドレターキューに入れて非同期再試行を実装することで実現できます。失敗した操作に関する情報も記録する必要があります。
- トランザクションを処理するシステムは、個々の更新がすべて実行されたか、まったく実行されないかを確認する必要があります。分散トランザクションでは、同じトランザクションの後の操作が失敗した場合に備えて、Saga パターンを使用して以前の操作をロールバックできます。ここでの中心的な機能は一貫性を維持することです。

- タイムクリティカルなシステムは、タイムリーに応答しない依存関係に対処できるはずです。このような場合は、サーキットブレーカーパターンを使用できます。依存関係からの応答がタイムアウトし始めると、システムは追加の呼び出しが行われないクローズ状態に切り替えることができます。
- アプリケーションはパラメータストアからパラメータを読み取ることができます。デフォルトのパラメータセットを使用してコンテナイメージを作成し、パラメータストアが利用できない場合にこれらを使用すると便利です。

なお、コンポーネントに障害が発生した場合の経路は検査が必要で、主要経路よりも大幅に簡潔でなければなりません。一般的に、フォールバック戦略は避けるべきです。

実装手順

外部依存関係と内部依存関係を特定します。どのような種類の障害が発生する可能性があるかを検討してください。障害発生時に上流と下流のシステムやカスタマーへの悪影響を最小限に抑える方法を考えてください。

依存関係の一覧と、失敗した場合に正常にデグレードする方法は次のとおりです。

1. 依存関係の部分的な障害: コンポーネントは、1 つのシステムへの複数の要求、または複数のシステムへの 1 つの要求のいずれかとして、下流システムに対して複数の要求を行うことができます。ビジネスの状況によっては、これに対するさまざまな処理方法が適切な場合があります (詳細については、実装ガイダンスの前述の例を参照してください)。
2. ダウンストリームシステムは、負荷が高いため、リクエストを処理できません。 ダウンストリームシステムへのリクエストが絶えず失敗している場合、再試行を続行しても意味がありません。これにより、既に過負荷になっているシステムに追加の負荷がかかり、回復が困難になる可能性があります。ここでは、ダウンストリームシステムへのコールの失敗を監視するサーキットブレーカーパターンを利用できます。大量のコールが失敗すると、ダウンストリームシステムへのリクエストの送信が停止され、ダウンストリームシステムが再び使用可能かどうかをテストするコールがたまにしか送信されません。
3. パラメータストアは使用できません: パラメータストアを変換するには、ソフト依存関係キャッシュを使用するか、コンテナイメージやマシンイメージに含まれる適切なデフォルトを使用できます。これらのデフォルトは最新の状態に保ち、テストスイートに含める必要があることに注意してください。
4. モニタリングサービスやその他の機能しない依存サービスは利用できません。 コンポーネントが断続的にログ、メトリクス、またはトレースを中央監視サービスに送信できない場合でも、通常どおりビジネス機能を実行するのが最善策です。メトリクスを長時間ログに記録したりプッシュ

したりしないことは、ほとんどの場合受け入れられません。また、ユースケースによっては、コンプライアンス要件を満たすために完全な監査エントリが必要になる場合があります。

5. リレーショナルデータベースのプライマリインスタンスが使用できない可能性があります。Amazon Relational Database Service は、ほとんどすべてのリレーショナルデータベースと同様に、プライマリライターインスタンスは 1 つしか設定できません。これにより、書き込みワークロードの単一障害点が生じ、スケーリングがより困難になります。これは、可用性を高めるためにマルチ AZ 構成を使用するか、スケーリングを向上させるために Amazon Aurora サーバーレス構成を使用することで部分的に軽減できます。可用性要件が非常に高い場合は、プライマリライターにまったく依存しない方が理にかなっています。読み取り専用のクエリには、リードレプリカを使用できます。これにより、冗長性が確保され、スケールアップだけでなくスケールアウトも可能です。書き込みは、例えば、Amazon Simple Queue Service キューにバッファリングできるため、プライマリが一時的に使用できなくなっても、カスタマーからの書き込み要求を引き続き受け付けることができます。

リソース

関連するドキュメント:

- Amazon API Gateway: スループット向上に向けた API リクエストのスロットリング
- CircuitBreaker (「Release It!」よりサーキットブレーカーをまとめたもの)
- AWS でのエラーの再試行とエクスポネンシャルバックオフ
- Michael Nygard「Release It! Design and Deploy Production-Ready Software」
- The Amazon Builders' Library: 分散システムでのフォールバックの回避
- The Amazon Builders' Library: 克服できないキューバックログの回避
- The Amazon Builders' Library: キャッシュの課題と戦略
- The Amazon Builders' Library: タイムアウト、再試行、ジッターによるバックオフ

関連動画:

- 再試行、バックオフ、ジッター: AWS re:Invent 2019: Introducing The Amazon Builders' Library (DOP328)

関連する例:

- Well-Architected ラボ: レベル 300: ヘルスチェックを実装し依存関係を管理して、信頼性を向上する

REL05-BP02 リクエストのスロットル

リクエストを制限して、予想外の需要の増加によるリソースの枯渇を緩和します。スロットリングレートを下回るリクエストは処理され、定義された制限を超えるリクエストは拒否され、リクエストがスロットリングされたことを示すメッセージが返されます。

期待される成果: 突然のカスタマートラフィックの増加、フラッディング攻撃、または再試行ストームによる大量のスパイクは、リクエストスロットリングによって軽減され、サポートされているリクエスト量の通常の処理をワークロードが継続できるようになります。

一般的なアンチパターン:

- API エンドポイントのスロットルは実装されていないか、予想される量を考慮せずにデフォルト値のままになっています。
- API エンドポイントは負荷テストされておらず、スロットリング制限もテストされていません。
- リクエストのサイズや複雑さを考慮せずにリクエストレートをスロットリングできます。
- 最大リクエストレートまたは最大リクエストサイズをテストしますが、両方を一緒にテストするわけではありません。
- リソースは、テストで設定したのと同じ制限にプロビジョニングされません。
- アプリケーション (A2A) API コンシューマーへの適用を目的とした使用プランは設定も検討もされていません。
- 水平方向にスケーリングするキューコンシューマーには、最大同時実行設定は設定されていません。
- IP アドレスごとのレート制限は実装されていません。

このベストプラクティスを活用するメリット: スロットル制限を設定したワークロードは、予期しない量のスパイクが発生しても、正常に動作し、受け入れられたリクエストの負荷を正常に処理できます。API やキューへのリクエストの急なスパイクや持続的なスパイクはスロットリングされ、リクエスト処理リソースを使い果たすことはありません。レート制限は、単一の IP アドレスまたは API コンシューマーからの大量のトラフィックがリソースを使い果たして他のコンシューマーに影響を与えないように、個々のリクエスタを制限します。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

サービスは、既知のキャパシティのリクエストを処理するように設計する必要があります。このキャパシティは、負荷テストによって確立できます。リクエストの到着率が制限を超えると、適切なレスポンスからリクエストがスロットリングされたことが通知されます。これにより、コンシューマーはエラーを処理して後で再試行できます。

サービスにスロットリングの実装が必要な場合は、トークンがリクエストにカウントされるトークンバケットアルゴリズムの実装を検討してください。トークンは 1 秒あたりのスロットルレートで補充され、リクエストごとに 1 つのトークンで非同期に空になります。

トークンバケットアルゴリズム。

Amazon API Gateway は、アカウントとリージョンの制限に従ってトークンバケットアルゴリズムを実装し、使用プランを使用してクライアントごとに構成できます。さらに、Amazon Simple Queue Service (Amazon SQS) および Amazon Kinesis は、リクエストをバッファリングしてリクエストレートをスムーズにし、対応可能なリクエストのスロットリングレートを高くすることができます。最後に、 AWS WAF を使用してレート制限を実装し、異常に高い負荷を発生させる特定の API コンシューマーを制限します。

実装手順

API のスロットリング制限を使用して API Gateway を設定し、制限を超えると `#429 Too Many Requests#(429 ##########)` エラーを返すことができます。AWS AppSync および API Gateway エンドポイントで AWS WAF を使用すると、IP アドレスごとにレート制限を有効にできます。さらに、システムが非同期処理に対応できる場合は、メッセージをキューまたはストリームに入

れてサービスクライアントへの応答を高速化できます。これにより、より高いスロットルレートにバーストできます。

非同期処理では、Amazon SQS を AWS Lambda のイベントソースとして設定すると、最大同時実行数を設定して、ワークロードやアカウント内の他のサービスに必要な、使用可能なアカウントの同時実行クォータを高いイベント率が消費することを回避できます。

API Gatewayトークンバケットのマネージド実装を提供していますが、API Gateway を使用できない場合は、サービス用のトークンバケットの言語固有のオープンソース実装 (「参考文献」の関連例を参照) を利用できます。

- リージョンごとのアカウントレベル、ステージごとの API、使用プランレベルごとの API キーでの API Gateway スロットリング制限 を理解して設定します。
- 以下の AWS WAF レート制限ルールを API Gateway および AWS AppSync エンドポイントに適用して、フラッドから保護し、悪意のある IP をブロックします。A2A コンシューマー向けの AWS AppSync API キーにレート制限ルールを設定することもできます。
- AWS AppSync API のレート制限よりも高度なスロットリング制御が必要かどうかを検討し、必要な場合は AWS AppSync エンドポイントの前に API Gateway を設定します。
- Amazon SQS キューが Lambda キューコンシューマーに対してアクティブとして設定されている場合は、最大同時実行数を サービスレベル目標を達成するのに十分な処理を行うが、他の Lambda 関数に影響を与える同時実行数の制限を消費しない値に設定します。Lambda でキューを使用する場合は、同じアカウントおよびリージョン内の他の Lambda 関数に予約された同時実行を設定することを検討してください。
- API Gateway と Amazon SQS または Kinesis へのネイティブサービス統合を使用して、リクエストをバッファリングします。
- API Gateway を使用できない場合は、言語固有のライブラリを調べて、ワークロード用のトークンバケットアルゴリズムを実装してください。サンプルセクションを確認して、適切なライブラリを見つけるために独自の調査を行ってください。
- 設定する予定の、または引き上げを許可する予定の制限をテストし、テストした制限を文書化します。
- テストで設定した上限を超えて制限を増やさないでください。制限を増やす場合は、増やす前に、プロビジョニングされたリソースが既にテストシナリオのものと同等かそれ以上であることを確認してください。

リソース

関連するベストプラクティス:

- REL04-BP03 継続動作を行う
- REL05-BP03 再試行呼び出しを制御および制限する

関連するドキュメント:

- Amazon API Gateway: スループット向上に向けた API リクエストのスロットリング
- AWS WAF: ルートベースのルールステートメント
- Amazon SQS をイベントソースとして使用する場合の AWS Lambda の最大同時実行数の導入
- AWS Lambda: 最大同時実行数

関連サンプル:

- 最も重要な 3 つの AWS WAF レートベースのルール
- Java Bucket4j
- Python トークンバケット
- ノードトークンバケット
- .NET システムスレッドのレート制限

関連動画:

- AWS AppSync を使用した GraphQL API セキュリティのベストプラクティスの実装

関連ツール:

- Amazon API Gateway
- AWS AppSync
- Amazon SQS
- Amazon Kinesis
- AWS WAF

REL05-BP03 再試行呼び出しを制御および制限する

エクスポネンシャルバックオフを使用して、各再試行の間隔を徐々に長くしてリクエストを再試行します。再試行間隔をランダム化するために、再試行間にジッターを導入します。最大再試行回数を制限します。

期待される成果: 分散ソフトウェアシステムの一般的なコンポーネントには、サーバー、ロードバランサー、データベース、DNS サーバーが含まれます。通常の操作では、これらのコンポーネントは一時的なエラーや限定的なエラーを含むリクエストに応答できます。また、再試行してもエラーが続くリクエストにも応答できます。クライアントがサービスにリクエストを行うと、そのリクエストはメモリ、スレッド、接続、ポート、またはその他の限られたリソースを含むリソースを消費します。再試行の制御と制限は、リソースを解放して消費を最小限に抑え、負荷がかかっているシステムコンポーネントに負荷がかからないようにするための戦略です。

クライアントのリクエストがタイムアウトになったり、エラーレスポンスが返されたりした場合は、再試行するかどうかを決定する必要があります。再試行する場合は、ジッターと最大再試行値によるエクスポネンシャルバックオフを行います。その結果、バックエンドのサービスとプロセスの負荷が軽減され、自己修復にかかる時間が短縮されるため、復旧が速くなり、リクエストサービスが正常に処理されます。

一般的なアンチパターン:

- エクスポネンシャルバックオフ、ジッター、最大再試行値を追加せずに再試行を実装します。バックオフとジッターは、同じ間隔で意図せずに調整された再試行による人為的なトラフィックのスパイクを防ぐのに役立ちます。
- その効果をテストしたり、再試行シナリオをテストせずに再試行が既に SDK に組み込まれていることを前提に再試行を実装します。
- 公開されている依存関係のエラーコードを理解できず、許可の欠如、設定エラー、または手動による介入なしでは解決できないと思われるその他の状態を示す明確な原因がある エラーを含め、すべてのエラーを再試行することになります。
- 根本的な問題を明らかにして対処できるように、繰り返し発生するサービス障害の監視や警告など、可観測性のプラクティスには触れていません。
- 組み込みまたはサードパーティの再試行機能で十分な場合は、カスタムの再試行メカニズムを開発します。
- アプリケーションスタックの複数のレイヤーで再試行すると、再試行が複雑になり、再試行の大混乱の中でさらにリソースを消費します。これらのエラーが依存しているアプリケーションの依存関係にどのように影響するかを必ず理解し、再試行は 1 つのレベルでのみ実装してください。

- べき等性を持たないサービスコールを再試行すると、結果が重複するなどの予期しない影響が発生します。

このベストプラクティスを活用するメリット: 再試行は、リクエストが失敗したときにクライアントが希望する結果を得るのに役立ちますが、必要な応答を得るまでにサーバーの時間を多く消費します。障害がまれな場合や一時的な場合は、再試行しても問題ありません。リソースの過負荷が原因で障害が発生した場合、再試行は事態を悪化させる可能性があります。クライアントの再試行にジッターを伴うエクスポネンシャルバックオフを追加することで、リソース過負荷が原因で障害が発生した場合でも、サーバーを回復できます。ジッターを使用すると、リクエストがスパイクに陥るのを防ぎ、バックオフによって通常のリクエスト負荷に再試行を追加することによる負荷のエスカレーションが軽減されます。最後に、メタステーブル障害の原因となるバックログが作成されないように、最大再試行回数または経過時間を設定することが重要です。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

再試行呼び出しを制御および制限します。エクスポネンシャルバックオフを使用して、徐々に長い間隔で再試行します。再試行間隔をランダム化するジッターを導入し、再試行の最大数を制限します。

一部の AWS SDK は、デフォルトで再試行とエクスポネンシャルバックオフを実装しています。ワークロードに該当する場合は、これらの組み込み AWS 実装を使用してください。べき等性を持たせて再試行することでクライアントの可用性が向上するサービスを呼び出すときも、同様のロジックをワークロードに実装します。タイムアウトの時間と、再試行をいつ停止するのかをユースケースに基づいて決めます。こうした再試行のユースケースに対応するテストシナリオを構築し、実行してください。

実装手順

- アプリケーションスタック内の最適なレイヤーを決定して、アプリケーションが依存するサービスの再試行を実装してください。
- 選択した言語に対してエクスポネンシャルバックオフとジッターを伴う実証済みの再試行戦略を実装している既存の SDK に注意し、独自の再試行実装を作成するよりもこれらを優先してください。
- 再試行を実装する前に、 サービスがべき等性を持っていることを 確認してください。再試行を実装したら、必ずテストを行い、本番環境で定期的に実行するようにしてください。

- AWS サービス API を呼び出すときは、 AWS SDK と AWS CLI を使用し、再試行設定オプションを理解してください。デフォルトがユースケースに適しているかどうかを判断し、テストし、必要に応じて調整します。

リソース

関連するベストプラクティス:

- REL04-BP04 すべての応答に冪等性を持たせる
- REL05-BP02 リクエストのスロットル
- REL05-BP04 フェイルファストとキューの制限
- REL05-BP05 クライアントタイムアウトを設定する
- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する

関連するドキュメント:

- AWS でのエラーの再試行とエクスポネンシャルバックオフ
- The Amazon Builders' Library: タイムアウト、再試行、ジッターによるバックオフ
- エクスポネンシャルバックオフとジッター
- べき等 API で再試行を安全に行う

関連サンプル:

- Spring Retry
- Resilience4j Retry

関連動画:

- 再試行、バックオフ、ジッター: AWS re:Invent 2019: Introducing The Amazon Builders’ Library (DOP328)

関連ツール:

- AWS SDK とツール: 再試行動作
- AWS Command Line Interface: AWS CLI 再試行

REL05-BP04 フェイルファストとキューの制限

サービスがリクエストに正常に応答できない場合は、すぐに失敗します。これにより、リクエストに関連付けられたリソースが解放され、リソースが不足した場合にサービスを復旧できます。フェイルファストは確立されたソフトウェア設計パターンであり、これを活用して信頼性の高いワークロードをクラウド上に構築できます。キューイングは、負荷をスムーズにし、非同期処理が許容できる場合にクライアントがリソースを解放できるようにする、確立されたエンタープライズ統合パターンでもあります。サービスが通常の状態では正常に応答できるが、リクエストのレートが高すぎると失敗する場合は、キューを使用してリクエストをバッファします。ただし、長いキューのバックログの蓄積は許可しないでください。クライアントが既に処理を停止している古いリクエストを処理する原因となる可能性があるためです。

期待される成果: システムにリソースの競合、タイムアウト、例外、またはグレー障害が発生してサービスレベル目標を達成できない場合、フェイルファスト戦略を使用するとシステムをより迅速に回復できます。トラフィックの急増を吸収する必要があり、非同期処理に対応できるシステムでは、バックエンドサービスへのリクエストをバッファリングするキューを使用して、クライアントがリクエストを迅速にリリースできるようにすることで信頼性を向上できます。リクエストをキューにバッファリングする際には、克服できないバックログを回避するためにキュー管理戦略が実装されます。

一般的なアンチパターン:

- メッセージキューを実装するが、システムに障害が発生したことを検出するデッドレターキュー (DLQ) やアラームを DLQ ボリュームに設定しない。
- キュー内のメッセージの経過時間を測定するのではなく、キューのコンシューマーが遅れたり、エラーが発生して再試行が発生したりするタイミングを把握するためのレイテンシーの測定です。
- 業務上の必要がなくなった場合に、これらのメッセージを処理する価値がない場合に、未処理のメッセージをキューから消去しない。
- 先入れ先出し (FIFO) キューを後入れ先出し (LIFO) キューに設定すると、クライアントのニーズにより適切に対応できます。例えば、厳密な順序付けが不要で、バックログ処理により新規リクエストや時間的制約のあるリクエストがすべて遅延し、その結果、すべてのクライアントでサービスレベル違反が発生するような場合です。
- 仕事の受け入れを管理してリクエストを内部キューに入れる API を公開する代わりに、内部キューをクライアントに公開します。
- 1 つのキューに多数の作業リクエストタイプをまとめると、リソース需要がリクエストタイプ全体に分散され、バックログの状態が悪化する可能性があります。
- 異なるモニタリング、タイムアウト、リソース割り当てが必要な場合でも、複雑なリクエストと単純なリクエストを同じキューで処理します。

- エラーを適切に処理できる上位レベルのコンポーネントに例外をバブリングするフェイルファストメカニズムをソフトウェアで実装するために、入力を検証したり、アサーションを使用しない。
- リクエストルーティングから障害のあるリソースを削除しない。特に、クラッシュや再起動、断続的な依存関係の障害、容量の低下、ネットワークのパケットロスなどにより、障害がグレーで成功と失敗の両方を示している場合。

このベストプラクティスを活用するメリット: すぐに失敗するシステムは、デバッグや修正が容易で、リリースが本稼働環境に公開される前にコーディングや構成の問題が明らかになることがよくあります。効果的なキューイング戦略を組み込んだシステムは、トラフィックの急増や断続的なシステム障害状態に対する回復力と信頼性が向上します。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

フェイルファスト戦略は、ソフトウェアソリューションにコード化することも、インフラストラクチャに構成することもできます。キューは、高速に障害が発生するだけでなく、システムコンポーネントを切り離してスムーズに負荷をかけるための単純でありながら強力なアーキテクチャ手法です。Amazon CloudWatch には、障害を監視して警告する機能があります。システムに障害が発生していることが判明したら、障害が発生したリソースからフェイルアウェイするなどの緩和策を講じることができます。システムが次の方法でキューを実装する場合 Amazon SQS およびその他のキューテクノロジーでは、負荷をスムーズにするために、キューのバックログやメッセージ消費の失敗を管理する方法を検討する必要があります。

実装手順

- プログラムによるアサーションや特定のメトリクスをソフトウェアに実装し、それらを使用してシステムの問題を明示的に警告します。Amazon CloudWatch は、アプリケーションログパターンと SDK 機器に基づいてメトリクスとアラームを作成するのに役立ちます。
- CloudWatchメトリクスとアラームを使用して、リソースに障害が発生して処理に遅延が発生したり、リクエストの処理が繰り返し失敗しないようにします。
- Amazon SQS を使用してリクエストを受け入れ、リクエストを内部キューに追加し、メッセージ生成クライアントに成功メッセージで応答する API を設計することで非同期処理を使用します。これにより、バックエンドキューのコンシューマーがリクエストを処理している間、クライアントはリソースを解放して他の作業に進むことができます。
- 現在とメッセージのタイムスタンプを比較することで、メッセージをキューから取り出すたびに CloudWatch メトリクスを生成し、キューの処理遅延を測定およびモニタリングします。

- 障害によってメッセージ処理が正常に行われなかった、またはサービスレベル契約の範囲内で処理できない量のトラフィックが急増した場合は、古いトラフィックや過剰なトラフィックをスピルオーバーキューに振り分けます。これにより、キャパシティに空きがあれば、新しい作業や古い作業を優先的に処理できます。この手法は LIFO 処理の近似値であり、すべての新規作業で通常のシステム処理が可能になります。
- 処理できないメッセージをバックログから後で調査して解決できる場所に移動するには、デッドレターキューまたはリドライブキューを使用します。
- 再試行するか、許容範囲内であれば、メッセージのタイムスタンプと現在を比較して、要求元のクライアントに関係のないメッセージは破棄して、古いメッセージを削除してください。

リソース

関連するベストプラクティス:

- REL04-BP02 疎結合の依存関係を実装する
- REL05-BP02 リクエストのスロットル
- REL05-BP03 再試行呼び出しを制御および制限する
- REL06-BP02 メトリクスを定義および計算する (集計)
- REL06-BP07 システムを通じたリクエストのエンドツーエンドのトレースをモニタリングする

関連するドキュメント:

- 乗り越えられないキューバックログの回避
- 速やかな失敗
- Amazon SQS キュー内のメッセージのバックログの増加を防ぐにはどうすればよいですか?
- Elastic Load Balancing: ゾーンシフト
- Amazon Route 53 アプリケーションリカバリコントローラ: トラフィックフェイルオーバーのルーティング制御

関連サンプル:

- エンタープライズ統合パターン: デッドレターチャネル

関連動画:

- AWS re:Invent 2022 - Operating highly available Multi-AZ applications (AWS re:Invent 2022 - 高可用性のマルチ AZ アプリケーションの運用)

関連ツール:

- Amazon SQS
- Amazon MQ
- AWS IoT Core
- Amazon CloudWatch

REL05-BP05 クライアントタイムアウトを設定する

接続とリクエストにタイムアウトを適切に設定し、体系的に検証します。また、デフォルト値には依存しないでください。これらはワークロードの詳細を認識していないためです。

期待される成果: クライアントのタイムアウトには、完了までに異常に時間がかかるリクエストを待つことに関連するクライアント、サーバー、およびワークロードにかかるコストを考慮する必要があります。タイムアウトの正確な原因を知ることはできないため、クライアントはサービスの知識を活用して、考えられる原因と適切なタイムアウトを予測する必要があります。

クライアント接続は、設定された値に基づいてタイムアウトします。タイムアウトが発生すると、クライアントはバックオフして再試行するか、 サーキットブレーカーを開くかを決定します。これらのパターンは、根本的なエラー状態を悪化させる可能性のあるリクエストの発行を回避します。

一般的なアンチパターン:

- システムタイムアウトまたはデフォルトタイムアウトを認識していない。
- 通常のリクエスト完了タイミングを認識していない。
- リクエストが完了するまでに異常に時間がかかる原因や、これらの完了を待つことによってクライアント、サービス、またはワークロードのパフォーマンスが低下する原因を認識していない。
- ネットワークに障害が発生して、タイムアウトに達したときだけリクエストが失敗する確率や、より短いタイムアウトを採用しないことでクライアントとワークロードのパフォーマンスにコストがかかることを認識していない。
- 接続とリクエストの両方のタイムアウトシナリオはテストされていません。
- タイムアウトの設定が高すぎると、待機時間が長くなり、リソースの使用率が高くなる可能性があります。

- タイムアウトの設定が低すぎると、人為的な障害が発生します。
- サーキットブレーカーや再試行などのリモート呼び出しのタイムアウトエラーを処理するパターンを見落としています。
- サービス呼び出しエラー率、遅延に関するサービスレベル目標、および遅延異常値のモニタリングは考慮していません。これらのメトリクスから、タイムアウトが積極的または許容範囲が広いかを判断できます。

このベストプラクティスを活用するメリット: リモート呼び出しのタイムアウトは、リモート呼び出しの応答が異常に遅い場合やタイムアウトエラーがサービスクライアントによって適切に処理される場合にリソースを節約できるように、タイムアウトを適切に処理するように設定され、システムがタイムアウトを適切に処理するように設計されています。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

サービス依存関係呼び出しに接続タイムアウトとリクエストタイムアウトの両方を設定します。またこの設定は、通常プロセス全体のすべての呼び出しにも行います。多くのフレームワークにはタイムアウト機能が組み込まれていますが、デフォルト値が無限であるか、サービス目標の許容範囲を超えているものもあるので注意してください。値が高すぎると、クライアントがタイムアウトの発生を待機している間もリソースが消費され続けるため、タイムアウトの有用性が低下します。値が小さすぎると、再試行されるリクエストが多くなりすぎるため、バックエンドのトラフィックが増加し、レイテンシーが高くなってしまいます。場合によっては、すべてのリクエストが再試行されることになるため、完全な機能停止につながる恐れもあります。

タイムアウト戦略を決定する際には、次の点を考慮してください。

- リクエストの内容、ターゲットサービスの障害、またはネットワークパーティションの障害により、リクエストの処理に通常よりも時間がかかる場合があります。
- 異常に高価なコンテンツを含むリクエストは、サーバーとクライアントのリソースを不必要に消費する可能性があります。この場合、これらのリクエストをタイムアウトさせて再試行しないことで、リソースを節約できます。また、サービスは、スロットルやサーバー側のタイムアウトにより、異常にコストが大きいコンテンツから身を守る必要があります。
- サービスの障害により異常に時間がかかるリクエストは、タイムアウトして再試行できます。リクエストと再試行のサービスコストを考慮する必要がありますが、原因が局所的な障害である場合は、再試行してもコストはかからず、クライアントリソースの消費量を削減できます。障害の性質によっては、タイムアウトによってサーバーリソースが解放されることもあります。

- リクエストまたはレスポンスがネットワークから配信されなかったために完了までに時間がかかるリクエストは、タイムアウトして再試行できます。リクエストまたはレスポンスが配信されなかったため、タイムアウトの長さに関係なく失敗に終わったことになります。この場合、タイムアウトしてもサーバーリソースは解放されませんが、クライアントリソースが解放され、ワークロードのパフォーマンスが向上します。

再試行やサーキットブレーカーなどの確立された設計パターンを活用して、タイムアウトをスムーズに処理し、フェイルファストアプローチをサポートします。AWS SDK そして AWS CLI 接続タイムアウトとリクエストタイムアウトの両方を設定でき、エクスポネンシャルバックオフとジッターによる再試行も可能です。AWS Lambda 関数はタイムアウトの設定をサポートしており、 AWS Step Functions を使用すると、AWS サービスおよび SDK との事前構築された統合を利用するローコードサーキットブレーカーを構築できます。AWS App Mesh Envoy はタイムアウトとサーキットブレーカー機能を提供します。

実装手順

- リモートサービス呼び出しのタイムアウトを設定し、組み込みの言語タイムアウト機能またはオープンソースのタイムアウトライブラリを活用してください。
- ワークロードが AWS SDK を使用して呼び出しを行う場合は、ドキュメントで言語固有のタイムアウト設定を確認してください。
  - Python
  - PHP
  - .NET
  - Ruby
  - Java
  - Go
  - Node.js
  - C++
- ワークロードで AWS SDK や AWS CLI コマンドを使用する場合、以下の AWS 設定デフォルト を設定し、デフォルトタイムアウト値を設定してください。(`connectTimeoutInMillis` そして `tlsNegotiationTimeoutInMillis`)。
- コマンドラインオプション cli-connect-timeout ### cli-read-timeout ###### AWS サービスへの 1 回限りの AWS CLI コマンドを制御します。

- リモートサービス呼び出しのタイムアウトをモニタリングし、エラーが続く場合はアラームを設定して、エラーシナリオにプロアクティブに対処できるようにします。
- タグ付けを CloudWatch Metrics cli-read-timeout CloudWatch 異常の検出 オンコールエラー率、レイテンシーに関するサービスレベル目標、レイテンシーの異常値から、過度にアグレッシブなタイムアウトや許容範囲のタイムアウトの管理に関するインサイトが得られます。
- タイムアウトを Lambda 関数に設定します。
- API Gateway クライアントは、タイムアウトを処理するときに独自の再試行を実装する必要があります。API Gateway は、 ダウンストリーム統合に対して 50 ミリ秒から 29 秒の統合タイムアウトをサポートし、 統合リクエストのタイムアウト時に再試行しません。
- タイムアウト時のリモート呼び出しを回避するには、 サーキットブレーカーを開くかを決定します パターンを実装します。呼び出しが失敗しないように回線を開き、呼び出しが正常に応答したら回線を閉じます。
- コンテナベースのワークロードの場合は、組み込みのタイムアウトとサーキットブレーカーを活用するための App Mesh Envoy 機能を確認してください。
- AWS Step Functions を使用して、リモートサービス呼び出し用のローコードサーキットブレーカーを構築します。特に、ワークロードを簡素化するために AWS ネイティブ SDK とサポートされている Step Functions 統合を呼び出す場合に使用します。

リソース

関連するベストプラクティス:

- REL05-BP03 再試行呼び出しを制御および制限する
- REL05-BP04 フェイルファストとキューの制限
- REL06-BP07 システムを通じたリクエストのエンドツーエンドのトレースをモニタリングする

関連するドキュメント:

- AWS SDK: 再試行とタイムアウト
- The Amazon Builders' Library: タイムアウト、再試行、ジッターによるバックオフ
- Amazon API Gateway クォータと重要事項
- AWS Command Line Interface: コマンドラインオプション
- AWS SDK for Java 2.x: API タイムアウトの設定

- 設定オブジェクトと設定リファレンスを使用した AWS Botocore
- AWS SDK for .NET: 再試行とタイムアウト
- AWS Lambda: Lambda 関数オプションの設定

関連サンプル:

- AWS Step Functions および Amazon DynamoDB を使用したサーキットブレーカーパターンの使用
- Martin Fowler: サーキットブレーカー

関連ツール:

- AWS SDK
- AWS Lambda
- Amazon SQS
- AWS Step Functions を使用すると
- AWS Command Line Interface

REL05-BP06 可能な限りシステムをステートレスにする

状態を必要としないシステム、または状態をオフロードするシステム (異なるクライアントリクエスト間にディスクやメモリ内のローカルに保存されたデータへの依存がない) にしてください。これにより、可用性に影響を与えることなく、サーバーをいつでも置き換えることができます。

ユーザーまたはサービスがアプリケーションと対話するとき、セッションを形成する一連のやりとりを頻繁に実行します。セッションは、ユーザーがアプリケーションを使用している間、リクエスト間で持続するユーザー固有のデータです。ステートレスアプリケーションは、以前のやりとりの知識を必要とせず、セッション情報を保存しません。

ステートレスな設計にすれば、あとは AWS Lambda や AWS Fargate などのサーバーレスコンピューティングサービスを利用できます。

サーバーの置き換えに加えて、ステートレスアプリケーションのもう 1 つの利点は、利用可能なコンピューティングリソース (EC2 インスタンスや AWS Lambda 関数など) がどのようなリクエストにも対応できるため、水平方向にスケーリングできることです。

このベストプラクティスを活用するメリット: ステートレスに設計されたシステムは水平スケーリングへの適応性が高いため、変動するトラフィックと需要に応じてキャパシティを追加したり削除したりできます。また、本質的に耐障害性に優れており、アプリケーション開発に柔軟性と俊敏性をもたらします。

このベストプラクティスが確立されていない場合のリスクレベル: 中

実装のガイダンス

アプリケーションをステートレスにします。ステートレスアプリケーションは、水平方向へのスケーリングが可能であり、個別ノードの障害に耐性があります。アーキテクチャ内の状態を維持するアプリケーションのコンポーネントを分析して理解します。これにより、ステートレス設計への移行で考えられる影響を評価できます。ステートレスアーキテクチャはユーザーデータを切り離し、セッションデータをオフロードします。各コンポーネントを個別にスケールし、変化するワークロードの需要に対応することでリソースの使用率を最適化できる柔軟性が得られます。

実装手順

- アプリケーション内のステートフルなコンポーネントを特定して理解します。
- ユーザーデータをコアアプリケーションロジックから分離して管理することで、データを切り離します。
  - Amazon Cognito は、アイデンティティプール、ユーザープール、Amazon Cognito Sync などの機能を使用して、ユーザーデータをアプリケーションコードから切り離すことができます。
  - AWS Secrets Manager を使用して、シークレットを安全で一元化された場所に保存することで、ユーザーデータを切り離すことができます。つまり、アプリケーションコードにシークレットを保存する必要がないため、安全性が高まります。
  - Amazon S3 を使用して、画像や文書などの大規模な非構造化データの保存を検討してください。アプリケーションは必要に応じてこのデータを取得できるため、メモリに保存する必要はありません。
  - Amazon DynamoDB を使用して、ユーザープロファイルなどの情報を保存します。アプリケーションでは、ほぼリアルタイムでこのデータをクエリできます。
- セッションデータをデータベース、キャッシュ、または外部ファイルにオフロードします。
  - Amazon ElastiCache、Amazon DynamoDB、Amazon Elastic File System (Amazon EFS)、Amazon MemoryDB for Redis は、セッションデータのオフロードに使用できる AWS のサービスの例です。
- 選択したストレージソリューションでは、どの状態とユーザーデータを持続しておく必要があるのかを特定した後、ステートレスアーキテクチャを設計します。

リソース

関連するベストプラクティス:

- REL11-BP03 すべてのレイヤーの修復を自動化する

関連するドキュメント:

- The Amazon Builders' Library: 分散システムでのフォールバックの回避
- The Amazon Builders' Library: 克服できないキューバックログの回避
- The Amazon Builders' Library: キャッシュの課題と戦略
- AWS のステートレスウェブ層のベストプラクティス

REL05-BP07 緊急レバーを実装する

緊急レバーは、ワークロードの可用性に対する影響を軽減できる迅速なプロセスです。

緊急レバーは、既知のテスト済みのメカニズムを使用して、コンポーネントや依存関係の動作を無効にしたり、制限したり、変更したりするためのものです。その効果として、想定外の需要増によるリソースの枯渇が原因となるワークロードの障害を軽減し、ワークロード内の重要ではないコンポーネントの障害の波及を抑制できます。

期待される成果: 緊急レバーを実装することで、ワークロードの重要なコンポーネントの可用性を維持するための、問題のないことが確認済みのプロセスを確立できます。緊急レバーが作動している間、ワークロードは意図的に性能を落とし (グレースフルデグラデーション)、ビジネスに不可欠な機能を引き続き実行します。グレースフルデグラデーションの詳細については、「REL05-BP01 該当するハードな依存関係をソフトな依存関係に変換するため、グレースフルデグラデーションを実装する」を参照してください。

一般的なアンチパターン:

- 重要ではない依存関係に障害が発生した場合に、主要ワークロードの可用性に影響が波及する。
- 重要ではないコンポーネントに障害が起きている間に、重要なコンポーネントの動作をテストまたは検証しない。
- 緊急レバーの作動または作動解除に関する決定的な基準が明確に定義されていない。

このベストプラクティスを活用するメリット: 緊急レバーを実装すると、想定外の需要の急増や重要ではない依存関係の障害に対処するためのプロセスを確立し、リゾルバーに提供できるため、ワークロードの重要なコンポーネントの可用性を向上させることができます。

このベストプラクティスが確立されていない場合のリスクレベル: 中

実装のガイダンス

- ワークロードの重要なコンポーネントを特定します。
- 重要ではないコンポーネントに障害が起きても耐えられるように、ワークロードの重要なコンポーネントを設計し、構築します。
- 重要ではないコンポーネントで障害が発生している最中に、重要なコンポーネントの動作を検証するためのテストを実施します。
- 緊急レバーの手続き開始の基準となる適切な指標やトリガーを定義し、監視します。
- 緊急レバーを構成する手順 (手動または自動) を定義します。

実装手順

- ワークロード内のビジネスクリティカルなコンポーネントを特定します。
  - ワークロードの技術的なコンポーネントをそれぞれ適切なビジネス機能にマッピングし、重要または非重要にランク分けします。Amazon の重要な機能と重要ではない機能の例については、「Any Day Can Be Prime Day: How Amazon.com Search Uses Chaos Engineering to Handle Over 84K Requests Per Second」を参照してください。
  - これは技術上の決定でもビジネス上の決定でもあり、組織やワークロードによって異なります。
- 重要ではないコンポーネントに障害が起きても耐えられるように、ワークロードの重要なコンポーネントを設計し、構築します。
  - 依存関係の分析では、想定される障害モードをすべて検討し、緊急レバーのメカニズムを通じて、ダウンストリームのコンポーネントも重要な機能を利用できるか検証します。
- 緊急レバーが作動している間に、重要なコンポーネントの動作を検証するためのテストを実施してください。
  - バイモーダル動作は防止してください。詳細については、REL11-BP05 静的安定性を使用してバイモーダル動作を防止する」を参照してください。
- 緊急レバーの手続き開始の基準となる指標を定義して監視し、警戒します。
  - ワークロードに応じて、監視対象として適切な指標を判断してください。指標の例としては、レイテンシーや、依存関係へのリクエストの失敗回数などが該当します。

- 緊急レバーを構成する手順 (手動または自動) を定義します。
  - これには、負荷制限、リクエストのスロットリング、グレースフルデグラデーションの実装などのメカニズムが使用される場合があります。

リソース

関連するベストプラクティス:

- REL05-BP01 該当するハードな依存関係をソフトな依存関係に変換するため、グレースフルデグラデーションを実装する
- REL05-BP02 リクエストのスロットル
- REL11-BP05 静的安定性を使用してバイモーダル動作を防止する

関連するドキュメント:

- 安全なハンズオフデプロイメントの自動化
- Any Day Can Be Prime Day: How Amazon.com Search Uses Chaos Engineering to Handle Over 84K Requests Per Second

関連動画:

- AWS re:Invent 2020: Reliability, consistency, and confidence through immutability

変更管理

ワークロードを信頼できる形で運用するには、ワークロードやその環境に対する変化を予測してそれに対応することが不可欠です。変更には、需要の急増などのワークロードに負荷がかかる変更や、機能のデプロイやセキュリティパッチの適用といった内部からの変更があります。

次のセクションでは、変更管理のベストプラクティスについて説明します。

トピック

ワークロードリソースをモニタリングする

ログとメトリクスは、ワークロードの状態についての洞察を得るための強力なツールです。ワークロードは、しきい値を超えたり重大なイベントが発生したりしたときに、ログとメトリクスがモニタリングされて通知が送信されるように構成できます。モニタリングにより、ワークロードは、低パフォーマンスのしきい値を超えたときや障害が発生したときにそれを認識できるため、それに応じて自動的に復旧できます。

モニタリングは、可用性の要件を満たしていることを確認する上で必要不可欠です。障害を効果的に検出するにはモニタリングが欠かせません。最悪の障害モードは「サイレント」障害です。この場合、機能は正常に機能しなくなっていますが、間接的なものを除き、検出する方法がありません。それにいち早く気付くのは、お客様ではなくてその顧客です。問題発生時にアラートを送信するのが、モニタリングの主な目的です。アラートは可能な限りシステムから分離する必要があります。サービスの中断によりアラートの機能が無効化されると、中断が長期化します。

AWS では、アプリケーションを複数のレベルで測定しています。これにより、各リクエスト、すべての依存関係、プロセス内の主要なオペレーションについて、レイテンシー、エラー率、可用性の記録を行っています。また、成功した操作のメトリクスも記録しています。これにより、切迫した問題が発生する前にそれを発見することができます。考慮するのは、平均レイテンシーだけではありません。99.9 パーセンタイルや 99.99 パーセンタイルなど、レイテンシーの外れ値により焦点を当てています。これは、1,000 または 10,000 のうちのたった 1 つのリクエストが遅かった場合でも、エクスペリエンスの満足度が低下するためです。また、平均値は許容できるかもしれませんが、リクエスト 100 件のうちの 1 件に極端なレイテンシーが発生すれば、トラフィックが増加したときに問題化します。

AWS のモニタリングは、次の 4 つの個別のフェーズで構成されています。

1. 生成 – ワークロードのすべてのコンポーネントをモニタリングする
2. 集計 – メトリクスを定義して計算する
3. リアルタイム処理とアラーム – 通知を送信し、応答を自動化する
4. ストレージと分析

ベストプラクティス

- REL06-BP01 ワークロードのすべてのコンポーネントをモニタリングする (生成)
- REL06-BP02 メトリクスを定義および計算する (集計)
- REL06-BP03 通知を送信する (リアルタイム処理とアラーム)
- REL06-BP04 レスポンスを自動化する (リアルタイム処理とアラーム)
- REL06-BP05 分析
- REL06-BP06 定期的にレビューを実施する
- REL06-BP07 システムを通じたリクエストのエンドツーエンドのトレースをモニタリングする

REL06-BP01 ワークロードのすべてのコンポーネントをモニタリングする (生成)

ワークロードのコンポーネントは、Amazon CloudWatch またはサードパーティー製ツールを使用してモニタリングします。AWS サービスを AWS Health ダッシュボードでモニタリングします。

フロントエンド、ビジネスロジック、ストレージ層など、ワークロードのすべてのコンポーネントをモニタリングする必要があります。主要なメトリクスと、必要に応じてそれをログから抽出する方法を定義し、対応するアラームイベントを起動させるためのしきい値を設定します。メトリクスがワークロードの重要業績評価指標 (KPI) に関連していることを確認し、メトリクスとログを使用して、サービス低下の早期警告サインを識別します。例えば、1 分間に正常に処理されたオーダー数など、ビジネス成果に関するメトリクスは、CPU 使用率などの技術的メトリクスより早く、ワークロード問題を示すことができます。AWS Health ダッシュボードは、AWS リソースの基盤となる AWS のサービスのパフォーマンスと可用性をパーソナライズして表示するために使用します。

クラウドでのモニタリングは新しい機会をもたらします。ほとんどのクラウドプロバイダーは、カスタマイズ可能なフックを開発して、ワークロードの複数のレイヤーをモニタリングする際に役立つインサイトを提供しています。Amazon CloudWatch などの AWS サービスは、統計的な機械学習アル

ゴリズムを応用して、システムとアプリケーションのメトリクスを継続的に分析し、正常なベースラインを決定し、最小限のユーザー介入で異常を表面化します。異常検出アルゴリズムでは、メトリクスの季節変動とトレンドの変化が考慮されます。

AWS では、豊富なモニタリングおよびログ情報を公開しており、これらを使用して、ワークロード固有のメトリクスと需要変化プロセスを定義し、機械学習の知識に関わらず、機械学習技法を適応させることができます。

さらに、すべての外部エンドポイントをモニタリングし、それらがベースとなる実装から独立していることを確認します。このアクティブモニタリングは、合成トランザクション (ユーザーカナリアと呼ばれることもありますが、canary デプロイと混同しないでください) で行うことができます。これは、ワークロードのクライアントによって実行されるアクションに相当する多数の一般的タスクを定期的に実行するというものです。これらのタスクは、短期間に保ち、テスト中にワークロードに負荷をかけすぎないようにしてください。Amazon CloudWatch Synthetics を使用すると、 Synthetics Canary を作成して、 エンドポイントと API をモニタリングできます。合成 Canary クライアントノードと AWS X-Ray コンソールを組み合わせて、選択した期間中にエラー、障害、スロットリング率で問題が発生している合成 Canary を特定することもできます。

期待される成果:

ワークロードのすべてのコンポーネントから重要なメトリクスを収集して使用し、ワークロードの信頼性と最適なユーザーエクスペリエンスを確保します。ワークロードがビジネス成果を達成していないことを検出した場合は、障害を迅速に宣言して、インシデントから復旧できます。

一般的なアンチパターン:

- ワークロードへの外部インターフェイスのみをモニタリングする。
- ワークロード固有のメトリクスを生成せず、ワークロードが使用している AWS から提供されるメトリクスにのみ依存する。
- ワークロードの技術的メトリクスを使用するだけで、ワークロードが貢献する非技術的な KPI に関するメトリクスをモニタリングしない。
- 本番トラフィックとシンプルなヘルスチェックに依存して、ワークロード状態をモニタリングし、評価する。

このベストプラクティスを確立するメリット: ワークロードのすべての階層でモニタリングすることで、ワークロードを構成するコンポーネントの問題をより迅速に予測し、解決できます。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

1. 可能な限りログを有効にします。 ワークロードのすべてのコンポーネントからモニタリングデータを取得する必要があります。S3 Access Logs など、追加のロギングをオンにして、ワークロードがワークロード固有のデータをログに記録できるようにします。Amazon ECS、Amazon EKS、Amazon EC2、Elastic Load Balancing、AWS Auto Scaling、Amazon EMR などのサービスから、CPU、ネットワーク I/O、およびディスク I/O の平均に関するメトリクスを収集します。把握 CloudWatch メトリクスを発行する AWS のサービス で、CloudWatch にメトリクスを発行する AWS のサービスのリストを確認できます。
2. デフォルトのメトリクスをすべてレビューし、データ収集にギャップがないか確認します。 すべてのサービスはデフォルトのメトリクスを生成します。デフォルトのメトリクスを収集することで、ワークロードのコンポーネント間の依存関係と、コンポーネントの信頼性とパフォーマンスがワークロードに及ぼす影響をより深く理解できます。また、独自のメトリクスを 作成して、AWS CLI または API を使用して CloudWatch に発行することもできます。 *** please leave this segment blank since the source does not make any sense in context ***
3. すべてのメトリクスを評価して、ワークロード内の各 AWS サービスに対してどのメトリクスでアラートを出すかを決定します。 ワークロードの信頼性に大きな影響を持つメトリクスのサブセットを選択することもできます。重要なメトリクスとしきい値に焦点を当てることで、アラート の数を絞り込み、偽陽性を最小化できます。
4. アラートを定義し、アラートが起動された後のワークロードの復旧プロセスを定義します。 アラートを定義することで、通知とエスカレーションを迅速に行い、インシデントからの復旧に必要なステップに従い、所定の目標復旧時間 (RTO) を満たすことができます。専用のインフラストラクチャで Amazon CloudWatch アラーム を使用すると、定義されたしきい値に基づいて自動ワークフローを起動し、回復手順を開始することができます。
5. 合成トランザクションを使用して、ワークロードの状態に関する関連データを収集することを検討しましょう。 合成モニタリングは、顧客と同じルートに従って同じアクションを実行するため、ワークロードに顧客のトラフィックがない場合でも、継続的にカスタマーエクスペリエンスを検証することが可能になります。また、 合成トランザクションを使用することで、顧客より先に問題を発見できます。

リソース

関連するベストプラクティス:

- REL11-BP03 すべてのレイヤーの修復を自動化する

関連するドキュメント:

- Getting started with your AWS Health Dashboard – Your account health (AWS Health ダッシュボードの使用開始 - アカウントのヘルス)
- CloudWatch メトリクスを発行する AWS のサービス
- Network Load Balancer のアクセスログ
- Application Load Balancer のアクセスログ
- Accessing Amazon CloudWatch Logs for AWS Lambda
- Amazon S3 サーバーアクセスのログ記録
- Classic Load Balancer のアクセスログの有効化
- Amazon S3 へのログデータのエクスポート
- CloudWatch エージェントを Amazon EC2 インスタンスにインストールする
- カスタムメトリクスの発行
- Amazon CloudWatch ダッシュボードの使用
- Using Amazon CloudWatch Metrics
- Canary の使用 (Amazon CloudWatch Synthetics)
- Amazon CloudWatch Logs とは

ユーザーガイド:

- 追跡の作成
- Amazon EC2 Linux インスタンスのメモリとディスクのメトリクスのモニタリング
- コンテナインスタンスでの CloudWatch Logs の使用
- VPC フローログ
- Amazon DevOps Guru とは
- 「AWS X-Ray とは何ですか。」

関連ブログ:

- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray

関連する例とワークショップ:

- AWS Well-Architected ラボ: 運用上の優秀性 - 依存関係のモニタリング
- The Amazon Builders' Library: 分散システムを装備して、運用の可視性を高める
- Observability workshop (可観測性ワークショップ)

REL06-BP02 メトリクスを定義および計算する (集計)

ログデータを保存し、必要に応じてフィルターを適用します。これにより、特定のログイベントのカウントや、ログイベントのタイムスタンプから計算されたレイテンシーなどのメトリクスを計算できます。

Amazon CloudWatch と Amazon S3 は、主要な集約レイヤーおよびストレージレイヤーとして機能します。AWS Auto Scaling や Elastic Load Balancing などの一部のサービスでは、クラスターまたはインスタンス全体の CPU 負荷または平均的なリクエストのレイテンシーについて、デフォルトのメトリクスが提供されます。VPC フローログや AWS CloudTrail などのストリーミングサービスの場合、イベントデータは CloudWatch Logs に転送されるため、メトリクスフィルターを定義して適用し、イベントデータからメトリクスを抽出する必要があります。これにより、時系列データが提供されます。これは、アラートをトリガーするために定義した CloudWatch アラームへの入力データとして機能します。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

- メトリクスを定義および計算します (集計)。特定のログイベントのカウントや、ログイベントのタイムスタンプから計算されたレイテンシーなどのメトリクスを計算するため、ログデータを保存し、必要に応じてフィルターを適用する
  - メトリクスフィルターは、CloudWatch Logs に送信されるログデータから検索する条件とパターンを定義します。CloudWatch Logs は、これらのメトリクスフィルターを使用して、ログデータを数値の CloudWatch メトリクスに変換し、グラフ化やアラームの設定を可能にします。
    - ログデータの検索およびフィルタリング
  - 信頼できるサードパーティーを使用してログを集計します。
    - サードパーティーの指示に従います。ほとんどのサードパーティー製品は、CloudWatch および Amazon S3 と統合されています。
  - 一部の AWS のサービスでは、ログを直接 Amazon S3 に発行できます。ログを Amazon S3 に保存することが主な要件であれば、追加のインフラを設定することなく、ログを生成するサービスに直接 Amazon S3 に送信させることが簡単にできます。

- Sending Logs Directly to Amazon S3 (Amazon S3 へのログの直接送信)

リソース

関連するドキュメント:

- Amazon CloudWatch Logs Insights Sample Queries
- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray
- 1 つの可観測性ワークショップ
- ログデータの検索およびフィルタリング
- Sending Logs Directly to Amazon S3 (Amazon S3 へのログの直接送信)
- The Amazon Builders' Library: 分散システムを装備して、運用の可視性を高める

REL06-BP03 通知を送信する (リアルタイム処理とアラーム)

組織は、潜在的な問題を検出すると、その問題に迅速かつ効果的に対応するために、適切な担当者とシステムにリアルタイムの通知とアラートを送信します。

期待される成果: サービスとアプリケーションのメトリクスに基づいて関連するアラームを設定することで、運用にかかわるイベントに迅速に対応できます。アラームのしきい値を超えると、適切な担当者とシステムに通知され、根本的な問題に対処できます。

一般的なアンチパターン:

- アラームのしきい値を過度に高く設定し、重要な通知が送信されなくなる。
- アラームのしきい値を低くしすぎたことにより、過剰な通知のノイズが原因で重要なアラートへの対処が行われなくなる。
- 使用率が変わってもアラームとそのしきい値を更新しない。
- 自動アクションで対処するのが最適なアラームに対して、自動アクションを生成する代わりに担当者に通知を送信することで、余計な通知が送信されてしまう。

このベストプラクティスを活用するメリット: 適切な担当者とシステムにリアルタイムの通知とアラートを送信することで、問題を早期に検出し、運用にかかわるインシデントに迅速に対応できます。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

アプリケーションの可用性に影響を与え、自動対応のトリガーとなる可能性のある問題を検出しやすくするために、ワークロードにはリアルタイム処理とアラーム機能が備わっている必要があります。組織は、重要なイベントが発生したり、メトリクスがしきい値を超えたりしたときに通知を受け取ることができるよう、定義されたメトリクスを使用してアラートを作成することで、リアルタイムの処理とアラームの発行を実施できます。

Amazon CloudWatch では、静的しきい値、異常検出、およびその他の基準に基づくCloudWatch アラームを使用して、 メトリクス アラームと複合アラームを作成できます。CloudWatch を使用して設定できるアラームの種類の詳細については、 CloudWatch ドキュメントのアラームに関するセクションを参照してください。

CloudWatch ダッシュボードを使用して、 チーム向けに AWS リソースのメトリクスとアラートをカスタマイズ表示できます。CloudWatch コンソールのカスタマイズ可能なホームページでは、複数のリージョンのリソースを 1 つのビューでモニタリングできます。

アラームは、 Amazon SNS トピックへの通知の送信、 Amazon EC2 アクションまたは Amazon EC2 Auto Scaling アクションの実行、 OpsItem または AWS Systems Manager のインシデントの作成など、1 つまたは複数のアクションを実行できます。

Amazon CloudWatch は Amazon SNS を使用して、アラームの状態が変化したときに通知を送信し、パブリッシャー (プロデューサー) からサブスクライバー (コンシューマー) にメッセージを配信します。Amazon SNS 通知設定の詳細については、 「Amazon SNS の設定」を参照してください。

CloudWatch は、 CloudWatch アラームが作成、更新、削除される、 またはその状態が変化するたびに EventBridge イベントを送信します。こうしたイベントで EventBridge を使用して、アラームの状態が変わるたびに通知したり、 Systems Manager Automation を使用してアカウント内のイベントを自動的にトリガーしたりするなどのアクションを実行するルールを作成できます。

EventBridge を使うべき場合と Amazon SNS を使うべき場合

EventBridge と Amazon SNS はどちらもイベント駆動型アプリケーションの開発に使用できます。どちらを選ぶかは、具体的なニーズによって異なります。

Amazon EventBridge は、自社アプリケーション、SaaS アプリケーション、AWS のサービスからのイベントに反応するアプリケーションを構築する場合に推奨されます。EventBridge は、サードパーティーの SaaS パートナーと直接統合できる唯一のイベントベースのサービスです。また、EventBridge では、デベロッパーが自分のアカウントにリソースを作成しなくても、200 を超える AWS サービスからイベントを自動的に取り込むことができます。

EventBridge では、定義済みの JSON ベースの構造がイベントに使用されており、ターゲットに転送するイベントを選択する際に イベント本文全体に適用されるルールを作成できます。EventBridge は現在、 AWS Lambda、Amazon SQS、Amazon SNS、Amazon Kinesis Data Streams、 Amazon Data Firehose など 20 を超える AWS のサービスをターゲットとしてサポートしています。

Amazon SNS は、高いファンアウトを必要とするアプリケーション (数千または数百万のエンドポイント) に推奨されます。よく見られるパターンは、お客様が Amazon SNS をルールのターゲットとして使用し、必要なイベントをフィルタリングして複数のエンドポイントに分散させるというものです。

メッセージは構造化されておらず、どのような形式でもかまいません。Amazon SNS では Lambda、Amazon SQS、HTTP/S エンドポイント、SMS、モバイルプッシュ、メールの 6 種類のターゲットへのメッセージ転送をサポートしています。Amazon SNS の通常のレイテンシーは 30 ミリ秒未満です。AWS のさまざまなサービス (Amazon EC2、Amazon S3、 Amazon RDS など 30 以上のサービス) で、Amazon SNS メッセージを送信するようにサービスを設定できます。

実装手順

1. Amazon CloudWatch アラームを 使用してアラームを作成します。
    a. メトリクスアラームは、単一の CloudWatch メトリクス、または CloudWatch メトリクスに依存する式をモニタリングします。アラームは、メトリクスまたは式の値としきい値との比較に基づいて、複数の時間間隔にわたって 1 つまたは複数のアクションを開始します。アクションは、 Amazon SNS トピックへの通知の送信、 Amazon EC2 アクションまたは Amazon EC2 Auto Scaling アクションの実行、 OpsItem または は AWS Systems Manager のインシデントの作成などで構成されます。
    b. 複合アラームは、作成した他のアラームのアラーム条件を考慮するルール式で構成されます。複合アラームは、すべてのルール条件が満たされた場合にのみアラーム状態になります。複合アラームのルール式で指定されるアラームには、メトリクスアラームや追加の複合アラームを含めることができます。複合アラームは、その状態が変更された場合にAmazon SNS 通知を送信でき、アラーム状態になった場合に Systems Manager の OpsItems を作成できる場合、 または インシデントを作成できますが、 Amazon EC2 または Auto Scaling アクションは実行できません。
2. Amazon SNS 通知を設定します。CloudWatch アラームを作成する際には、アラームの状態が変化したときに通知を送信する Amazon SNS トピックを含めることができます。
3. 指定された CloudWatch アラームと一致する ルールを EventBridge に作成します。各ルールは、Lambda 関数を含む複数のターゲットをサポートします。例えば、使用可能なディスク容量が少なくなったときに起動するアラームを定義できます。このアラームにより、領域をクリー

ンアップする Lambda 関数が EventBridge ルールを介してトリガーされます。EventBridge ターゲットの詳細については、「EventBridge ターゲット」を参照してください。

リソース

関連する Well-Architected のベストプラクティス:

- REL06-BP01 ワークロードのすべてのコンポーネントをモニタリングする (生成)
- REL06-BP02 メトリクスを定義および計算する (集計)
- REL12-BP01 プレイブックを使用して障害を調査する

関連するドキュメント:

- Amazon CloudWatch
- CloudWatch Logs のインサイト
- Amazon CloudWatch アラームの使用
- Amazon CloudWatch ダッシュボードの使用
- Amazon CloudWatch メトリクスを使用する
- Amazon SNS 通知の設定
- CloudWatch 異常の検出
- CloudWatch Logs データの保護
- Amazon EventBridge
- Amazon Simple Notification Service

関連動画:

- reinvent 2022 observability videos
- AWS re:Invent 2022 - Amazon におけるオブザーバビリティのベストプラクティス

関連する例:

- One Observability ワークショップ
- Amazon EventBridge to AWS Lambda with feedback control by Amazon CloudWatch Alarms

REL06-BP04 レスポンスを自動化する (リアルタイム処理とアラーム)

自動化を使用して、イベントが検出されたときにアクションを実行します (例えば、障害が発生したコンポーネントを交換します)。

アラームの自動リアルタイム処理が実装されているため、アラームがトリガーされたときにシステムが迅速に是正措置を講じ、障害やサービスの低下を防ぐことができます。アラームへの自動対応には、障害が起きたコンポーネントの交換、コンピューティングキャパシティの調整、正常なホスト、アベイラビリティーゾーン、その他のリージョンへのトラフィックのリダイレクト、オペレーターへの通知などがあります。

期待される成果: リアルタイムのアラームが特定され、アラームの自動処理が設定され、サービスレベル目標とサービスレベルアグリーメント (SLA) を達成するための適切なアクションが呼び出されます。自動処理は、単一コンポーネントの自己修復アクティビティからサイト全体のフェイルオーバーまで多岐にわたります。

一般的なアンチパターン:

- 主要なリアルタイムアラームの明確なインベントリまたはカタログがない。
- 重大なアラームへの自動対応 (例えば、コンピューティングが枯渇しそうになると、オートスケーリングが行われる) が欠如している。
- アラームへの対応が矛盾している。
- オペレーターがアラート通知を受け取ったときに従うべき標準作業手順書 (SOP) がない。
- 構成変更がモニタリングされていない。構成変更が検出されないと、ワークロードのダウンタイムが生じる可能性があります。
- 意図しない構成変更を取り消す戦略がない。

このベストプラクティスを活用するメリット: アラーム処理を自動化することで、システムの回復力を向上させることができます。システムが自動的に是正措置を講じるため、人が介入することでミスが生じやすい手作業を減らすことができます。ワークロードの可用性の目標を達成し、サービスの中断を低減します。

このベストプラクティスが確立されていない場合のリスクレベル: 中

実装のガイダンス

アラートを効果的に管理し、対応を自動化するには、重要度と影響に基づいてアラートを分類し、対応手順を文書化し、対応計画を立ててからタスクをランク付けします。

特定のアクション (たいていはランブックに詳細が記載されている) が必要なタスクを特定し、ランブックとプレイブックをすべて調べて、どのタスクを自動化できるか判断します。アクションを定義できる場合、たいていは自動化できます。アクションを自動化できない場合は、手作業による手順を SOP に記録し、オペレーターにその手順の訓練をします。手作業のプロセスは継続的に見直し、アラートへの対応を自動化する計画を立て、実践できる余地がないか検討してください。

実装手順

1. アラームのインベントリを作成する: すべてのアラームのリストを入手するには、AWS CLI で Amazon CloudWatch コマンド `describe-alarms` を使用できます。設定したアラームの数によっては、ページ分割を使用して、呼び出しごとにアラームのサブセットを取得しなければならない場合があります。または、AWS SDK を使用して API を呼び出し、アラームを取得することもできます。
2. すべてのアラームを文書化する: ランブックを更新し、すべてのアラームとそのアクションを (手作業でも自動処理でも) 記録します。AWS Systems Manager には、定義済みのランブックが用意されています。ランブックの詳細については、「独自のランブックの作成」を参照してください。ランブックの内容の表示方法については、「View runbook content」を参照してください。
3. アラームアクションを設定して管理する: アクションが必要なアラームについては、CloudWatch SDK を使用して自動アクションを指定します。例えば、アラームに応じてアクションを作成して有効にする、またはアラームに応じてアクションを無効にする形で、CloudWatch アラームに基づいて Amazon EC2 インスタンスの状態を自動的に変更できます。

   また、Amazon EventBridge を使用して、アプリケーションの可用性の問題やリソースの変更といったシステムイベントに自動的に対応することもできます。ルールを作成し、関心のあるイベントと、イベントがルールに合致したときに実行するアクションを指定できます。自動的に開始できるアクションには、AWS Lambda 関数の呼び出し、Amazon EC2 Run Command の呼び出し、イベントの Amazon Kinesis Data Streams へのリレーなどがあります。「EventBridge を使用してAmazon EC2 を自動化する」を参照してください。
4. 標準作業手順書 (SOP): アプリケーションコンポーネントに基づいて、AWS Resilience Hub は複数の SOP テンプレートを推奨します。これらの SOP を使用して、アラートが発生した場合にオペレーターが従うべきプロセスをすべて文書化できます。また、Resilience Hub のレコメンデーションに基づいて SOP を作成することもできます。その場合は、回復ポリシーを関連付けた Resilience Hub のアプリケーションと、そのアプリケーションに対する回復力評価の履歴が必要です。SOP のレコメンデーションは、回復力の評価を受けて作成されます。

   Resilience Hub は Systems Manager と連携して、SOP のひな型として参考になる多数の SSM ドキュメントを提供し、SOP の手順を自動化します。例えば、Resilience Hub は、自動化に関する

既存の SSM ドキュメントに基づいて、ディスク容量を増量するための SOP を提案する場合があります。

5. Amazon DevOps Guru:を使用して自動化アクションを実行する: Amazon DevOps Guru は、アプリケーションリソースに異常な挙動がないか自動的にモニタリングし、的を絞ったレコメンデーションを提供することにより、問題の特定を速めて修復時間を短縮します。DevOps Guru では、Amazon CloudWatch のメトリクス、AWS Config、AWS CloudFormation、AWS X-Ray など、複数のソースからの運用データのストリームをほぼリアルタイムで監視できます。また、DevOps Guru を使用して OpsCenter で OpsItems を自動作成し、イベントを EventBridge に送信してさらに自動化することもできます。

リソース

関連するベストプラクティス:

- REL06-BP01 ワークロードのすべてのコンポーネントをモニタリングする (生成)
- REL06-BP02 メトリクスを定義および計算する (集計)
- REL06-BP03 通知を送信する (リアルタイム処理とアラーム)
- REL08-BP01 デプロイなどの標準的なアクティビティにランブックを使用する

関連するドキュメント:

- AWS Systems Manager Automation
- AWS リソースからのイベントでトリガーする EventBridge ルールの作成
- 1 つの可観測性ワークショップ
- The Amazon Builders' Library: 分散システムを装備して、運用の可視性を高める
- What is Amazon DevOps Guru?
- オートメーションドキュメント (プレイブック) の使用

関連動画:

- AWS re:Invent 2022 - Observability best practices at Amazon
- AWS re:Invent 2020: Automate anything with AWS Systems Manager
- Introduction to AWS Resilience Hub
- Create Custom Ticket Systems for Amazon DevOps Guru Notifications

- Enable Multi-Account Insight Aggregation with Amazon DevOps Guru

関連する例:

- 「Reliability」ワークショップ
- Amazon CloudWatch and Systems Manager Workshop

REL06-BP05 分析

ログファイルとメトリクスの履歴を収集し、これらを分析して、幅広いトレンドとワークロードの洞察が得られます。

Amazon CloudWatch Logs Insights は、 シンプルかつ強力なクエリ言語をサポートし、 ログデータの分析に使用できます。Amazon CloudWatch Logs ではさらに、シームレスにデータを Amazon S3 に送ってデータを使用したり、または Amazon Athena に送ってデータをクエリしたりできるサブスクリプションもサポートしています。豊富な種類のフォーマットのクエリがサポートされています。把握 サポートされる SerDes とデータ形式 詳細については、Amazon Athena ユーザーガイドを参照してください。巨大なログファイルセットの分析では、Amazon EMR クラスターを実行してペタバイト規模の分析を実行できます。

集計、処理、保存、分析を実行できる多数のツールが AWS パートナーや
サードパーティによって提供されています。このようなツールには、New
Relic、Splunk、Loggly、Logstash、CloudHealth、Nagios などがあります。ただし、システムやアプリケーションログの外で行うデータ生成は各クラウドプロバイダーに固有であり、また多くの場合サービスごとに固有です。

モニタリングプロセスで見落とされがちな点は、データ管理です。モニタリングのためのデータ保存要件を決定し、それに応じたライフサイクルポリシーを適用する必要があります。Amazon S3 は S3 バケットレベルのライフサイクル管理をサポートしています。このライフサイクル管理には、バケット内のパスごとに異なる管理方法を適用できます。ライフサイクルの最終段階では、データを Amazon S3 Glacier に移行して長期保存し、保存期間の終了後には期限切れにすることができます。S3 Intelligent-Tiering ストレージクラスは、パフォーマンスへの影響や運用のオーバーヘッドなしに、データを最も費用対効果の高いアクセス階層に自動的に移動することにより、コストを最適化できるように設計されています。

このベストプラクティスを活用しない場合のリスクレベル: ミディアム

実装のガイダンス

- CloudWatch Logs Insights を使用すると、Amazon CloudWatch Logs でログデータをインタラクティブに検索して分析できます。
  - CloudWatch Logs Insights を使用したログデータの分析
  - Amazon CloudWatch Logs Insights Sample Queries
- 使用できる場合は、Amazon CloudWatch Logs を使用してログを Amazon S3 に送信するか、Amazon Athena を使用してデータをクエリします。
  - Athena を使用して Amazon S3 サーバーのアクセスログを分析するにはどうすればよいですか?
    - サーバーアクセスログバケットの S3 ライフサイクルポリシーを作成します。ライフサイクルポリシーを設定して、定期的にログファイルを削除します。そうすることで、Athena が各クエリについて分析するデータ量が削減されます。
      - S3 バケットのライフサイクルポリシーを作成する方法

リソース

関連するドキュメント:

- Amazon CloudWatch Logs Insights Sample Queries
- CloudWatch Logs Insights を使用したログデータの分析
- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray
- S3 バケットのライフサイクルポリシーを作成する方法
- Athena を使用して Amazon S3 サーバーのアクセスログを分析するにはどうすればよいですか?
- 1 つの可観測性ワークショップ
- The Amazon Builders' Library: 分散システムを装備して、運用の可視性を高める

REL06-BP06 定期的にレビューを実施する

ワークロードモニタリングがどのように実装されているかを頻繁に確認し、重要なイベントや変更に基づいて更新します。

効果的なモニタリングは、主要なビジネスメトリクスが原動力になります。ビジネスの優先順位が変化したときに、メトリクスがワークロードに確実に対応できるようにします。

モニタリングを監査することで、アプリケーションがどのタイミングで可用性の目標を満たしているかを確実に把握できます。根本原因の分析には、障害発生時に何が起こったかを発見する機能が必要です。AWS は、インシデント時にサービスの状態を追跡できるサービスを提供しています。

- Amazon CloudWatch Logs: このサービスにログを保存してその内容を調査できます。
- Amazon CloudWatch Logs Insights: 数秒で大量のログを分析できるフルマネージドサービスです。高速でインタラクティブなクエリと視覚化が行えます。
- AWS Config: さまざまな時点でどの AWS インフラストラクチャが使用されているかを確認できます。
- AWS CloudTrail: どの AWS API が、いつどのプリンシパルに呼び出されたかを確認できます。

AWS では、週に一度のミーティングを実施して、 運用パフォーマンスをレビューし、 学んだ教訓をチーム間で共有しています。AWS には多数のチームが存在するため、 私たちは The Wheel を作成し、 ワークロードをランダムに選んで確認できるようにしました。運用パフォーマンスのレビューと知識の共有を定期的に行うことで、運用チームのパフォーマンスを向上させることができます。

一般的なアンチパターン:

- デフォルトのメトリクスのみを収集する。
- モニタリング戦略を設定し、見直さない。
- 主要な変更がデプロイされる際に、モニタリングについて話し合わない。

このベストプラクティスを活用するメリット: モニタリングを定期的にレビューすることで、予期される問題が実際に発生したときに通知に反応する代わりに、潜在的な問題を予測できるようになります。

このベストプラクティスを活用しない場合のリスクレベル: ミディアム

実装のガイダンス

- ワークロード用に複数のダッシュボードを作成します。主要なビジネスメトリクスと、使用状況の変化に応じて予測されるワークロードの状態に最も関連性があるものとして特定した技術メトリクスを含む最上位のダッシュボードが必要です。また、検査が可能なさまざまなアプリケーション層や依存関係のダッシュボードも必要があります。
  - Amazon CloudWatch ダッシュボードの使用

- ワークロードダッシュボードの定期的なレビューをスケジュールし、実施します。ダッシュボードの定期的な検査を行います。検査する深度に応じて異なる頻度にすることができます。
  - メトリクスの傾向を検査します。メトリクス値と履歴値を比較して、調査が必要なものを示唆している可能性がある傾向があるかどうかを確認します。これには、レイテンシーの増加、主要なビジネス機能の減少、失敗レスポンスの増加などがあります。
  - メトリクスの外れ値/異常を検査します。平均値または中央値は、外れ値と異常値を覆い隠すことがあります。期間中の最大値と最低値を調べ、極端なスコアの原因を調査します。これらの原因の排除を続行しながら、極値の定義を低くしていくことで、ワークロードパフォーマンスの一貫性を継続して向上させることができます。
  - 行動の急変を探します。メトリクスの数量または方向性の突然の変化は、アプリケーションに変更があったこと、または追跡するためにさらなるメトリクスを追加する必要がある外部要因があることを示唆している可能性があります。

リソース

関連するドキュメント:

- Amazon CloudWatch Logs Insights Sample Queries
- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray
- 1 つの可観測性ワークショップ
- The Amazon Builders' Library: 分散システムを装備して、運用の可視性を高める
- Amazon CloudWatch ダッシュボードの使用

REL06-BP07 システムを通じたリクエストのエンドツーエンドのトレースをモニタリングする

サービスコンポーネントで処理されるリクエストをトレースすることで、製品チームではより簡単に問題の分析とデバッグを行い、パフォーマンスを向上させることができます。

期待される成果: すべてのコンポーネントを網羅的にトレースできるワークロードは、デバッグが容易で、根本原因の発見を簡略化することで、エラーやレイテンシーの 解決までの平均時間 (MTTR) を改善します。エンドツーエンドのトレースによって影響を受けるコンポーネントを検出し、エラーやレイテンシーの根本原因の詳細調査にかかる時間を短縮できます。

一般的なアンチパターン:

- トレースは一部のコンポーネントに使用されますが、すべてのコンポーネントで使用されるわけではありません。例えば、AWS Lambda のトレースを行わない場合は、ワークロードの急増でのコールドスタートが原因で生じたレイテンシーを明確に把握できない可能性があります。
- Synthetic Canaries やリアルユーザーモニタリング (RUM) には、トレースは設定されていません。Canary や RUM を使用しない場合、クライアントインタラクションのテレメトリがトレース分析から除外され、パフォーマンスプロファイルが不完全な状態になります。
- ハイブリッドワークロードには、クラウドネイティブとサードパーティのトレースツールの両方が含まれていますが、単一のトレースソリューションを選択し、完全に統合する手段は講じられていません。選択したトレースソリューションに基づいて、クラウドネイティブのトレース SDK を使用して、クラウドネイティブではないコンポーネントを測定するか、サードパーティツールを使用してクラウドネイティブのトレーステレメトリを取り込むように設定する必要があります。

このベストプラクティスを活用するメリット: 開発チームでは、問題についてアラートを受けることで、ロギング、パフォーマンス、障害に対するコンポーネントごとの相関関係など、システムコンポーネントの相互作用の全体像を把握できます。トレースによって根本原因を視覚的に把握しやすくなるため、根本原因の究明に費やす時間を短縮できます。チームではコンポーネントの相互作用を詳細に理解することで、問題を解決する際に、より適切で迅速な意思決定を行うことができます。システムトレースの分析によって、ディザスタリカバリ (DR) フェイルオーバーの開始時期、自己修復戦略を実行する場所などの意思決定を改善することで、最終的にサービスに対する顧客満足度の向上につながります。

このベストプラクティスを活用しない場合のリスクレベル: 中

実装のガイダンス

分散アプリケーションを運用するチームでは、トレースツールを使用して相関識別子を設定し、リクエストのトレースを収集して、接続されたコンポーネントのサービスマップを作成できます。サービスクライアント、ミドルウェアゲートウェイ、イベントバス、コンピューティングコンポーネント、キーバリューストアやデータベースを含むストレージなど、すべてのアプリケーションコンポーネントをリクエストトレースに含める必要があります。エンドツーエンドのトレース設定に Synthetic Canaries とリアルユーザーモニタリングを組み込んで、リモートクライアントとのやり取りやレイテンシーを測定することで、サービスレベル契約や目標に対するシステムのパフォーマンスを正確に評価できます。

そのため、 AWS X-Ray および Amazon CloudWatch アプリケーションモニタリング の測定サービスを使用すると、アプリケーションを通過するリクエストの完全なビューを提供できます。X-Ray は、アプリケーションのテレメトリを収集し、ペイロード、関数、トレース、サービス、API 全般の

可視化およびフィルター処理が可能で、ノーコードまたはローコードのシステムコンポーネントに対して有効にできます。CloudWatch アプリケーションのモニタリングには ServiceLens が含まれており、トレースをメトリクス、ログ、アラームと統合します。CloudWatch アプリケーションモニタリングには、エンドポイントと API をモニタリングするための Synthetics や、ウェブアプリケーションクライアントを測定するためのリアルユーザーモニタリングも含まれています。

実装手順

- サポートされているすべてのネイティブサービスで AWS X-Ray (Amazon S3、AWS Lambda、Amazon API Gateway など) を使用します。これらの AWS サービスでは、インフラストラクチャをコードとして、AWS SDK、または AWS Management Console を使用して設定を切り替え、X-Ray を有効にできます。
- 測定アプリケーション AWS Distro for Open Telemetry および X-Ray またはサードパーティの収集エージェント。
- プログラミング言語固有の実装については、 AWS X-Ray 開発者ガイド を参照してください。これらのドキュメントでは、HTTP リクエスト、SQL クエリ、アプリケーションのプログラミング言語固有のその他のプロセスを測定する方法について詳しく説明します。
- X-Ray トレース (Amazon CloudWatch Synthetic Canaries および Amazon CloudWatch RUM) を使用して、エンドユーザークライアントからダウンストリームの AWS インフラストラクチャを経由するリクエストパスを分析します。
- リソースの健全性と Canary テレメトリに基づき CloudWatch メトリクスとアラームを設定することで、チームでは迅速に問題についてアラートを発し、ServiceLens でトレースやサービスマップを詳しく調査できます。
- 次のようなサードパーティのトレースツールと X-Ray の統合を有効にします。 Datadog、New Relic、 Dynatrace (主要なトレースソリューションにサードパーティツールを使用している場合)。

リソース

関連するベストプラクティス:

- REL06-BP01 ワークロードのすべてのコンポーネントをモニタリングする (生成)
- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する

関連するドキュメント:

- AWS X-Ray とは?

- Amazon CloudWatch: アプリケーションモニタリング
- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray
- The Amazon Builders' Library: 分散システムを装備して、運用の可視性を高める
- 他の AWS サービスと AWS X-Ray の統合
- AWS Distro for OpenTelemetry および AWS X-Ray
- Amazon CloudWatch: 合成モニタリングを使用
- Amazon CloudWatch: CloudWatch RUM を使用
- Amazon CloudWatch Synthetic Canaries と Amazon CloudWatch アラームの設定
- 可用性を超えて: AWS での分散システムの回復力の理解と向上

関連サンプル:

- 1 つの可観測性ワークショップ

関連動画:

- AWS re:Invent 2022 - How to monitor applications across multiple accounts
- AWS アプリケーションをモニタリングする方法

関連ツール:

- AWS X-Ray
- Amazon CloudWatch
- Amazon Route 53

需要の変化に適応するようにワークロードを設計する

スケーラブルなワ ークロード には、リソースを自動で追加または削除する伸縮性があるので、リソースは任意の時点で常に、現行の需要に厳密に適合します。

ベストプラクティス

- REL07-BP01 リソースの取得またはスケーリング時に自動化を使用する
- REL07-BP02 ワークロードの障害を検出したときにリソースを取得する

- REL07-BP03 ワークロードにより多くのリソースが必要であることを検出した時点でリソースを取得する
- REL07-BP04 ワークロードの負荷テストを実施する

REL07-BP01 リソースの取得またはスケーリング時に自動化を使用する

障害のあるリソースを交換したり、ワークロードをスケーリングしたりする場合は、Amazon S3 や AWS Auto Scaling などのマネージド型の AWS のサービスを使用してプロセスを自動化します。サードパーティのツールや AWS SDK を使用して、スケーリングを自動化することもできます。

マネージド型の AWS のサービスとしては、Amazon S3、Amazon CloudFront、AWS Auto Scaling、AWS Lambda、Amazon DynamoDB、AWS Fargate、および Amazon Route 53 があります。

AWS Auto Scaling では、障害のあるインスタンスを検出して置き換えることができます。また、以下を含むリソースのスケーリングプランを構築することもできます。 Amazon EC2 インスタンスとスポットフリート、 Amazon ECS タスク、 Amazon DynamoDB テーブルとインデックス、および Amazon Aurora レプリカ。

EC2 インスタンスをスケーリングする場合は、複数のアベイラビリティゾーン (できれば少なくとも 3 つ) を使用し、容量を追加または削除して、これらのアベイラビリティゾーン間のバランスを維持します。ECS タスクまたは Kubernetes ポッド (Amazon Elastic Kubernetes Service を使用しているとき) も複数のアベイラビリティゾーンに分散してください。

AWS Lambda を使用しているときには、インスタンスは自動的にスケーリングされます。AWS Lambda は、関数のイベント通知を受信するたびに、コンピューティングフリート内の空き容量をすばやく見つけ、割り当てられた同時実行数までコードを実行します。特定の Lambda とService Quotas で、必要な同時実行数が確実に設定されているようにしてください。

Amazon S3 は、高いリクエスト頻度を処理できるように自動的にスケーリングします。たとえば、アプリケーションはバケット内のプレフィックスごとに 1 秒あたり 3,500 件以上の PUT/COPY/POST/DELETE リクエストまたは 5,500 件以上の GET/HEAD リクエストを送信できます。バケット内のプレフィックス数に制限はありません。読み取りを並列化することで、読み取りまたは書き込みのパフォーマンスを向上させることができます。例えば、Amazon S3 バケットに 10 個のプレフィックスを作成して読み取りを並列化する場合、読み取りパフォーマンスを 1 秒あたり 55,000 件の読み取りリクエストにスケーリングできます。

Amazon CloudFront または信頼できるコンテンツ配信ネットワーク (CDN) を設定して使用します。CDN は、より迅速なエンドユーザーレスポンスタイムを提供でき、コンテンツのリクエストをキャッシュから処理できるため、ワークロードをスケーリングする必要性が少なくなります。

一般的なアンチパターン:

- 自動ヒーリングのために Auto Scaling グループを実装しますが、伸縮性は実装しません。
- トラフィックの大幅な増加に対応するために自動スケーリングを使用する。
- ステートフル性が高いアプリケーションをデプロイし、伸縮性を排除する。

このベストプラクティスを活用するメリット: 自動化により、リソースのデプロイと廃棄で手動エラーが発生する可能性がなくなります。自動化は、デプロイや廃棄のニーズへの応答が遅いことによるコストの超過やサービス拒否のリスクを排除します。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

- AWS Auto Scaling を設定して使用します。これにより、アプリケーションをモニタリングし、安定した予測可能なパフォーマンスを可能な限り低いコストで維持するためのキャパシティーを自動的に調整します。AWS Auto Scaling を使用すると、複数のサービスにまたがる複数のリソースに対してアプリケーションのスケーリングをセットアップできます。
  - 「AWS Auto Scaling とは何ですか?」
    - Amazon EC2 インスタンスとスポットフリート、Amazon ECS タスク、Amazon DynamoDB のテーブルとインデックス、Amazon Aurora のレプリカ、および AWS Marketplace アプライアンスなど、該当する者に対して Auto Scaling を設定します。
      - DynamoDB Auto Scaling によるスループットキャパシティの自動管理
        - サービス API を操作して、アラーム、スケーリングポリシー、ウォームアップ時間、およびクールダウン時間を指定します。
- Elastic Load Balancing を使用します。ロードバランサーは、パスまたはネットワーク接続ごとに負荷を分散することができます。
  - 「Elastic Load Balancing とは何ですか?」
    - Application Load Balancers は、負荷をパスごとに分散できます。
      - Application Load Balancer とは?
        - Application Load Balancer を設定して、ドメイン名の下のパスに基づいてトラフィックをさまざまなワークロードに分散します。

        - Application Load Balancers を使用すると、AWS Auto Scaling と統合して需要を管理するという方法で負荷を分散できます。
            - Auto Scaling グループでロードバランサーを使用する
    - Network Load Balancers は、接続ごとに負荷を分散することができます。
        - Network Load Balancer とは?
            - Network Load Balancer は、TCP を使用してトラフィックをさまざまなワークロードに分散するか、ワークロードの IP アドレスの一定のセットが含まれるように設定します。
            - Network Load Balancer を使用すると、AWS Auto Scaling と統合して需要を管理するという方法で負荷を分散できます。
- 可用性の高い DNS プロバイダーを使用します。DNS 名により、ユーザーは、IP アドレスの代わりに DNS 名を入力してワークロードにアクセスでき、この情報を、定義されたスコープ (通常はワークロードのユーザーに対してグローバルに定義されたスコープ) に分散できます。
    - Amazon Route 53 または信頼できる DNS プロバイダーを使用します。
        - 「Amazon Route 53 とは何ですか?」
    - Route 53 を使用して、CloudFront ディストリビューションとロードバランサーを管理します。
        - 管理する予定のドメインとサブドメインを決定します。
        - ALIAS レコードまたは CNAME レコードを使用して適切なレコードセットを作成します。
            - レコードの操作
- AWS グローバルネットワークを使用して、ユーザーからアプリケーションへのパスを最適化します。AWS Global Accelerator は、アプリケーションエンドポイントの状態を継続的にモニタリングし、トラフィックを 30 秒未満で正常なエンドポイントにリダイレクトします。
    - AWS Global Accelerator は、ローカルまたはグローバルユーザーが使用するアプリケーションの可用性とパフォーマンスを向上させるサービスです。Application Load Balancers、Network Load Balancer、Amazon EC2 インスタンスなど、単一または複数の AWS リージョンのアプリケーションエンドポイントへの固定エントリポイントとして機能する静的 IP アドレスが提供されます。
        - AWS Global Accelerator とは?
- Amazon CloudFront または信頼できるコンテンツ配信ネットワーク (CDN) を設定して使用します。コンテンツ配信ネットワークは、エンドユーザーの応答時間を短縮し、ワークロードの不要なスケーリングを引き起こす原因となるコンテンツのリクエストを処理できます。
    - 「Amazon CloudFront とは」
        - ワークロード用の Amazon CloudFront ディストリビューションを設定するか、サードパーティの CDN を使用します。

- エンドポイントセキュリティグループまたはアクセスポリシーで CloudFront の IP 範囲を使用することで、ワークロードへのアクセスを CloudFront からのみに制限できます。

リソース

関連するドキュメント:

- APN Partner: partners that can help you create automated compute solutions
- AWS Auto Scaling: スケーリングプランの仕組み
- AWS Marketplace: Auto Scaling で使用できる製品
- Managing Throughput Capacity Automatically with DynamoDB Auto Scaling
- Auto Scaling グループでロードバランサーを使用する
- AWS Global Accelerator とは?
- 「What Is Amazon EC2 Auto Scaling?」
- 「AWS Auto Scaling とは何ですか?」
- 「Amazon CloudFront とは」
- 「Amazon Route 53 とは何ですか?」
- 「Elastic Load Balancing とは何ですか?」
- Network Load Balancer とは?
- Application Load Balancer とは?
- レコードの操作

REL07-BP02 ワークロードの障害を検出したときにリソースを取得する

可用性が影響を受ける場合、必要に応じてリソースをリアクティブにスケールし、ワークロードの可用性を復元します。

まず、ヘルスチェックとこのチェックの基準を設定して、リソースの不足が可用性に影響を与えるタイミングを示す必要があります。次に、適切な担当者に通知してリソースを手動でスケーリングするか、オートメーションを開始してリソースを自動的にスケーリングします。

スケーリングはワークロードに合わせて手動で調整できます (例えば、Auto Scaling グループの EC2 インスタンスの数の変更や、DynamoDB テーブルのスループットの変更は、AWS Management

Console または AWS CLI で行うことができます)。ただし、可能な限りオートメーションを使用してください (「リソースの取得またはスケーリング時に自動化を使用する」を参照)。

期待される成果: 障害やカスタマーエクスペリエンスの低下が検知された時点で、可用性を回復するためのスケーリングアクティビティ (自動または手動) が開始されます。

このベストプラクティスが確立されていない場合のリスクレベル: 中

実装のガイダンス

ワークロードのすべてのコンポーネントにオブザーバビリティとモニタリングを実装して、カスタマーエクスペリエンスを監視し、障害を検知します。必要なリソースをスケーリングする手順 (手動または自動) を定義します。詳細については、「REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する」を参照してください。

実装手順

- 必要なリソースをスケーリングする手順 (手動または自動) を定義します。
  - スケーリングの手順は、ワークロード内のさまざまなコンポーネントの設計方法に応じて異なります。
  - また、使用されている基盤のテクノロジーによっても異なります。
    - AWS Auto Scaling を使用するコンポーネントでは、スケーリングプランを使用して、リソースをスケーリングするための一連の指示を設定できます。AWS CloudFormation を操作する場合や、タグを AWS リソースに追加する場合、アプリケーションごとに、さまざまなリソースセットのスケーリングプランを設定できます。Auto Scaling はリソースごとにスケーリング戦略をカスタマイズし、それに応じたレコメンデーションを提示します。スケーリングプランを作成すると、Auto Scaling は、動的スケーリングと予測スケーリングの手法を適宜組み合わせて、スケーリング戦略をサポートします。詳細については、「スケーリングプランの仕組み」を参照してください。
    - Amazon EC2 Auto Scaling を使用すると、適正数の Amazon EC2 インスタンスを用意して、アプリケーションの負荷に対処できます。Auto Scaling グループと呼ばれる EC2 インスタンスのコレクションを作成します。Auto Scaling グループごとにインスタンスの最小数と最大数を指定でき、グループがこれらの制限を下回る/上回ることがないように Amazon EC2 Auto Scaling が調整します。詳細については、「Amazon EC2 Auto Scaling とは」を参照してください。
    - Amazon DynamoDB Auto Scaling は、Application Auto Scaling サービスを使用して、実際のトラフィックパターンに応じて、プロビジョニングされたスループットキャパシティを動的に調整します。そのため、テーブルまたはグローバルセカンダリインデックスで、プロビジョ

ニング済みの読み込みおよび書き込みキャパシティを増やすことができ、スロットリングを行うことなくトラフィックの急増に対処できます。詳細については、「DynamoDB Auto Scaling によるスループットキャパシティの自動管理」を参照してください。

リソース

関連するベストプラクティス:

- REL07-BP01 リソースの取得またはスケーリング時に自動化を使用する
- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する

関連するドキュメント:

- AWS Auto Scaling: スケーリングプランの仕組み
- DynamoDB Auto Scaling によるスループットキャパシティの自動管理
- Amazon EC2 Auto Scaling とは

REL07-BP03 ワークロードにより多くのリソースが必要であることを検出した時点でリソースを取得する

需要に合わせてリソースをプロアクティブにスケールし、可用性への影響を回避します。

多くの AWS サービスは、需要に合わせて自動的にスケールします。Amazon EC2 インスタンスまたは Amazon ECS クラスターを使用している場合、ワークロードの需要に対応する使用状況のメトリクスに基づいて Auto Scaling を実行するように設定できます。Amazon EC2 では、平均 CPU 使用率、ロードバランサーリクエスト数、またはネットワーク帯域幅を使用して、EC2 インスタンスをスケールアウト (またはスケールイン) できます。Amazon ECS では、平均 CPU 使用率、ロードバランサーリクエスト数、およびメモリ使用率を使用して、ECS タスクをスケールアウト (またはスケールイン) できます。AWS で Target Auto Scaling を使用すると、オートスケーラーは家庭用サーモスタットのように機能し、指定したターゲット値 (例えば、CPU 使用率 70%) を維持するためにリソースを追加または削除します。

AWS Auto Scaling はまた、 Predictive Auto Scaling も実行できます。これは、機械学習を使用して各リソースの過去のワークロードを分析し、次の 2 日間の負荷を定期的に予測します。

リトルの法則は、必要なコンピューティングインスタンス (EC2 インスタンス、同時実行の Lambda 関数など) 数を計算するのに役立ちます。

L = λW

L = インスタンス数 (またはシステムの平均同時実行数)

λ = リクエストが到着する平均レート (リクエスト/秒)

W = 各リクエストがシステムで費やす平均時間 (秒)

例えば、100 rps では、各リクエストの処理に 0.5 秒かかる場合、需要に対応するには 50 インスタンスが必要です。

このベストプラクティスを活用しない場合のリスクレベル: ミディアム

実装のガイダンス

- ワークロードにより多くのリソースが必要であることを検出した時点でリソースを取得します。需要に合わせてリソースをプロアクティブにスケールし、可用性への影響を回避します。
  - 特定のリクエストレートを処理するために必要なコンピューティングリソースの数 (コンピューティングの同時実行) を計算します。
    - リトルの法則について語る
  - 使用状況の履歴パターンがあるときには、Amazon EC2 Auto Scaling のスケジュールされたスケーリングをセットアップします。
    - Amazon EC2 Auto Scaling のスケジュールされたスケーリング
  - AWS 予測スケーリングを使用します。
    - 機械学習を利用した EC2 の予測スケーリング

リソース

関連するドキュメント:

- AWS Auto Scaling: スケーリングプランの仕組み
- AWS Marketplace: Auto Scaling で使用できる製品
- Managing Throughput Capacity Automatically with DynamoDB Auto Scaling
- 機械学習を利用した EC2 の予測スケーリング
- Amazon EC2 Auto Scaling のスケジュールされたスケーリング
- リトルの法則について語る
- 「What Is Amazon EC2 Auto Scaling?」

REL07-BP04 ワークロードの負荷テストを実施する

負荷テスト手法を採用して、スケーリングアクティビティがワークロード要件を満たすかどうかを測定します。

持続的な負荷テストを実行することが重要です。負荷テストによって、ブレークポイントを発見し、ワークロードのパフォーマンスをテストします。AWS は、生産ワークロードのスケールをモデル化する一次的なテスト環境のセットアップを容易にします。クラウド上では、本稼働スケールのテスト環境をオンデマンドで作成し、テスト完了後にリソースを解放できます。テスト環境の#払いは実#時にのみ発#するため、オンプレミスでテストを実施する場合と比べてわずかなコストで、本番環境をシミュレートできます。

本番環境での負荷テストは、ゲームデーの一部として考える必要もあります。その中で、顧客の使用率が低い時間帯に本番システムに負荷をかけ、担当者全員がテスト結果を解釈して発生した問題に対処できるようにします。

一般的なアンチパターン:

- 本稼働環境と同じ設定ではないデプロイで負荷テストを実行する。
- ワークロード全体ではなく、ワークロードの個々の部分に対してのみ負荷テストを実行する。
- 実際のリクエストの代表的なセットではなく、リクエストのサブセットを使用して負荷テストを実行する。
- 予想される負荷を超える小さな安全率に対して負荷テストを実行する。

このベストプラクティスを活用するメリット: 負荷がかかっているときにアーキテクチャのどのコンポーネントが失敗するかを把握し、問題に対処すべく、その負荷が近づいていることを示すために監視するメトリクスを特定し、その障害の影響を防ぐことができます。

このベストプラクティスを活用しない場合のリスクレベル: 中

実装のガイダンス

- 負荷テストを実行して、キャパシティを追加または削除する必要があるワークロードの側面を特定します。負荷テストには、本稼働環境で受け取るものと同様の代表的なトラフィックを用いる必要があります。設定したメトリクスを監視しながら負荷を増やし、リソースを追加または削除する必要があるタイミングをどのメトリクスが示唆しているのかを判断します。
  - AWS での分散負荷テスト: 接続された数千のユーザーをシミュレートする

- リクエストの組み合わせを特定します。なされるリクエストの組み合わせはさまざまであるため、トラフィックの混在を組み合わせを特定する際は、さまざまな時間枠を確認する必要があります。
- ロードドライバーを実装します。カスタムコード、オープンソース、または商用ソフトウェアを使用して、ロードドライバーを実装できます。
- 最初は小さなキャパシティを使用して負荷テストを実施します。1 つのインスタンスまたはコンテナと同じくらいの少なさのキャパシティーに負荷をかけることで、すぐに効果が現れます。
- 大きなキャパシティに対して負荷テストを実施します。この効果は分散された負荷によって異なるため、できるだけ製品環境に近いに対してテストする必要があります。

リソース

関連するドキュメント:

- AWS での分散負荷テスト: 接続された数千のユーザーをシミュレートする
- Load testing applications

関連動画:

- AWS Summit ANZ 2023: Accelerate with confidence through AWS Distributed Load Testing

変更の実装

新しい機能をデプロイし、ワークロードとオペレーティング環境が既知の適切にパッチが適用されたソフトウェアを実行していることを確認するには、変更を制御する必要があります。変更が制御されていないと、変更の影響を予測したり、変更によって発生した問題に対処したりすることが困難になります。

ベストプラクティス

- REL08-BP01 デプロイなどの標準的なアクティビティにランブックを使用する
- REL08-BP02 デプロイの一部として機能テストを統合する
- REL08-BP03 デプロイの一部として回復力テストを統合する
- REL08-BP04 イミュータブルなインフラストラクチャを使用してデプロイする
- REL08-BP05 オートメーションを使用して変更をデプロイする

REL08-BP01 デプロイなどの標準的なアクティビティにランブックを使用する

ランブックは、特定の成果を達成するための事前定義された手順です。手動または自動のどちらでも、標準的なアクティビティを実行するにはランブックを使用します。例えば、ワークロードのデプロイ、ワークロードへのパッチの適用、DNS の変更などがあります。

例えば、デプロイ中のロールバックの安全性を確保するためのプロセスを導入します。顧客側の中断なしでデプロイをロールバックできるようにすることは、サービスの信頼性を高める上で重要です。

ランブックの手順については、有効で効果的な手動プロセスから開始し、それをコードで実装して、適切な場合は自動実行をトリガーします。

高度に自動化された高機能のワークロードの場合でも、ランブックは本番環境で実行したり、厳格なレポートや監査の要件を満たしたりするのに役立ちます。

プレイブックは特定のインシデントに対応するために用いられ、ランブックは特定の結果を達成するために使用されます。多くの場合、ランブックは日常的なアクティビティ用で、プレイブックは非日常的なイベントに応えるために使用します。

一般的なアンチパターン:

- 本稼働環境の設定に対して、計画されていない変更を実行する。
- デプロイを高速化するために計画の手順をスキップすることで、デプロイを失敗させる。
- 変更を戻すことができるかどうかをテストせずに変更を加える。

このベストプラクティスを確立するメリット: 効果的な変更計画は、影響を受けるすべてのシステムを考慮しているため、変更を正常に実行する能力を強化します。テスト環境で変更を検証すると、信頼が強化されます。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

- ランブックに手順を文書化することにより、一貫性を保ち、汎用イベントにすみやかに対応できるようになります。
  - AWS Well-Architected Framework: Concepts: Runbook (AWS Well-Architected フレームワーク: 概念: ランブック)

- Infrastructure as Code の原則を適用して、インフラストラクチャを定義します。AWS CloudFormation (または信頼できるサードパーティ) を使用してインフラストラクチャを定義することにより、バージョン管理ソフトウェアを使用して、バージョン管理および変更の追跡を行うことができます。
  - AWS CloudFormation (または信頼できるサードパーティプロバイダー) を使用して、インフラストラクチャを定義します。
    - AWS CloudFormation とは
  - 優れたソフトウェア設計の原則を使用して、単一または分離されたテンプレートを作成します。
    - 実装にあたって、必要な権限、テンプレート、責任者を決定します。
      - AWS Identity and Access Management によるアクセスの制御
    - バージョン管理には、AWS CodeCommit や信頼できるサードパーティ製ツールのようなバージョン管理用ソースコントロールを使用します。
      - AWS CodeCommit とは

リソース

関連するドキュメント:

- APN パートナー: 自動化されたデプロイソリューションの作成を支援できるパートナー
- AWS Marketplace: products that can be used to automate your deployments
- AWS Well-Architected Framework: Concepts: Runbook (AWS Well-Architected フレームワーク: 概念: ランブック)
- AWS CloudFormation とは
- AWS CodeCommit とは

関連する例:

- Automating operations with Playbooks and Runbooks (プレイブックとランブックによるオペレーションの自動化)

REL08-BP02 デプロイの一部として機能テストを統合する

機能テストは、自動デプロイの一部として実行されます。成功条件を満たさない場合、パイプラインは停止またはロールバックされます。このようなテストは、パイプラインの本番稼働前にステージングされた本番稼働前環境で実行されます。これは、デプロイパイプラインの一部として行うのが理想的です。

期待される成果: 自動化を使用して機能をテストし、関連付けられたテストデータによってテスト期間および費用を削減して、テスト結果の精度を高めます。デプロイプロセスの一部として機能テストを組み込むことで、リリースパイプラインを自動化して、アプリケーションとインフラストラクチャを迅速かつ確実に更新できます。

一般的なアンチパターン:

- テストをデプロイパイプラインの外部で手動で実行する。
- 手動の緊急ワークフローにより、自動化のテストステップを省略する。
- スケジュールを短縮するために、確立されたテスト計画やプロセスを無視する。

このベストプラクティスを活用するメリット: 機能テストでは、指定された要件に従ってシステムが動作することを検証します。テストでは、ユーザーインターフェース、API、データベース、ソースコードなどのコンポーネントの意図された動作順序を一貫して検証します。システムのこれらのコンポーネントを検証する際、機能テストで各機能が期待どおりに動作することを検証します。これにより、ユーザーの期待とソフトウェアの整合性の両方を満たすことができます。通常のデプロイの一部として機能テストを組み込み、すべての変更を自動化してデプロイすることで、人為的ミスの発生の可能性を減らすことができます。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

デプロイの一部として機能テストを統合します。機能テストは、自動デプロイの一部として実行されます。成功基準が満たされない場合、パイプラインは停止またはロールバックされます。AWS CodePipeline は、自動テスト用の継続的デリバリーパイプラインを提供するので、テスターはテストとデプロイのプロセス全体を自動化できます。AWS CodeBuild や AWS CodeDeploy などの AWS サービスと統合して、ソフトウェア開発ライフサイクルの構築、テスト、デプロイの各フェーズを自動化します。

実装手順

- パイプラインの設定: AWS CodePipeline コンソールまたは AWS Command Line Interface (CLI) を使用して、ソース、構築、テスト、デプロイの各ステージを設定します。
  - ソースの定義: AWS CodePipeline を使用すると、GitHub、AWS CodeCommit、Bitbucket などのバージョン管理システムからソースコードを自動的に取得して、常に最新のコードがテストに使用されていることを確認できます。

- 構築とテストの自動化: AWS CodeBuild はコードを自動的に構築してテストし、テストレポートを生成します。JUnit、NUnit、TestNG などの一般的なテストフレームワークをサポートしています。
- コードのデプロイ: 構築およびテスト済のコードは、AWS CodeDeploy を使用して Amazon EC2 インスタンス、AWS Lambda 関数、オンプレミスサーバーなどのテスト環境にデプロイできます。
- パイプラインのモニタリング: AWS CodePipeline は、パイプラインの進行状況と各ステージのステータスを追跡できます。品質チェックを使用して、テストの実行ステータスに基づいてパイプラインをブロックすることもできます。パイプラインステージの障害やパイプラインの完了に関する通知を受け取ることもできます。

リソース

関連するドキュメント:

- Use AWS CodePipeline with AWS CodeBuild to test code and run builds
- AWS CodeBuild でのログ記録とモニタリング
- 機能テストの指標

REL08-BP03 デプロイの一部として回復力テストを統合する

システム障害を意図的に導入して障害発生時の能力を測定することで、回復力テストを統合します。回復力テストは、システムでの予期しない障害の特定に重点を置いているため、通常デプロイサイクルに統合されるユニットテストや機能テストとは異なります。本番稼働前の回復力テストの統合から始めても問題ありませんが、ゲームデーの一環として、これらのテストを本番環境に実装するという目標を設定します。

期待される成果: 回復力テストは、本番稼働時の障害に耐えるシステムの能力の信頼性を高めるのに役立ちます。テストによって、障害につながる可能性のある弱点が特定され、システムを改善して障害や劣化を自動的かつ効率的に軽減できます。

一般的なアンチパターン:

- デプロイプロセスにおけるオブザーバビリティとモニタリングの欠如
- 人によるシステム障害の解決
- 低品質の分析メカニズム

- システムの既知の問題にフォーカスし、未知の問題点を特定するためのテストを行わない
- 障害は特定できるが、解決できない
- 調査結果とランブックが文書化されていない

このベストプラクティスを活用することのメリット: デプロイに統合された回復力テストは、本番環境のダウンタイムにつながる可能性のある、他の方法では気付かれないシステムの未知の問題を特定するのに役立ちます。システムでのこれらの未知の問題の特定は、結果の文書化、テストの CI/CD プロセスへの統合、およびランブックの作成に役立ち、効率的で反復可能なメカニズムを通じて、障害の緩和を簡素化します。

このベストプラクティスが確立されていない場合のリスクレベル: 中

実装のガイダンス

システムのデプロイに統合できる最も一般的な回復力テストの形式は、ディザスタリカバリとカオスエンジニアリングです。

- 大規模なデプロイには、ディザスタリカバリ計画と標準運用手順 (SOP) の更新を含めます。
- 信頼性テストを自動デプロイパイプラインに統合します。AWS Resilience Hub などのサービスを CI/CD パイプラインに統合して、継続的な回復力評価を確立できます。この評価は、すべてのデプロイで自動的に評価されます。
- AWS Resilience Hub でアプリケーションを定義します。回復力評価では、アプリケーションの AWS Systems Manager ドキュメントとして復旧手順を作成するのに役立つコードスニペットが生成され、推奨される Amazon CloudWatch モニターとアラームのリストが提供されます。
- ディザスタリカバリ計画と SOP が更新されたら、ディザスタリカバリテストを実施して効果を確認します。ディザスタリカバリテストは、イベント後にシステムを復旧して通常の運用に戻ることができるかどうかを判断するのに役立ちます。さまざまなディザスタリカバリ戦略をシミュレートし、計画が稼働時間の要件を満たすのに十分であるかどうかを確認できます。一般的なディザスタリカバリ戦略には、バックアップと復元、パイロットライト、コールドスタンバイ、ウォームスタンバイ、ホットスタンバイ、アクティブ - アクティブなどがあり、コストと複雑さは戦略によって異なります。ディザスタリカバリテストを行う前に、シミュレートする戦略の選択を簡素化するために、目標復旧時間 (RTO) と目標復旧時点 (RPO) を定義することをお勧めします。AWS は、AWS Elastic Disaster Recovery など、計画とテストを始めるのに役立つディザスタリカバリツールを提供しています。
- カオスエンジニアリングのテストでは、ネットワーク障害やサービス障害などのシステムの中断を発生させます。制御された障害を使用してシミュレーションを行うことで、発生した障害の影響を

抑えながら、システムの脆弱性を発見できます。他の戦略と同様に、AWS Fault Injection Service などのサービスを使用して、非本番環境で制御された障害シミュレーションを実行して、本番環境にデプロイする前に検証を行います。

リソース

関連するドキュメント:

- Experiment with failure using resilience testing to build recovery preparedness
- Continually assessing application resilience with AWS Resilience Hub and AWS CodePipeline
- Disaster recovery (DR) architecture on AWS, part 1: Strategies for recovery in the cloud
- Verify the resilience of your workloads using Chaos Engineering
- Principles of Chaos Engineering
- Chaos Engineering Workshop

関連動画:

- AWS re:Invent 2020: Testing Resilience using Chaos Engineering
- Improve Application Resilience with AWS Fault Injection Service
- Prepare & Protect Your Applications From Disruption With AWS Resilience Hub

REL08-BP04 イミュータブルなインフラストラクチャを使用してデプロイする

イミュータブルなインフラストラクチャは、本番ワークロードで更新、セキュリティパッチ、または設定変更がインプレースで行われないように義務付けるモデルです。変更が必要な場合、アーキテクチャは新しいインフラストラクチャに構築され、本番環境にデプロイされます。

イミュータブルインフラストラクチャのデプロイ戦略に従って、ワークロードデプロイの信頼性、一貫性、再現性を高めましょう。

期待される成果: イミュータブルインフラストラクチャでは、インプレース変更を行って、ワークロード内でインフラストラクチャリソースを実行することはできません。代わりに、変更の必要が生じた場合は、必要な変更をすべて適用した新しいインフラストラクチャリソース一式を既存のリソースと並行してデプロイします。このデプロイは自動的に検証され、検証に合格すると、トラフィックが新しいリソース一式に徐々にシフトします。

このデプロイ戦略は、ソフトウェアの更新、セキュリティパッチ、インフラストラクチャの変更、構成の更新、アプリケーションの更新などに適用されます。

一般的なアンチパターン:

- 実行中のインフラストラクチャリソースにインプレース変更を実装する。

このベストプラクティスを活用するメリット:

- 環境間の整合性の向上: 環境間でインフラストラクチャリソースに違いがなくなるため、整合性が向上し、テストが簡素化されます。
- 構成のドリフトの低減: バージョン管理された既知の構成でインフラストラクチャリソースを置き換えるので、インフラストラクチャがテスト済みで信頼できる既知の状態に保たれ、構成のドリフトを回避できます。
- 信頼性の高いアトミックデプロイ: デプロイは正常に完了するか、一切変更されないかの二択です。デプロイプロセスの一貫性と信頼性が高まります。
- 簡単なデプロイ: アップグレードをサポートする必要がないため、デプロイが簡素化されます。単に新たにデプロイすることが、アップグレードになります。
- 高速なロールバックと復旧プロセスによる安全なデプロイ: 以前動作していたバージョンは変更されないため、デプロイの安全性が高まります。エラーが検出された場合は、ロールバックできます。
- セキュリティ体制の強化: インフラストラクチャの変更を許容しないことで、リモートアクセスメカニズム (SSH など) を無効にすることができます。これにより攻撃ベクトルが減少し、組織のセキュリティ体制が強化されます。

このベストプラクティスが確立されていない場合のリスクレベル: 中

実装のガイダンス

オートメーション

イミュータブルインフラストラクチャのデプロイ戦略を定義する際には、再現性を高め、人為的ミスを極力抑えるために、可能な限りオートメーションを使用することが推奨されます。詳細については、「REL08-BP05 オートメーションを使用して変更をデプロイする」および「安全なハンズオンデプロイメントの自動化」を参照してください。

Infrastructure as Code (IaC) では、インフラストラクチャのプロビジョニング、オーケストレーション、デプロイの手順がプログラム的、記述的、宣言的に定義され、ソース管理システムに保存されま

す。IaC を活用することで、インフラストラクチャのデプロイを簡単に自動化し、インフラストラクチャの不変性を実現できます。

デプロイパターン

ワークロードの変更が必要な場合、イミュータブルインフラストラクチャのデプロイ戦略では、必要な変更をすべて適用済みの新しいインフラストラクチャリソース一式をデプロイすることが義務付けられています。この新しいリソースセットでは、ユーザーへの影響を最小限に抑えるロールアウトパターンに従うことが重要です。このデプロイには、主に 2 つの戦略があります。

Canary デプロイ: 通常は単一のサービスインスタンス (Canary) で実行される新しいバージョンに、少数の顧客を誘導する方法です。次に、生じた動作の変更やエラーを詳細に調べます。重大な問題が発生した場合、Canary からトラフィックを削除して、ユーザーを以前のバージョンに戻すことができます。デプロイが成功したら、希望の速度でデプロイを続行できます。変更やエラーをモニタリングしながら、デプロイがすべて完了するまで続けます。AWS CodeDeploy では、Canary デプロイを可能にするデプロイ設定を構成できます。

ブルー/グリーンデプロイ: Canary デプロイに似ていますが、アプリケーション全体が並行してデプロイされる点が異なります。2 つのスタック (青と緑) 間でデプロイを交互に行います。この場合も、トラフィックを新しいバージョンに送信したときにデプロイに問題が発生した場合は、古いバージョンにフォールバックできます。通常は、すべてのトラフィックを一度に切り替えますが、Amazon Route 53 の加重 DNS ルーティング機能を使用して、トラフィックの一部を各バージョンに切り替え、新しいバージョンを段階を踏みながら導入していくこともできます。AWS CodeDeploy と AWS Elastic Beanstalk では、ブルー/グリーンデプロイを可能にするデプロイ設定を構成できます。

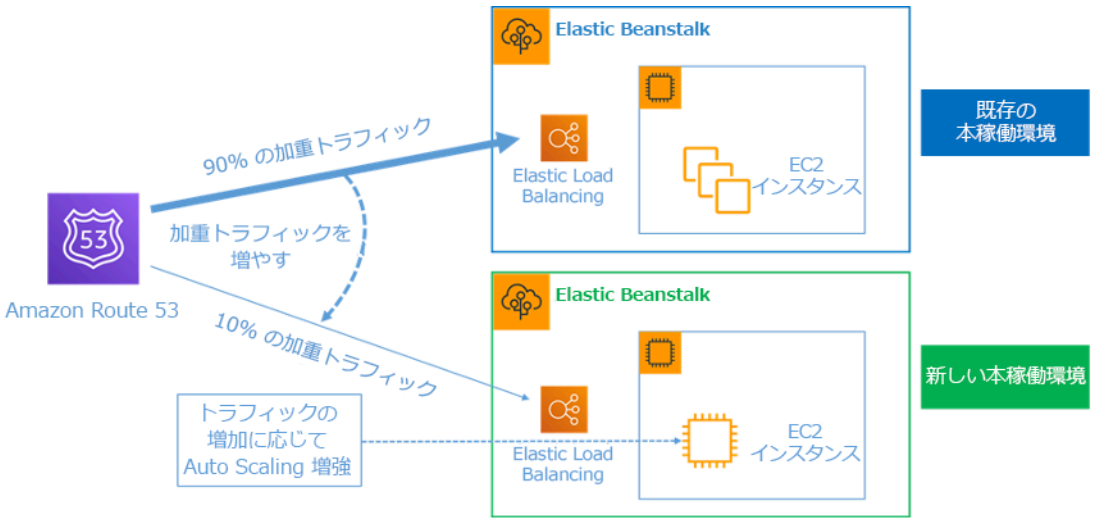


図 8: AWS Elastic Beanstalk と Amazon Route 53 によるブルー/グリーンデプロイ

ドリフトの検知

ドリフトとは、何らかの変化によって、インフラストラクチャリソースが予測とは異なる状態や構成になることです。適切に管理されていない構成変更は、イミュータブルインフラストラクチャの概念と矛盾します。イミュータブルインフラストラクチャを首尾よく実装するためには、そうした構成変更を検出し、対処する必要があります。

実装手順

- 実行中のインフラストラクチャリソースのインプレース変更は禁止します。
  - AWS Identity and Access Management(IAM) を使用して、AWS でサービスやリソースにアクセスできるユーザーやアクセスの対象を指定し、アクセス許可をきめ細かく一元管理し、アクセスを分析して AWS 全体のアクセス管理を強化することができます。
- インフラストラクチャリソースのデプロイを自動化して、再現性を高め、人為的ミスを極力抑えます。
  - 「AWS での DevOps の概要入門」ホワイトペーパーで説明されているように、オートメーションは AWS サービスの基本理念の 1 つであり、すべてのサービス、機能、オファリングの内部でサポートされています。
  - Amazon マシンイメージ (AMI) を プリベイクすると、起動時間を短縮できます。EC2 Image Builder はフルマネージド型の AWS サービスで、安全で最新の Linux または Windows のカスタム AMI の作成、保守、検証、共有、デプロイを自動化できます。
  - 次のサービスは、オートメーションに対応しています。
    - AWS Elastic Beanstalk は、Java、.NET、PHP、Node.js、Python、Ruby、Go、Docker で開発されたウェブアプリケーションを、Apache、NGINX、Passenger、IIS などの使い慣れたサーバーに迅速にデプロイおよびスケーリングするサービスです。
    - AWS Proton を使用すれば、開発チームがインフラストラクチャのプロビジョニング、コードのデプロイ、モニタリング、更新に必要とするさまざまなツールをプラットフォームチームがすべて接続し、調整できます。AWS Proton では、サーバーレスアプリケーションとコンテナベースアプリケーションのプロビジョニングとデプロイを Infrastructure as Code (IaC) で自動化できます。
  - IaC を活用することで、インフラストラクチャのデプロイを簡単に自動化し、インフラストラクチャの不変性を実現できます。AWS は、プログラム的、記述的、宣言的な方法でインフラストラクチャの作成、デプロイ、保守を可能にするサービスを提供しています。
    - AWS CloudFormation では、開発者が予測可能な方法で順序立てて AWS リソースを作成できます。リソースは JSON または YAML 形式でテキストファイルに書き込まれます。テンプレートには、作成および管理されるリソースのタイプに応じて、特定の構文と構造が必要

です。AWS Cloud9 などのコードエディタを使用して JSON または YAML でリソースを作成し、バージョン管理システムにチェックインすると、CloudFormation が指定されたサービスを安全で繰り返し可能な方法で構築します。

- AWS Serverless Application Model(AWS SAM) は、AWS でサーバーレスアプリケーションの構築に使用できるオープンソースのフレームワークです。AWS SAM は、AWS の他のサービスと統合でき、AWS CloudFormation の拡張機能です。
- AWS Cloud Development Kit (AWS CDK) は、使い慣れたプログラミング言語でクラウドアプリケーションリソースをモデル化し、プロビジョニングできるオープンソースのソフトウェア開発フレームワークです。AWS CDK を使用して、TypeScript、Python、Java、.NET を使用してアプリケーションインフラストラクチャをモデル化できます。AWS CDK は AWS CloudFormation をバックグラウンドで使用して、安全で繰り返し可能な方法でリソースをプロビジョニングします。
- AWS Cloud Control API は作成、読み取り、更新、削除、一覧表示 (CRUDL) の共通 API を提供し、開発者はこの API 一式を使用して、クラウドインフラストラクチャを簡単かつ一貫した方法で管理できます。開発者は Cloud Control API の共通 API を使用して、AWS およびサードパーティのサービスのライフサイクルを一様に管理できます。

- ユーザーへの影響を最小限に抑えるデプロイパターンを実装します。
  - Canary デプロイ:
    - API Gateway の Canary リリースデプロイの設定
    - AWS App Mesh を使用した Amazon ECS でのカナリアデプロイパイプラインの作成
  - ブルー/グリーンデプロイ: AWS でのブルー/グリーンデプロイに関するホワイトペーパーでは、ブルー/グリーンデプロイ戦略を実装するテクニックの例を紹介しています。
- 構成または状態のドリフトを検出します。詳細については、「スタックとリソースに対するアンマネージド型構成変更の検出」を参照してください。

リソース

関連するベストプラクティス:

- REL08-BP05 オートメーションを使用して変更をデプロイする

関連するドキュメント:

- 安全なハンズオフデプロイメントの自動化
- Leveraging AWS CloudFormation to create an immutable infrastructure at Nubank

- コードとしてのインフラストラクチャ
- Implementing an alarm to automatically detect drift in AWS CloudFormation stacks

関連動画:

- AWS re:Invent 2020: Reliability, consistency, and confidence through immutability

REL08-BP05 オートメーションを使用して変更をデプロイする

デプロイとパッチ適用は自動化されて、悪影響を排除します。

本番システムに変更を加えることは、多くの企業にとって最大級のリスクの 1 つです。当社は、ソフトウェアで解決するビジネス課題と同じくらい、デプロイを最優先課題としてとらえています。これは今日、変更のテストと導入、容量の追加と削除、データの移行など、実運用のあらゆる場所における自動化の導入を意味します。

期待される成果: 大規模な運用前テスト、自動ロールバック、段階的な本番環境へのデプロイによって、リリースプロセスに自動デプロイの安全性を組み込むことができます。この自動化により、デプロイの失敗による本番環境への潜在的な影響が最小限に抑えられ、開発者は本番環境へのデプロイを活発にモニタリングする必要がなくなります。

一般的なアンチパターン:

- 手動で変更を行う。
- 手動の緊急ワークフローにより、自動化を省略する。
- スケジュールを短縮するため、確立されたテスト計画やプロセスを無視する。
- ベイク時間を考慮せずに、急激なフォローオンデプロイを実行する。

このベストプラクティスを活用するメリット: 自動化を使用してすべての変更をデプロイすることで、人為的ミスが生じる可能性がなくなり、本番環境を変更する前にテストできるようになります。本番環境の稼働前にこのプロセスを実行することで、計画が完了していることを確認できます。さらに、リリースプロセスに自動ロールバックを組み込むことで、本番環境の問題を特定し、ワークロードを以前の動作状態に戻すことができます。

このベストプラクティスが確立されていない場合のリスクレベル: 中

実装のガイダンス

デプロイパイプラインを自動化します。デプロイパイプラインを使用すると、自動テストおよび異常の検出を呼び出せるようになります。また、本番環境へのデプロイを行う前の特定のステップでパイプラインを休止したり、変更を自動的にロールバックしたりできます。そのために欠かせないのが、継続的インテグレーションと継続的デリバリー/デプロイ (CI/CD) の文化の採用です。CI/CD では、コミットやコードの変更が、構築やテストの段階から本番環境へのデプロイまで、自動化されたさまざまな段階を経て導入されます。

運用上の最も難しい手順に人を関与させることが一般通念で推奨されていますが、最も難しい手順については、まさにこの理由から自動化を推奨します。

実装手順

デプロイを自動化して手動操作をなくすには、以下の手順に従います。

- コードを安全に保存するためのコードリポジトリを設定する: AWS CodeCommit を使用して、Git ベースの安全なリポジトリを作成します。
- ソースコードのコンパイル、テストの実行、デプロイアーティファクトの作成を行う継続的な統合サービスを構成する: この目的でビルドプロジェクトを設定するには、「Getting started with AWS CodeBuild using the console」を参照してください。
- エラーが発生しやすい手動デプロイに依存することなく、アプリケーションのデプロイを自動化し、複雑なアプリケーションの更新を処理するデプロイサービスを設定する: AWS CodeDeploy は、Amazon EC2、AWS Fargate、AWS Lambda、オンプレミスサーバーなどのさまざまなコンピューティングサービスへのソフトウェアのデプロイを自動化します。これらの手順を設定するには、「Getting started with CodeDeploy」を参照してください。
- リリースパイプラインを自動化する継続的デリバリーサービスを設定して、より早く、より信頼性の高いアプリケーションおよびインフラストラクチャの更新を実現する: AWS CodePipeline を使用してリリースパイプラインを自動化することを検討します。詳細については、「CodePipeline tutorials」を参照してください。

リソース

関連するベストプラクティス:

- OPS05-BP04 構築およびデプロイ管理システムを使用する
- OPS05-BP10 統合とデプロイを完全自動化する
- OPS06-BP02 デプロイをテストする

- OPS06-BP04 テストとロールバックを自動化する

関連するドキュメント:

- Continuous Delivery of Nested AWS CloudFormation Stacks Using AWS CodePipeline
- Complete CI/CD with AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, and AWS CodePipeline
- APN パートナー: 自動化されたデプロイソリューションの作成を支援できるパートナー
- AWS Marketplace: products that can be used to automate your deployments
- Webhook を使用してチャットメッセージを自動化する
- The Amazon Builders' Library: デプロイ時におけるロールバックの安全性の確保
- The Amazon Builders' Library: 継続的デリバリーによる高速化
- AWS CodePipeline とは
- CodeDeploy とは
- AWS Systems Manager Patch Manager
- Amazon SES とは
- Amazon Simple Notification Service とは

関連動画:

- AWS Summit 2019: CI/CD on AWS

障害管理

> 障害は発生するものであり、最終的にはすべてが時間の経過とともにフェイルオーバーします。つまり、ルーターからハードディスクまで、TCP パケットを破壊するオペレーティングシステムからメモリユニットまで、そして一時的なエラーから永続的な障害まで、どれもが対象となるのです。これは、最高品質のハードウェアを使用しているか、最低料金のコンポーネントを使用しているかにかかわらず、当たり前のことです - Werner Vogels、CTO、Amazon.com

低レベルのハードウェアコンポーネントの障害は、オンプレミスのデータセンターで毎日対処する必要がある問題です。ただし、クラウドでは、お客様はこれらのタイプの障害のほとんどから保護されるはずです。例えば、Amazon EBS ボリュームは特定のアベイラビリティーゾーンに配置され、そこで自動的にレプリケートされます。これにより、単一のコンポーネントに障害が発生した場合でもユーザーは保護されます。すべての EBS ボリュームは 99.999% の可用性を実現するよう設計されています。Amazon S3 オブジェクトは、最低 3 つのアベイラビリティーゾーンに保存され、年間 99.999999999% のオブジェクト耐久性を実現しています。クラウドプロバイダーに関係なく、障害がワークロードに影響を与える可能性があります。したがって、ワークロードの信頼性を確保する必要がある場合は、回復力を持たせる手順を実行しなければなりません。

ここで説明するベストプラクティスを適用するための前提条件として、ワークロードを設計、実装、および運用する担当者がビジネス目標とこれを達成するための信頼性目標を確実に把握しているようにする必要があります。その担当者は、信頼性要件を認識し、トレーニングを受ける必要があります。

以下のセクションでは、障害を管理してワークロードに影響を与えるのを防ぐためのベストプラクティスについて説明します。

トピック

- データのバックアップ方法
- 障害部分を切り離してワークロードを保護する
- コンポーネントの障害に耐えられるようにワークロードを設計する
- テストの信頼性
- 災害対策 (DR) を計画する

データのバックアップ方法

目標復旧時間 (RTO) と目標復旧時点 (RPO) の要件を満たすように、データ、アプリケーション、設定をバックアップします。

ベストプラクティス

- REL09-BP01 バックアップが必要なすべてのデータを特定し、バックアップする、またはソースからデータを再現する
- REL09-BP02 バックアップを保護し、暗号化する
- REL09-BP03 データバックアップを自動的に実行する
- REL09-BP04 データの定期的な復旧を行ってバックアップの完全性とプロセスを確認する

REL09-BP01 バックアップが必要なすべてのデータを特定し、バックアップする、またはソースからデータを再現する

ワークロードが使用するデータサービスとリソースのバックアップ機能を理解し、使用します。ほとんどのサービスは、ワークロードデータをバックアップする機能を提供します。

期待される成果: データソースが識別され、重要性に基づいて分類されています。次に、RPO に基づいてデータ復旧戦略を確立します。この戦略には、これらのデータソースをバックアップするか、他のソースからデータを再生成する能力を持つことが含まれます。データ損失の場合、実装された戦略によって、定義された RPO および RTO 内でのデータの復旧または再生成が可能になります。

クラウド成熟度フェーズ: Fondational

一般的なアンチパターン:

- ワークロードのすべてのデータソースとそれらの重要性を認識していない。
- 重要なデータソースのバックアップを取っていない。
- 重要性を評価基準として使用せずに、一部のデータソースのみのバックアップを取る。
- RPO が定義されていないか、バックアップの頻度が RPO を満たしていない。
- バックアップが必要かどうか、またはデータを他のソースから再生成できるかどうかを評価していない。

このベストプラクティスを活用するメリット: バックアップが必要な場所を特定し、バックアップを作成するメカニズムを実装するか、外部ソースからデータを再生成できるようにすることで、停止時にデータを復元し、復旧する能力が高まります。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

すべての AWS データストアは、バックアップ機能を備えています。Amazon RDS や Amazon DynamoDB などのサービスは、ポイントインタイムリカバリ (PITR) を有効にする自動バックアップを追加でサポートします。これにより、現在時刻の 5 分前までの任意の時刻にバックアップを復元することができます。多くの AWS サービスは、バックアップを別の AWS リージョン にコピーする機能を備えています。AWS Backup は、AWS サービス全体にわたるデータ保護を一元化して自動化する機能を提供するツールです。AWS Elastic Disaster Recovery を使用すると、サーバーのワークロード全体をコピーして、オンプレミス、クロス AZ、またはクロスリージョンから継続的なデータ保護を維持できます。目標復旧時点 (RPO) は秒単位で測定されます。

Amazon S3 をセルフマネージドおよび AWS マネージドデータソースのバックアップ先として使用できます。Amazon EBS、Amazon RDS、Amazon DynamoDB などの AWS サービスには、バックアップを作成する機能が組み込まれています。サードパーティー製のバックアップソフトウェアも使用できます。

オンプレミスのデータは、AWS Storage Gateway または AWS DataSync を使用して AWS クラウドにバックアップできます。このデータを AWS で保管するには、Amazon S3 バケットを使用できます。Amazon S3 は、Amazon S3 Glacier や S3 Glacier Deep Archive などの複数のストレージ層を提供し、データストレージコストを低減します。

他のソースからデータを再生成することによって、データリカバリのニーズを満たすこともできます。例えば、Amazon ElastiCache レプリカノードまたは Amazon RDS リードレプリカ を使用して、プライマリが失われた場合にデータを再生成できます。このようなソースを使用して目標復旧時点 (RPO) と目標復旧時間 (RTO) を満たすことができる場合には、バックアップは必要でないことがあります。別の例として、Amazon EMR を使用している場合、データを Amazon S3 から Amazon EMR に再生成できる限り、HDFS データストアをバックアップする必要がないことがあります。

バックアップ戦略を選択するときには、データの復旧にかかる時間を考慮してください。データの復旧に必要な時間は、バックアップのタイプ (バックアップ戦略の場合) やデータ再生成メカニズムの複雑性に依存します。この時間は、ワークロードの RTO 以内でなければなりません。

実装手順

1. ワークロードのすべてのデータソースを特定します。データは、データベース、ボリューム、ファイルシステム、ログ記録システム、オブジェクトストレージなどのさまざまなリソースに保存できます。「リソース」セクションを参照して、データが保存されているさまざまな AWS サービスに関する関連ドキュメントと、これらのサービスが提供するバックアップ機能を確認してください。
2. 重要度に基づいてデータソースを分類します。データセットごとにワークロードにとっての重要度が異なるため、回復力の要件も異なります。例えば、一部のデータは重要であり、ゼロに近い RPO を必要とするかもしれませんが、その他のデータは重要度が低く、より高い RPO や部分的なデータ損失に耐えられるかもしれません。同様に、データセットごとに RTO 要件も異なります。
3. AWS またはサードパーティーサービスを使用して、データのバックアップを作成します。AWS Backup は、AWS でさまざまなデータソースのバックアップを作成できるマネージドサービスです。AWS Elastic Disaster Recovery は、AWS リージョン への自動サブセカンドデータレプリケーションを処理します。また、AWS サービスのほとんどは、バックアップを作成する機能をネイティブで備えています。AWS Marketplace には、これらの機能を提供する多数のソリューションも用意されています。さまざまな AWS サービスからデータのバックアップを作成する方法に関する情報については、以下に記載されているリソースを参照してください。
4. バックアップしないデータについては、データ再生成メカニズムを確立します。さまざまな理由から、他のソースから再生成できるデータはバックアップしないという選択をすることもあるでしょう。バックアップの保管にコストがかかるため、バックアップを作成するよりも、必要なときにソースからデータを再現したほうが安いという状況もあるかもしれません。別の例としては、バックアップからの復元にかかる時間が、ソースからデータを再生成するよりも長く、結果として RTO に反する場合です。このような状況では、トレードオフを考慮して、データ復旧が必要なときに、これらのソースからデータを再生成する方法について、十分に定義されたプロセスを確立してください。例えば、データの分析を行うために、Amazon S3 からデータウェアハウス (Amazon Redshift など) 、または MapReduce クラスター (Amazon EMR など) にデータをロードしてある場合、これは他のソースから再生成できるデータの例になるかもしれません。これらの分析の結果がどこかに保存されているか再現可能である限り、データウェアハウスまたは MapReduce クラスターで発生した障害でデータが失われることはありません。ソースから再現できる例には他にも、キャッシュ (Amazon ElastiCache など) や RDS リードレプリカなどがあります。
5. データをバックアップするサイクルを確立します。データソースのバックアップの作成は定期的なプロセスであり、頻度は RPO に依存します。

実装計画に必要な工数レベル: 中

リソース

関連するベストプラクティス:

REL13-BP01 ダウンタイムやデータ消失に関する復旧目標を定義する

REL13-BP02 復旧目標を満たすため、定義された復旧戦略を使用する

関連するドキュメント:

- What Is AWS Backup? (AWS Backup とは)
- What is AWS DataSync? (AWS DataSync とは)
- ボリュームゲートウェイとは?
- APN パートナー: バックアップを支援できるパートナー
- AWS Marketplace: products that can be used for backup (AWS Marketplace: バックアップに使用できる製品)
- Amazon EBS スナップショット
- Backing Up Amazon EFS (Amazon EFS のバックアップ)
- バックアップの使用 - Amazon FSx for Windows File Server
- ElastiCache for Redis のバックアップと復元
- Creating a DB Cluster Snapshot in Neptune (Neptune での DB クラススナップショットの作成)
- DB スナップショットの作成
- Creating an EventBridge Rule That Triggers on a Schedule (スケジュールに従ってトリガーする Amazon EventBridge ルールの作成)
- Cross-Region Replication with Amazon S3 (Amazon S3 を使用したクロスリージョンレプリケーション)
- EFS-to-EFS AWS Backup (AWS の EFS-to-EFS バックアップ)
- Amazon S3 へのログデータのエクスポート
- オブジェクトのライフサイクル管理
- DynamoDB のオンデマンドバックアップと復元の使用
- DynamoDB のポイントインタイムリカバリ
- Amazon OpenSearch Service でのインデックススナップショットの作成

- What is AWS Elastic Disaster Recovery? (AWS Elastic Disaster Recovery とは)

関連動画:

- AWS re:Invent 2021 - Backup, disaster recovery, and ransomware protection with AWS (AWS re:Invent 2021 - AWS を使用したバックアップ、ディザスタリカバリ、ランサムウェア保護)
- AWS Backup Demo: Cross-Account and Cross-Region Backup (AWS Backup デモ: クロスアカウントおよびクロスリージョンバックアップ)
- AWS re:Invent 2019: Deep dive on AWS Backup, ft.Rackspace (STG341)

関連する例:

- Well-Architected Lab - Implementing Bi-Directional Cross-Region Replication (CRR) for Amazon S3 (Well-Architected ラボ - Amazon S3 の双方向クロスリージョンレプリケーション (CRR) の実装)
- Well-Architected Lab - Testing Backup and Restore of Data (Well-Architected ラボ - データのバックアップと復元のテスト)
- Well-Architected Lab - Backup and Restore with Failback for Analytics Workload (Well-Architected ラボ - 分析ワークロードのフェイルバックによるバックアップと復元)
- Well-Architected Lab - Disaster Recovery - Backup and Restore (Well-Architected ラボ - ディザスタリカバリ - バックアップと復元)

REL09-BP02 バックアップを保護し、暗号化する

認証と承認を使用して、バックアップへのアクセスを制御し、検出します。暗号化によりバックアップのデータ保全性が損なわれることを防止、検出します。

一般的なアンチパターン:

- データに対するのと同一の、バックアップおよび復元オートメーションへのアクセスを設定する。
- バックアップを暗号化しない。

このベストプラクティスを活用するメリット: バックアップを保護することで、データの改ざんを防止し、データの暗号化により、誤って公開されたデータへのアクセスが防止されます。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

AWS Identity and Access Management (IAM) などの認証と承認を使用して、バックアップへのアクセスを制御し、検出します。暗号化によりバックアップのデータ保全性が損なわれることを防止、検出します。

Amazon S3 は、保管時のデータを暗号化するための方法をいくつかサポートしています。Amazon S3 はサーバー側の暗号化を使用して、オブジェクトを暗号化されていないデータとして受け入れてから、保存時に暗号化します。クライアント側の暗号化を使用すると、ワークロードアプリケーションはデータを Amazon S3 に送信する前に暗号化することに対して責任を負います。どちらの方法でも、AWS Key Management Service (AWS KMS) を使ってデータキーを作成して保存することもできますし、自分でキーを用意し、そのキーに責任を持つこともできます。AWS KMS を使用すると、IAM を使用してポリシーを設定し、データキーと復号化されたデータにアクセスできるユーザーとアクセスできないユーザーにわけることができます。

Amazon RDS では、データベースの暗号化を選択すると、バックアップも暗号化されます。DynamoDB のバックアップは常に暗号化されます。AWS Elastic Disaster Recovery を使用すると、転送中および保管中のすべてのデータが暗号化されます。Elastic Disaster Recovery を使用すると、デフォルトの Amazon EBS 暗号化ボリューム暗号化キーまたはカスタムのカスタマーマネージドキーのいずれかを使用して、保管中のデータを暗号化できます。

実装手順

1. 各データストアで暗号化を使用します。ソースデータが暗号化されている場合、バックアップも暗号化されます。
   - Amazon RDS で暗号化を使用します。RDS インスタンスの作成時に、AWS Key Management Service を使用して、保管時の暗号化を設定できます。
   - Amazon EBS ボリュームで暗号化を使用します。デフォルトの暗号化を設定するか、ボリュームの作成時に一意のキーを指定できます。
   - 必要な Amazon DynamoDB 暗号化を使用します。DynamoDB は保管中のすべてのデータを暗号化します。AWS 所有の AWS KMS キーを使用するか、AWS マネージド KMS キーを使用して、アカウントに保存されるキーを指定できます。
   - Amazon EFS に保存しているデータを暗号化します。ファイルシステムを作成するときに暗号化を設定します。
   - 送信元と送信先のリージョンで暗号化を設定します。KMS に保存されているキーを使用して Amazon S3 で保管時の暗号化を設定できますが、キーはリージョン固有です。レプリケーションを設定するときに、送信先キーを指定できます。

- デフォルトの暗号化を設定するか、Elastic Disaster Recovery 向けの Amazon EBS 暗号化を使用するかを選択します。このオプションでは、ステージングエリアのサブネットディスクとレプリケートしたディスク上に保存されているレプリケートされたデータを暗号化します。

2. バックアップにアクセスするための最小特権のアクセス許可を実装します。「セキュリティのベストプラクティス」に従って、バックアップ、スナップショット、およびレプリカへのアクセスを制限します。

リソース

関連するドキュメント:

- AWS Marketplace: products that can be used for backup (AWS Marketplace: バックアップに使用できる製品)
- Amazon EBS 暗号化
- Amazon S3: 暗号化を使用したデータの保護
- CRR 追加のレプリケーション設定: AWS KMS に保存された暗号化キーでサーバー側の暗号化 (SSE) で作成されたオブジェクトをレプリケートする
- 保管時の DynamoDB 暗号化
- Encrypting Amazon RDS Resources (Amazon RDS リソースの暗号化)
- Encrypting Data and Metadata in Amazon EFS (EFS でのデータとメタデータの暗号化)
- Encryption for Backups in AWS (AWS でのバックアップの暗号化)
- 暗号化されたテーブルの管理
- AWS Well-Architected Framework - セキュリティの柱
- What is AWS Elastic Disaster Recovery? (AWS Elastic Disaster Recovery とは)

関連する例:

- Well-Architected Lab - Implementing Bi-Directional Cross-Region Replication (CRR) for Amazon S3 (Well-Architected ラボ - Amazon S3 の双方向クロスリージョンレプリケーション (CRR) の実装)

REL09-BP03 データバックアップを自動的に実行する

目標復旧時点 (RPO) によって、またはデータセット内の変更によって通知される定期的なスケジュールに基づいて、バックアップが自動的に行われるように設定します。データ損失の少ない重要

なデータセットは頻繁に自動バックアップする必要がありますが、多少の損失は許容できる重要度の低いデータは、バックアップの頻度を少なくすることができます。

期待される成果: 一定の周期でデータソースのバックアップを作成する自動化されたプロセス。

一般的なアンチパターン:

- バックアップを手動で実行する。
- バックアップ機能があるが、自動化にバックアップが含まれていないリソースを使用する。

このベストプラクティスを活用するメリット: バックアップを自動化することで、バックアップが RPO に基づいて定期的に作成され、作成されない場合はアラートが送信されます。

このベストプラクティスを確立しない場合のリスクレベル: 中

実装のガイダンス

AWS Backup を使用して、さまざまな AWS データソースの自動化されたデータバックアップを作成できます。Amazon RDS インスタンスは 5 分ごとにほぼ連続的にバックアップでき、Amazon S3 オブジェクトは 15 分ごとにほぼ連続的にバックアップできます。これにより、バックアップ履歴内の特定の時点へのポイントインタイムリカバリ (PITR) が可能になります。Amazon EBS ボリューム、Amazon DynamoDB テーブル、Amazon FSx ファイルシステムなど、その他の AWS データソースについては、AWS Backup は 1 時間ごとの頻度で自動バックアップを実行できます。これらのサービスでは、バックアップ機能もネイティブに提供されています。ポイントインタイムリカバリを備えた自動バックアップを提供する AWS サービスとしては、Amazon DynamoDB、Amazon RDS、Amazon Keyspaces (for Apache Cassandra) があります。これらはバックアップ履歴内の特定の時点への復元が可能です。その他の AWS データストレージサービスのほとんどが、1 時間ごとの定期バックアップをスケジュールする機能を提供しています。

Amazon RDS と Amazon DynamoDB は、ポイントインタイムリカバリを使用して継続的なバックアップを提供しています。Amazon S3 バージョニングは一度有効にすると、自動で実行されます。Amazon Data Lifecycle Manager は、Amazon EBS スナップショットの作成、コピー、削除を自動化するために使用できます。また、Amazon EBS-backed Amazon マシンイメージ (AMI) とその基盤となる Amazon EBS スナップショットの作成、コピー、廃止、および登録解除も自動化できます。

AWS Elastic Disaster Recovery は、ソース環境 (オンプレミスまたは AWS) からターゲットの復旧リージョンへの継続的なブロックレベルのレプリケーションを提供します。ポイントインタイム Amazon EBS スナップショットは、このサービスが自動的に作成し、管理します。

バックアップの自動化と履歴を一元的に確認できるようにするために、AWS Backup は完全マネージド型の、ポリシーベースのバックアップソリューションを提供します。AWS Storage Gateway を使用して、クラウド内およびオンプレミスの複数の AWS のサービスにわたってデータのバックアップを一元化および自動化します。

バージョン管理に加えて、Amazon S3 はレプリケーション機能も備えています。S3 バケット全体を同じまたは異なる AWS リージョン にある別のバケットに自動的にレプリケートできます。

実装手順

1. 現在手動でバックアップされているデータソースを特定します。詳細については、以下を参照してください REL09-BP01 バックアップが必要なすべてのデータを特定し、バックアップする、またはソースからデータを再現する.
2. ワークロードの RPO を決定します。詳細については、以下を参照してください REL13-BP01 ダウンタイムやデータ消失に関する復旧目標を定義する.
3. 自動バックアップソリューションまたはマネージドサービスを使用します。AWS Backup は、フルマネージドサービスで、AWS サービス、クラウド、オンプレミス全体にわたるデータ保護の一元化と自動化を容易にします。AWS Backup のバックアッププランを使用して、バックアップするリソースと、これらのバックアップを作成する頻度を定義するルールの作成します。この頻度は、ステップ 2 で確立した RPO によって通知される必要があります。AWS Backup を使用した自動バックアップの作成方法に関するハンズオントレーニングについては、「Testing Backup and Restore of Data」(データのバックアップと復元のテスト) を参照してください。データを保存するほとんどの AWS サービスでは、バックアップ機能がネイティブで提供されています。例えば、RDS は、ポイントインタイムリカバリ (PITR) 付きの自動バックアップに利用できます。
4. オンプレミスのデータソースやメッセージキューなどの自動バックアップソリューションやマネージドサービスでサポートされていないデータソースについては、信頼できるサードパーティーソリューションを使用して自動バックアップを作成することを検討してください。または、AWS CLI または SDK を使用してこれを行うオートメーションを作成することができます。AWS Lambda 関数または AWS Step Functions を使用して、データバックアップの作成にかかわるロジックを定義し、Amazon EventBridge を使用して RPO に基づく頻度で実行することができます。

実装計画に必要な工数レベル: 低。

リソース

関連するドキュメント:

- APN パートナー: バックアップを支援できるパートナー
- AWS Marketplace: products that can be used for backup (AWS Marketplace: バックアップに使用できる製品)
- Creating an EventBridge Rule That Triggers on a Schedule (スケジュールに従ってトリガーする Amazon EventBridge ルールの作成)
- What Is AWS Backup? (AWS Backup とは)
- What Is AWS Step Functions? (AWS Step Functions とは)
- What is AWS Elastic Disaster Recovery? (AWS Elastic Disaster Recovery とは)

関連動画:

- AWS re:Invent 2019: Deep dive on AWS Backup, ft.Rackspace (STG341)

関連する例:

- Well-Architected Lab - Testing Backup and Restore of Data (Well-Architected ラボ - データのバックアップと復元のテスト)

REL09-BP04 データの定期的な復旧を行ってバックアップの完全性とプロセスを確認する

復旧テストを実行して、バックアッププロセスの実装が目標復旧時間 (RTO) と目標復旧時点 (RPO) を満たしていることを検証します。

期待される成果: バックアップからのデータを、十分に定義されたメカニズムを使用して定期的に復旧することで、ワークロードについて確立された目標復旧時間 (RTO) 内での復旧が可能であることを確認できます。バックアップからの復元により、オリジナルデータを含むリソースになり、破損したりアクセス不能になっていたりするデータがなく、目標復旧時点 (RPO) 内のデータ損失であることを確認します。

一般的なアンチパターン:

- バックアップを復元しますが、復元が使用可能であることを確認するためのデータのクエリや取得は行いません。
- バックアップが存在することを前提とする。

- システムのバックアップが完全に動作可能であり、そこからデータを復旧できることを前提とする。
- バックアップからデータを復元または復旧する時間がワークロードの RTO の範囲内であることを前提とする。
- バックアップに含まれるデータがワークロードの RPO の範囲内であることを前提とする。
- ランブックを使用せずに、または確立された自動手順の外部で、必要に応じて復元する。

このベストプラクティスを活用するメリット: バックアップの復旧をテストすると、データの紛失や破損を心配せずに、必要なときにデータを復元できること、ワークロードの RTO 内で復元と復旧が可能であること、ならびにデータ損失がワークロードの RPO の範囲内であることを確認できます。

このベストプラクティスを確立しない場合のリスクレベル: 中

実装のガイダンス

バックアップおよび復元機能をテストすることで、停止時にこれらのアクションを実行できるという安心感が得られます。定期的にバックアップを新しい場所に復元して、テストを実行し、データの完全性を確認します。実行する必要がある一般的なテストには、すべてのデータが利用可能かどうか、破損していないかどうか、アクセス可能かどうか、データ損失がワークロードの RPO 内に収まるかどうかを確認することなどがあります。そのようなテストは、復旧メカニズムが十分に高速であり、ワークロードの RTO に対応できることを確認するのにも役立ちます。

AWS を使用して、テスト環境を立ち上げ、そこにバックアップを復元して RTO および RPO が機能するかを評価し、データコンテンツと完全性のテストを実行できます。

さらに、Amazon RDS および Amazon DynamoDB はポイントインタイムリカバリ (PITR) を許可します。継続的バックアップを使用すると、データセットを指定された日時の状態に復元できます。

すべてのデータが使用可能であり、破損しておらず、アクセス可能であり、データ損失がワークロードの RPO の範囲内であることを確認します。そのようなテストは、復旧メカニズムが十分に高速であり、ワークロードの RTO に対応できることを確認するのにも役立ちます。

AWS Elastic Disaster Recovery は、Amazon EBS ボリュームの継続的なポイントインタイムリカバリのスナップショットを提供します。ソースサーバーがレプリケートされると、設定済みのポリシーに基づいて、ポイントインタイムの状態が経時的に記録されます。Elastic Disaster Recovery を使用すると、トラフィックをリダイレクトせずにテストおよびドリル目的でインスタンスを起動することにより、これらのスナップショットの整合性を検証できます。

実装手順

1. 現在バックアップされているデータソースを特定し、バックアップが保存されている場所を確認します。実装のガイダンスについては、以下を参照してください REL09-BP01 バックアップが必要なすべてのデータを特定し、バックアップする、またはソースからデータを再現する.
2. 各データソースのデータ検証に使用する基準を確立します。データのタイプが異なると、プロパティも異なり、異なる検証メカニズムが必要になることがあります。本番環境での使用を決定する前に、このデータを検証する方法を考慮してください。データを検証するための一般的な方法としては、データタイプ、フォーマット、チェックサム、サイズ、またはこれらの組み合わせなど、データとバックアッププロパティをカスタム検証ロジックで使用することです。例えば、復元されたリソースと、バックアップが作成された時点でのデータソースの間でチェックサム値を比較します。
3. データの重要度に基づいて、データ復元の RTO と RPO を確立します。実装のガイダンスについては、以下を参照してください REL13-BP01 ダウンタイムやデータ消失に関する復旧目標を定義する.
4. データ復旧機能を評価します。バックアップおよび復元戦略をレビューして、RTO および RPO を満たせるかどうかを理解し、必要に応じて戦略を調整します。AWS Resilience Hub を使用して、ワークロードの評価を実行できます。アセスメントは、回復力ポリシーに対してアプリケーション設定を評価し、RTO および RPO 目標を満たすことができるかどうかを報告します。
5. 本番環境で現在確立されているデータ復元プロセスを使用してテスト復元を行います。これらのプロセスは、オリジナルデータソースのバックアップ方法、バックアップそのもののフォーマットとストレージ場所、またはデータが他のソースから再生成されるかどうかによって異なります。例えばこれは、AWS Backup などのマネージドサービスを使用している場合は、新しいリソースでバックアップを復元するという簡単な作業となります。AWS Elastic Disaster Recovery を使用した場合、リカバリドリルを開始できます。
6. 前のステップで確立したデータ検証の基準に基づいて、復元したリソースからのデータ回復を検証します。復元され、復旧されたデータには、バックアップ時点で最新のレコードまたはアイテムが含まれていますか? このデータはワークロードの RPO の範囲内ですか?
7. 復元と復旧に必要とした時間を測定して、決定済みの RTO と比較します。このプロセスは、ワークロードの RTO の範囲内ですか? 例えば、復元プロセスが開始されたときのタイムスタンプと復旧検証が完了したときのタイムスタンプを比較して、このプロセスの所要時間を計算します。すべての AWS API コールにはタイムスタンプが付けられており、この情報は、AWS CloudTrail で提供されています。この情報から復元プロセスが開始したときの詳細がわかりますが、検証が完了したときの終了タイムスタンプが検証ロジックによって記録される必要があります。自動化されたプロセスを使用している場合は、この情報の保存に Amazon DynamoDB などのサービスを使用できます。さらに、多くの AWS のサービスは、特定のアクションが発生したときのタイムスタンプ付きの情報を提供するイベント履歴を備えています。AWS Backup では、バックアップと

復元アクションは、ジョブと呼ばれており、これらのジョブにはメタデータの一部としてタイムスタンプ情報が含まれ、復元と復旧の所要時間の測定に使用できます。

8. データの検証に失敗した場合や、復元と復旧に必要な時間がワークロードについて確立された RTO を超えている場合は、ステークホルダーに通知します。このラボのように、このプロセスのオートメーションを実装する場合、Amazon Simple Notification Service (Amazon SNS) などのサービスを使用して、E メールや SMS などのプッシュ通知をステークホルダーに送信できます。これらのメッセージは、Amazon Chime、Slack、Microsoft Teams などのメッセージングアプリケーションに発行したり、AWS Systems Manager OpsCenter を使用して、OpsItems としてタスクを作成するために使用したりすることができます。
9. このプロセスを定期的に実行するように自動化します。例えば、AWS Lambda や AWS Step Functions の状態マシンなどのサービスを使用して、復元および復旧プロセスを自動化でき、Amazon EventBridge を使用して、下のアーキテクチャ図に示されているように、このオートメーションワークフローを定期的にトリガーすることができます。Automate data recovery validation with AWS Backup (AWS Backup を使用して復元データの検証を自動化する) 方法を確認してください。さらに、この Well-Architected ラボでは、いくつかのステップのオートメーション方法の 1 つについてのハンズオンエクスペリエンスを提供しています。

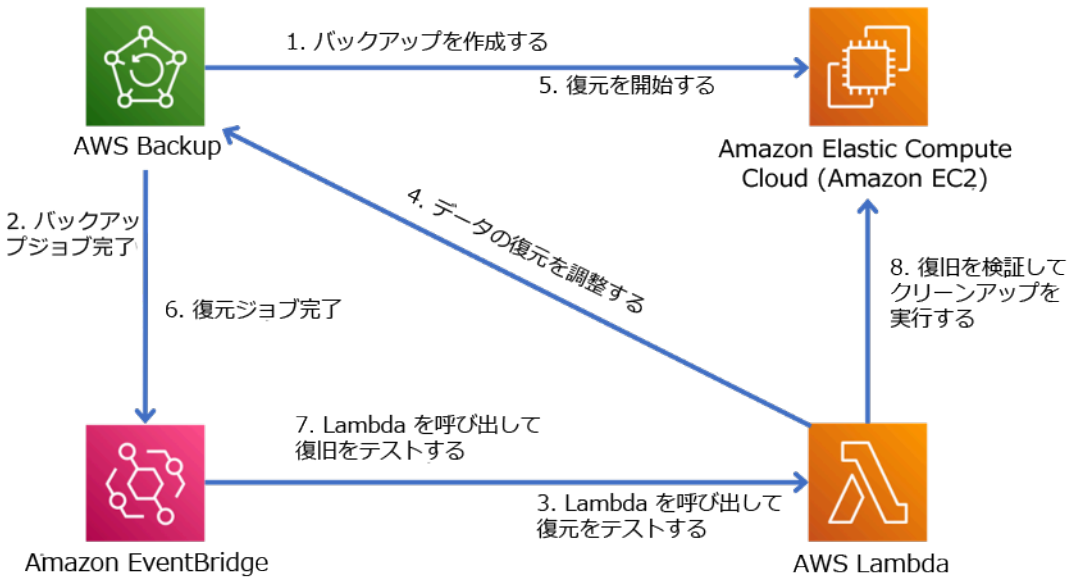


図 9.自動化されたバックアップおよび復元プロセス

実装計画に必要な工数レベル: 中 ~ 高 (検証基準の複雑性に依存)。

リソース

関連するドキュメント:

- Automate data recovery validation with AWS Backup (AWS Backup を使用して復元データの検証を自動化する)
- APN パートナー: バックアップを支援できるパートナー
- AWS Marketplace: products that can be used for backup (AWS Marketplace: バックアップに使用できる製品)
- Creating an EventBridge Rule That Triggers on a Schedule (スケジュールに従ってトリガーする Amazon EventBridge ルールの作成)
- DynamoDB のオンデマンドバックアップと復元の使用
- What Is AWS Backup? (AWS Backup とは)
- What Is AWS Step Functions? (AWS Step Functions とは)
- What is AWS Elastic Disaster Recovery (Elastic Disaster Recovery とは)
- AWS Elastic Disaster Recovery

関連する例:

- Well-Architected ラボ: データのバックアップと復元のテスト

障害部分を切り離してワークロードを保護する

障害部分を分離した境界は、ワークロード内の障害の影響を限られた数のコンポーネントに限定します。境界外のコンポーネントは、障害の影響を受けません。障害部分を切り離した境界を複数使用すると、ワークロードへの影響を制限できます。

ベストプラクティス

- REL10-BP01 複数の場所にワークロードをデプロイする
- REL10-BP02 マルチロケーションデプロイ用の適切な場所の選択
- REL10-BP03 単一のロケーションに制約されるコンポーネントのリカバリを自動化する
- REL10-BP04 バルクヘッドアーキテクチャを使用して影響範囲を制限する

REL10-BP01 複数の場所にワークロードをデプロイする

ワークロードのデータとリソースを複数のアベイラビリティゾーンに分散するか、必要に応じて複数の AWS リージョン にまたがって分散します。これらのロケーションは、必要に応じて多様化できます。

AWS のサービス設計の基本原則の 1 つは、基盤となる物理インフラストラクチャの単一障害点を回避することです。これによって、複数のアベイラビリティーゾーンを使用して単一ゾーンで起こる障害に耐えられるソフトウェアおよびシステムを構築することができます。これと同様に、システムは単一のコンピューティングノード、単一のストレージボリューム、単一のデータベースインスタンスの障害に対する弾力性を持つように設計されています。冗長コンポーネントに依存するシステムを構築するときには、それぞれのコンポーネントが独立して動作すること、また、AWS リージョン の場合は、それぞれのリージョンが自律して動作することが重要です。冗長化されたコンポーネントを使用した理論的な可用性の計算から得られるメリットは、これが当てはまる場合にのみ有効です。

アベイラビリティゾーン (AZ)

AWS リージョン は、互いに独立するように設計された 2 つ以上のアベイラビリティゾーンで構成されます。各アベイラビリティゾーンは、火災、洪水、竜巻などの自然災害による障害の影響を回避するため、ほかのゾーンから物理的に大きな距離を隔てています。各アベイラビリティゾーンは、商用電源への専用接続、スタンドアロンのバックアップ電源、独立したメカニカルサービス、アベイラビリティゾーン内外の独立したネットワーク接続など、独立した物理インフラストラクチャも備えています。この設計により、これらのシステムのいずれかに障害が発生した場合、影響を受ける AZ は 1 だけで済みます。地理的には分離されていても、アベイラビリティゾーンは、同じリージョンのエリアに配置されているため、高スループットで低レイテンシーなネットワーク接続が可能です。AWS リージョン 全体 (すべてのアベイラビリティゾーンにまたがり、複数の物理的に独立したデータセンターで構成される) をワークロードの単一の論理的なデプロイ先として扱うことができ、同期したデータレプリケーション (例えば、データベース間で) が可能です。このようにして、アクティブ/アクティブまたはアクティブ/スタンバイの設定でアベイラビリティゾーンを使用することができます。

アベイラビリティゾーンは独立しているため、ワークロードが複数のゾーンを使用するように設定された場合、ワークロードの可用性が向上します。一部の AWS サービス (Amazon EC2 インスタンスデータプレーンを含む) は、厳密なゾーンサービスとしてデプロイされるため、属するアベイラビリティゾーンに左右されます。ただし、他の AZ 内の Amazon EC2 インスタンスは影響を受けず、機能し続けます。同様に、アベイラビリティゾーンの障害によって Amazon Aurora データベースに障害が発生した場合、影響を受けない AZ のリードレプリカ Aurora インスタンスは自動的にプライマリに昇格できます。一方、Amazon DynamoDB などのリージョンにおける AWS のサービスは、サービスの可用性の設計目標を達成するために、内部ではアクティブ/アクティブ設定で複数のアベイラビリティゾーンを使用するため、AZ 配置を設定する必要はありません。

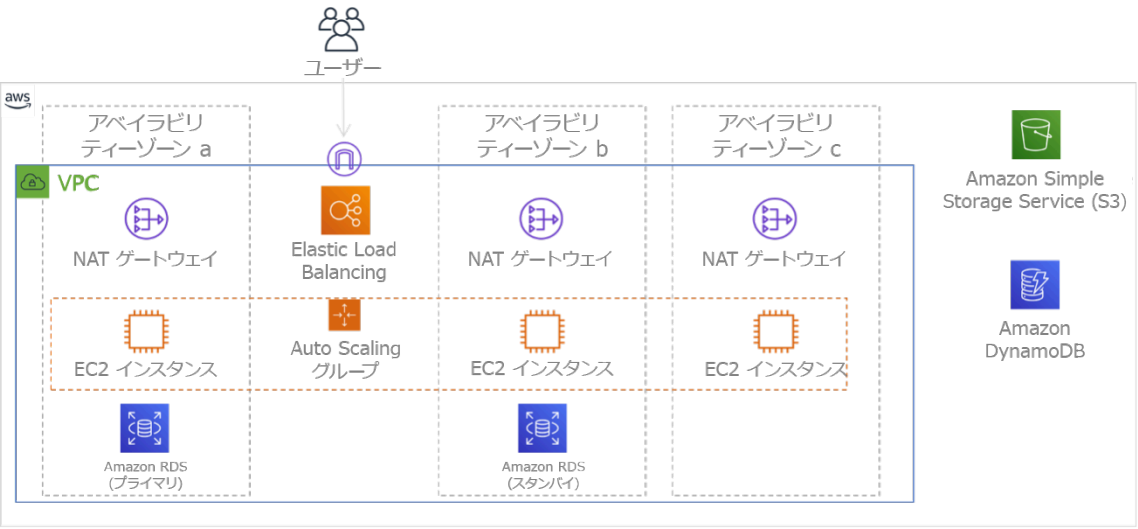


図 9: 3 つのアベイラビリティーゾーンにまたがってデプロイされる多階層アーキテクチャ。Amazon S3 と Amazon DynamoDB は常に自動的にマルチ AZ であることに注意してください。ELB も 3 つのゾーンすべてにデプロイされます。

AWS コントロールプレーンは、通常、リージョン全体 (複数のアベイラビリティゾーン) 内のリソースを管理する機能を提供しますが、特定のコントロールプレーン (Amazon EC2 および Amazon EBS を含む) は、結果を単一のアベイラビリティゾーンにフィルタリングする機能を備えています。これを実行すると、指定されたアベイラビリティーゾーン内でのみリクエストが処理されるため、その他のアベイラビリティーゾーンで起こる障害からの影響を軽減できます。この AWS CLI の例は、us-east-2c アベイラビリティゾーンからのみ Amazon EC2 インスタンス情報を取得する例を示しています。

```
AWS ec2 describe-instances --filters Name=availability-zone,Values=us-east-2c
```

AWS ローカルゾーン

AWS ローカルゾーンは、サブネットや EC2 インスタンスなど、ゾーン内の AWS リソースの配置場所として選択できるという点で、それぞれの AWS リージョン リージョン内でアベイラビリティゾーンと同じように機能します。それらが特別なのは、関連する AWS リージョン リージョンにあるのではなく、現在 AWS リージョン が存在しない大規模な人口、産業、IT センターの近くにあることです。それでも、ローカルゾーンのローカルワークロードと AWS リージョン で実行されているワークロードの間で、高帯域幅で安全な接続を維持します。ワークロードをユーザーに近い場所にデプロイし、低レイテンシー要件を満たすには、AWS ローカルゾーンを使用する必要があります。

Amazon Global Edge Network

Amazon Global Edge Network は、世界中の都市のエッジロケーションで構成されます。Amazon CloudFront は、このネットワークを使用して、コンテンツをエンドユーザーに低レイテンシーで配信します。AWS Global Accelerator を使用すると、これらのエッジロケーションにワークロードエンドポイントを作成して、ユーザーに近い AWS グローバルネットワークへのオンボーディングを提供できます。Amazon API Gateway は、CloudFront ディストリビューションを使用してエッジ最適化された API エンドポイントを有効にし、最も近いエッジロケーションを経由してクライアントアクセスを容易にします。

AWS リージョン

AWS リージョン は自律するように設計されているため、マルチリージョンアプローチを使用するには、各リージョンにサービスの専用コピーをデプロイします。

マルチリージョンアプローチは、1 回限りの大規模なイベントが発生したときに復旧目標を満たすための ディザスタリカバリ 戦略に一般的です。把握 災害対策 (DR) を計画する これらの戦略の詳細について確認してください。ただし、ここでは、 可用性に焦点を当てて、平均アップタイム目標の実現を目指します。高可用性目標については、マルチリージョンアーキテクチャは、一般に、アクティブ/アクティブとして設計され、(それぞれのリージョンの) 各サービスコピーはアクティブです (リクエストを処理します)。

> 推奨
>
> ほとんどのワークロードの可用性目標は、単一の AWS リージョン 内でマルチ AZ 戦略を使用して満たすことができます。マルチリージョンアーキテクチャは、ワークロードの可用性要件が極端に高いか、その他のビジネス目標のために、マルチリージョンアーキテクチャが必要とされる場合のみ検討してください。

AWS は、お客様がリージョンをまたいでサービスを運用できるようにしています。例えば、AWS は、Amazon Simple Storage Service (Amazon S3) レプリケーション、Amazon RDS リードレプリカ (Aurora リードレプリカを含む)、Amazon DynamoDB グローバルテーブルを使用して、連続的な非同期データレプリケーションを提供します。連続レプリケーションでは、アクティブリージョンのそれぞれでデータのバージョンをほとんどすぐに使用できます。

AWS CloudFormation を使用して、インフラストラクチャを定義し、複数の AWS アカウント と複数の AWS リージョン にまたがって一貫してデプロイできます。また、AWS CloudFormation StackSets は、この機能を拡張し、複数のアカウントとリージョンにまたがって単一のオペレーションで AWS CloudFormation スタックの作成、更新、または削除が可能です。Amazon EC2 インスタンスのデプロイの場合、AMI (Amazon マシンイメージ) は、ハードウェア設定やインストールされ

たソフトウェアなどの情報を提供するために使用されます。Amazon EC2 Image Builder パイプラインを実装して、必要な AMI を作成し、これらをアクティブリージョンにコピーできます。これにより、これらの ゴールデン AMI は、新しい各リージョンにワークロードをデプロイし、スケールアウトするために必要なすべてを備えることになります。

トラフィックをルーティングするために、Amazon Route 53 と AWS Global Accelerator の両方により、どのユーザーをどのアクティブリージョンエンドポイントに向かわせるかを決定するポリシーを定義できます。Global Accelerator では、トラフィックダイヤルを設定して、各アプリケーションエンドポイントに向かうトラフィックのパーセンテージを制御します。Route 53 は、このパーセンテージアプローチをサポートするほか、地理的近接性やレイテンシーに基づくものなど、他の複数の可用性ポリシーもサポートします。Global Accelerator は、AWS エッジサーバーの拡張ネットワークを自動的に利用して、AWS ネットワークバックボーンへのトラフィックを可能な限りすぐにオンボードするため、リクエストのレイテンシーが低下します。

これらの機能はすべて、各リージョンの自律性を保つように動作します。このアプローチには、ごくわずかな例外があり、AWS Identity and Access Management (IAM) サービスのコントロールプレーンと共に、グローバルエッジデリバリーを提供するサービス (Amazon CloudFront や Amazon Route 53 など) が含まれます。大部分のサービスが、完全に単一リージョン内で運用されています。

オンプレミスのデータセンター

オンプレミスのデータセンターで実行されるワークロードに関しては、可能な場合はハイブリッドエクスペリエンスを設計します。AWS Direct Connect は、オンプレミスから AWS への専用ネットワーク接続を提供し、両方での実行が可能になります。

もう 1 つのオプションは、AWS Outposts を使用して、AWS インフラストラクチャとサービスをオンプレミスで実行することです。AWS Outposts は、AWS インフラストラクチャ、AWS のサービス、API、ツールをデータセンターに拡張する完全マネージド型サービスです。AWS クラウドで使用されているのと同じハードウェアインフラストラクチャがデータセンターにインストールされます。その後、AWS Outposts は最も近い AWS リージョン に接続されます。その結果、AWS Outposts を使用して、低レイテンシーまたはローカルデータ処理を要求されるワークロードをサポートできます。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

- 複数のアベイラビリティゾーンと AWS リージョン を使用します。ワークロードのデータとリソースを複数のアベイラビリティゾーンに分散するか、必要に応じて複数の AWS リージョン にまたがって分散します。これらの口ケーションは、必要に応じて多様化できます。

- リージョンサービスは初めからアベイラビリティーゾーンにデプロイされます。
  - これには、Amazon S3、Amazon DynamoDB、AWS Lambda (VPC に接続されていない場合) が含まれます。
- コンテナ、インスタンス、機能ベースのワークロードを複数のアベイラビリティーゾーンにデプロイします。キャッシュを含め、マルチゾーンデータストアを使用します。EC2 Auto Scaling、ECS タスク配置、AWS Lambda 関数の設定 (VPC で実行するとき)、および ElastiCache クラスターの機能を使用します。
  - Auto Scaling グループをデプロイするときには、アベイラビリティーゾーンの異なるサブネットを使用します。
    - 例: 複数のアベイラビリティーゾーンにインスタンスを分散する
    - Amazon ECS タスク配置戦略
    - Amazon VPC 内のリソースにアクセスできるように AWS Lambda 関数を設定する
    - リージョンとアベイラビリティーゾーンを選択する
  - Auto Scaling グループをデプロイするときには、アベイラビリティーゾーンの異なるサブネットを使用します。
    - 例: 複数のアベイラビリティーゾーンにインスタンスを分散する
  - タスク配置パラメータを使用して、DB サブネットグループを指定します。
    - Amazon ECS タスク配置戦略
  - VPC で実行する関数を設定するには、複数のアベイラビリティーゾーンでサブネットを使用します。
    - Amazon VPC 内のリソースにアクセスできるように AWS Lambda 関数を設定する
  - ElastiCache クラスターには複数のアベイラビリティーゾーンを使用します。
    - リージョンとアベイラビリティーゾーンを選択する
- ワークロードを複数のリージョンにデプロイする必要がある場合は、マルチリージョン戦略を選択します。信頼性に関するほとんどのニーズは、マルチアベイラビリティゾーン戦略を使用して単一の AWS リージョン 内で満たすことができます。必要に応じて、ビジネスニーズを満たすためにマルチリージョン戦略を使用します。
  - AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)
    - 別の AWS リージョン にバックアップすると、必要なときにデータが利用可能になるという保証のレイヤーを追加できます。
    - ワークロードによっては、マルチリージョン戦略の使用を必要とする規制要件があります。

- ワークロードの AWS Outposts を評価します。ワークロードでオンプレミスのデータセンターへの低レイテンシーが必要な場合、またはローカルのデータ処理要件がある場合があります。その場合、AWS Outposts を使用して、オンプレミスで AWS インフラストラクチャとサービスを実行します。
  - 「What is AWS Outposts?」
- AWS Local Zones がユーザーにサービスを提供するのに役立つかどうかを判断します。低レイテンシー要件がある場合は、AWS Local Zones がユーザーの近くにあるかどうかを確認してください。近くにある場合は、それらのユーザーの近くにワークロードをデプロイするのに使用します。
  - AWS Local Zones のよくある質問

リソース

関連するドキュメント:

- AWS グローバルインフラストラクチャ
- AWS Local Zones のよくある質問
- Amazon ECS タスク配置戦略
- リージョンとアベイラビリティーゾーンを選択する
- 例: 複数のアベイラビリティーゾーンにインスタンスを分散する
- グローバルテーブル: DynamoDB を使用したマルチリージョンレプリケーション
- Amazon Aurora グローバルデータベースの使用
- AWS のサービスによるマルチリージョンアプリケーションの作成 (ブログシリーズ)
- 「What is AWS Outposts?」

関連動画:

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)
- AWS re:Invent 2019: Innovation and operation of the AWS global network infrastructure (NET339)

REL10-BP02 マルチロケーションデプロイ用の適切な場所の選択

期待される成果

高可用性のためには、(可能なときには) 常に、図 10 に示されているように、ワークロードコンポーネントを複数のアベイラビリティゾーン (AZ) にデプロイします。回復力要件が極端に高いワークロードについては、マルチリージョンアーキテクチャのオプションを慎重に評価してください。

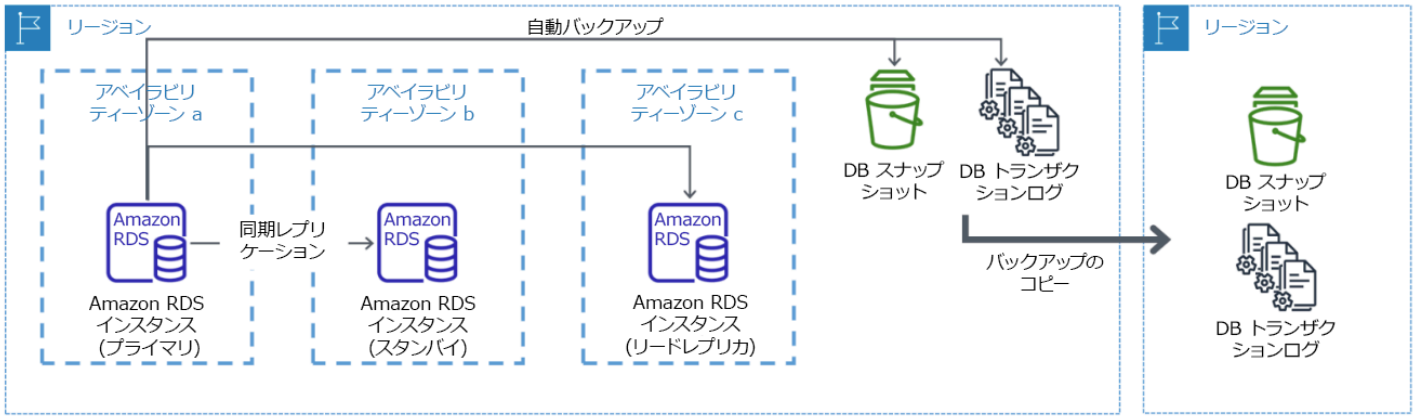


図 10: 別の AWS リージョンへのバックアップを備えた回復力の高いマルチ AZ データベースデプロイ

一般的なアンチパターン:

- マルチ AZ アーキテクチャで要件を満たせるときに、マルチリージョンアーキテクチャの設計を選択する。
- 回復力要件とマルチロケーション要件がアプリケーションコンポーネント間で異なる場合に、コンポーネント間の依存関係を考慮しない。

このベストプラクティスを活用するメリット:

回復力のためには、複数の防御層を構築するアプローチを使用する必要があります。1 つの層では、複数の AZ を使用して高可用性アーキテクチャを構築することによって、小規模で一般的な混乱に対して保護します。もう 1 つの防御層では、広範囲の自然災害やリージョンレベルの混乱など、まれな出来事に対して保護します。この 2 番目の層には、複数の AWS リージョン にまたがるアプリケーションの設計が必要です。

- 99.5% の可用性と 99.9% の可用性の違いは、1 か月あたり 3.5 時間に相当します。複数の AZ で期待されるワークロードの可用性を達成するには、99.99% が必要です。

- 複数の AZ でワークロードを実行することにより、電力、冷却、ネットワークの障害、および火災や洪水などの自然災害のほとんどを分離できます。
- ワークロードのためのマルチリージョン戦略の実装は、国の広い範囲に影響を与える自然災害や、リージョン全体に及ぶ技術的障害に対する保護に役立ちます。マルチリージョンアーキテクチャの実装はかなり複雑になることがあり、通常、ほとんどのワークロードには不要である点に注意してください。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

1 つのアベイラビリティゾーンの混乱または部分的損失に基づく災害イベントについては、単一の AWS リージョン 内の複数のアベイラビリティゾーンで高可用性ワークロードを実装すると、自然災害または技術的な災害の軽減に役立ちます。各 AWS リージョン は複数のアベイラビリティゾーンで構成され、各ゾーンは他のゾーンの障害から隔離され、かなりの距離によって分離されます。ただし、相互にかなりの距離を隔てた複数のアベラビリティゾーンコンポーネントの損失リスクがある災害については、リージョン規模の障害を緩和するディザスタリカバリオプションを実装する必要があります。極端に高い回復力を必要とするワークロードについては (重要なインフラストラクチャ、医療関連のアプリケーション、財務システムインフラストラクチャなど)、マルチリージョン戦略が必要なことがあります。

実装手順

1. ワークロードを評価して、回復力ニーズをマルチ AZ アプローチ (単一の AWS リージョン) で満たせるか、それともマルチリージョンアプローチが必要かを判断します。これらの要件を満たすためにマルチリージョンアーキテクチャを実装すると、複雑性が増すため、ユースケースと要件を慎重に考慮してください。回復力の要件は、ほぼ常に、単一の AWS リージョン を使用して満たすことができます。複数のリージョンを使用する必要があるかどうかを判断するときには、次のような要件を考慮してください。
   a. ディザスタリカバリ (DR): 1 つのアベイラビリティゾーンの混乱または部分的損失に基づく災害イベントについては、単一の AWS リージョン 内の複数のアベイラビリティゾーンで高可用性ワークロードを実装すると、自然災害または技術的な災害の軽減に役立ちます。相互にかなりの距離を隔てた複数のアベイラビリティゾーンコンポーネントの損失リスクがある災害については、リージョン規模の自然災害や技術的障害を緩和するために、複数のリージョンにまたがるディザスタリカバリを実装する必要があります。
   b. 高可用性 (HA): マルチリージョンアーキテクチャ (各リージョンで複数の AZ を使用) を使用して、99.99% 以上の可用性を達成できます。

c. スタックローカリゼーション: 世界中のオーディエンスにワークロードをデプロイするときには、異なる AWS リージョン にローカライズしたスタックをデプロイして、それらのリージョンのオーディエンスに対応できます。ローカリゼーションには、言語、通貨、および保存されるデータのタイプが含まれます。

d. ユーザーへの近接性: 世界中のオーディエンスにワークロードをデプロイするときには、エンドユーザーに近い AWS リージョン にスタックをデプロイすることによって、レイテンシーを軽減できます。

e. データレジデンシー: 一部のワークロードはデータレジデンシー要件の対象であり、特定のユーザーからのデータは特定の国境内にとどめる必要があります。該当する規制に基づいて、それらの国境内の AWS リージョン にスタック全体をデプロイするか、データだけをデプロイするかを選ぶことができます。

2. AWS のサービスによって提供されるマルチ AZ 機能の例をいくつか示します。

a. EC2 または ECS を使用するワークロードを保護するには、コンピューティングリソースの前に Elastic Load Balancer をデプロイします。Elastic Load Balancing は、異常のあるゾーンのインスタンスを検出して、正常なゾーンにトラフィックをルーティングするソリューションを提供します。

i. Application Load Balancers の開始方法

ii. Network Load Balancer の開始方法

b. ロードバランシングをサポートしない市販のソフトウェアを実行する EC2 インスタンスの場合、マルチ AZ ディザスタリカバリ方法論を実装することによって、一種の耐障害性を達成できます。

i. the section called “REL13-BP02 復旧目標を満たすため、定義された復旧戦略を使用する”

c. Amazon ECS タスクの場合は、3 つの AZ に均等にサービスをデプロイして、可用性とコストのバランスを達成します。

i. Amazon ECS の可用性のベストプラクティス | コンテナ

d. 非 Aurora Amazon RDS の場合、マルチ AZ を設定オプションとして選ぶことができます。プライマリデータベースインスタンスに障害が発生した場合、Amazon RDS はスタンバイデータベースを自動的に昇格させて、別のアベイラビリティゾーンのトラフィックを受け取ります。マルチリージョンリードレプリカを作成して、回復力を高めることもできます。

i. Amazon RDS マルチ AZ デプロイ

ii. 別の AWS リージョン でのリードレプリカの作成

3. AWS のサービスによって提供されるマルチリージョン機能の例をいくつか示します。

a. マルチ AZ の可用性がサービスによって自動的に提供される Amazon S3 ワークロードの場合、マルチリージョンデプロイが必要な場合は、マルチリージョンアクセスポイントを検討してください。

   i. Amazon S3 のマルチリージョンアクセスポイント

b. マルチ AZ の可用性がサービスによって自動的に提供される DynamoDB テーブルの場合、既存のテーブルをグローバルテーブルに容易に変換して、複数リージョンの利点を活かすことができます。

   i. 単一リージョン Amazon DynamoDB テーブルをグローバルテーブルに変換する

c. ワークロードの前に Application Load Balancers または Network Load Balancer がある場合には、AWS Global Accelerator を使用し、正常なエンドポイントを含んでいる複数のリージョンにトラフィックを向けることで、アプリケーションの可用性を高めます。

   i. AWS Global Accelerator にける標準アクセラレーター用エンドポイント - AWS Global Accelerator (amazon.com)

d. AWS EventBridge を利用するアプリケーションの場合、クロスリージョンバスによってイベントを、選択したほかのリージョンに転送することを検討してください。

   i. Amazon EventBridge イベントの AWS リージョン 間での送受信

e. Amazon Aurora データベースの場合、複数の AWS リージョンにまたがる Aurora グローバルデータベースを検討してください。既存のクラスターを変更して、新しいリージョンも追加できます。

   i. Amazon Aurora グローバルデータベースの開始方法

f. ワークロードに AWS Key Management Service (AWS KMS) 暗号化キーが含まれる場合は、マルチリージョンキーがアプリケーションに適切かどうかを考慮してください。

   i. AWS KMS のマルチリージョンキー

g. その他の AWS サービスの機能については、このブログシリーズの AWS のサービスによるマルチリージョンアプリケーションの作成シリーズを参照してください。

実装計画の工数レベル: 中 ~ 高

リソース

関連するドキュメント:

- AWS のサービスによるマルチリージョンアプリケーションの作成シリーズを参照してください。

- AWS のディザスタリカバリ (DR) アーキテクチャ、パート IV: マルチサイトアクティブ/アクティブ
- AWS グローバルインフラストラクチャ
- AWS Local Zones のよくある質問
- AWS のディザスタリカバリ (DR) アーキテクチャ、パート I: クラウドでのリカバリ戦略
- クラウド上の災害対策はさまざまある
- グローバルテーブル: DynamoDB を使用したマルチリージョンレプリケーション

関連動画:

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)
- Auth0: 自動フェイルオーバーにより、月あたり 15 億回以上のログインにスケールするマルチリージョンの高可用性アーキテクチャ

関連する例:

- AWS のディザスタリカバリ (DR) アーキテクチャ、パート I: クラウドでのリカバリ戦略
- DTCC はオンプレミスで実行できることをはるかに超えた回復力を達成
- Expedia Group は専有 DNS サービスでマルチリージョン、マルチアベイラビリティゾーンを使用して、アプリケーションに回復力を追加
- Uber: マルチリージョン Kafka の災害対策
- Netflix: マルチリージョンの回復力のためのアクティブ/アクティブ
- Atlassian Cloud のデータ回復力を構築する方法
- Intuit TurboTax は 2 つのリージョンで実行

REL10-BP03 単一のロケーションに制約されるコンポーネントのリカバリを自動化する

ワークロードのコンポーネントを実行できるのが単一のアベイラビリティーゾーンまたはオンプレミスのデータセンターでのみである場合は、定義した復旧目標内でワークロードを全面的に再構築する機能を実装する必要があります。

このベストプラクティスを確立しない場合のリスクレベル: 中

実装のガイダンス

技術的な制約のためにワークロードを複数のロケーションにデプロイするベストプラクティスが不可能な場合は、回復性を確保するための代替パスを採り入れる必要があります。このような場合、必要なインフラストラクチャを再作成し、アプリケーションを再デプロイし、必要なデータを再作成する機能を自動化する必要があります。

例えば、Amazon EMR は同じアベイラビリティーゾーンで特定のクラスターのすべてのノードを起動します。これは、同じゾーンでクラスターを実行すると、データアクセス率が高くなり、ジョブフローのパフォーマンスが向上するためです。このコンポーネントがワークロードの回復力のために必要な場合は、クラスターとそのデータを再デプロイする方法が必要です。また、Amazon EMR では、マルチ AZ を使用する以外の方法で冗長性をプロビジョニングする必要があります。複数のノードをプロビジョニングできます。EMR File System (EMRFS)を使用すると、EMR のデータを Amazon S3 に保存でき、今度はそのデータを複数のアベイラビリティーゾーンまたは複数の AWS リージョン にわたりレプリケートできます。

Amazon Redshift も同様に、デフォルトでは、選択した AWS リージョン 内のランダムに選択されたアベイラビリティゾーンにクラスターをプロビジョニングします。すべてのクラスターノードが同じゾーンにプロビジョニングされます。

オンプレミスのデータセンターにデプロイされたサーバベースのステートフルなワークロードの場合、AWS Elastic Disaster Recovery を使用して AWS のワークロードを保護できます既に AWS でホストされてる場合、Elastic Disaster Recovery を使用してワークロードを別のアベイラビリティゾーンまたはリージョンに保護できます。Elastic Disaster Recovery は、軽量のステージングエリアへのブロックレベルの継続的なレプリケーションを行い、オンプレミスおよびクラウドベースのアプリケーションの高速かつ信頼性の高い復旧を提供します。

実装手順

1. 自己修復を実装します。可能であれば自動スケーリングを利用して、インスタンスとコンテナをデプロイします。自動スケーリングを利用できない場合は、EC2 インスタンスの自動復旧機能を利用するか、Amazon EC2 または ECS のコンテナのライフサイクルイベントに基づいた自己修復自動化を実装します。
    - 単一インスタンス IP アドレスや、プライベート IP アドレス、Elastic IP アドレス、インスタンスメタデータを必要としないインスタンスとコンテナのワークロードには、Amazon EC2 Auto Scaling グループを使用します。
        - 起動テンプレートのユーザーデータを使用して、ほとんどのワークロードを自己修復できるオートメーションを実装できます。

- 単一インスタンス IP アドレスや、プライベート IP アドレス、elastic IP アドレス、インスタンスメタデータを必要とするワークロードには、Amazon EC2 インスタンスの自己復旧機能を使用します。
  - 自動復旧は、インスタンスの障害が検出されると、復旧ステータスアラートを SNS トピックに送信します。
- オートスケーリングや EC2 の復旧機能を利用できない場合は、Amazon EC2 インスタンスのライフサイクルイベントや Amazon ECS イベントを利用して、自己修復を自動化します。
  - 必要なプロセスロジックに従ってコンポーネントを修復するオートメーションを呼び出すには、イベントを利用します。
- 単一のロケーションに制限したステートフルワークロードは、AWS Elastic Disaster Recovery を使用して保護します。

リソース

関連するドキュメント:

- Amazon ECS イベント
- Amazon EC2 Auto Scaling のライフサイクルフック
- インスタンスの復旧
- サービスのオートスケーリング
- Amazon EC2 Auto Scaling とは
- AWS Elastic Disaster Recovery

REL10-BP04 バルクヘッドアーキテクチャを使用して影響範囲を制限する

バルクヘッドアーキテクチャ (セルベースアーキテクチャとも呼ばれる) を実装して、ワークロード内の障害の影響を限られた数のコンポーネントに制限します。

期待される成果: セルベースアーキテクチャでは、複数の分離されたワークロードのインスタンスが使用されます。各インスタンスはセルと呼ばれます。各セルは独立しており、その他のセルとは状態を共有せず、ワークロードのリクエスト全体のサブセットを処理します。これにより、不適切なソフトウェア更新などの障害による個別のセルおよび処理中のリクエストへの予測される影響が軽減されます。例えばワークロードが 10 個のセルを使用して 100 個のリクエストを処理していると、障害が発生した場合、リクエスト全体の 90% は障害の影響を受けません。

一般的なアンチパターン:

- セルを際限なく増殖させる。
- コードの更新やデプロイをすべてのセルに同時に適用する。
- セル間で状態またはコンポーネントを共有する (ルーターレイヤーを除く)。
- 複雑なビジネスロジックやルーティングロジックをルーターレイヤーに追加する。
- セル間のインタラクションを最小に抑えない。

このベストプラクティスを活用するメリット: セルベースアーキテクチャでは、多数の一般的なタイプの障害がセル自体に封じ込めるため、障害分離が向上します。このような障害境界は、コードのデプロイの失敗や、破損したリクエスト、または特定の障害モードをトリガーするリクエスト (ポイズンピルリクエストとも呼ばれる) など、これ以外の方法では封じ込めが困難なタイプの障害への回復力を実現します。

実装のガイダンス

船ではバルクヘッドが使用されており、船体が破損しても、船体の一部のセクションのみに封じ込めることができます。複雑なシステムでは、このパターンを模して障害分離を実現する場合がよくあります。障害部分を分離した境界は、ワークロード内の障害の影響を限られた数のコンポーネントに限定します。境界外のコンポーネントは、障害の影響を受けません。障害部分を切り離した境界を複数使用すると、ワークロードへの影響を制限できます。AWS では、複数のアベイラビリティーゾーンとリージョンを使用して障害分離を実現できます。この障害分離の概念はワークロードのアーキテクチャにも拡張できます。

ワークロード全体は、パーティションキーによってセルに分割されます。このキーは、サービスの粒度またはサービスのワークロードを最小限のクロスセルインタラクションで分割できる自然な方法に沿って分割を行う必要があります。パーティションキーの例には、カスタマー ID、リソース ID、またはほとんどの API コールで簡単にアクセスできるその他のパラメータなどがあります。セルルーティングレイヤーは、パーティションキーに基づいて個々のセルにリクエストを分散し、クライアントに単一のエンドポイントを提示します。

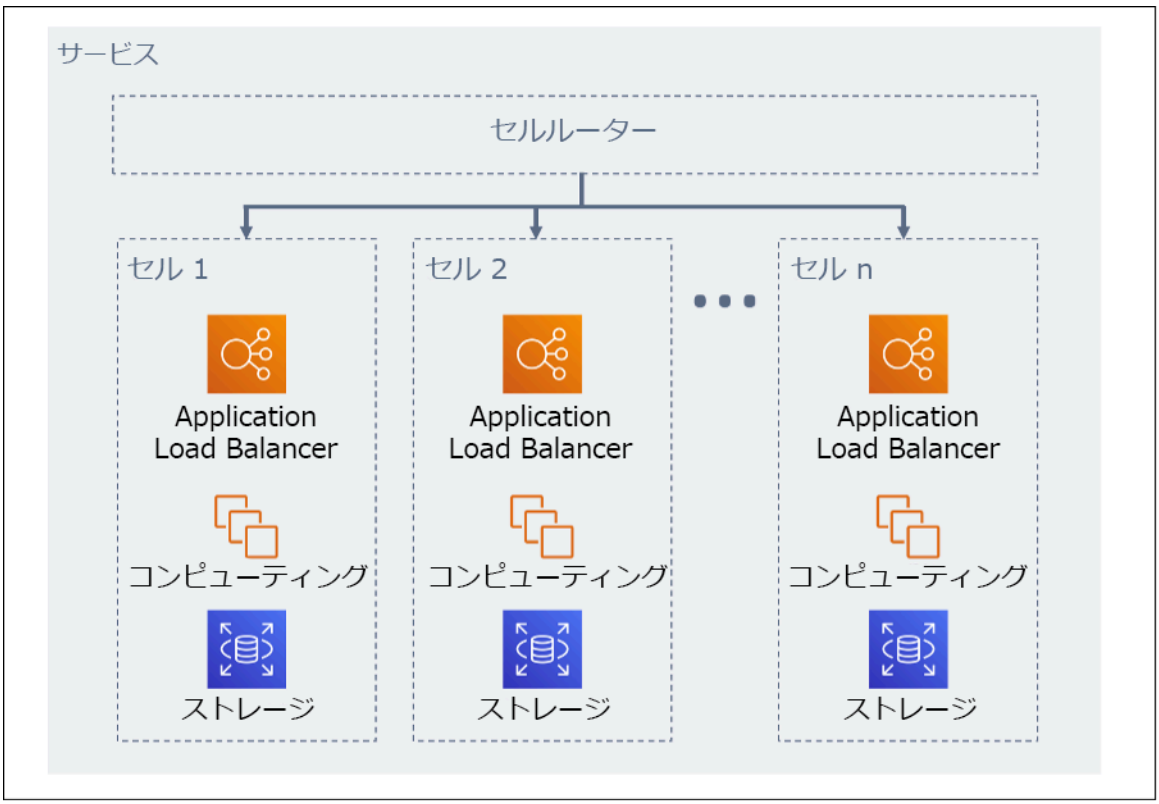


図 11: セルベースアーキテクチャ

実装手順

セルベースアーキテクチャを設計する場合、以下のとおり、考慮すべき設計上の考慮事項がいくつかあります。

1. パーティションキー: パーティションキーを選択する際は、特に配慮が必要です。
   - このキーは、サービスの粒度またはサービスのワークロードを最小限のクロスセルインタラクションで分割できる自然な方法に沿って分割を行う必要があります。例としては、 `##### ID` や `#### ID #####`.
   - パーティションキーは、あらゆるリクエストにおいて、直接またはその他のパラメータによって決定論的に容易に推測できる方法で使用できる必要があります。
2. 永続的なセルマッピング: アップストリームサービスは、リソースのライフサイクル全体にわたり、単一のセルとのみインタラクションを行う必要があります。
   - ワークロードによっては、あるセルから別のセルにデータを移行するためにセル移行戦略が必要となる場合があります。セルの移行が必要になる可能性のあるシナリオには、ワークロード内の特定のユーザーまたはリソースが過剰に増大して、専用のセルが必要になる、といった場合があります。

- セルは、セル間で状態またはコンポーネントを共有すべきではありません。
- つまり、セル間のインタラクションは回避するか、最小限に抑える必要があります。これは、このようなインタラクションにより、セル間の依存関係が形成され、障害分離による改善が妨げられるためです。

3. ルーターレイヤー: ルーターレイヤーはセル間の共有コンポーネントであるため、セルと同様の区分化戦略に従うことはできません。
   - ルーターレイヤーでは、パーティションマッピングアルゴリズムを計算効率の高い方法で使用して、リクエストを個別のセルに分散することをお勧めします。例えば、暗号化ハッシュ関数と合同算術を組み合わせてパーティションキーをセルにマップするなどの方法があります。
   - マルチセルへの影響を回避するには、ルーティングレイヤーを可能な限りシンプルかつ水平方向にスケーラブルなものとする必要があります。この場合、このレイヤー内での複雑なビジネスロジックは避ける必要があります。この方法には期待される動作をいつでも簡単に把握できるという利点があり、完全なテスト容易性が実現します。Colm MacCárthaigh が『Reliability, constant work, and a good cup of coffee (信頼性、動作の継続、一杯のコーヒー)』で説明しているとおり、シンプルな設計と一定の作業パターンは、信頼性の高いシステムを生み出し、アンチフラジャイルの低減につながります。
4. セルのサイズ: セルにはサイズ制限が必要であり、最大サイズを超えて拡大すべきではありません。
   - 最大サイズを特定するには、限界点に達して安全なオペレーションの限界が明らかになるまで徹底的にテストを実施する必要があります。テストプラクティスの実装方法の詳細については、以下を参照してください。 REL07-BP04 ワークロードの負荷テストを実施する
   - 全体的なワークロードの増大は、セルの追加によるべきです。これにより、需要の拡大に合わせてワークロードをスケーリングできるようになります。
5. マルチ AZ またはマルチリージョン戦略: さまざまな障害ドメインからの保護として、マルチレイヤーの回復力を活用する必要があります。
   - 回復力のためには、複数の防御層を構築するアプローチを使用する必要があります。1 つの層では、複数の AZ を使用して高可用性アーキテクチャを構築することによって、小規模で一般的な混乱に対して保護します。もう 1 つの防御層では、広範囲の自然災害やリージョンレベルの混乱など、まれな出来事に対して保護します。この 2 番目の層には、複数の AWS リージョン にまたがるアプリケーションの設計が必要です。ワークロードのためのマルチリージョン戦略の実装は、国の広い範囲に影響を与える自然災害や、リージョン全体に及ぶ技術的障害に対する保護に役立ちます。マルチリージョンアーキテクチャの実装はかなり複雑になることがあり、通常、ほとんどのワークロードには不要である点に注意してください。詳細については、以下を参照してください REL10-BP02 マルチロケーションデプロイ用の適切な場所の選択.

6. コードのデプロイ: コードの変更をすべてのセルに同時にデプロイせずに、タイミングをずらしたコードのデプロイ戦略を優先する必要があります。
   - これにより、不適切なデプロイや人的エラーによる複数のセルでの障害発生可能性を最小限に抑えることができます。詳細については、「安全なハンズオフデプロイメントの自動化」を参照してください。

このベストプラクティスが確立されていない場合のリスクレベル: 高

リソース

関連するベストプラクティス:

- REL07-BP04 ワークロードの負荷テストを実施する
- REL10-BP02 マルチロケーションデプロイ用の適切な場所の選択

関連するドキュメント:

- Reliability, constant work, and a good cup of coffee (信頼性、動作の継続、一杯のコーヒー)
- AWS and Compartmentalization (AWS と区分化)
- シャッフルシャーディングを使用したワークロードの分離
- 安全なハンズオフデプロイメントの自動化

関連動画:

- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small (AWS re:Invent 2018: ループをクローズして柔軟に対応: サイズを問わず、システムを制御する方法)
- AWS re:Invent 2018: How AWS Minimizes the Blast Radius of Failures (ARC338) (AWS re:Invent 2018: AWS が障害の影響範囲を最小限に抑える方法 (ARC338))
- Shuffle-sharding: AWS re:Invent 2019: Introducing The Amazon Builders' Library (DOP328) (AWS re:Invent 2020: Amazon Builders' Library の紹介 (DOP328))
- AWS Summit ANZ 2021 - Everything fails, all the time: Designing for resilience (Summit ANZ 2021 - すべてが常に失敗する: 回復力に向けた設計)

関連する例:

- Well-Architected Lab - Fault isolation with shuffle sharding (Well-Architected ラボ - シャッフルシャーディングを使用した障害分離)

コンポーネントの障害に耐えられるようにワークロードを設計する

高い可用性と低い平均復旧時間 (MTTR) の要件を持つワークロードは、回復力を考慮した設計をする必要があります。

ベストプラクティス

- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する
- REL11-BP02 正常なリソースにフェイルオーバーする
- REL11-BP03 すべてのレイヤーの修復を自動化する
- REL11-BP04 復旧中はコントロールプレーンではなくデータプレーンを利用する
- REL11-BP05 静的安定性を使用してバイモーダル動作を防止する
- REL11-BP06 イベントが可用性に影響する場合に通知を送信する
- REL11-BP07 可用性の目標と稼働時間のサービスレベルアグリーメント (SLA) を満たす製品を設計する

REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する

ワークロードの状態を継続的にモニタリングすることで、お客様と自動化システムが障害やパフォーマンスの低下の発生を速やかに把握できるようにします。ビジネス価値に基づいて主要業績評価指標 (KPI) をモニタリングします。

復旧および修復メカニズムはすべて、問題を迅速に検出する機能から開始する必要があります。技術的な障害を最初に検出して、解決できるようにするのです。ただし、可用性はワークロードがビジネス価値を提供する能力に基づいているため、これを測定する主要業績評価指標 (KPI) が検出および修正戦略の一部である必要があります。

期待される成果: ワークロードの重要なコンポーネントは個別にモニタリングされ、障害が発生したタイミングと場所で障害を検出してアラートを出します。

一般的なアンチパターン:

- アラームが設定されていないため、停止は通知なしで発生する。

- アラームは存在しますが、そのしきい値では対応するために十分な時間がない。
- メトリクスは、目標復旧時間 (RTO) を満たすのに十分な頻度で収集されない。
- ワークロードの顧客向けインターフェイスのみがアクティブにモニタリングされる。
- 技術的なメトリクスのみ収集し、ビジネス関数のメトリクスは収集しない。
- ワークロードのユーザーエクスペリエンスを測定するメトリクスがない。
- 作成されたモニターが多すぎる。

このベストプラクティスを活用するメリット: すべてのレイヤーで適切なモニタリングを行うことで、検出までの時間を短縮して、復旧時間を短縮できます。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

モニタリングの対象となるすべてのワークロードを特定します。モニタリングする必要があるワークロードのすべてのコンポーネントを特定したら、次はモニタリングの間隔を決定します。モニタリングの間隔は、障害の検出にかかる時間に基づいた復旧を開始する速さに直接影響します。平均検出時間 (MTTD) は、障害が発生してから修理作業が開始されるまでの時間です。サービスのリストは広範囲かつ完全でなければなりません。

モニタリングは、アプリケーション、プラットフォーム、インフラストラクチャ、ネットワークを含むアプリケーションスタックのすべてのレイヤーを対象とする必要があります。

モニタリング戦略では、 グレー障害の影響を考慮する必要があります。グレー障害の詳細については、Advanced Multi-AZ Resiliance Patterns whitepaper (高度なマルチ AZ 回復性パターンについてのホワイトペーパー) の グレー障害 を参照してください。

実装手順

- モニタリング間隔は、どの程度迅速に復旧する必要があるかによって異なります。復旧時間は復旧にかかる時間によって決まるため、この時間と目標復旧時間 (RTO) を考慮して、収集の頻度を決定する必要があります。
- コンポーネントとマネージドサービスの詳細なモニタリングを設定します。
  - EC2 インスタンスの詳細モニタリング と Auto Scaling が必要かどうかを判断します。詳細モニタリングは 1 分間隔メトリクスを提供し、デフォルトのモニタリングは 5 分間隔メトリクスを提供します。

  - EC2 拡張モニタリング が必要かどうかを判断します。拡張モニタリングでは、RDS インスタンスのエージェントを使用して、さまざまなプロセスまたはスレッドに関する有益な情報を取得します。
  - 以下における重要なサーバーレスコンポーネントのモニタリング要件を決定します。 Lambda、API Gateway、 Amazon EKS、 Amazon ECS、およびすべてのタイプの ロードバランサー。
  - 以下におけるストレージコンポーネントのモニタリング要件を決定します。 Amazon S3、Amazon FSx、 Amazon EFS、 Amazon EBS。
- ビジネス主要業績評価指標 (KPI) を測定する カスタムメトリクス を作成します。ワークロードには主要なビジネス機能が実装されており、いつ間接的な問題が発生したのかを特定するのに役立つ KPI として使用される必要があります。
- ユーザー Canary を使用して、障害に対するユーザーエクスペリエンスをモニタリングします。合成トランザクションテスト (「Canary テスト」とも呼ばれますが、Canary デプロイとは異なります) は、最も重要なテストプロセスの 1 つです。さまざまなリモートロケーションからワークロードエンドポイントに対してこれらのテストを常に実行します。
- ユーザーエクスペリエンスを追跡する カスタムメトリクスを 作成します。顧客のエクスペリエンスを測定できる場合は、コンシューマーエクスペリエンスが低下するタイミングを判断できます。
- アラームを設定 し、ワークロードの一部が正常に動作していないことを検出して、リソースを自動的にスケールするタイミングを示します。アラームをダッシュボードに視覚的に表示したり、Amazon SNS または E メールでアラートを送信したり、Auto Scaling と連携してワークロードリソースをスケールアップ/スケールダウンしたりできます。
- メトリクスを視覚化する ダッシュボード を作成します。ダッシュボードは、傾向や異常値などの潜在的な問題の指標を視覚的に確認したり、調査対象となり得る問題の存在を示すために使用できます。
- サービスに 分散型トレースモニタリング を作成します。分散型モニタリングにより、アプリケーションと基盤となるサービスがどのように動作しているかを理解して、パフォーマンスの問題やエラーの根本原因を特定し、トラブルシューティングすることができます。
- 別のリージョンとアカウントでモニタリングシステム (CloudWatch または X-Rayを使用) ダッシュボードとデータ収集を作成します。
- Amazon Health Aware モニタリング を統合することで、パフォーマンスの低下の可能性がある AWS リソースをモニタリングする可視性を得られます。ビジネスに不可欠なワークロードの場合、このソリューションによって、AWS サービスに対するプロアクティブでリアルタイムのアラートへのアクセスが可能になります。

リソース

関連するベストプラクティス:

- 可用性の定義
- REL11-BP06 イベントが可用性に影響する場合に通知を送信する

関連するドキュメント:

- Amazon CloudWatch Synthetics により模擬モニタリングを作成可能
- インスタンスの詳細モニタリングの有効化/無効化
- 拡張モニタリング
- Amazon CloudWatch を使用した Auto Scaling グループおよびインスタンスのモニタリング
- カスタムメトリクスの発行
- Amazon CloudWatch アラームの使用
- CloudWatch ダッシュボードの使用
- クロスリージョンのクロスアカウント CloudWatch ダッシュボードを使用する
- クロスリージョンのクロスアカウント X-Ray トレースを使用する
- 可用性を理解する
- Amazon Health Aware (AHA) を実装する

関連動画:

- グレー障害の軽減

関連する例:

- Well-Architected ラボ: レベル 300: ヘルスチェックを実装し依存関係を管理して、信頼性を向上する
- One Observability Workshop: X-Ray の探求

関連ツール:

- CloudWatch
- CloudWatch X-Ray

REL11-BP02 正常なリソースにフェイルオーバーする

リソース障害の発生時に、正常なリソースが引き続きリクエストに対応します。ロケーション障害 (アベイラビリティーゾーンや AWS リージョン など) に対しては、障害のないロケーションの正常なリソースにフェイルオーバーするシステムを用意します。

サービスを設計するときは、リソース、アベイラビリティーゾーン、またはリージョンに負荷を分散します。そのため、個々のリソースの障害は、残りの正常なリソースにトラフィックをシフトすることによって緩和できます。障害発生時にサービスがどのように検出され、ルーティングされるかを検討してください。

障害復旧を念頭に置いてサービスを設計します。AWS では、障害からの復旧時間とデータへの影響を最小限に抑えるサービスを設計しています。当社のサービスは主にデータストアを使用しており、リクエストが認識されるのは、リージョン内の複数のレプリカにわたりデータが永続的に保存された後です。これらのサービスは、セル単位の分離とアベイラビリティーゾーンにより提供される障害切り分けを活用するように構成されています。当社は、運用上の手順の多くで自動化を幅広く使用しています。また、中断から迅速に復旧するために、置換と再起動の機能を最適化しています。

フェイルオーバーを可能にするパターンとデザインは、AWS プラットフォームサービスごとに異なります。AWS ネイティブマネージドサービスの多くは、ネイティブに複数のアベイラビリティーゾーン (Lambda や API Gateway など) に対応しています。他の AWS サービス (EC2 や EKS など) では、AZ 間でのリソースまたはデータストレージのフェイルオーバーをサポートするための特定のベストプラクティス設計が必要です。

モニタリングは、フェイルオーバーリソースが正常であることを確認し、リソースのフェイルオーバーの進行状況を追跡して、ビジネスプロセスの復旧をモニタリングするために設定する必要があります。

期待される成果: システムは、新しいリソースを自動または手動で使用してパフォーマンスの低下から復旧できます。

一般的なアンチパターン:

- 障害を想定した計画が、計画と設計の段階に含まれていない。
- RTO と RPO が確立されていない。
- モニタリングが不十分で、障害が発生しているリソースを検出できない。
- 障害ドメインの適切な隔離。
- マルチリージョンのフェイルオーバーが考慮されていない。
- フェイルオーバーを決定する際の障害検出の感度が高すぎる、または過剰である。

- フェイルオーバー設計のテストや検証を行っていない。
- オートヒーリングのオートメーションを実行したが、ヒーリングが必要とされたことは通知しない。
- すぐにフェイルバックするのを防ぐための減衰期間を十分に設けていない。

このベストプラクティスを活用するメリット: 適切に機能低下し、迅速に回復することで、障害発生時でも信頼性を維持する耐障害性の高いシステムを構築できます。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

AWS のサービス (Elastic Load Balancing および Amazon EC2 Auto Scaling) は、複数のリソースおよびアベイラビリティーゾーンへの負荷分散に役立ちます。そのため、個々のリソース (EC2 インスタンスなど) の障害や、アベイラビリティーゾーンの障害を、残りの正常なリソースにトラフィックをシフトすることによって緩和できます。

マルチリージョンのワークロードの場合、設計はさらに複雑です。たとえば、クロスリージョンリードレプリカを使用すると、データを複数の AWS リージョン にデプロイできます。ただし、リードレプリカをプライマリに昇格させ、トラフィックを新しいエンドポイントに向けるには、やはりフェイルオーバーが必要です。Amazon Route 53、Route 53 Route 53 ARC、CloudFront、AWS Global Accelerator は複数の AWS リージョン へのトラフィックのルーティングに役立ちます。

Amazon S3、Lambda、API Gateway、Amazon SQS、Amazon SNS、Amazon SES、Amazon Pinpoint、Amazon ECR、AWS Certificate Manager、EventBridge、Amazon DynamoDB などの AWS サービスは、AWS によって複数のアベイラビリティーゾーンに自動的にデプロイされます。障害が発生した場合、これらの AWS サービスはトラフィックを正常なロケーションに自動的にルーティングします。データは複数のアベイラビリティーゾーンに冗長的に保存され、使用可能な状態を維持します。

Amazon RDS、Amazon Aurora、Amazon Redshift、Amazon EKS、または Amazon ECS の場合、マルチ AZ は設定オプションです。フェイルオーバーが開始されると、AWS はトラフィックを正常なインスタンスに転送できます。このフェイルオーバーアクションは、AWS が行うことも、必要に応じてお客様が実行することもできます。

Amazon EC2 インスタンス、Amazon Redshift、Amazon ECS タスク、または Amazon EKS ポッドの場合、デプロイ先のアベイラビリティーゾーンを選択します。一部の設計の場合、Elastic Load Balancing は異常なゾーンのインスタンスを検出し、正常なゾーンにトラフィックをルーティングす

るソリューションを提供します。Elastic Load Balancing は、オンプレミスのデータセンターのコンポーネントにトラフィックをルーティングすることもできます。

マルチリージョントラフィックフェイルオーバーの場合、再ルーティングでは Amazon Route 53、Route 53 ARC、AWS Global Accelerator、Route 53 Private DNS for VPCs、または CloudFront を活用して、インターネットドメインを定義し、ヘルスチェックなどのルーティングポリシーを割り当て、トラフィックを正常なリージョンにルーティングできます。AWS Global Accelerator は、アプリケーションへの固定エントリポイントとして機能する静的 IP アドレスを提供し、インターネットの代わりに AWS グローバルネットワークを使用して任意の AWS リージョン のエンドポイントにルーティングすることで、パフォーマンスと信頼性を向上させます。

実装手順

- すべての適切なアプリケーションとサービスのフェイルオーバー設計を作成します。各アーキテクチャコンポーネントを分離し、各コンポーネントの RTO と RPO を満たすフェイルオーバー設計を作成します。
- フェイルオーバープランに必要なすべてのサービスを使用して、下位環境 (開発環境やテスト環境など) を構成します。Infrastructure as Code (IaC) を使用してソリューションをデプロイし、再現性を確保します。
- フェイルオーバー設計を実装してテストするために、2 つ目のリージョンなどの復旧サイトを設定します。必要に応じて、テスト用のリソースを一時的に設定して、追加コストを抑えることができます。
- どのフェイルオーバープランを AWS で自動化するか、どのフェイルオーバープランを DevOps プロセスで自動化できるか、どのフェイルオーバープランを手動で行うかを判断します。各サービスの RTO と RPO を文書化して測定します。
- フェイルオーバープレイブックを作成し、各リソース、アプリケーション、サービスをフェイルオーバーするためのすべての手順を含めます。
- フェイルバックプレイブックを作成し、各リソース、アプリケーション、サービスをフェイルバックするためのすべての手順を (タイミングとともに) 含めます。
- プレイブックを開始してリハーサルするための計画を立てます。シミュレーションとカオステストを使用して、プレイブックの手順と自動化をテストします。
- ロケーション障害 (アベイラビリティーゾーンや AWS リージョン など) に対しては、障害のないロケーションの正常なリソースにフェイルオーバーするシステムを用意します。フェイルオーバーテストの前に、クォータ、自動スケーリングのレベル、実行中のリソースを確認してください。

リソース

関連する Well-Architected のベストプラクティス

- REL13- ディザスタリカバリの計画
- REL10 - 障害部分を切り離してワークロードを保護する

関連するドキュメント:

- RTO と RPO の目標の設定
- アプリケーションロードバランサーによる Route 53 ARC の設定
- Route 53 加重ルーティングを使用したフェイルオーバー
- DR と Route 53 ARC
- 自動スケーリング機能付き EC2
- EC2 デプロイ - マルチ AZ
- ECS デプロイ - マルチ AZ
- Route 53 ARC を使用したトラフィックの切り替え
- Application Load Balancer とフェイルオーバーによる Lambda
- ACM レプリケーションとフェイルオーバー
- パラメータストアのレプリケーションとフェイルオーバー
- ECR クロスリージョンレプリケーションとフェイルオーバー
- シークレットマネージャーのクロスリージョンレプリケーションの設定
- EFS とフェイルオーバーのクロスリージョンレプリケーションの有効化
- EFS クロスリージョンレプリケーションとフェイルオーバー
- ネットワークフェイルオーバー
- MRAP を使用した S3 エンドポイントフェイルオーバー
- S3 のクロスリージョンレプリケーションの作成
- Route 53 ARC によるリージョンの API Gateway のフェイルオーバー
- マルチリージョンのグローバルアクセラレーターによるフェイルオーバー
- DRS によるフェイルオーバー
- Amazon Route 53 を使用したディザスタリカバリメカニズムの作成

関連する例:

- AWS でのディザスタリカバリ
- AWS での Elastic Disaster Recovery

REL11-BP03 すべてのレイヤーの修復を自動化する

障害を検出したら、自動化機能を使用して修復するアクションを実行します。パフォーマンスの低下は、内部のサービスメカニズムによって自動的に修復される場合もあれば、修復アクションによってリソースを再起動または削除する必要がある場合もあります。

セルフマネージドアプリケーションやクロスリージョン修復では、復旧設計と自動修復プロセスを既存のベストプラクティスから引き出すことができます。

リソースを再起動または削除する機能は、障害を修復するための重要なツールです。ベストプラクティスは、可能な限りサービスをステートレスにすることです。これにより、リソースの再起動時のデータまたは可用性が失われるのを防ぎます。クラウドでは、再起動の一環として、リソース全体 (コンピューティングインスタンス、サーバーレス関数など) を置き換えることができます (通常はそうする必要があります)。再起動自体は、障害から復旧するための簡単で信頼できる方法です。ワークロードでは、さまざまなタイプの障害が発生します。障害は、ハードウェア、ソフトウェア、通信、オペレーションなどさまざまな部分で発生する可能性があります。

再起動または再試行は、ネットワークリクエストにも適用されます。依存関係にあるシステムからエラーが返された場合、ネットワークのタイムアウトの場合と依存関係にあるシステムの障害の両方に同じ復旧アプローチを適用します。どちらのイベントもシステムに類似の影響を与えるため、どちらかのイベントを特例とするのではなく、エクスポネンシャルバックオフとジッターで限定的に再試行するという類似の戦略を適用します。再起動の機能は、復旧指向コンピューティングと高可用性クラスターアーキテクチャを特徴とする復旧メカニズムです。

期待される成果: 障害の検出を修正するために、自動アクションが実行されます。

一般的なアンチパターン:

- 自動スケーリングなしでリソースをプロビジョニングする。
- インスタンスまたはコンテナにアプリケーションを個別にデプロイする。
- 自動復旧を使用せずに、複数のロケーションにデプロイできないアプリケーションをデプロイします。
- 自動スケーリングと自動復旧が修復に失敗するアプリケーションを手動で修復する。

- フェイルオーバーデータベースの自動化はない。
- トラフィックを新しいエンドポイントに再ルーティングする自動化された方法が足りない。
- ストレージのレプリケーションはない。

このベストプラクティスを活用するメリット: 自動修復により、平均回復時間を短縮し、可用性を向上させることができます。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

Amazon EKS やその他の Kubernetes サービスの設計には、レプリカセットまたはステートフルセットの最小値と最大値の両方、およびクラスターとノードグループの最小サイズが含まれている必要があります。これらのメカニズムは、Kubernetes コントロールプレーンを使用して障害を自動的に修正しながら、継続的に利用可能な処理リソースを最小限に抑えます。

コンピューティングクラスターを使用してロードバランサーを介してアクセスされる設計パターンでは、Auto Scaling グループを活用する必要があります。Elastic Load Balancing (ELB) は、受信アプリケーショントラフィックを 1 つ以上のアベイラビリティーゾーン (AZ) にある複数のターゲットと仮想アプライアンスに自動的に分散します。

ロードバランシングを使用しないクラスター化されたコンピューティングベースの設計では、少なくとも 1 台のノードが失われることを想定したサイズにする必要があります。これにより、新しいノードを復旧している間も、サービスが容量が減少する可能性がある状態で稼働し続けることができます。サービスの例としては、Mongo、DynamoDB Accelerator、Amazon Redshift、Amazon EMR、Cassandra、Kafka、MSK-EC2、Couchbase、ELK、Amazon OpenSearch Service などがあります。これらのサービスの多くは、追加の自動修復機能を使用して設計できます。一部のクラスターテクノロジーでは、ノードが失われたときにアラートを生成し、新しいノードを再作成するための自動または手動のワークフローをトリガーする必要があります。このワークフローは、AWS Systems Manager を使用して自動化でき、問題を迅速に修正できます。

Amazon EventBridge を使用して、CloudWatch アラームなどのイベントや他の AWS のサービスの状態の変化をモニタリングおよびフィルタリングできます。イベント情報に基づいて、AWS Lambda、Systems Manager オートメーションやその他のターゲットを呼び出して、ワークロード上でカスタム修復ロジックを実行できます。Amazon EC2 Auto Scaling は EC2 インスタンスの状態をチェックするように設定できます。インスタンスが実行中以外の状態にある場合、またはシステムステータスが損なわれている場合、Amazon EC2 Auto Scaling はインスタンスが異常であると見な

し、代替インスタンスを起動します。大規模な置き換え (アベイラビリティーゾーン全体の喪失など) の場合、静的安定性が高可用性のために優先されます。

実装手順

- Auto Scalingグループを使用してワークロードに階層をデプロイします。 Auto Scaling ステートレスなアプリケーションで自己修復を実行し、キャパシティーを追加および削除できます。
- 前述のコンピューティングインスタンスの場合は、 ロードバランシング 適切なロードバランサーのタイプを選択します。
- Amazon RDS の修復を検討してください。スタンバイインスタンスでは、スタンバイインスタンスへの 自動フェイルオーバー を設定します。Amazon RDS リードレプリカの場合、リードレプリカをプライマリにするには自動化されたワークフローが必要です。
- 複数のロケーションにデプロイできないアプリケーションがデプロイされている EC2 インスタンスでの自動復旧 を実装し、障害時の再起動を許容できます。自動復旧は、アプリケーションが複数のロケーションにデプロイできない場合に、障害が発生したハードウェアを交換してインスタンスを再起動するために使用できます。インスタンスのメタデータと関連する IP アドレスの他、 EBS ボリューム 、そして Amazon Elastic File System 、 Lustre 用ファイルシステム や Windowsから引き出すことができます。 AWS OpsWorksを使用することで、レイヤーレベルで EC2 インスタンスの自動修復を設定できます。
- 自動スケーリングまたは自動復旧を使用できない場合、または自動復旧が失敗した場合は、 AWS Step Functions および AWS Lambda を使用して自動復旧を実装します。自動スケーリングを使用できず、さらに、自動復旧が使用できないか、自動復旧が失敗した場合は、AWS Step Functions と AWS Lambda を使用して修復を自動化できます。
- Amazon EventBridge は、 CloudWatch アラーム や他の AWS サービスの状態の変化などのイベントを監視し、フィルタリングするために使用できます。イベント情報に基づいて、AWS Lambda (または他のターゲット) を呼び出し、ワークロードに対してカスタム修正ロジックを実行できます。

リソース

関連するベストプラクティス:

- 可用性の定義
- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する

関連するドキュメント:

- AWS Auto Scaling の仕組み
- Amazon EC2 自動復旧
- Amazon Elastic Block Store (Amazon EBS)
- Amazon Elastic File System (Amazon EFS)
- Amazon FSx for Lustre とは?
- Amazon FSx for Windows File Server とは?
- AWS OpsWorks: 自動ヒーリングを使用して、障害のあるインスタンスを置き換える
- AWS Step Functions とは?
- AWS Lambda とは?
- Amazon EventBridge とは?
- Amazon CloudWatch アラームの使用
- Amazon RDS フェイルオーバー
- SSM - Systems Manager オートメーション
- 回復力のあるアーキテクチャのベストプラクティス

関連動画:

- OpenSearch Service の自動プロビジョニングとスケーリング
- Amazon RDS の自動フェイルオーバー

関連する例:

- Auto Scaling に関するワークショップ
- Amazon RDS フェイルオーバーのワークショップ

関連ツール:

- CloudWatch
- CloudWatch X-Ray

REL11-BP04 復旧中はコントロールプレーンではなくデータプレーンを利用する

コントロールプレーンはリソースの作成、読み取り、記述、更新、削除、一覧表示 (CRUDL) に使用される管理 API を提供し、データプレーンは日々のサービストラフィックを処理します。回復力に影響を与える可能性のあるイベントへの回復または軽減対応を実装するときは、サービスの回復、再スケーリング、復元、修復、またはフェイルオーバーに最小限のコントロールプレーンのオペレーションを使用することに焦点を当ててください。これらのパフォーマンスの低下イベント中は、データプレーンのアクションがどのアクティビティよりも優先されるはずです。

たとえば、新しいコンピューティングインスタンスの起動、ブロックストレージの起動、キューサービスの記述などは、すべてコントロールプレーンのアクションです。コンピューティングインスタンスを起動後、コントロールプレーンは、容量のある物理ホストの検索、ネットワークインターフェースの割り当て、ローカルブロックストレージボリュームの準備、認証情報の生成、セキュリティルールの追加など、複数のタスクを実行する必要があります。コントロールプレーンは複雑なオーケストレーションになりがちです。

期待される成果: リソースに障害が発生すると、システムは、トラフィックを障害のあるリソースから正常なリソースに移行することで、自動または手動で回復できます。

一般的なアンチパターン:

- トラフィックを再ルーティングするための DNS レコードの変更への依存。
- リソースのプロビジョニングが不十分なため、障害が発生したコンポーネントの交換をコントロールプレーンのスケーリング操作に依存。
- あらゆるカテゴリの障害を修復するために、広範囲にわたるマルチサービス、マルチ API コントロールプレーンアクションに依存。

このベストプラクティスを活用するメリット: 自動修復の成功率が高くなると、平均復旧時間が短縮され、ワークロードの可用性が向上します。

このベストプラクティスを活用しない場合のリスクレベル: 中程度: 特定の種類のサービス低下では、コントロールプレーンが影響を受けます。コントロールプレーンを広範囲に使用して修復すると、復旧時間 (RTO) と平均復旧時間 (MTTR) が長くなる可能性があります。

実装のガイダンス

データプレーンのアクションを制限するには、サービスの復元に必要なアクションについて各サービスを評価します。

Amazon Route 53 Application Recovery Controller を活用して DNS トラフィックをシフトします。これらの機能により、障害から回復するアプリケーションの機能を継続的にモニタリングし、複数の AWS リージョン、アベイラビリティーゾーン、およびオンプレミスにまたがってアプリケーションの回復を管理できます。

Route 53 ルーティングポリシーではコントロールプレーンを使用するため、復旧のためにコントロールプレーンに依存しないでください。Route 53 データプレーンは、DNS クエリに応答し、ヘルスチェックを実行して、評価します。グローバルに分散され、 100% の可用性サービスレベルアグリーメント (SLA) として設計されています。.

Route 53 のリソースを作成、更新、削除する Route 53 管理 API およびコンソールは、コントロールプレーンで実行します。コントロールプレーンは、DNS の管理に必要な強力な一貫性と耐久性を重視するように設計されています。これを達成するために、コントロールプレーンは単一のリージョン、米国東部 (バージニア北部) に配置されています。どちらのシステムも非常に高い信頼性で構築されていますが、コントロールプレーンは SLA には含まれません。まれに、データプレーンの回復力設計によって可用性を維持できるときでも、コントロールプレーンでは維持でない場合があります。ディザスタリカバリおよびフェイルオーバーメカニズムについては、データプレーンの機能を使用して、可能な限り最善の信頼性を提供してください。

Amazon EC2 については、静的安定設計を使用してコントロールプレーンのアクションを制限してください。コントロールプレーンのアクションには、リソースを個別に、またはAuto Scaling グループ (ASG) を使用してスケールアップすることが含まれます。回復性を最大限に高めるには、フェイルオーバーに使用するクラスターに十分な容量をプロビジョニングしてください。この容量のしきい値を制限する必要がある場合は、エンドツーエンドのシステム全体にスロットルを設定して、限られたリソースに到達するトラフィックの合計を安全に制限してください。

Amazon DynamoDB、Amazon API Gateway、ロードバランサー、AWS Lambda サーバーレスなどのサービスでは、これらのサービスを使用するとデータプレーンを活用できます。ただし、新しい関数、ロードバランサー、API ゲートウェイ、またはDynamoDB テーブルの作成はコントロールプレーンのアクションであり、イベントの準備やフェイルオーバーアクションのリハーサルとして、パフォーマンス低下の前に完了しておく必要があります。Amazon RDS では、データプレーンのアクションはデータへのアクセスを可能にします。

データプレーン、コントロールプレーン、および AWS が高可用性目標を満たすためにサービスを構築する方法の詳細については、 アベイラビリティーゾーンを使用した静的安定性を参照してください。

データプレーンでの運用と、コントロールプレーンでの運用を理解します。

実装手順

パフォーマンス低下のイベント後に復元する必要のある各ワークロードについて、フェイルオーバーランブック、高可用性設計、自動修復設計、または HA リソース復元プランを評価します。コントロールプレーンのアクションと見なされる可能性のある各アクションを特定します。

コントロールアクションをデータプレーンアクションに変更することを検討します。

- Auto Scaling (コントロールプレーン) と事前にスケーリングされた Amazon EC2 リソース (データプレーン) の比較
- Lambda およびそのスケーリング方法 (データプレーン) または Amazon EC2 および ASG (コントロールプレーン) への移行
- Kubernetes を使用するあらゆる設計とコントロールプレーンのアクションの性質を評価します。ポッドの追加は Kubernetes のデータプレーンのアクションです。アクションはノードの追加ではなく、ポッドの追加に限定する必要があります。 オーバープロビジョニングされたノード を使用することが、コントロールプレーンのアクションを制限するための推奨方法です

データプレーンのアクションが同じ修復に影響を与えられる代替アプローチを検討してください。

- Route 53 のレコード変更 (コントロールプレーン) または Route 53 ARC (データプレーン)
- より自動化されたアップデートのための Route 53 ヘルスチェック

サービスがミッションクリティカルな場合は、影響を受けていないリージョンでより多くのコントロールプレーンとデータプレーンのアクションを実行できるように、セカンダリリージョンのサービスを検討してください。

- プライマリリージョンの Amazon EC2 Auto Scaling または Amazon EKS と、セカンダリリージョンの Amazon EC2 Auto Scaling または Amazon EKS を比較し、セカンダリリージョンにトラフィックをルーティングする (コントロールプレーンのアクション)
- セカンダリプライマリでリードレプリカを作成するか、プライマリリージョンで同じアクションを試みる (コントロールプレーンのアクション)

リソース

関連するベストプラクティス:

- 可用性の定義

- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する

関連するドキュメント:

- APN パートナー: 耐障害性のオートメーションを支援できるパートナー
- AWS Marketplace: 耐障害性に関して活用できる商品
- The Amazon Builders' Library: より小規模なサービスを管理して、分散システムの過負荷を回避する
- Amazon DynamoDB API (コントロールプレーンとデータプレーン)
- AWS Lambda の実行 (コントロールプレーンとデータプレーンに分割されています)
- AWS Elemental MediaStore データプレーン
- Amazon Route 53 Application Recovery Controller を使用した回復力の高いアプリケーションの構築、パート 1: シングルリージョンスタック
- Amazon Route 53 を使用した回復力の高いアプリケーションの構築、パート 1: シングルリージョンスタック
- Amazon Route 53 を使用したディザスタリカバリメカニズムの作成
- Route 53 Application Recovery Controller とは
- Kubernetes コントロールプレーンとデータプレーン

関連動画:

- 基本に戻る - 静的安定性の使用
- AWS グローバルサービスを使用して回復力のあるマルチサイトワークロードを構築

関連する例:

- Amazon Route 53 Application Recovery Controller の概要
- The Amazon Builders' Library: より小規模なサービスを管理して、分散システムの過負荷を回避する
- Amazon Route 53 Application Recovery Controller を使用した回復力の高いアプリケーションの構築、パート 1: シングルリージョンスタック
- Amazon Route 53 を使用した回復力の高いアプリケーションの構築、パート 1: シングルリージョンスタック

- アベイラビリティーゾーンを使用した静的安定性

関連ツール:

- Amazon CloudWatch
- AWS X-Ray

REL11-BP05 静的安定性を使用してバイモーダル動作を防止する

ワークロードは静的に安定し、1 つの通常モードでのみ動作する必要があります。バイモーダル動作とは、通常モードと障害モードでワークロードが異なる動作を示す場合をいいます。

例えば、別のアベイラビリティーゾーン (AZ) で新しいインスタンスを起動して、AZ の障害からの復旧を試みることができます。これにより、障害モード中にバイモーダル応答が発生する可能性があります。バイモーダル動作を防止するために、静的に安定し、1 つのモードでのみ動作するワークロードを構築する必要があります。この例では、これらのインスタンスは障害発生前に 2 番目の AZ でプロビジョニングしておく必要があります。この静的に安定した設計により、確実にワークロードが単一のモードでのみ動作するようになります。

期待される成果: ワークロードは、通常モードと障害モードでバイモーダル動作を示しません。

一般的なアンチパターン:

- 障害の範囲に関係なく、リソースが常にプロビジョニング可能であると仮定する。
- 障害発生時にリソースを動的に取得しようとする。
- 障害が発生するまで、ゾーンまたはリージョン間で適切なリソースをプロビジョニングしない。
- コンピューティングリソースにのみ静的に安定した設計を検討する。

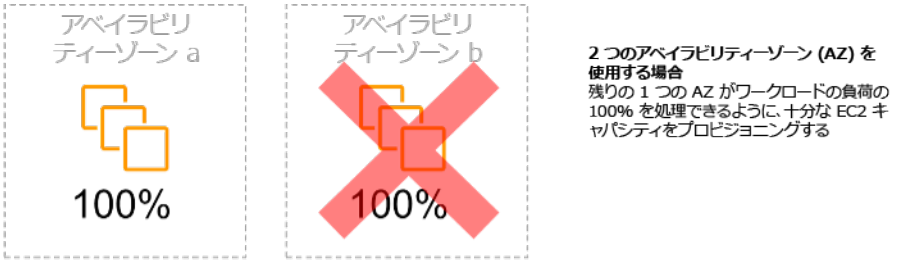
このベストプラクティスを活用するメリット: 静的に安定した設計で実行されるワークロードは、通常のイベント時でも障害発生時でも予測可能な結果を得ることができます。

このベストプラクティスを活用しない場合のリスクレベル: 中

実装のガイダンス

バイモーダル動作とは、例えばアベイラビリティーゾーン (AZ) に障害が発生した場合に新しいインスタンスの起動に依存するなど、通常モードと障害モードでワークロードが異なる動作を示す場合をいいます。バイモーダル動作の例は、安定した Amazon EC2 設計により、1 つの AZ が削除され

た場合にワークロードの負荷を処理するのに十分な数のインスタンスが各 AZ にプロビジョニングされる場合です。Elastic Load Balancing または Amazon Route 53 ヘルスチェックを使用して、障害のあるインスタンスから負荷を分散します。トラフィックがシフトした後、AWS Auto Scaling を使用して、障害が発生したゾーンのインスタンスを非同期で置き換え、正常なゾーンで起動します。EC2 インスタンスやコンテナなどのコンピューティングデプロイの静的安定性があると、信頼性が最も高くなります。



複数のアベイラビリティーゾーン (AZ) にわたる EC2 インスタンスの静的安定性

これは、このモデルのコストと、すべてのレジリエンスケースでワークロードを維持することのビジネス価値に照らして検討する必要があります。コンピューティングキャパシティを少なくプロビジョニングし、障害発生時に新しいインスタンスの起動に依存するほうがコストは低くなりますが、大規模な障害 (AZ やリージョンの障害など) の場合、このアプローチは、運用プレーンと、影響を受けていないゾーンまたはリージョンでリソースが十分に利用可能であることの両方に依存するため、あまり効果的ではありません。

また、ソリューションでは、信頼性とワークロードに必要なコストを比較検討する必要があります。静的に安定したアーキテクチャは、複数の AZ に分散されているコンピューティングインスタンス、データベースリードレプリカ設計、Kubernetes (Amazon EKS) クラスター設計、マルチリージョンフェイルオーバーアーキテクチャなど、さまざまなアーキテクチャに適用されます。

各ゾーンでより多くのリソースを使用することにより、より静的に安定した設計を実装することもできます。ゾーンを追加することで、静的安定性に必要な追加のコンピューティング量を減らすことができます。

バイモーダル動作のもう 1 つの例に、ネットワークのタイムアウトにより、システム全体の設定状態の再読み込みが始まることがあります。これにより想定外の負荷が別のコンポーネントに加わり、

そのコンポーネントで障害が発生し、想定外の結果につながる可能性があります。この負のフィードバックループは、ワークロードの可用性に影響を与えます。そこで、静的に安定し、1 つのモードでのみ動作するシステムを構築する必要があります。静的に安定した設計は、一定の作業を行い、常に一定の周期で設定状態を更新します。呼び出しに失敗すると、ワークロードは以前にキャッシュされた値を使用し、アラームを開始します。

バイモーダル動作のもう 1 つの例は、障害発生時にクライアントがワークロードキャッシュをバイパスできるようにすることです。これは、クライアントのニーズに対応するソリューションのように思われるかもしれませんが、ワークロードの需要を大幅に変更し、障害が発生する可能性が高くなります。

重要なワークロードを評価して、どのワークロードにこのタイプのレジリエンス設計が必要かを判断します。重要と思われるものについては、各アプリケーションコンポーネントを確認する必要があります。静的安定性評価を必要とするサービスの種類の例は次のとおりです。

- コンピューティング: Amazon EC2、EKS-EC2、ECS-EC2、EMR-EC2
- データベース: Amazon Redshift、Amazon RDS、Amazon Aurora
- ストレージ: Amazon S3 (単一ゾーン)、Amazon EFS (マウント)、Amazon FSx (マウント)
- ロードバランサー: 特定の設計で

実装手順

- 静的に安定し、1 つのモードでのみ動作するシステムを構築します。この場合、1 つのアベイラビリティーゾーンまたはリージョンが削除された場合にワークロード容量を処理するのに十分な数のインスタンスを、各アベイラビリティーゾーンまたはリージョンにプロビジョニングします。正常なリソースへのルーティングには、次のようなさまざまなサービスを使用できます。
  - クロスリージョン DNS ルーティング
  - MRAP Amazon S3 マルチリージョンのルーティング
  - AWS Global Accelerator
  - Amazon Route 53 Application Recovery Controller
- 単一のプライマリインスタンス またはリードレプリカが失われた場合に備えて、 データベースリードレプリカを設定します。トラフィックがリードレプリカによって処理されている場合、各アベイラビリティーゾーンと各リージョンでの量は、そのゾーンまたはリージョンに障害が発生した場合の全体的な必要量と同じにします。
- アベイラビリティーゾーンに障害が発生した場合に備えて保存されるデータに対し静的に安定するように設計された Amazon S3 ストレージに、重要なデータを設定します。Amazon S3 1 ゾーン –

IA ストレージクラスを使用する場合、そのゾーンが失われると、保存されたデータへのアクセスが最小限に抑えられるため、静的に安定していると見なすべきではありません。

- ロードバランサー は、誤って、または設計により、特定のアベイラビリティーゾーン (AZ) を処理するように設定されている場合があります。この場合、静的に安定した設計では、ワークロードをより複雑な設計の複数の AZ に分散することが考えられます。セキュリティ、レイテンシー、またはコスト上の理由から、元の設計を使用してゾーン間のトラフィックを削減できる場合があります。

リソース

関連する Well-Architected のベストプラクティス

- 可用性の定義
- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する
- REL11-BP04 復旧中はコントロールプレーンではなくデータプレーンを利用する

関連するドキュメント:

- 災害対策プランでの依存関係の最小化
- The Amazon Builders' Library: アベイラビリティーゾーンを使用した静的安定性
- 障害分離境界
- アベイラビリティーゾーンを使用した静的安定性
- マルチゾーン RDS
- 災害対策プランでの依存関係の最小化
- クロスリージョン DNS ルーティング
- MRAP Amazon S3 マルチリージョンのルーティング
- AWS Global Accelerator
- Route 53 ARC
- 単一ゾーン Amazon S3
- クロスゾーン負荷分散

関連動画:

- Static stability in AWS: AWS re:Invent 2019: Introducing The Amazon Builders' Library (DOP328)

関連する例:

- The Amazon Builders' Library: アベイラビリティーゾーンを使用した静的安定性

REL11-BP06 イベントが可用性に影響する場合に通知を送信する

しきい値の違反が検出されると、イベントによって引き起こされた問題が自動的に解決された場合でも、通知が送信されます。

自動ヒーリング機能により、ワークロードの信頼性を高めることができます。ただし、対処する必要のある根本的な問題もあいまいになる可能性があります。根本原因の問題を解決できるように、自動ヒーリングによって対処されたものを含む問題のパターンを検出できるように、適切なモニタリングとイベントを実装します。

回復力を備えたシステムは、低下イベントがすぐに適切なチームに通知されるよう設計されています。これらの通知は、1 つまたは複数の通信チャネルを通じて送信する必要があります。

期待される成果: エラー率、レイテンシー、その他の主要業績評価指標 (KPI) メトリクスなどのしきい値を超えると、直ちにオペレーションチームにアラートが送信されます。これにより、これらの問題をできるだけ早く解決し、ユーザーへの影響を回避するか最小限に抑えることができます。

一般的なアンチパターン:

- 送信するアラームが多すぎる。
- 実行できないアラームを送信する。
- アラームのしきい値の設定が高すぎる (感度が高すぎる) か、低すぎる (感度が低すぎる)。
- 外部依存関係に対するアラームを送信しない。
- モニタリングとアラームを設計する際に グレー障害 を考慮していない。
- ヒーリングのオートメーションを実行しているが、ヒーリングが必要となったことを適切なチームに通知しない。

このベストプラクティスを活用するメリット: 復旧の通知により、運用チームやビジネスチームはサービスの低下を認識し、直ちに対応して平均検出時間 (MTTD) と平均修復時間 (MTTR) の両方を最小限に抑えることができます。復旧イベントの通知により、発生頻度の低い問題を無視することもなくなります。

このベストプラクティスを活用しない場合のリスクレベル: 中程度適切なモニタリングとイベント通知のメカニズムを実装しないと、自動ヒーリングで対処されるものも含め、問題のパターンを検出で

きなくなる可能性があります。チームは、ユーザーがカスタマーサービスに連絡した場合、または偶然に見つけた場合にしか、システムの低下を認識できなくなります。

実装のガイダンス

モニタリング戦略の定義において、アラームのトリガーは一般的なイベントです。このイベントには、アラームの識別子、アラームの状態 (`IN ALARM` または `OK`など)、およびアラームをトリガーした原因の詳細が含まれる可能性があります。多くの場合、1 件のアラームイベントが検出されると、1 件の E メール通知が送信されます。これは、1 件のアラームにつき 1 つのアクションの例です。アラーム通知は、問題があることを適切な担当者に通知するため、オブザーバビリティにおいて非常に重要です。ただし、オブザーバビリティソリューションでイベントに対するアクションが洗練されると、人間の介入を必要とせずに問題を自動的に修正できます。

KPI モニタリングアラームを設定すると、しきい値を超えたときに適切なチームにアラートが送信されます。これらのアラートを使用して、低下の修復を試みる自動プロセスをトリガーすることもできます。

より複雑なしきい値のモニタリングには、複合アラームを検討する必要があります。複合アラームでは、多くの KPI モニタリングアラームを使用して、運用上のビジネスロジックに基づいてアラートを作成します。CloudWatch アラームは、Amazon SNS 統合または Amazon EventBridge を使用して、E メールを送信するか、サードパーティのインシデント追跡システムにインシデントを記録するように設定できます。

実装手順

モニタリング対象のワークロードに応じて、次のようなさまざまなタイプのアラームを作成します。

- アプリケーションアラームは、ワークロードの一部が正常に動作していないことを検出するために使用します。
- インフラストラクチャアラーム は、リソースをスケーリングするタイミングを示します。アラームをダッシュボードに視覚的に表示したり、Amazon SNS または E メールでアラートを送信したり、Auto Scaling と連携してワークロードリソースをスケールイン/スケールアウトしたりできます。
- シンプルな 静的アラーム を作成して、メトリクスが指定された評価期間の静的しきい値を超える状況をモニタリングできます。
- 複合アラーム は、複数のソースからの複雑なアラームに対応できます。
- アラームを作成したら、適切な通知イベントを作成します。Amazon SNS API を 直接呼び出して、 通知を送信し、修正やコミュニケーションのための自動化をリンクすることができます。

- <!--ATMS sidestep. Remove this--> モニタリング を統合することで、パフォーマンスの低下の可能性がある AWS リソースをモニタリングする可視性を得られます。ビジネスに不可欠なワークロードの場合、このソリューションによって、AWS サービスに対するプロアクティブでリアルタイムのアラートへのアクセスが可能になります。

リソース

関連する Well-Architected のベストプラクティス

- 可用性の定義

関連するドキュメント:

- 静的しきい値に基づいて CloudWatch アラームを作成する
- 「Amazon EventBridge とは?」
- Amazon Simple Notification Service とは
- カスタムメトリクスの発行
- Amazon CloudWatch でのアラームの使用
- Amazon Health Aware (AHA)
- Setup CloudWatch Composite alarms
- What's new in AWS Observability at re:Invent 2022

関連ツール:

- CloudWatch
- CloudWatch X-Ray

REL11-BP07 可用性の目標と稼働時間のサービスレベルアグリーメント (SLA) を満たす製品を設計する

可用性の目標と稼働時間のサービスレベルアグリーメント (SLA) を満たすように製品を設計します。可用性目標またはアップタイム SLA を公開するか、非公開で同意する場合は、アーキテクチャと運用プロセスが SLA をサポートするように設計されていることを確認します。

期待される成果: 各アプリケーションには、可用性の目標とパフォーマンスメトリクスの SLA が定義されています。これらのメトリクスは、ビジネス上の成果を満たすためにモニタリングされ、維持されることがあります。

一般的なアンチパターン:

- SLA を設定せずにワークロードを設計およびデプロイする。
- 合理的な理由やビジネス要件なしに SLA メトリクスが高く設定されている。
- 依存関係とその基盤となる SLA を考慮せずに SLA を設定する。
- 回復力の共有責任モデルを考慮せずにアプリケーションが設計される。

このベストプラクティスを活用するメリット: 主要な復元力の目標に基づいてアプリケーションを設計すると、ビジネス目標と顧客の期待を満たすことができます。このような目標は、さまざまなテクノロジーを評価し、さまざまなトレードオフを考慮に入れたアプリケーション設計プロセスの促進につながります。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

アプリケーションの設計では、ビジネス、オペレーション、財務上の目的に基づくさまざまな要件を考慮する必要があります。運用上の要件の範囲内で、ワークロードを適切にモニタリングおよびサポートできるように、ワークロードには特定の回復性メトリクス目標が必要です。ワークロードのデプロイ後は、回復性メトリクスの設定や派生の作成を行うべきではありません。これは設計段階で定義し、さまざまな決定とトレードオフのガイドとしての役割を果たす必要があります。

- 各ワークロードには、独自の回復性メトリクスセットが必要です。このようなメトリクスは、その他のビジネスアプリケーションとは異なる場合があります。
- 依存関係を低減すると、可用性にプラスの影響が生まれる可能性があります。各ワークロードは、独自の依存関係と SLA を考慮に入れる必要があります。通常、ワークロードの目標以上の可用性目標を持つ依存関係を選択します。
- 可能であれば、依存関係が損なわれてもワークロードが正常に動作できるように、疎結合の設計を検討します。
- コントロールプレーンの依存関係を減らします。特に、復旧時または機能低下時の依存関係を低減します。ミッションクリティカルなワークロードに対して静的安定性がある設計を評価します。リソースを節約して、ワークロード内のこのような依存関係の可用性を向上します。

- オブザーバビリティと計測は、平均検出時間 (MTTD) と平均修復時間 (MTTR) の短縮と SLA の達成のために重要です。
- 障害の頻度が低い (MTBF が長い)、障害検出時間が短い (MTTD が短い)、修復時間が短い (MTTR が短い) という 3 つの要素が、分散システムの可用性の向上のために使用されます。
- ワークロードの回復性メトリクスを確立し、それを満たすことは、効果的な設計の基本となります。このような設計では、設計の複雑性、サービスの依存関係、パフォーマンス、スケーリング、コストのトレードオフを考慮する必要があります。

実装手順

- 以下の質問を検討し、ワークロードの設計を確認して、文書化します。
  - コントロールプレーンはワークロードのどの個所で使用されますか。
  - ワークロードはどのように耐障害性を実装しますか。
  - どのようなスケーリング、自動スケーリング、冗長性、高可用性コンポーネントの設計パターンがありますか。
  - どのようなデータ整合性と可用性の要件がありますか。
  - リソース節約またはリソースの静的安定性に関する考慮事項はありますか。
  - どのようなサービスの依存関係がありますか。
- ステークホルダーと協力して、ワークロードアーキテクチャに基づいて SLA メトリクスを定義します。ワークロードで使用されるすべての依存関係の SLA を検討します。
- SLA 目標の設定後、SLA を満たすようにアーキテクチャを最適化します。
- SLA を満たす設計が定まったら、運用上の変更、プロセスの自動化、MTTD および MTTR の短縮についても重視するランブックを導入します。
- デプロイ後、SLA についてモニタリングしてレポートを作成します。

リソース

関連するベストプラクティス:

- REL03-BP01 ワークロードをセグメント化する方法を選択する
- REL10-BP01 複数の場所にワークロードをデプロイする
- REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する
- REL11-BP03 すべてのレイヤーの修復を自動化する
- REL12-BP05 カオスエンジニアリングを使用して回復力をテストする

- REL13-BP01 ダウンタイムやデータ消失に関する復旧目標を定義する
- ワークロードの状態の把握

関連するドキュメント:

- Availability with redundancy (冗長性を備えた可用性)
- 可用性 - 信頼性の柱
- Measuring availability (可用性の測定)
- AWS Fault Isolation Boundaries (AWS の障害分離境界)
- 回復性に関する責任共有モデル
- アベイラビリティーゾーンを使用した静的安定性
- AWSAWS サービスレベルアグリーメント (SLA)
- Guidance for Cell-based Architecture on AWS(AWS のセルベースアーキテクチャについてのガイダンス)
- AWS infrastructure (AWS インフラストラクチャ)
- Advanced Multi-AZ Resiliance Patterns whitepaper (高度なマルチ AZ 回復性パターンについてのホワイトペーパー)

関連サービス:

- Amazon CloudWatch
- AWS Config
- AWS Trusted Advisor

テストの信頼性

本番環境のストレスに耐えられるようにワークロードを設計した後、ワークロードが意図したとおりに動作し、期待する弾力性を実現することを確認する唯一の方法が、テストを行うことです。

バグまたはパフォーマンスのボトルネックがワークロードの信頼性に影響を与える可能性があるため、ワークロードが機能の要件と非機能要件を満たしていることを検証するためにテストします。ワークロードの弾#性をテストして、本番環境でしか表面化しない潜在的なバグを見つけられるようにします。このようなテストを定期的に行います。

ベストプラクティス

REL12-BP01 プレイブックを使用して障害を調査する

調査プロセスをプレイブックに文書化することで、よく理解されていない障害シナリオに対する一貫性のある迅速な対応が可能になります。プレイブックは、障害シナリオの原因となる要因を特定するために実行される事前定義されたステップです。プロセスステップの結果は、問題が特定されるか、エスカレーションされるまで、次のステップを決定するために使用されます。

プレイブックは、対応措置を効果的に実行できるようにするために立てる必要があるプロアクティブな計画です。本番環境でプレイブックに含まれていない障害シナリオが発生した場合は、まず問題に対処します (火を消します)。その後、振り返って問題に対処するために実行した手順を見て、これらの手順を用いてプレイブックに新しいエントリを追加します。

プレイブックは特定のインシデントに対応するために用いられる一方、ランブックは特定の結果を達成するために使用されます。多くの場合、ランブックは日常的なアクティビティに用いられる一方、プレイブックは非日常的なイベントに応えるために使用します。

一般的なアンチパターン:

- 問題の診断やインシデントへの対応を行うためのプロセスを知ることなくワークロードのデプロイを計画する。
- イベントを調査するときに、ログとメトリクスを収集するシステムに関する計画外の決定。
- データを取得するためにメトリクスとイベントを十分な期間保持していない。

このベストプラクティスを活用するメリット: プレイブックをキャプチャすることで、プロセスへの一貫した遵守が実現できます。プレイブックを成文化することによって、手動のアクティビティによるエラーの発生が抑制されます。プレイブックのオートメーションは、チームメンバーの介入の必要性をなくし、または介入の開始時に追加情報を提供することによって、イベントへの対応時間を短縮します。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

- プレイブックを使用して問題を特定します。プレイブックは、問題を調査するための文書化されたプロセスです。プロセスをプレイブックに文書化することで、障害シナリオに対する一貫性のある迅速な対応が可能になります。プレイブックには、十分なスキルを持った人物が該当する情報の収集、障害の潜在的原因の特定、障害の切り分け、寄与する要因の特定 (インシデント後の分析の実行) を行うために必要な情報とガイダンスが含まれている必要があります。
  - プレイブックをコードとして実装します。プレイブックをスクリプト化することにより、運用をコードとして実行し、一貫性を保ち、手動プロセスによって発生するエラーを抑制または低減します。プレイブックは、問題に寄与する要因を特定するために必要となり得るさまざまなステップを表す複数のスクリプトで構成できます。ランブックのアクティビティは、プレイブックのアクティビティの一部としてトリガーまたは実行するか、特定されたイベントへの応答としてプレイブックの実行を引き起こす場合があります。
    - AWSSystems Manager を使用して運用上のプレイブックを自動化する
    - AWS Systems Manager Run コマンド
    - AWS Systems Manager Automation をトリガーして
    - 「AWS Lambda とは何ですか?」
    - 「Amazon EventBridge とは?」
    - Amazon CloudWatch アラームの使用

リソース

関連するドキュメント:

- AWS Systems Manager Automation をトリガーして
- AWS Systems Manager Run コマンド
- AWSSystems Manager を使用して運用上のプレイブックを自動化する
- Amazon CloudWatch アラームの使用
- カナリアの使用 (Amazon CloudWatch Synthetics)
- 「Amazon EventBridge とは?」
- 「AWS Lambda とは何ですか?」

関連する例:

- プレイブックとランブックによるオペレーションの自動化

REL12-BP02 インシデント後の分析を実行する

顧客に影響を与えるイベントを確認し、寄与する要因と予防措置を特定します。この情報を使用して、再発を制限または回避するための緩和策を開発します。迅速で効果的な対応のための手順を開発します。対象者に合わせて調整された、寄与する要因と是正措置を必要に応じて伝えます。必要に応じて根本原因を他の人に伝える方法を確立します。

既存のテストで問題が見つからなかった理由を評価します。テストがまだ存在しない場合は、このケースのテストを追加します。

期待される成果: チームが合意済みの一貫したアプローチで、インシデント後の分析を取り扱います。そのメカニズムの 1 つがエラーの修正 (COE) プロセスです。COE プロセスは、チームがインシデントの根本原因を特定、理解、対処するのに役立つと同時に、同じインシデントの再発を防止するメカニズムとガードレールの構築にも有益です。

一般的なアンチパターン:

- 寄与因子を見つけるが、他の潜在的な問題やリスクの軽減策についてさらに詳しく調べない。
- 人的エラーの原因を特定するだけで、人的ミスを防止し得るトレーニングやオートメーションを実施しない。
- 原因究明よりも責任を追及するばかりで恐怖心を煽り、オープンなコミュニケーションが妨げられる。
- インサイトを共有できない。インシデント分析の結果を少人数のグループで抱え込み、他の人が教訓を活かせなくなります。
- 組織の知識をキャプチャするメカニズムがない。学んだ教訓を最新のベストプラクティスという形で保存しないと貴重なインサイトが失われ、同様または類似の根本原因でインシデントが再発することになります。

このベストプラクティスを活用するメリット: インシデント後の分析を実施し、結果を共有することで、他のワークロードが同じ寄与因子を実装した場合のリスクを軽減し、インシデントが発生する前に軽減策または自動復旧を実装できます。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

優れたインシデント後の分析は、システムの別の場所で使用されているアーキテクチャパターンの問題に対して、共通のソリューションを提案する機会になります。

COE プロセスの基礎は、問題を文書化して対処することです。重要な根本原因を文書化し、その原因をレビューして確実に解決するための標準化した手法を定義しておくことを推奨します。インシデント後の分析プロセスには明確に担当者を割り当てます。インシデントの調査とフォローアップの監督を担当するチームまたは個人を指名してください。

責任を追及するのではなく、学習と改善を重視する文化を醸成してください。個人の責任を問うのではなく、インシデントの再発防止が目標であることを強調します。

インシデント後の分析を実施するための明確な手順を策定します。これらの手順では、実行すべき手順、収集する情報、分析中に対処すべき重要な問題をまとめておく必要があります。インシデントを徹底的に調査し、直接的な原因だけでなく、根本原因と寄与因子を特定します。5 つの why 分析などの手法を用いて、根本的な問題を掘り下げてください。

インシデント分析から学んだ教訓のリポジトリを維持します。こうした組織の知識は、将来のインシデントや予防策の参考になります。インシデント後の分析から得られた結果とインサイトを共有し、誰でも参加できるインシデント後のレビューミーティングを開催して、学んだ教訓について話し合うことを検討してください。

実装手順

- インシデント後の分析を行う際は、非難の場にならないようにします。これにより、インシデントにかかわった人々は、提案された是正措置を冷静に検討し、誠実な自己評価とチーム間のコラボレーションに努めることができます。
- 重大な問題を文書化する標準化された方法を定義します。このようなドキュメントの構造の例は次のとおりです。
  - 何が起こったのでしょうか?
  - 顧客とビジネスにどのような影響がありましたか?
  - 根本原因は何でしたか?
  - これをサポートするデータはありますか?
    - 例えば、メトリクスとグラフ
  - 重要な柱、特にセキュリティとは何を意味するでしょうか?
    - ワークロードを設計するときには、ビジネスの状況に応じて、柱の間でトレードオフを行います。これらのビジネス上の決定は、エンジニアリング面での優先事項を推進させることがで

きます。開発環境では信頼性を妥協することでコストを削減するという最適化を#う場合や、ミッションクリティカルなソリューションでは、信頼性を最適化するためにコストをかける場合などがあります。セキュリティは常に最優先事項です。顧客を保護する必要があるからです。

  - どのような教訓を学びましたか?
  - どのような是正措置を講じましたか?
    - アクションアイテム
    - 関連項目
- インシデント後の分析を実施するための明確に定義された標準運用手順を作成します。
- 標準化されたインシデント報告プロセスを用意します。初回のインシデントレポート、ログ、コミュニケーション、インシデントの発生中に取られたアクションなど、すべてのインシデントを包括的に文書化します。
- インシデントが生じても必ずしもシステム停止を伴うわけではありません。ニアミスである場合や、システムがビジネス機能を果たしながら予想外の方法で動作している場合があります。
- フィードバックと教訓に基づいて、インシデント後の分析プロセスを継続的に改善します。
- 重要な検出結果をナレッジ管理システムでキャプチャし、開発者ガイドやデプロイ前のチェックリストに追加すべきパターンを検討します。

リソース

関連するドキュメント:

- エラーの修正 (COE) を開発すべき理由

関連動画:

- Amazon's approach to failing successfully
- AWS re:Invent 2021 - Amazon Builders' Library: Operational Excellence at Amazon

REL12-BP03 機能要件をテストする

必要な機能を検証する単体テストや統合テストなどの技法を使用します。

これらのテストがビルドおよびデプロイアクションの一部として自動的に実行されると、最良の結果が得られます。例えば、デベロッパーは AWS CodePipeline を使用して、CodePipeline が変更を自

動的に検出するソースリポジトリに変更をコミットします。このような変更が構築されたら、テストが実行されます。テストが完了すると、ビルドされたコードがテストのためステージングサーバーにデプロイされます。CodePipeline はステージングサーバーから統合テストや負荷テストなど、より多くのテストを実行します。これらのテストが正常に完了すると、CodePipeline はテストおよび承認されたコードを本番稼働インスタンスにデプロイします。

また、経験上、合成トランザクションテスト (カナリアテストとも呼ばれますが、カナリアデプロイと混同しないでください) は、顧客の行動を実行およびシミュレートでき、最も重要なテストプロセスの 1 つです。さまざまなリモートロケーションからワークロードエンドポイントに対してこれらのテストを常に実行します。Amazon CloudWatch Synthetics を使用すると、 Canary を作成して エンドポイントと API をモニタリングできます。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

- 機能要件をテストします。これには、必要な機能を検証する単体テストと統合テストが含まれます。
  - AWS CodeBuild で CodePipeline を使用してコードをテストし、ビルドを実行する
  - AWS CodePipeline が、AWS CodeBuild を使用した単体テストとカスタム統合テストのサポートを追加
  - 継続的デリバリーと継続的インテグレーション
  - カナリアの使用 (Amazon CloudWatch Synthetics)
  - ソフトウェアテストのオートメーション

リソース

関連するドキュメント:

- APN パートナー: 継続的インテグレーションパイプラインの実装を支援できるパートナー
- AWS CodePipeline が、AWS CodeBuild を使用した単体テストとカスタム統合テストのサポートを追加
- AWS Marketplace: 継続的インテグレーションに利用できる製品
- 継続的デリバリーと継続的インテグレーション
- ソフトウェアテストのオートメーション
- AWS CodeBuild で CodePipeline を使用してコードをテストし、ビルドを実行する

- カナリアの使用 (Amazon CloudWatch Synthetics)

REL12-BP04 スケーリングおよびパフォーマンス要件をテストする

負荷テストなどの技法を使用して、ワークロードがスケーリングおよびパフォーマンス要件を満たしていることを検証します。

クラウドでは、ワークロードに合わせて、本番稼働働規模のテスト環境を作成できます。スケールダウンしたインフラストラクチャでこれらのテストを実行する場合、観測された結果を、本番環境で予想される事態にスケーリングする必要があります。実際のユーザーに影響を与えないように注意する場合は、本番環境でも負荷テストとパフォーマンステストを行います。その際、実際のユーザーデータと混合したり、使用統計や本番レポートを破損しないないようにテストデータにタグを付けます。

テストでは、ベースリソース、スケーリング設定、サービスクォータ、および弾力性設計が負荷がかかる時に想定どおりに動作することを確認します。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

- スケーリングおよびパフォーマンス要件をテストします。ワークロードがスケーリングおよびパフォーマンスの要件を満たしていることを検証するための負荷テストを実行します。
  - AWS での分散負荷テスト: 接続された数千のユーザーをシミュレートする
  - Apache JMeter
    - 本番環境と同じ環境にアプリケーションをデプロイして、負荷テストを実施します。
      - コードとしてのインフラストラクチャの概念を使用して、できるだけ本番環境と類似した環境を作成する

リソース

関連するドキュメント:

- AWS での分散負荷テスト: 接続された数千のユーザーをシミュレートする
- Apache JMeter

REL12-BP05 カオスエンジニアリングを使用して回復力をテストする

不利な条件下でシステムがどのように反応するかを理解するために、本番環境またはできるだけそれに近い環境で定期的にカオス実験を行います。

期待される成果:

イベント発生時のワークロードの既知の期待動作を検証する回復力テストに加え、フォールトインジェクション実験や想定外の負荷の注入という形でカオスエンジニアリングを適用し、ワークロードの回復力を定期的に検証します。カオスエンジニアリングと回復力テストの両方を組み合わせることで、ワークロードがコンポーネントの故障に耐え、予期せぬ中断から影響を最小限に抑えて回復できることへの信頼を得ることができます。

一般的なアンチパターン:

- 耐障害性を考慮した設計でありながら、障害発生時にワークロードが全体としてどのように機能するかを検証していない。
- 実際の条件および予期された負荷の下で実験を一切行わない。
- 実験をコードとして処理しないか、開発サイクルを通して維持しない。
- CI/CD パイプラインの一部、またはデプロイの外部のどちらとしても、カオス実験を実行しない。
- どの障害で実験するかを考慮する際に、過去のインシデント後の分析を軽視する。

このベストプラクティスを活用するメリット: ワークロードの回復力を検証するために障害を発生させることにより、耐障害性設計の回復手順が実際の障害発生時にも機能するという信頼性を得られます。

このベストプラクティスを活用しない場合のリスクレベル: ミディアム

実装のガイダンス

カオスエンジニアリングは、サービスプロバイダ、インフラストラクチャ、ワークロード、コンポーネントレベルにおいて、現実世界の障害 (シミュレーション) を継続的に発生させる機能を提供し、顧客には最小限の影響しか与えません。これにより、チームは障害から学び、ワークロードの回復力を観察、測定、改善することができます。また、イベント発生時にアラートが発せられ、チームに通知されることを確認することもできます。

カオスエンジニアリングを継続的に実施することで、ワークロードの欠陥が浮き彫りになり、そのままにしておくと可用性や オペレーションに悪影響が及ぶ可能性があります。

> **Note**
>
> カオスエンジニアリングとは、あるシステムで実験を行い、実稼働時の混乱状態に耐えることができるかどうかの確信を得るための手法です。 カオスエンジニアリングの原則

もし、システムがこれらの混乱に耐えられるなら、カオス実験は自動化された回帰テストとして維持されるはずです。このように、カオス実験はシステム開発ライフサイクル (SDLC) の一部および CI/CD の一部として実行される必要があります。

ワークロードがコンポーネントの障害に耐えられることを確認するために、実際のイベントを実験の一部として挿入します。たとえば、Amazon EC2 インスタンスの喪失やプライマリ Amazon RDS データベースインスタンスのフェイルオーバーを実験し、ワークロードに影響がないこと (または最小限の影響にとどまること) を確認します。コンポーネントの障害の組み合わせを使用して、アベイラビリティーゾーンで中断によって発生する可能性のあるイベントをシミュレートします。

アプリケーションレベルの障害 (クラッシュなど) では、メモリや CPU の枯渇などのストレス要因から始めます。

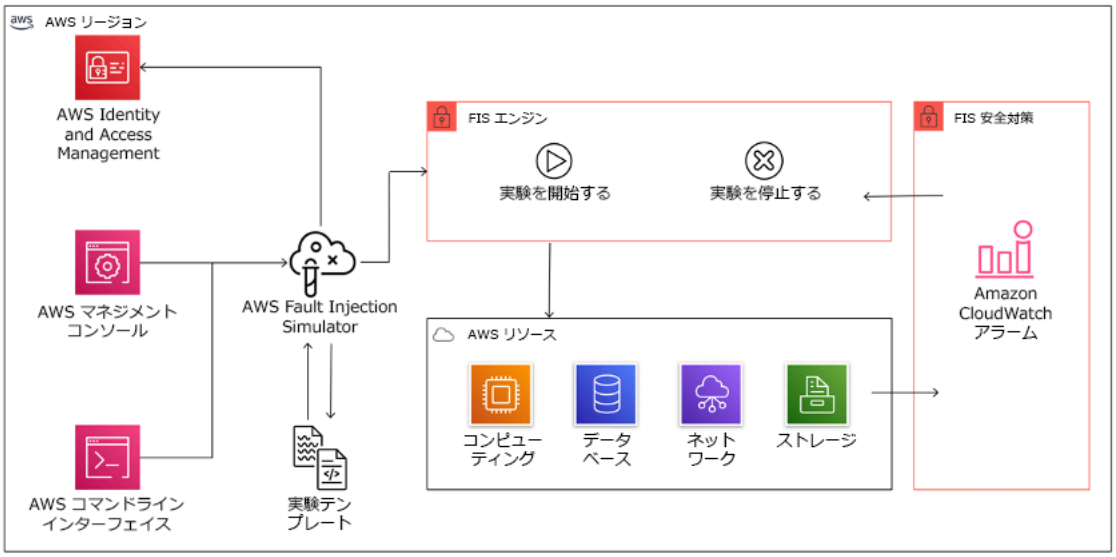
断続的なネットワークの中断による外部依存のための フォールバックまたはフェイルオーバーメカニズム を検証するために、コンポーネントは、数秒から数時間の指定された期間、サードパーティプロバイダへのアクセスをブロックすることにより、そのようなイベントをシミュレートする必要があります。

その他の劣化モードでは、機能の低下や応答の遅れが発生し、サービスの中断につながることがよくあります。このパフォーマンス低下の一般的な原因は、主要サービスのレイテンシー増加と、信頼性の低いネットワーク通信 (パケットのドロップ) です。レイテンシー、メッセージのドロップ、DNS 障害などのネットワーク効果を含むこれらの障害の実験には、名前を解決できない、DNS サービスヘリーチできない、または依存サービスへの接続を確立できないなどの可能性があります。

カオスエンジニアリングのツール:

AWS Fault Injection Service (AWS FIS) は、フォールトインジェクション実験を実行する完全マネージド型サービスであり、CD パイプラインの一部として、またはパイプラインの外で使用することができます。AWS FIS は、カオスエンジニアリングゲームデー中に使用するのに適しています。Amazon EC2、Amazon Elastic Container Service (Amazon ECS)、Amazon Elastic Kubernetes Service (Amazon EKS)、および Amazon RDS などを含む、異なるタイプのリソースに同時に障害を導入することをサポートします。これらの障害には、リソースの終了、強制フェイルオーバー、CPU にストレスをかける、スロットリング、またはメモリー、レイテンシー、およびパケッ

トの損失が含まれます。Amazon CloudWatch アラームと連携しているため、ガードレールとして停止条件を設定し、予期せぬ影響を与えた場合に実験をロールバックすることができます。



ワークロードのフォールトインジェクション実験を実行するため、AWS リソースと統合された AWS Fault Injection Service。

フォールトインジェクション実験のためのサードパーティオプションもいくつかあります。これには、次のようなオープンソースのツールが含まれます。 Chaos Toolkit、Chaos Mesh、および Litmus Chaos、Gremlin などの商用オプションです。AWS に挿入できる障害のスコープを拡張するために、AWS FIS は Chaos Mesh および Litmus Chaos と統合して、複数のツール間でフォールトインジェクションワークフローの調整を可能にします。たとえば、AWS FIS 障害アクションを使用して、ランダムに選択した割合のクラスターノードを終了する間に、Chaos Mesh または Litmus 障害を使用してポッドの CPU のストレステストを実行することができます。

実装手順

- どの障害を実験に使用するかを決定する。

  回復力に関してワークロードの設計を評価します。そのような設計 (Well-Architected フレームワークのベストプラクティスを使用して作成) では、重要な依存関係、過去のイベント、既知の問題、およびコンプライアンス要件に基づくリスクが考慮されています。回復力を維持するために意図された設計の各要素と、それを軽減するために設計された障害をリストアップします。そのようなリストの作成に関する詳細については、 運用準備状況の確認に関するホワイトペーパー を確認し、過去のインシデントの再発を防ぐためのプロセスを作成する方法を学びます。障害モードと影響の分析 (FMEA) プロセスにより、障害とそれがワークロードに与える影響をコンポーネントレベ

ルで解析するためのフレームワークが提供されます。FMEA については Adrian Cockcroft 氏による「 Failure Modes and Continuous Resilience」内に詳しく記載されています。

- それぞれの障害に優先度を割り当てる。

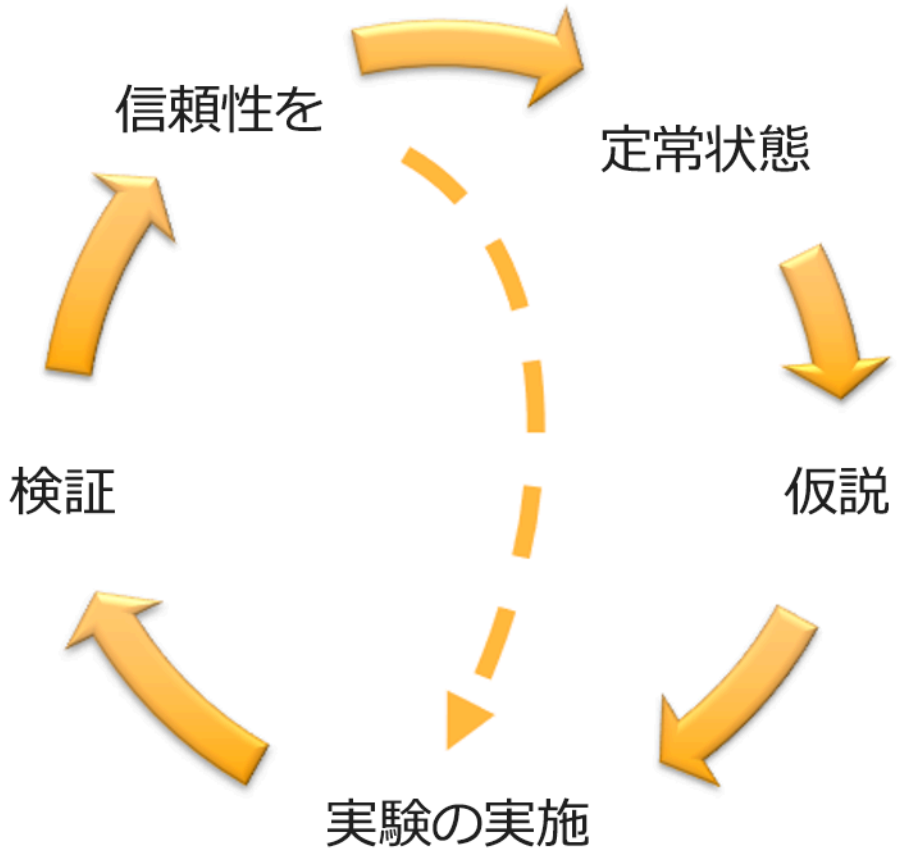
  「高」「中」「低」などの大まかな分類から始めます。優先度を評価するためには、障害の発生頻度と障害によるワークロード全体への影響度を考慮します。

  ある障害の発生頻度を考慮する場合、利用可能であれば、そのワークロードの過去のデータを分析します。利用できない場合は、類似の環境において実行されている他のワークロードのデータを使用します。

  ある障害の影響を考慮する場合、一般に、障害の範囲が大きければ大きいほど、影響も大きくなります。また、ワークロードの設計と目的も考慮します。たとえば、ソースデータストアにアクセスする機能は、データ変換や分析を行うワークロードにとって重要です。この場合、アクセス障害、スロットルアクセス、レイテンシー挿入の実験を優先させることになります。

  障害発生後の分析は、障害モードの頻度と影響の両方を理解するための良いデータソースとなります。

  割り当てた優先度を使用して、どの障害を最初に実験するか、および新しいフォールトインジェクション実験を開発する順序を決定します。
- 実行するそれぞれの実験に対して、カオスエンジニアリングと継続的な回復力のフライホイールに従います。



Adrian Hornsby 氏による、科学的手法を用いたカオスエンジニアリングと継続的な回復力のフライホイール。

- 定常状態とは、正常な動作を示すワークロードの測定可能な出力であると定義する。

  ワークロードは、信頼性が高く、期待通りに動作していれば、定常状態を示しています。したがって、定常状態を定義する前に、ワークロードが正常であることを検証します。障害が発生した場合、一定の割合で許容範囲内である可能性があるため、定常状態は、必ずしもワークロードに影響を与えないことを意味するものではありません。定常状態は、実験中に観察する基準値であり、次のステップで定義した仮説が予想通りにならない場合に、異常が浮き彫りになります。

  たとえば、決済システムの定常状態は、成功率 99%、往復時間 500ms で 300TPS を処理することと定義することができます。

- ワークロードがどのように障害に対応するかについての仮説を立てる。

良い仮説とは、定常状態を維持するために、ワークロードがどのように障害を軽減すると予想されるかに基づいています。仮説は、特定のタイプの障害が発生した場合、システムまたはワークロードが定常状態を維持することを示しています。なぜなら、ワークロードは特定の緩和策を講じて設計されているからです。特定の種類の障害および緩和策は、仮説の中で特定さ れる必要があります。

次のテンプレートが仮説に使用できます (ただし、他の表現も許容されます)。

> **Note**
>
> もし ##### が発生した場合、 ####### ワークロードは ####### して ########## ##########。

例:

- Amazon EKS ノードグループの 20% のノードが停止しても、[Transaction Create API] は 100 ミリ秒未満で 99% のリクエストに対応し続けます (定常状態)。Amazon EKS ノードは 5 分以内に回復し、ポッドは実験開始後 8 分以内にスケジューリングされてトラフィックを処理するようになります。3 分以内にアラートが発せられます。
- Amazon EC2 インスタンスの単一障害が発生した場合、注文システムの Elastic Load Balancing ヘルスチェックにより、Amazon EC2 Auto Scaling が障害インスタンスを置き換える間、Elastic Load Balancing は残りの健全なインスタンスにのみリクエストを送信し、サーバーサイド (5xx) エラーの増加を 0.01% 未満に維持します (定常状態)。
- プライマリ Amazon RDS データベースインスタンスに障害が発生した場合、サプライチェーンデータ収集ワークロードはフェイルオーバーし、スタンバイ Amazon RDS データベースインスタンスに接続して、データベースの読み取りまたは書き込みエラーを 1 分未満に維持します (定常状態)。

- 障害を挿入して実験を実行する。

  実験はデフォルトでフェイルセーフであり、ワークロードが耐えることができる必要があります。ワークロードが失敗することが分かっている場合は、実験を実行しないでください。カオスエンジニアリングは、既知の未知、または未知なる未知を見つけるために使用されるべきです。既知の未知 とは、認識はしているが完全に理解していないことであり、 未知なる未知 は、認識も完全な理解もしていないことです。壊れていると分かっているワークロードに対して実験を行っても、新しいインサイトを得ることはできません。実験は慎重に計画し、影響の範囲を明確にし、予期せぬ乱れが発生した場合に適用できるロールバックメカニズムを提供する必要があ

ります。デューディリジェンスにより、ワークロードが実験に耐えられることが分かったら、実験を進めてください。障害を挿入するには、いくつかの方法があります。AWS 上のワークロードに対して、AWS FIS は多くの事前定義された障害シミュレーションを挿入します。これは、アクションと呼ばれます。また、AWS FIS で実行するカスタムアクションも定義します。これには AWS Systems Manager ドキュメントを使用します。

カオス実験にカスタムスクリプトを使用することは、スクリプトがワークロードの現在の状態を理解する機能を持ち、ログを出力でき、可能であればロールバックと停止条件のメカニズムを提供しない限り、推奨されません。

カオスエンジニアリングをサポートする効果的なフレームワークやツールセットは、実験の現在の状態を追跡し、ログを出力し、実験の制御された実行をサポートするためのロールバックメカニズムを提供する必要があります。AWS FIS のように、実験範囲を明確に定義し、実験によって予期せぬ乱れが生じた場合に実験をロールバックする安全なメカニズムを備えた実験を行うことができる、確立されたサービスから始めてみてください。AWS FIS を使用した、より幅広い実験に関する詳細については、「カオスエンジニアリングラボでの回復力と Well-Architected アプリ」も参照してください。また、AWS Resilience Hub はワークロードを分析し、AWS FIS で実装、実行することを選択できるような実験を作成します。

> **Note**
>
> すべての実験について、その範囲と影響を明確に理解します。本番環境で実行する前に、まず非本番環境で障害をシミュレートすることをお勧めします。

実験は、可能な限り、対照系と実験系の両方をスピンアップする canary デプロイ を使用して、実世界の負荷で実稼働させる必要があります。オフピークの時間帯に実験を行うことは、本番で初めて実験を行う際に潜在的な影響を軽減するための良い習慣です。また、実際の顧客トラフィックを使用するとリスクが高すぎる場合は、本番インフラストラクチャの制御環境と実験環境に対して合成トラフィックを使用して実験を実行することができます。本番環境での実験が不可能な場合は、できるだけ本番環境に近いプレ本番環境で実験を行ってください。

実験が本番トラフィックや他のシステムに許容範囲を超えて影響を与えないように、ガードレールを確立してモニタリングする必要があります。停止条件を設定し、定義したガードレールのメトリクスでしきい値に達した場合に実験を停止するようにします。これには、ワークロードの定常状態のメトリクスと、障害を注入するコンポーネントに対するメトリク スを含める必要があります。ユーザー canary とも呼ばれる 合成モニタリング は、通常、ユーザープロキシとして含む必要がある 1 つのメトリクスです。AWS FIS への停止条件 は、実験テンプレートの一部

としてサポートされており、1 テンプレートあたり最大 5 つの停止条件を設定することが可能です。

カオスの原則の 1 つは、実験の範囲とその影響を最小化することです。

短期的な悪影響は許容されなければなりませんが、実験の影響を最小化し、抑制することはカオスエンジニアの責任であり義務です。

範囲や潜在的な影響を検証する方法としては、本番環境で直接実験を行うのではなく、まず非本番環境で実験を行い、実験中に停止条件のしきい値が想定通りに作動するか、例外をキャッチするための観測性があるかどうかを確認することが挙げられます。

フォールトインジェクション実験を実施する場合、すべての責任者に情報が十分に提供されるようにします。オペレーションチーム、サービス信頼性チーム、カスタマーサポートなどの適切なチームとコミュニケーションをとり、実験がいつ実行され、何を期待されるかを伝えます。これらのチームには、何か悪影響が見られた場合に、実験を行っているチームに知らせるためのコミュニケーションツールを与えます。

ワークロードとその基盤となるシステムを元の既知の良好な状態に復元する必要があります。多くの場合、ワークロードの回復力のある設計が自己回復を行います。しかし、一部の障害設計や実験の失敗により、ワークロードが予期せぬ障害状態に陥ることがあります。実験が終了するまでに、このことを認識し、ワークロードとシステムを復旧させなければなりません。AWS FIS では、アクションのパラメータ内にロールバック設定 (ポストアクションとも呼ばれる) を設定することができます。ポストアクションは、アクションが実行される前にある状態にターゲットを返します。自動化 (AWS FIS の使用など) であれ手動であれ、これらのポストアクションは、障害を検出し処理する方法を説明するプレイブックの一部であるべきです。

- 仮説を検証する。

  カオスエンジニアリングの原則 は、ワークロードの定常状態を検証する方法について、このようなガイダンスを示しています。

  システムの内部属性ではなく、測定可能な出力に焦点を当てます。その出力を短期間で測定することによって、システムの安定状態のプロキシが構成されます。システム全体のスループット、エラーレート、レイテンシーのパーセンタイルはすべて、定常状態の動作を表す重要なメトリクスになり得ます。実験中にシステム的な動作パターンに注目することで、カオスエンジニアリングは、システムがどのように動作するかを検証するのではなく、システムが動作していることを検証します。

先の 2 つの例では、サーバーサイド (5xx) エラーの増加率が 0.01% 未満、データベースの読み取りと書き込みのエラーが 1 分未満という定常状態の測定基準が含まれています。

5xx エラーは、ワークロードのクライアントが直接経験する障害モードの結果であるため、良いメトリクスです。データベースエラーの測定は、障害の直接的な結果として適切ですが、顧客からのリクエストの失敗や、顧客に表面化したエラーなど、顧客への影響も測定して補足する必要があります。さらに、ワークロードのクライアントが直接アクセスする API や URI に、合成モニタリング (ユーザー canary としても知られる) を含めるようにしましょう。

- 回復力を高めるためのワークロード設計を改善する。

  定常状態が維持されなかった場合、障害を軽減するためにワークロード設計をどのように改善できるかを調査し、 AWS Well-Architected 信頼性の柱のベストプラクティスを適用します。その他のガイダンスとリソースは AWS Builder's Libraryにあり、ここでは、他の記事と共に ヘルスチェックを見直す 方法、または アプリケーションコード内のバックオフを使用した再試行の採用に関する記事が掲載されています。

  これらの変更を実施した後、再度実験を行い (カオスエンジニアリングフライホイールの点線で表示)、その効果を判断します。検証の結果、仮説が正しいことがわかれば、ワークロードは定常状態になり、このサイクルが続きます。

- 実験を定期的に実施する。

  カオス実験はサイクルであり、実験はカオスエンジニアリングの一環として定期的に実施される必要があります。ワークロードが実験の仮説を満たした後、CI/CD パイプラインの回帰部分として継続的に実行されるように実験を自動化する必要があります。この方法については、このブログの「 how to run AWS FIS experiments using AWS CodePipeline」を参照してください。このラボでは、 CI/CD パイプラインで AWS FIS 実験 を繰り返し行うことで、実践的に作業することができます。

  フォールトインジェクション実験は、ゲームデーの一部でもあります (参照: REL12-BP06 定期的にゲームデーを実施する)。ゲームデーでは、障害やイベントをシミュレートし、システム、プロセス、チームの対応を検証します。その目的は、例外的な出来事が発生した場合にチームが実行することになっているアクションを実際に実行することです。

- 実験結果をキャプチャし、保存する。

  フォールトインジェクション実験の結果は、キャプチャおよび保持される必要があります。実験結果や傾向を後で分析できるように、必要なデータ (時間、ワークロード、条件など) をすべて含めておきましょう。結果の例には、ダッシュボードのスクリーンショット、メトリクスのデータ

ベースからの CSV ダンプ、実験中のイベントや観察結果を手書きで記録したものなどがあります。AWS FIS での実験記録 もデータキャプチャの一部となり得ます。

リソース

関連するベストプラクティス:

- REL08-BP03 デプロイの一部として回復力テストを統合する
- REL13-BP03 ディザスタリカバリの実装をテストし、実装を検証する

関連するドキュメント:

- AWS Fault Injection Service とは
- AWS Resilience Hub とは
- カオスエンジニアリングの原則
- カオスエンジニアリング: 最初の実験を計画する
- 回復力エンジニアリング: 失敗から学ぶ
- カオスエンジニアリングのストーリー
- 分散システムでのフォールバックの回避
- カオス実験の canary デプロイ

関連動画:

- AWS re:Invent 2020: Testing resiliency using chaos engineering (ARC316)
- AWS re:Invent 2019: Improving resiliency with chaos engineering (DOP309-R1)
- AWS re:Invent 2019: Performing chaos engineering in a serverless world (CMY301)

関連する例:

- Well-Architected ラボ: レベル 300: Amazon EC2 Amazon RDS と Amazon S3 の回復力をテストする
- AWS ラボでのカオスエンジニアリング
- カオスエンジニアリングラボでの回復力と Well-Architected アプリ

- サーバーレスカオスラボ
- アプリケーションの回復力を AWS Resilience Hub ラボを使用して測定し、向上させる

関連ツール:

- AWS Fault Injection Service
- AWS Marketplace: Gremlin Chaos Engineering Platform
- Chaos Toolkit
- Chaos Mesh
- Litmus

REL12-BP06 定期的にゲームデーを実施する

ゲームデーを使用して、実際の障害シナリオに関わる人々と、可能な限り本番環境に近い環境 (本番環境を含む) でのイベントと障害の対処手順を定期的に実行します。ゲームデーでは、本番環境のイベントがユーザーに影響を与えないようにするための対策を講じます。

ゲームデーでは、障害やイベントをシミュレーションして、システム、プロセス、チームの対応をテストします。その目的は、例外的な出来事が発生した場合にチームが実行することになっているアクションを実際に実行することです。これは、改善できる箇所を把握し、組織がイベントに対応することを経験するのに役#ちます。こうしたゲームデーを定期的に実施することで、チームは対応方法に関する 「基礎体力」 をつけることができます。

弾力性を考慮した設計が整い、本番環境以外の環境でテストした後、本番環境ですべてが計画どおりに機能することを確認するのがゲームデーです。ゲームデー、特に初日は、「全員が総力を挙げた」取り組みです。いつ起こるか、そして何が起こるかについてエンジニアと運用担当者に通知します。ランブックを用意します。障害イベントも含めて、シミュレートされたイベントが本番稼働システムで所定の方法で実行され、影響が評価されます。すべてのシステムが設計どおりに動作すると、検出と自己修復が行われ、影響はほとんどありません。ただし、負の影響が観察された場合、テストはロールバックされ、ワークロードの問題が必要に応じて (ランブックを参照して) 手動で修正されます。ゲームデーは本番環境で行われることが多いため、顧客の可用性に影響を与えないように、あらゆる予防策を講じる必要があります。

一般的なアンチパターン:

- 手順は文書化するが、実行しない。

- テスト演習にビジネス上の意思決定者を含めない。

このベストプラクティスを活用するメリット: ゲームデーを定期的に実施することで、実際のインシデントが発生したときにすべてのスタッフがポリシーと手順に従っていることを確認し、それらのポリシーと手順が適切であることを検証できます。

このベストプラクティスを活用しない場合のリスクレベル: ミディアム

実装のガイダンス

- ゲームデーをスケジュールして定期的にランブックおよびプレイブックを使ってみるゲームデーには、事業主、開発スタッフ、運用スタッフ、インシデント対応チームといった、本番環境でのイベントに関与すると思われるすべての人員が参加する必要があります。
  - 負荷テストやパフォーマンステストを実施した後、障害注入を実施します。
  - ランブックのおかしな点やプレイブックを使う機会を探します。
    - ランブックから逸脱したら、対応マニュアルを改善するか行動を修正します。プレイブックを使用したら、使用すべきだったランブックを特定するか新しいランブックを作成します。

リソース

関連するドキュメント:

- AWS GameDay とは?

関連動画:

- AWS re:Invent 2019: Improving resiliency with chaos engineering (DOP309-R1)

  関連する例:

- AWS Well-Architected ラボ - Testing Resiliency

災害対策 (DR) を計画する

バックアップと冗長ワークロードコンポーネントを配置することは、DR 戦略の出発点です。RTO と RPO は ワークロードの復元目標です。これは、ビジネスニーズに基づいて設定します。ワークロードのリソースとデータのロケーションと機能を考慮して、目標を達成するための戦略を実装しま

す。ワークロードの災害対策を提供することのビジネス価値を伝えるには、中断の可能性と復旧コストも重要な要素となります。

可用性とディザスタリカバリは、どちらも、障害のモニタリング、複数のロケーションへのデプロイ、自動フェイルオーバーなど、同じベストプラクティスに依存しています。ただし、可用性がワークロードのコンポーネントに焦点を合わせているのに対して、ディザスタリカバリはワークロード全体の個別のコピーに焦点を合わせています。ディザスタリカバリの目標はアベイラビリティとは異なり、災害後の復旧時間が焦点です。

ベストプラクティス

- REL13-BP01 ダウンタイムやデータ消失に関する復旧目標を定義する
- REL13-BP02 復旧目標を満たすため、定義された復旧戦略を使用する
- REL13-BP03 ディザスタリカバリの実装をテストし、実装を検証する
- REL13-BP04 DR サイトまたはリージョンでの設定ドリフトを管理する
- REL13-BP05 復旧を自動化する

REL13-BP01 ダウンタイムやデータ消失に関する復旧目標を定義する

ワークロードには、目標復旧時間 (RTO) と目標復旧時点 (RPO) が定義されます。

目標復旧時間 (RTO) RTO は、サービスの中断からサービスの復元までの最大許容遅延です。これにより、サービスが利用できないときに許容可能と見なされる時間枠が決まります。

目標復旧時点 (RPO) RPO は、最後のデータ復旧ポイントからの最大許容時間です。これにより、最後の復旧ポイントからサービスの中断までの間に許容可能と見なされるデータ損失が決まります。

RTO 値と RPO 値は、ワークロードに適したディザスタリカバリ (DR) 戦略を選択する際の重要な考慮事項です。これらの目標は企業によって決定され、技術チームによって DR 戦略の選択と実装のために使用されます。

期待される成果:

すべてのワークロードに、ビジネスへの影響に基づいて定義された RTO と RPO が割り当てられます。ワークロードが事前に定義された改装に割り当てられ、該当する RTO および RPO とともに、サービスの可用性と許容可能なデータ損失を定義します。そのような階層化ができない場合には、後で階層を作成する目的で、ワークロードごとに別注を割り当てることもできます。RTO と RPO は、ワークロードのディザスタリカバリ戦略実装を選択する際の主要な考慮事項の 1 つとして使用

されます。DR 戦略を選択する際のその他の考慮事項としては、コストの制約、ワークロードの依存関係、運用要件があります。

RTO については、停止時間に基づく影響を理解してください。線形か、それとも非線形の意味合いがあるか (例えば、4 時間後に、次のシフトの開始まで製造ラインをシャットダウンしておく) 。

次のようなディザスタリカバリマトリックスは、ワークロードが復旧目標にどの程度関係しているかを理解するのに役立ちます。(X 軸と Y 軸の実際の値は、組織のニーズに合わせてカスタマイズしてください)。

ディザスタリカバリマトリックス						
		目標復旧時点				
		1 分未満	1 時間未満	6 時間未満	1 日未満	1 日以上
目標復旧時間	10 分未満	重要	重要	高	中	中
	2 時間未満	重要	高	中	中	低
	8 時間未満	高	中	中	低	低
	24 時間未満	中	中	低	低	低
	24 時間以上	中	低	低	低	低

図16: ディザスタリカバリマトリックス

一般的なアンチパターン:

- 復旧目標を定義していない。
- 任意の復旧目標を選択する。
- 過度に寛大で、ビジネス目標を満たさない復旧目標を選択する。
- ダウンタイムとデータ損失の影響を理解していない。
- 復旧時間ゼロやデータ損失ゼロなど、ワークロード設定では達成できない恐れのある非現実的な復旧目標を選択する。
- 実際のビジネス目標よりも厳格な復旧目標を選択する。これにより、ワークロードが必要とするよりもコストが高く、複雑な DR 実装を強いられます。
- 依存するワークロードの復旧目標とは互換性のない復旧目標を選択する。
- 復旧目標で規制コンプライアンス要件が考慮されていない。
- ワークロードの RTO と RPO は定義されたが、テストされていない。

このベストプラクティスを活用するメリット: 時間とデータ損失の復旧目標は、DR 実装の指針とするために必要です。

このベストプラクティスを活用しない場合のリスクレベル: 高

実装のガイダンス

特定のワークロードについて、ダウンタイムとデータ損失がビジネスに与える影響を理解する必要があります。一般に、ダウンタイムが長いほど、またはデータ損失が大きいほど、影響は増加しますが、この増加の形状はワークロードのタイプによって異なります。例えば、1 時間までのダウンタイムなら耐えられ、影響もほとんどないかもしれませんが、その後は影響が急増するかもしれません。ビジネスへの影響は、金銭的コスト (減益など)、顧客の信頼 (と評判への影響)、運用上の問題 (給与未払いや生産性の低下など)、規制リスクなど、多くの形態をとります。以下のステップを使用して、これらの影響を理解し、ワークロードの RTO と RPO を設定してください。

実装手順

1. このワークロードのビジネスステークホルダーを決め、これらのステップを実装するように促します。ワークロードの復旧目標は、ビジネス上の決定です。技術チームはビジネスステークホルダーと協力して、これらの目標に基づいて DR 戦略を選択します。

> **Note**
>
> ステップ 2 と 3 については、以下を使用してください。 the section called “実装ワークシート”.

2. 以下の質問に答えることによって、決定を下すために必要な情報を集めます。
3. ワークロードが組織に与える影響について、重要度のカテゴリまたは階層がありますか?
   a. ある場合、このワークロードをカテゴリに割り当てます。
   b. ない場合は、これらのカテゴリを確立します。5 つ以下のカテゴリを作成し、それぞれの目標復旧時間の範囲を絞り込みます。カテゴリの例としては、重要、高、中、低などがあります。ワークロードがどのようにカテゴリにマッピングされるかを理解するには、ワークロードがミッションクリティカルであるか、ビジネスにとって重要であるか、それともビジネスを駆動するものではないかを考慮します。
   c. カテゴリに基づいて、ワークロードの RTO と RPO を設定します。このステップに入るときに計算した元の値より厳しいカテゴリ (低い RTO および RPO) を選ぶようにします。この結果、値の変化が不適切に大きくなる場合には、新しいカテゴリの作成を検討します。

4. これらの回答に基づいて、RTO 値と RPO 値をワークロードに割り当てます。これは直接行うか、ワークロードを事前定義のサービス階層に割り当てることで行うことができます。
5. このワークロードのディザスタリカバリプラン (DRP) を文書化し、組織の ビジネス継続性計画 (BCP)の一部とし、ワークロードチームとステークホルダーがアクセスできる場所に保管します。
   a. RTO および RPO と、これらの値を決めるために使用した情報を記録します。ワークロードがビジネスに与える影響を評価するために使用した戦略も含めます。
   b. RTO と RPO のほかに、ディザスタリカバリ目標のために追跡しているか、追跡する予定のその他のメトリクスも記録します。
   c. DR 戦略とランブックを作成したときには、これらの詳細をこのプランに追加します。
6. 図 15 のようなマトリックスでワークロードの重要性を調べることで、組織で定義される事前定義のサービス階層の確立を開始できます。
7. に従って DR 戦略 (または DR 戦略の概念実証) を実装した後the section called “REL13-BP02 復旧目標を満たすため、定義された復旧戦略を使用する”、この戦略をテストして、ワークロードの実際の RTC (復旧時間機能) と RPC (復旧時点機能) を判断します。これらがターゲットの復旧目標を満たさない場合は、ビジネスステークホルダーと協力して目標を調整するか、DR 戦略に変更を加えて、ターゲット目標を満たします。

主な質問

1. ワークロードがダウンしてからビジネスに重大な影響が出るまでの最大時間はどのくらいですか。
   a. ワークロードが中断した場合にビジネスに及ぼす 1 分間あたりの金銭的コスト (直接的な経済的影響) を判断します。
   b. 影響が常に線形とは限らないことを考慮します。影響は最初は限定的でも、臨界時点を超えると急増することがあります。
2. ビジネスに重大な影響が出るデータ損失の最大量はどのくらいですか。
   a. 最も重要なデータストアについて、この値を考慮します。その他のデータストアのそれぞれの重要度を特定します。
   b. ワークロードデータが失われた場合、再作成できますか? これがバックアップと復元よりも運用上容易な場合は、ワークロードデータの再作成に使用されるソースデータの重要度に基づいて RPO を選びます。
3. このワークロードに依存するワークロード (ダウンストリーム) またはこのワークロードが依存するワークロード (アップストリーム) の復旧目標と可用性期待は何ですか?

a. このワークロードがアップストリームの依存関係の要件を満たすことができる復旧目標を選びます。

b. ダウンストリームの依存関係の復旧機能を前提として達成可能な復旧目標を選びます。重要でないダウンストリームの依存関係 (「対処」できるもの) は除外できます。または、必要な場合は、ダウンストリームの重要な依存関係と協力して、復旧能力を高めます。

その他の質問

以下の質問と、これらがこのワークロードにどのように適用されるか考慮してください。

4. 停止のタイプ (リージョン対AZ など) に応じた異なる RTO および RPO がありますか?
5. RTO/RPO が変更される特定の時期 (季節性、販売イベント、製品の発売) がありますか? その場合、異なる測定と時間境界は何ですか?
6. ワークロードが中断した場合、何人の顧客が影響を受けますか?
7. ワークロードが中断した場合、評判への影響は何ですか?
8. ワークロードが中断した場合に発生する可能性のある、その他の運用上の影響は何ですか? 例えば、E メールシステムが使用できない場合や、給与システムがトランザクションを送信できない場合の従業員の生産性への影響などです。
9. ワークロードの RTO および RPO は基幹業務および組織の DR 戦略とどのように連携しますか?
10. サービスの提供に関する内部契約上の義務がありますか? 満たすことができなかった場合の罰則はありますか?
11. データに関する規制またはコンプライアンス制約は何ですか?

実装ワークシート

このワークシートは、実装ステップ 2 および 3 に使用できます。質問を追加するなど、特定のニーズに応じてこのワークシートを調整することができます。

ステップ 2: 主な質問	ワークロードに適用されますか？	ワークロード RTO	ワークロード RPO	RTO 調整	RPO 調整	手順
[1] ワークロードの最大ダウン時間						サービス停止の発生から復旧までの時間で測定
[2] 失われるデータの最大量						復元可能な最後の既知のデータセットからの時間で測定
[3a] アップストリームの依存関係						最も厳しいアップストリームリカバリ目標を入力します
[3b] ダウンストリームの依存関係						最も緩いダウンストリームリカバリ目標を入力します
[3a] 調整されたアップストリームの依存関係						アップストリームの値が現在の値よりも小さく、ダウンストリームの値が大きい場合は、依存関係を調整し、調整した値をここに入力します
[3b] 調整されたダウンストリームの依存関係						
[3] 依存関係						アップストリームの依存関係を満たすために値を下げるか、ダウンストリームの依存関係の機能に基づいて値を上げます
ステップ 2: その他の質問						質問に該当する場合は、ご記入ください。該当しない場合は、飛ばしてください
ベースの RTO/RPO						上記の RTO と RPO の値をここに入れます
[4] サービス停止の種類	[ ]はい / [ ]いいえ					イベントタイプについて要件が最も厳しい復旧目標を入力します
[5] 特定の時間ベースの目標	[ ]はい / [ ]いいえ					時間について要件が最も厳しい復旧目標を入力します
[6] 顧客の混乱	[ ]はい / [ ]いいえ					ダウンタイムまたはデータ損失の関数として影響を受ける顧客をグラフ化します。これをもとに、顧客への影響を考慮して許容される最大 RTO と RPO を入力します
[7] 評判への影響	[ ]はい / [ ]いいえ					ビジネスと連携し、評判への影響を考慮した最大の RTO と RPO を決定します
[8] 運用上の影響	[ ]はい / [ ]いいえ					運用上の影響に基づいて、最大 RTO と RPO を入力します
[9] 組織の調整	[ ]はい / [ ]いいえ					LOB および組織の要件に従って、このタイプのワークロードの最大 RTO と RPO を入力します
[10] 契約上の義務	[ ]はい / [ ]いいえ					契約上の義務に基づいて、最大 RTO と RPO を入力します
[11] 規制コンプライアンス	[ ]はい / [ ]いいえ					適用される規制コンプライアンスに基づいて、最大 RTO と RPO を入力します
その他の質問に基づくターゲット						Q の 4 ～ 11 から最小値 (より厳しい値) をここに入力します
調整済みターゲット						上記の目標に対応できない場合は、ステークホルダーと協力して制約を緩め、ここに新しい最小値を入力します
調整済み RTO/RPO						ベースの RPO/RTO 値、または調整済みターゲットのいずれか低い方を入力します
ステップ 3						
事前定義されたカテゴリまたは層にマッピング						両方の値を下方 (より厳密) に調整して、最も近い定義された層に合わせます

ワークシート

実装計画の工数レベル: 低

リソース

関連するベストプラクティス:

- the section called “REL09-BP04 データの定期的な復旧を行ってバックアップの完全性とプロセスを確認する”
- the section called “REL13-BP02 復旧目標を満たすため、定義された復旧戦略を使用する”
- the section called “REL13-BP03 ディザスタリカバリの実装をテストし、実装を検証する”

関連するドキュメント:

- AWS アーキテクチャブログ: Disaster Recovery Series
- AWS でのワークロードのディザスタリカバリ: クラウドでの復旧 (AWS ホワイトペーパー)
- AWS レジリエンスハブによる回復力ポリシーの管理

- APN パートナー: 災害対策を支援できるパートナー
- AWS Marketplace: 災害対策に活用できる商品

関連動画:

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)
- Disaster Recovery of Workloads on AWS

REL13-BP02 復旧目標を満たすため、定義された復旧戦略を使用する

ワークロードの復旧目標を満たすディザスタリカバリ (DR) 戦略を定義します。バックアップと復元、スタンバイ (アクティブ/パッシブ)、またはアクティブ/アクティブなどの戦略を選択します。

期待される成果: 各ワークロードについて、定義され、実装された DR 戦略があり、ワークロードは DR 目標を達成できます。ワークロード間の DR 戦略では、再利用可能なパターンを利用します (以前に記載された戦略など)。

一般的なアンチパターン:

- 同じような DR 目標を持つワークロードについて、一貫性のない復旧手順を実装する。
- DR 戦略は、災害が発生したときにアドホックに実装すればよいとする。
- ディザスタリカバリが計画されていない。
- 復旧時にコントロールプレーンのオペレーションに依存する。

このベストプラクティスを活用するメリット:

- 定義された復旧戦略を使用すると、一般的なツールとテスト手順を使用できます。
- 定義された復旧戦略を使用すると、チーム間のナレッジ共有と、チームが所有するワークロードでの DR の実装が改善します。

このベストプラクティスが確立されていない場合のリスクレベル: 高計画され、実装され、テストされた DR 戦略がなければ、災害発生時に復旧目標を達成できない可能性があります。

実装のガイダンス

DR 戦略は、プライマリロケーションでワークロードを実行できなくなった場合に復旧サイトでワークロードに耐えられる能力に依存します。最も一般的な復旧目標は、RTO と RPO です REL13-BP01 ダウンタイムやデータ消失に関する復旧目標を定義する.

単一の AWS リージョン 内の複数のアベイラビリティゾーン (AZ) にまたがる DR 戦略は、火災、洪水、大規模な停電などの災害イベントに対して影響を緩和できます。ワークロードを特定の AWS リージョン で実行できなくなるような、可能性の低いイベントに対する保護を実装する必要がある場合には、複数のリージョンを使用する DR 戦略を使用できます。

複数のリージョンにまたがる DR 戦略を設計するときには、以下のいずれかの戦略を選んでください。戦略は、コストと複雑さの昇順、および RTO と RPO の降順でリストされています。 復旧リージョンとは、ワークロードで使用されるプライマリ以外の AWS リージョン を指します。

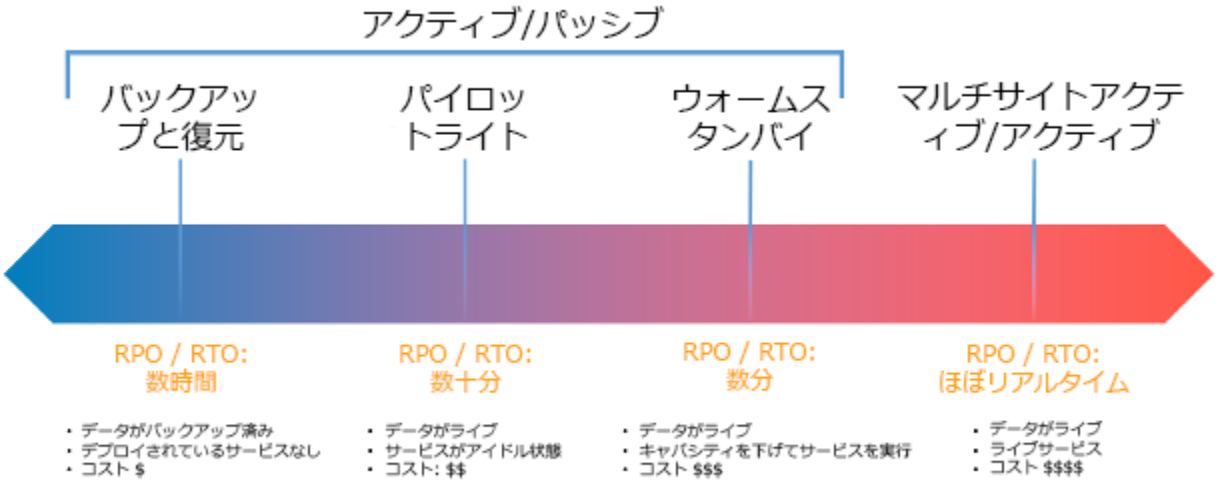


図 17: ディザスタリカバリ (DR) 戦略

- バックアップと復元 (RPO は時間単位、RTO は 24 時間以内): データとアプリケーションを復旧リージョンにバックアップします。自動化されたバックアップまたは連続バックアップを使用すると、ポイントインタイムリカバリが可能であり、場合によっては RPO を 5 分間に短縮できます。災害の際には、インフラストラクチャをデプロイし (インフラストラクチャをコードとして使用して RTO を削減)、コードをデプロイし、バックアップされたデータを復元して、復旧リージョンで災害から復旧します。
- パイロットライト （ 数分間の RPO、数十分間の RTO): コアワークロードインフラストラクチャのコピーを復旧リージョンにプロビジョニングします。データを復旧リージョンにレプリケートして、そこでバックアップを作成します。データベースやオブジェクトストレージなど、データのレ

プリケーションとバックアップのサポートに必要なリソースは、常にオンです。アプリケーションサーバーやサーバーレスコンピューティングなど、その他の要素はデプロイされませんが、必要なときには、必須の設定とアプリケーションコードで作成できます。

- ウォームスタンバイ (数秒間の RPO、数分間の RTO): 完全に機能する縮小バージョンのワークロードを復旧リージョンで常に実行している状態に保ちます。ビジネスクリティカルなシステムは完全に複製され、常に稼働していますが、フリートは縮小されています。データは復旧リージョンでレプリケートされ、使用可能です。復旧時には、システムをすばやくスケールアップして本番環境の負荷を処理できるようにします。ウォームスタンバイの規模が大きいほど、RTO とコントロールプレーンへの依存は低くなります。これを完全にスケールアップしたものはホットスタンバイと呼ばれます。
- マルチリージョン (マルチサイト) アクティブ-アクティブ (ゼロに近い RPO、ほぼゼロの RTO): ワークロードは複数の AWS リージョン にデプロイされ、そこからトラフィックにアクティブに対応します。この戦略では、複数のリージョン間でデータを同期する必要があります。2 つの異なるリージョンレプリカ内の同じレコードへの書き込みによって生じる矛盾を回避または処理する必要があり、これは複雑になることがあります。データレプリケーションは、データの同期に便利であり、特定のタイプの災害から保護しますが、ソリューションがポイントインタイムリカバリのためのオプションを含んでいない限り、データの破損や破壊からは保護しません。

> **Note**
>
> パイロットライトとウォームスタンバイの違いは、理解しにくいかもしれません。どちらも、プライマリリージョンアセットのコピーがある復旧リージョン内の環境を含みます。その違いは、パイロットライトが最初に追加アクションを取らなければリクエストを処理できないのに対して、ウォームスタンバイはトラフィックを直ちに (削減された能力レベルで) 処理できることです。パイロットライトでは、サーバーをオンにして、おそらく追加の (非コア) インフラストラクチャをデプロイし、スケールアップする必要があるのに対して、ウォームスタンバイでは、スケールアップするだけです (すべてが既にデプロイされ、実行しています)。RTO と RPO のニーズに基づいて、両者の中から選択してください。
> コストが懸念事項であり、ウォームスタンバイ戦略での定義と同様の RPO および RTO 目標の達成を目指す場合は、パイロットライトアプローチを採用して、改善された RPO および RTO 目標を提供する AWS Elastic Disaster Recovery などのクラウドネイティブソリューションを検討することができます。

実装手順

1. このワークロードの復旧要件を満たす DR 戦略を決定します。

DR 戦略を選ぶということは、ダウンタイムとデータ損失の削減 (RTO と RPO) 、戦略を実装するコストと複雑性のトレードオフです。必要以上に厳格な戦略の実装は、不要なコストにつながるため避けてください。

例えば、次の図では、許容可能な最大 RTO と、サービス復元戦略に費やすことができるコストの限界を決定しています。特定のビジネス目標の場合、パイロットライトまたはウォームスタンバイの DR 戦略は、RTO とコスト基準の両方を満たすことができます。

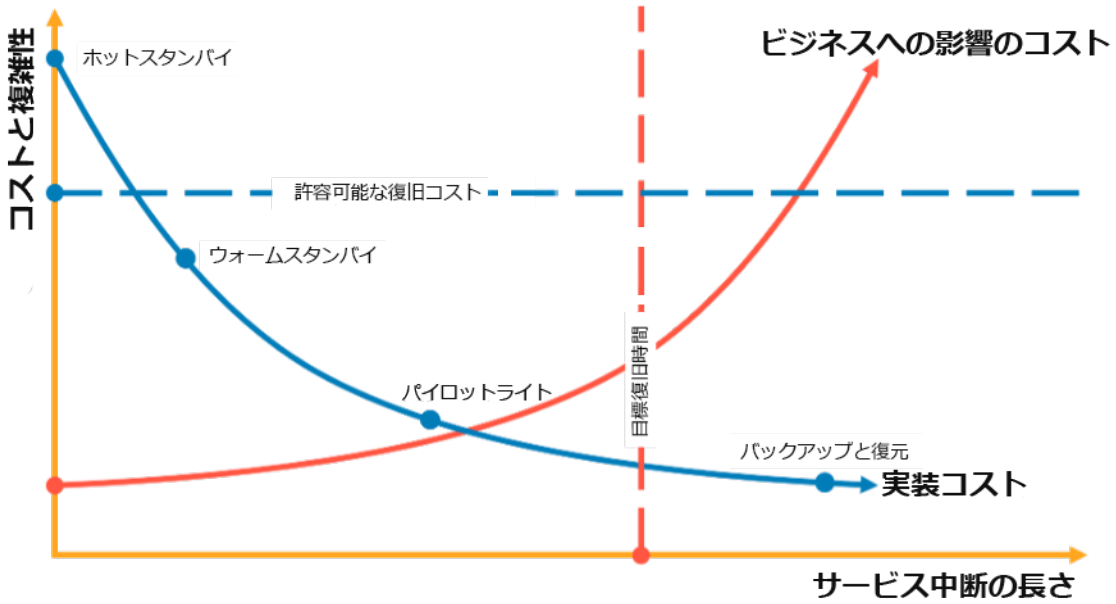


図 18: RTO とコストに基づく DR 戦略の選択を示すグラフ

詳細については、「Business Continuity Plan (BCP)」(ビジネス継続性計画 (BCP)) を参照してください。

2. 選択した DR 戦略の実装パターンをレビューします。

このステップでは、選択した戦略の実装方法を理解します。戦略は、プライマリサイトと復旧サイトとしての AWS リージョン を使用して説明されています。ただし、単一リージョン内のアベイラビリティゾーンを DR 戦略として使用することもでき、その場合は、これら複数の戦略の要素を利用します。

以下の手順では、戦略を特定のワークロードに適用できます。

バックアップと復元

バックアップと復元は、最も実装の複雑性が低い戦略であるとはいえ、ワークロードの復元に必要な時間と労力が多く、より高い RTO と RPO につながります。常にデータのバックアップを取り、これらを別のサイト (別の AWS リージョン など) にコピーすることをお勧めします。

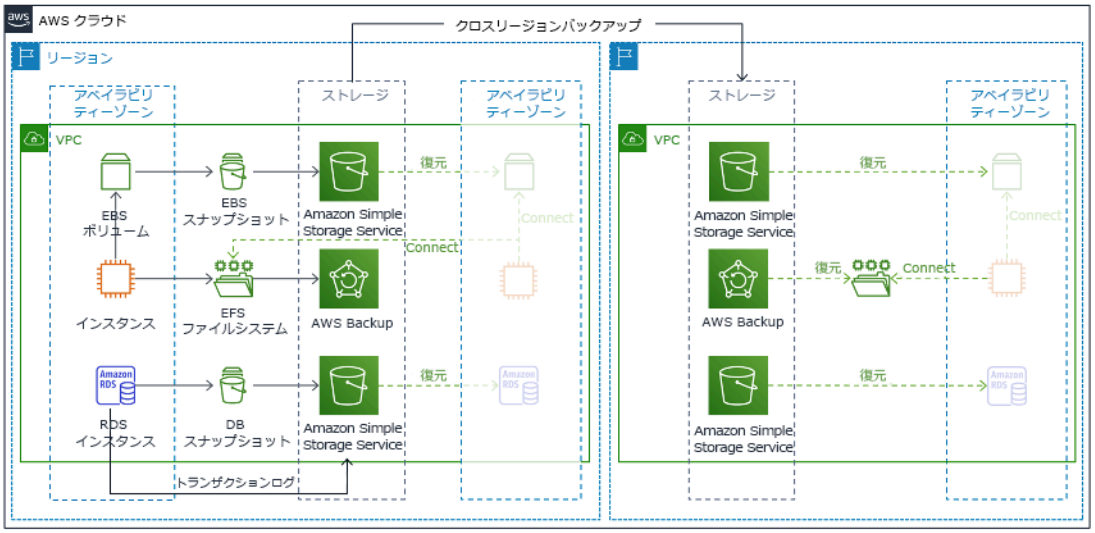


図 19: バックアップと復元アーキテクチャ

この戦略の詳細については、「AWS のディザスタリカバリ (DR) アーキテクチャ、パート II: 迅速なリカバリによるバックアップと復元」を参照してください。

パイロットライト

パイロットライトアプローチでは、プライマリリージョンから復旧リージョンにデータをレプリケートします。ワークロードインフラストラクチャに使用されるコアリソースは復旧リージョンにデプロイされますが、これを機能するスタックにするには、やはり追加のリソースと依存関係が必要です。例えば、図 20 では、コンピューティングインスタンスはデプロイされていません。

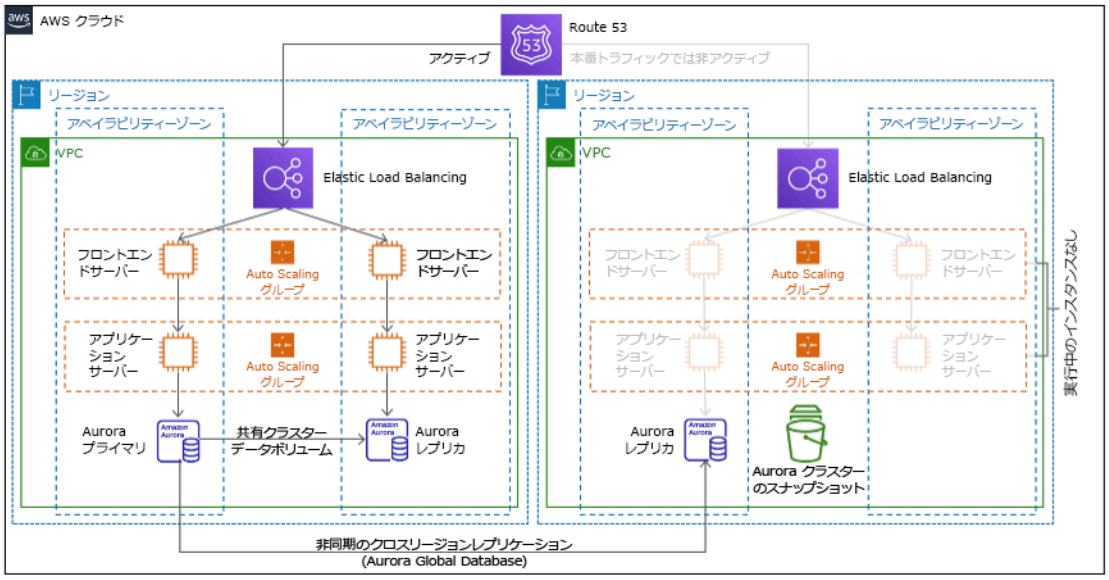


図 20: パイロットライトアーキテクチャ

この戦略の詳細については、「AWS のディザスタリカバリ (DR) アーキテクチャ、パート III: パイロットライトとウォームスタンバイ」を参照してください。

ウォームスタンバイ

ウォームスタンバイのアプローチでは、本番稼働環境の完全に機能するスケールダウンしたコピーを別のリージョンに用意する必要があります。このアプローチは、パイロットライトの概念を拡張して、ワークロードが別のリージョンに常駐するため、復旧時間が短縮されます。復旧リージョンが完全なキャパシティでデプロイされた場合は、ホットスタンバイと呼ばれます。

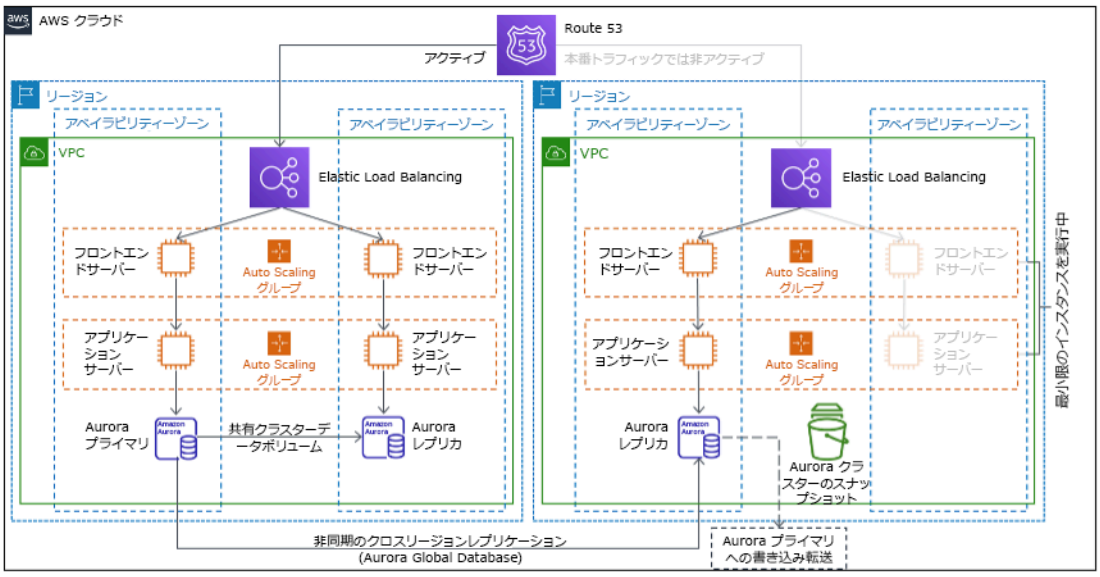


図 21: ウォームスタンバイアーキテクチャ

ウォームスタンバイまたはパイロットライトを使用するには、復旧リージョンのリソースをスケールアップする必要があります。必要なときにキャパシティが利用可能であることを確認するには、EC2 インスタンスのキャパシティ予約の使用を検討します。AWS Lambda を使用する場合は、プロビジョニングした同時実行が、関数の呼び出しにすぐに応答できるように実行環境を提供します。

この戦略の詳細については、「AWS のディザスタリカバリ (DR) アーキテクチャ、パート III: パイロットライトとウォームスタンバイ」を参照してください。

マルチサイトアクティブ/アクティブ

マルチサイトアクティブ/アクティブ戦略の一環として、ワークロードを複数リージョンで同時に実行することができます。マルチサイトアクティブ/アクティブは、デプロイされたすべてのリージョンからのトラフィックを処理します。顧客は、DR 以外の理由でこの戦略を選択することもあります。可用性を高めるためや、グローバルオーディエンスにワークロードをデプロイするときに (エンドポイントをエンドユーザーに近づけるためや、そのリージョン内のオーディエンスに対してローカライズされたスタックをデプロイするため) 使用できます。DR 戦略としては、ワークロードがデプロイされている AWS リージョン の 1 つでワークロードをサポートできない場合、そのリージョンは隔離され、残りのリージョンを使用して可用性を維持します。マルチサイトアクティブ/アクティブは、運用が最も複雑な DR 戦略であり、ビジネス要件上、必須の場合のみ選択してください。

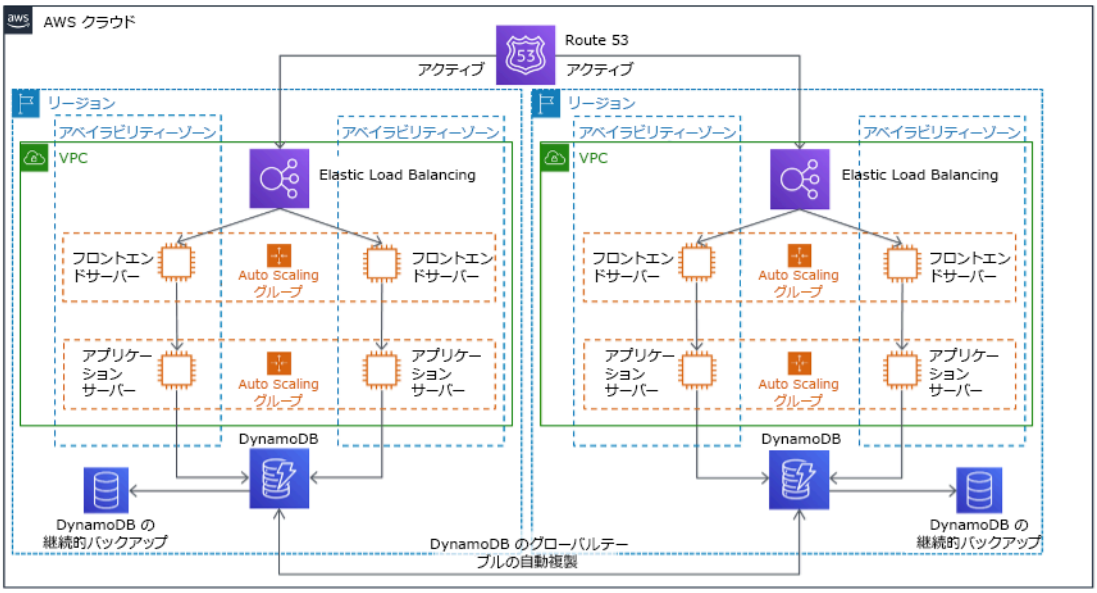


図 22: マルチサイトアクティブ/アクティブアーキテクチャ

この戦略の詳細については、「AWS のディザスタリカバリ (DR) アーキテクチャ、パート IV: マルチサイトアクティブ/アクティブ」を参照してください。

AWS Elastic Disaster Recovery

ディザスタリカバリのためにパイロットライト戦略またはウォームスタンバイ戦略を検討している場合、AWS Elastic Disaster Recovery を使用すると、さらに優れた利点を提供する代替アプローチが提供される場合があります。Elastic Disaster Recovery は、ウォームスタンバイと同様の RPO および RTO 目標を提供することができ、パイロットライトの低コストアプローチを維持します。Elastic Disaster Recoveryは、継続的なデータ保護を使用してプライマリリージョンから復旧リージョンにデータをレプリケートし、秒単位で測定される RPO と分単位で測定できる RTO を達成します。パイロットライト戦略と同様、データのレプリケートに必要となるリソースのみが復旧リージョンにデプロイされるため、コストが抑えられます。Elastic Disaster Recovery を使用する場合、サービスはフェイルオーバーまたはドリルの一環として開始されたコンピューティングリソースの回復を調整します。

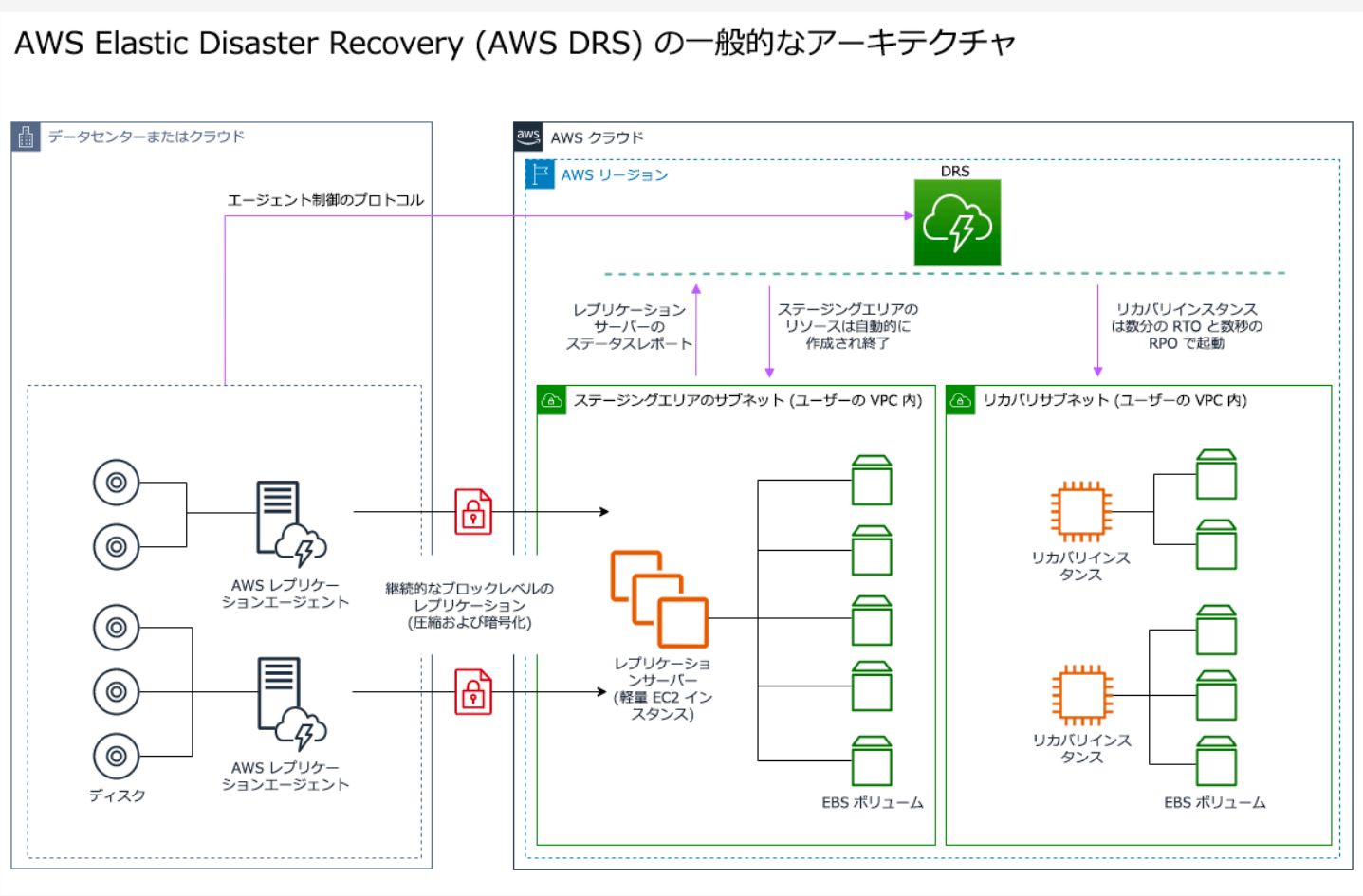


図 23: AWS Elastic Disaster Recoveryアーキテクチャ

データを保護するためのその他のプラクティス

どの戦略でも、データ災害に対する緩和も必要です。連続的なデータレプリケーションは、特定のタイプの災害から保護しますが、戦略に、保存データのバージョニングまたはポイントインタイムリカバリのためのオプションが含まれていない限り、データの破損や破壊からは保護しません。復旧サイトにレプリケートしたデータもバックアップして、レプリカに加えて、ポイントインタイムバックアップを作成する必要があります。

単一の AWS リージョン 内での複数のアベイラビリティーゾーン (AZ) の使用

単一のリージョン内の複数の AZ を使用する場合、DR 実装は上記の戦略の複数の要素を使用します。まず、図 23 に示されているとおり、複数の AZ を使用して、高可用性 (HA) アーキテクチャを作成する必要があります。このアーキテクチャは、マルチサイトアクティブ/アクティブアプローチを活用し、Amazon EC2 インスタンス と Elastic Load Balancer は複数の AZ にデプロイされたリソースを備え、アクティブにリクエストを処理します。このアーキテクチャは、プライマリ Amazon RDS インスタンスに障害が発生した場合 (または AZ 自体に障害が発生した場合)、スタンバイインスタンスがプライマリに昇格するホットスタンバイについても説明しています。

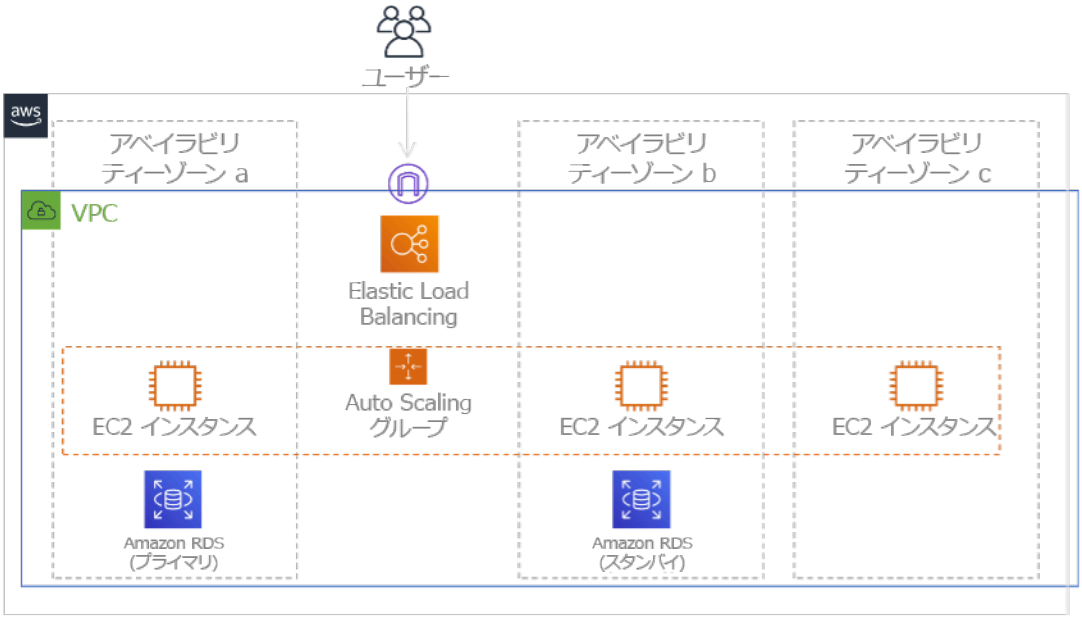


図 24: マルチ AZ アーキテクチャ

この HA アーキテクチャに加えて、ワークロードの実行に必要なすべてのデータのバックアップを追加する必要があります。これは、Amazon EBS ボリューム や Amazon Redshift クラスター などの単一のゾーンに制約されているデータの場合に特に重要です。AZ に障害が発生した場合、このデータを別の AZ に復元する必要があります。可能な場合には、追加の保護層として、データバックアップも別の AWS リージョン にコピーしてください。

単一リージョン、マルチ AZ DR に対する、あまり一般的でない代替アプローチが下記のブログ投稿で説明されています。「Building highly resilient applications using Amazon Route 53 Application Recovery Controller, Part 1: Single-Region stack」(Amazon Route 53 Application Recovery Controller を使用した回復力の高いアプリケーションの構築、パート 1: シングルリージョンスタック)。ここでは、戦略は、AZ 間の分離をできるだけ高く維持して、リージョンのように動作させることです。この代替戦略を使用すると、アクティブ/アクティブまたはアクティブ/パッシブアプローチを選ぶことができます。

> **Note**
>
> ワークロードによっては、規制によるデータレジデンシー要件があります。現在 AWS リージョン が 1 つだけの地域のワークロードにこれが該当する場合、マルチリージョンではビジネスニーズに適しません。マルチ AZ 戦略は、ほとんどの災害に対して良好な保護を提供します。

3. ワークロードのリソースと、それらの設定がフェイルオーバー前 (正常なオペレーション時) に復旧リージョンでどうなるかを評価します。

インフラストラクチャと AWS リソースについては、AWS CloudFormation や、Hashicorp Terraform のようなサードパーティーツールなどの Infrastructure as Code (IaC) を使用します。複数のアカウントとリージョンに単一の操作でデプロイするには、AWS CloudFormation StackSets を使用します。マルチサイトアクティブ/アクティブとホットスタンバイ戦略の場合、復旧リージョンにデプロイされるインフラストラクチャはプライマリリージョンと同じリソースを持ちます。パイロットライトとウォームスタンバイ戦略の場合、デプロイされたインフラストラクチャを本番稼働で使用するには追加のアクションが必要です。CloudFormation のパラメータと条件付きロジックを使用すると、デプロイされるスタックがアクティブかスタンバイかを単一のテンプレートで制御できます。Elastic Disaster Recovery を使用する場合、サービスがアプリケーション設定とコンピューティングリソースの復元をレプリケートおよび調整します。

すべての DR 戦略では、データソースが AWS リージョン の範囲内にバックアップされ、その後、それらのバックアップが復旧リージョンにコピーされる必要があります。AWS Backup は、これらのリソースのバックアップの設定、スケジュール、モニタリングできる一元的なビューを提供します。パイロットライト、ウォームスタンバイ、およびマルチサイトアクティブ/アクティブについては、プライマリリージョンのデータを復旧リージョンのデータリソース、例えば、Amazon Relational Database Service (Amazon RDS) DB インスタンスや Amazon DynamoDB テーブルなど

にもレプリケートする必要があります。したがって、これらのデータリソースはライブであり、復旧リージョンのリクエストに対応できます。

複数のリージョンにまたがる AWS サービスの動作の詳細については、下記に関するこのブログシリーズを参照してください。Creating a Multi-Region Application with AWS Services (AWS サービスを使用したマルチリージョンアプリケーションの作成)。

4. 必要なとき (災害発生時) に復旧リージョンをフェイルオーバーに備える方法を決定し、実装します。

マルチサイトアクティブ/アクティブの場合、フェイルオーバーとは、リージョンを隔離して、残りのアクティブリージョンに頼ることを意味します。一般に、これらのリージョンはトラフィックを受け入れる準備ができています。パイロットライトとウォームスタンバイ戦略の場合、復旧アクションとして、図 20 の EC2 インスタンスなど、不足しているリソースやその他の不足リソースをデプロイする必要があります。

上記の戦略のすべてで、データベースの読み取り専用インスタンスを昇格して、プライマリの読み書きインスタンスにしなければならない場合があります。

バックアップと復元の場合、バックアップからのデータの復元によって、EBS ボリューム、RDS DB インスタンス、DynamoDB テーブルなど、そのデータのリソースを作成します。インフラストラクチャを復元し、コードをデプロイする必要もあります。AWS Backup を使用して、データを復旧リージョンに復元できます。把握 REL09-BP01 バックアップが必要なすべてのデータを特定し、バックアップする、またはソースからデータを再現する をご覧ください。インフラストラクチャの再構築には、Amazon Virtual Private Cloud (Amazon VPC)、サブネット、必要となるセキュリティグループに加え、EC2 インスタンスなどのリソースの作成も含まれます。復元プロセスの大部分を自動化できます。方法の詳細については、このブログ投稿を参照してください。

5. 必要なとき (災害発生時) にフェイルオーバーするトラフィックを再ルーティングする方法を決定し、実装します。

このフェイルオーバー操作は、自動または手動で開始できます。ヘルスチェックまたはアラームに基づくフェイルオーバーの自動開始を使用するときには、不要なフェイルオーバー (誤ったアラーム) によって、使用できないデータやデータ損失などのコストが発生するため、注意が必要です。そのため、多くの場合、手動によるフェイルオーバーの開始が使用されます。この場合でも、フェイルオーバーのステップを自動化できるため、手動開始はボタンを押すようなものです。

AWS サービスを使用するときに検討すべき、いくつかのトラフィック管理オプションがあります。オプションの 1 つに、Amazon Route 53 の使用があります。Amazon Route 53 を使用すると、1 つ以上の AWS リージョン の複数の IP エンドポイントを Route 53 ドメイン名に関連付けることができます。手動で開始するフェイルオーバーを実装するには、Amazon Route 53 Application Recovery Controller を使用できます。Application Recovery Controller は、高可用性データプレーン API を提供して、トラフィックを復旧リージョンに再ルーティングします。フェイルオーバーを実装するときには、火気で説明されているように、データプレーン操作を使用し、コントロールプレーンを避けてください REL11-BP04 復旧中はコントロールプレーンではなくデータプレーンを利用する.

このオプションおよびその他のオプションの詳細については、ディザスタリカバリに関するホワイトペーパーのこのセクションを参照してください。

6. ワークロードをフェイルバックする方法のプランを設計します。

フェイルバックとは、災害イベントの終息後、ワークロード操作をプライマリリージョンに戻すことを言います。インフラストラクチャとコードをプライマリリージョンにプロビジョニングするときには、一般に、最初に使用したのと同じステップに従い、コードとしてのインフラストラクチャとコードデプロイパイプラインに依存します。フェイルバックでの課題は、データストアを復元し、動作中の復旧リージョンとの一貫性を確認することです。

フェイルオーバー状態では、復旧リージョンのデータベースはライブであり、最新データを保持しています。目的は、復旧リージョンからプライマリリージョンへ再同期して、最新であることを確認することです。

いくつかの AWS のサービスは、これを自動的に行います。Amazon DynamoDB のグローバルテーブルを使用している場合、プライマリリージョンのテーブルが使用できなくなった場合でも、オンラインに復帰すると、DynamoDB が保留中の書き込みの反映を再開します。Amazon Aurora Global Database を使って、マネージドプランドフェイルオーバーを使用している場合、Aurora のグローバルデータベースの既存のレプリケーショントポロジが維持されます。そのため、プライマリリージョンの以前の読み書きインスタンスがレプリカになり、復旧リージョンから更新を受け取ります。

これが自動でない場合、プライマリリージョンで復旧リージョンのデータベースのレプリカとしてデータベースを再確立する必要があります。多くの場合、これには、古いプライマリデータベースを削除して、新しいレプリカを作成する必要があります。例えば、計画外のフェイルオーバーを前提として、Amazon Aurora Global Database でこれを行う方法の説明については、以下のラボを参照してください。Fail Back a Global Database (Global Database のフェイルバック)。

フェイルオーバー後、復旧リージョンでの実行を続行できる場合は、これを新しいプライマリリージョンにすることを検討してください。その場合でも、上記のすべてのステップを実行して、前のプ

ライマリリージョンを復旧リージョンにします。一部の組織は、計画的ローテーションを実行して、プライマリリージョンと復旧リージョンを定期的に (3 か月ごとなど) 交換しています。

フェイルオーバーとフェイルバックに必要なすべてのステップをプレイブックに記載して、チームのメンバー全員が使用できるようにし、定期的にレビューする必要があります。

Elastic Disaster Recovery を使用する場合、サービスはフェイルバックプロセスの調整と自動化のサポートを提供します。詳細については、「Performing a failback」(フェイルバックの実行) を参照してください。

実装計画に必要な工数レベル: 高。

リソース

関連するベストプラクティス:

- the section called “REL09-BP01 バックアップが必要なすべてのデータを特定し、バックアップする、またはソースからデータを再現する”
- the section called “REL11-BP04 復旧中はコントロールプレーンではなくデータプレーンを利用する”
- the section called “REL13-BP01 ダウンタイムやデータ消失に関する復旧目標を定義する”

関連するドキュメント:

- AWS Architecture Blog: Disaster Recovery Series (AWS アーキテクチャに関するブログ: ディザスタリカバリシリーズ)
- AWS でのワークロードのディザスタリカバリ: クラウド内での復旧 (AWS ホワイトペーパー)
- クラウド内の災害対策オプション
- サーバーレス、マルチリージョン、アクティブ/アクティブのバックエンドソリューションを 1 時間で構築する
- マルチリージョンのサーバーレスバックエンド – 再ロード
- RDS: リージョン間でのリードレプリカのレプリケーション方法
- Route 53: DNS フェイルオーバーの設定
- S3: クロスリージョンレプリケーション
- What Is AWS Backup? (AWS Backup とは)

- What is Route 53 Application Recovery Controller? (Amazon Route 53 Application Recovery Controller とは)
- AWS Elastic Disaster Recovery
- HashiCorp Terraform: Get Started - AWS (HashiCorp Terraform: 開始方法 - AWS)
- APN パートナー: 災害対策を支援できるパートナー
- AWS Marketplace: products that can be used for disaster recovery (AWS Marketplace: ディザスタリカバリに活用できる商品)

関連動画:

- Disaster Recovery of Workloads on AWS (AWS 上のワークロードのディザスタリカバリ)
- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2) (AWS re:Invent 2018: マルチリージョンアクティブ/アクティブアプリケーションのアーキテクチャパターン (ARC209-R2))
- Get Started with AWS Elastic Disaster Recovery | Amazon Web Services (AWS Elastic Disaster Recovery の使用を開始する | Amazon Web Services)

関連する例:

- Well-Architected Lab - Disaster Recovery - Series of workshops illustrating DR strategies (Well-Architected ラボ - ディザスタリカバリ - DR 戦略を説明するワークショップシリーズ)

REL13-BP03 ディザスタリカバリの実装をテストし、実装を検証する

復旧サイトへの定期的なテストフェイルオーバーにより、適切な動作と、RTO および RPO が満たされることを確認します。

一般的なアンチパターン:

- 本番環境ではフェイルオーバーを実行しない。

このベストプラクティスを活用するメリット: ディザスタリカバリプランを定期的にテストすることで、必要なときに機能することや、チームが戦略の実行方法を把握していることを確認できます。

このベストプラクティスが確立されていない場合のリスクレベル: 高

実装のガイダンス

回避すべきパターンは、まれにしか実行されない復旧経路を作ることです。たとえば、読み取り専用のクエリに使用されるセカンダリデータストアがあるとします。データストアの書き込み時にプライマリデータストアで障害が発生した場合、セカンダリデータストアにフェイルオーバーします。もしこのフェイルオーバーを頻繁にテストしない場合、セカンダリデータストアの機能に関する前提が正しくない可能性があります。セカンダリデータストアの容量は、最後にテストしたときには十分だったかもしれませんが、このシナリオでは負荷に耐えられなくなる可能性があります。エラー復旧がうまくいくのは頻繁にテストする経路のみであることは、これまでの経験からも明らかです。少数の復旧経路を用意することがベストであるのはそのためです。復旧パターンを確立して定期的にテストできます。復旧経路が複雑な場合や重大な場合に復旧経路が正常に機能するという確信を持つには、本番環境でその障害を定期的に実行する必要があります。前述の例では、その必要性に関係なく、スタンバイへのフェイルオーバーを定期的に行う必要があります。

実装手順

1. ワークロードを復旧用にエンジニアリングします。復旧経路を定期的にテストします。復旧指向コンピューティングは、回復を強化するシステムの以下の特性を特徴としています。隔離と冗長性、システム全体の変更のロールバック機能、正常性を監視し判断する機能、診断する機能、自動的な復旧、モジュラー設計、再起動する機能。復旧経路を訓練して、指定された時間内に指定された状態に復旧できるようにします。この復旧中にランブックを使用して問題を文書化し、次のテストの前に解決策を見つけます。
2. Amazon EC2 ベースのワークロードの場合、AWS Elastic Disaster Recovery を使用して、DR 戦略のためのドリルインスタンスを導入して起動します。AWS Elastic Disaster Recovery を使用すると、フェイルオーバーイベントに備えて、ドリルを効率的に実行することができます。また、Elastic Disaster Recovery を使用すると、トラフィックをリダイレクトせずに、テストおよびドリル目的でインスタンスを頻繁に起動できます。

リソース

関連するドキュメント:

- APN パートナー: 災害対策を支援できるパートナー
- AWS Architecture Blog: Disaster Recovery Series (AWS アーキテクチャに関するブログ: ディザスタリカバリシリーズ)
- AWS Marketplace: products that can be used for disaster recovery (AWS Marketplace: ディザスタリカバリに活用できる商品)

- AWS Elastic Disaster Recovery
- AWS でのワークロードのディザスタリカバリ: クラウド内での復旧 (AWS ホワイトペーパー)
- AWS Elastic Disaster Recovery Preparing for Failover (フェイルオーバーの準備)
- バークレー/スタンフォード大学の復旧指向コンピューティングプロジェクト
- What is AWS Fault Injection Simulator? (AWS Fault Injection Simulator とは)

関連動画:

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (AWS re:Invent 2018: マルチリージョンアクティブ-アクティブアプリケーションのアーキテクチャパターン)
- AWS re:Invent 2019: Backup-and-restore and disaster-recovery solutions with AWS (AWS re:Invent 2019: AWS を使用したバックアップと復元およびディザスタリカバリソリューション)

関連する例:

- Well-Architected Lab - Testing for Resiliency (Well-Architected ラボ - 回復力テスト)

REL13-BP04 DR サイトまたはリージョンでの設定ドリフトを管理する

インフラストラクチャ、データ、設定が DR サイトまたはリージョンで必要とされるとおりであることを確認します。たとえば、AMI と Service Quotas が最新であることを確認します。

AWS Config は AWS リソース設定を継続的にモニタリングおよび記録します。これにより AWS Systems Manager Automation のドリフトを検出、トリガーでき、 修正してアラームを発生させます。AWS CloudFormation は、さらにデプロイしたスタックのドリフトを検出できます。

一般的なアンチパターン:

- プライマリロケーションで設定またはインフラストラクチャに変更を加えたときに、復旧ロケーションの更新を行わない。
- プライマリロケーションと復旧ロケーションの潜在的な制限 (サービスの違いなど) を考慮しない。

このベストプラクティスを確立するメリット: DR 環境が既存の環境と一致していることを確認することで、完全な復旧が保証されます。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

- デリバリーパイプラインがプライマリサイトとバックアップサイトの両方に配信しているようにします。アプリケーションを本番環境にデプロイするための配信パイプラインは、開発環境やテスト環境など、指定されたすべての災害対策戦略の口ケーションに分散する必要があります。
- AWS Config を有効にして、潜在的なドリフト口ケーションを追跡します。AWS Config ルールを使用して、ディザスタリカバリ戦略を実施するシステムを構築し、ドリフトを検出したときにアラートを生成します。
  - AWS Config ルール による非準拠 AWS リソースの修復
  - AWS Systems Manager Automation をトリガーして
- AWS CloudFormation を使用して、インフラストラクチャをデプロイします。AWS CloudFormation は、CloudFormation テンプレートが指定するものと実際にデプロイされているものとの間のドリフトを検出できます。
  - AWS CloudFormation: CloudFormation スタック全体のドリフトを検出する

リソース

関連するドキュメント:

- APN パートナー: 災害対策を支援できるパートナー
- AWS アーキテクチャブログ: ディザスタリカバリシリーズ
- AWS CloudFormation: CloudFormation スタック全体のドリフトを検出する
- AWS Marketplace: 災害対策に活用できる商品
- AWS Systems Manager Automation をトリガーして
- AWS でのワークロードの災害対策: クラウド内での復旧 (AWS ホワイトペーパー)
- How do I implement an Infrastructure Configuration Management solution on AWS? (AWS でインフラストラクチャ設定管理ソリューションを実装するにはどうすればよいですか?)
- AWS Config ルール による非準拠 AWS リソースの修復

関連動画:

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2) (マルチリージョンアクティブ/アクティブアプリケーションのアーキテクチャパターン)

REL13-BP05 復旧を自動化する

AWS またはサードパーティ製のツールを使用して、システムの復旧を自動化し、トラフィックを DR サイトまたはリージョンにルーティングします。

設定されたヘルスチェックに基づいて、Elastic Load Balancing や AWS Auto Scaling などの AWS サービスは、正常なアベイラビリティゾーンに負荷を分散できますが、Amazon Route 53、や AWS Global Accelerator などのサービスは、正常な AWS リージョン に負荷をルーティングできます。Route 53 Application Recovery Controller は、準備状況のチェックとルーティングコントロール機能を使用して、フェイルオーバーの管理と調整を支援します。これらの機能は、障害から回復するアプリケーションの能力を継続的にモニタリングするため、複数の AWS リージョン、アベイラビリティゾーン、およびオンプレミスにまたがってアプリケーションの回復を管理できます。

既存の物理または仮想データセンターまたはプライベートクラウド上のワークロードについては、 AWS Elastic Disaster Recovery(AWS Marketplace から入手可能) により、組織は自動ディザスタリカバリ戦略を AWS にセットアップできます。CloudEndure は、AWS のクロスリージョン/クロス AZ ディザスタリカバリもサポートしています。

一般的なアンチパターン:

- 同一の自動フェイルオーバーとフェイルバックを実装すると、障害が発生したときにフラッピングが発生する可能性があります。

このベストプラクティスを活用するメリット: 自動復旧により、手動エラーの可能性が排除され、復旧時間が短縮されます。

このベストプラクティスを活用しない場合のリスクレベル: ミディアム

実装のガイダンス

- 復旧経路を自動化します。復旧時間が短い場合に人が判断して対処する方法は、高い可用性シナリオには利用できません。システムはあらゆる状況下で自動的に復旧する必要があります。
  - CloudEndure Disaster Recover を自動化したフェイルオーバーとフェイルバックに使用する CloudEndure Disaster Recovery は、マシン (オペレーティングシステム、システム状態設定、データベース、アプリケーション、ファイルなど) をターゲット AWS アカウント および希望するリージョンの低コストのステージングエリアに継続的にレプリケートします。災害が発生した場合、CloudEndure Disaster Recovery に指示して、数千台のマシンを数分で完全にプロビジョニングされた状態で自動的に起動できます。

- 災害対策フェイルオーバーとフェイルバックの実行
- CloudEndure Disaster Recovery

リソース

関連するドキュメント:

- APN パートナー: 災害対策を支援できるパートナー
- AWS アーキテクチャブログ: ディザスタリカバリシリーズ
- AWS Marketplace: 災害対策に活用できる商品
- AWS Systems Manager Automation をトリガーして
- AWS への CloudEndure Disaster Recovery
- AWS でのワークロードのディザスタリカバリ: クラウドでの復旧 (AWS ホワイトペーパー)

関連動画:

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)

可用性目標の実装例

このセクションでは、リバースプロキシ、Amazon S3 上の静的コンテンツ、アプリケーションサーバー、およびデータを永続的に保存する SQL データベースで構成される典型的なウェブアプリケーションのデプロイを使用して、ワークロード設計のレビューを行います。それぞれの可用性の目標ごとに、実装例を示します。このワークロードでは、コンテナまたはコンピューティング用の AWS Lambda とデータベース用の NoSQL (Amazon DynamoDB など) を使用することもできますが、その場合もアプローチは似ています。各シナリオでは、以下のトピックのワークロード設計を通じて可用性の目標を達成する方法を示します。

トピック	詳細については、このセクションを参照してください
リソースをモニタリングする	ワークロードリソースをモニタリングする
需要の変化に対する適応方法	需要の変化に適応するようにワークロードを設計する
変更の実装	変更の実装
データのバックアップ方法	データのバックアップ方法
弾力性のためのアーキテクト	障害部分を切り離してワークロードを保護する コンポーネントの障害に耐えられるようにワークロードを設計する
回復力をテストする方法	テストの信頼性
災害対策 (DR) を計画する	災害対策 (DR) を計画する

依存関係の選択

アプリケーションには Amazon EC2 を使用することにしました。今回は、Amazon RDS と複数のアベイラビリティーゾーンを使用して、アプリケーションの可用性を向上させる方法について説明します。DNS には Amazon Route 53 を使用します。複数のアベイラビリティーゾーンを使用する場合は、Elastic Load Balancing を使用します。 Amazon S3 はバックアップと静的コンテンツに使用さ

れます。より高い信頼性を実現するためには、より高い可用性を持つサービスを使用しなければなりません。

単一リージョンのシナリオ

トピック

99% のシナリオ

これらのワークロードはビジネスに役立ちますが、 利用できない場合には 不便でしかありません。このタイプのワークロードには、内部ツール、内部ナレッジ管理やプロジェクト追跡があります。または、実験的なサービスから提供され、必要に応じてサービスを非表示にできる機能トグルを備えたものであれば、実際の顧客向けワークロードでもかまいません。

このようなワークロードは、1 つのリージョンと 1 つのアベイラビリティーゾーンでデプロイできます。

リソースをモニタリングする

サービスのホームページが HTTP 200 OK ステータスを返しているかどうかを確認するには、簡単なモニタリングを行います。問題が発生した場合、当社のプレイブックは、インスタンスのログ記録を使って根本原因を特定することを提案します。

需要の変化に対する適応方法

一般的なハードウェア障害、緊急のソフトウェア更新、その他の破壊的な変更に関するプレイブックも用意されています。

変更の実装

AWS CloudFormation を使用してインフラストラクチャをコードとして定義し、特に障害発生時の再構築を高速化できるようにします。

ソフトウェアの更新は、ランブックを使用して手動で実行され、サービスのインストールと再開にはダウンタイムが発生します。デプロイの最中に問題が発生した場合、ランブックでは旧バージョンにロールバックする方法を説明しています。

問題の修正は運用チームと開発チームがログを分析することで行われ、その修正が優先されて作業が完了した後、修正プログラムがデプロイされます。

データのバックアップ方法

暗号化されたバックアップデータは、ランブックを使って、ベンダーや専用バックアップソリューションにより Amazon S3 に送信されます。そのバックアップが正常に機能するかどうかは、ランブックを使って、データの復元およびデータを使用できるかどうかの確認を定期的に行います。Amazon S3 オブジェクトのバージョニング設定を行い、バックアップデータを削除できる権限を削除します。要件に従いデータをアーカイブまたは完全に削除するため、Amazon S3 バケットのライフサイクルポリシーを利用します。

弾力性のためのアーキテクト

ワークロードは、1 つのリージョンと 1 つのアベイラビリティーゾーンでデプロイされます。アプリケーションについては、データベースも含めて単一インスタンスにデプロイします。

回復力をテストする方法

新しいソフトウェアにはデプロイパイプラインが計画されており、ユニットテストも含まれていますが、そのほとんどは組み立てられたワークロードのホワイトボックスまたはブラックボックステストです。

災害対策 (DR) を計画する

故障が発生している間は、故障状態が解消するまで待ちつつ、必要に応じてランブックを介した DNS の修正により静的ウェブサイトへリクエストをルーティングさせます。これにかかる復旧時間は、インフラストラクチャがデプロイされ、データベースが最も直近のバックアップに復元される速度によって決まります。これは、同じアベイラビリティーゾーン内にデプロイすることも、アベイラビリティーゾーンに障害が発生した場合、ランブックを使用して異なるアベイラビリティーゾーン内にデプロイすることもできます。

可用性の設計目標

問題について理解して復旧を実行する判断をするまで 30 分、全体のスタックを AWS CloudFormation にデプロイするまで 10 分、新しいアベイラビリティーゾーンにデプロイし、データベースを復元させるまで 30 分かかったとします。この場合は障害から復旧するまでに 70 分かかることになります。四半期ごとに障害が発生すると仮定すると、年間の推定影響時間は 280 分 (4 時間 40 分) になります。

つまり、可用性の上限は 99.9% です。実際の可用性は、実際の障害発生率、障害の持続期間、各障害からの実際の復旧速度によっても異なります。このアーキテクチャでは、プログラム更新と実際のイベントのためにアプリケーションをオフラインにする (一度の更新は 4 時間 x 1年に 6 回 = 年間 24 時間) 必要があります。したがって、このホワイトペーペーで前述したアプリケーションの可用性の表を参照すると、この 可用性の設計目標 は 99% であることがわかります。

要約

トピック	導入
リソースをモニタリングする	サイトのヘルスチェックのみ (アラートなし)
需要の変化に対する適応方法	再デプロイによる垂直スケーリング
変更の実装	デプロイとロールバックに関するランブック
データのバックアップ方法	バックアップと復元のためのランブック
弾力性のためのアーキテクト	全面的な再構築、バックアップから復元
回復力をテストする方法	全面的な再構築、バックアップから復元
災害対策 (DR) を計画する	暗号化されたバックアップ、必要に応じて別のアベイラビリティーゾーンに復元

99.9% のシナリオ

次に高いレベルの可用性が求められるのは、求められる可用性は高くても、短い期間であればサービス停止に耐えられるアプリケーションです。このタイプのワークロードは、通常、ダウンタイムによる影響を受けるのが従業員であるような内部運用で使用されます。ビジネスの収益性は高くありませんが、より長い復旧時間や復旧時点を許容できれば、このタイプのワークロードは顧客向けにもできます。このようなワークロードには、アカウント管理または情報管理のためのアプリケーションなどがあります。

アベイラビリティーゾーンを 2 つ使用してデプロイし、アプリケーションを個別の階層に分離することで、ワークロードの可用性を向上させることができます。

リソースをモニタリングする

ホームページで HTTP 200 OK ステータスをチェックすると、モニタリング機能が拡張されてウェブサイト全体の可用性を監視します。さらに、ウェブサーバーを交換したとき、またはデータベースがフェイルオーバーしたときにはアラートを出します。また、Amazon S3 で静的コンテンツの可用性をモニタリングし、利用不可になったときにアラートを出します。ログ記録の集約は、管理業務を容易にし、また根本原因の分析に役立ちます。

需要の変化に対する適応方法

自動スケーリング は、EC2 インスタンスの CPU 使用率をモニタリングし、インスタンスを追加または削除して CPU ターゲットを 70% に維持するように設定されていますが、アベイラビリティーゾーンごとに EC2 インスタンスが 1 つしかありません。RDS インスタンスの負荷パターンからスケールアップが必要であることが示されている場合、メンテナンスウィンドウ中にインスタンスタイプを変更します。

変更の実装

インフラストラクチャのデプロイテクノロジーは、前のシナリオと同じです。

新しいソフトウェアの提供は、2 ~ 4 週間ごとの定期スケジュールで行われます。ソフトウェアのアップデートは Canary デプロイ やブルーグリーンデプロイのパターンではなく、一括の置き換えによって自動化されます。ロールバックをする判断は、ランブックに従って行います。

問題の根本原因を特定するためにプレイブックがあります。根本原因がわかると、運用チームと開発チームが一体となって エラーの修正方法を特定します。修正は、それが開発された後で反映されます。

データのバックアップ方法

データのバックアップと復元には、Amazon RDS を使います。復旧要件を確実に満たすため、これはランブックを使用して定期的に実行されます。

弾力性のためのアーキテクト

アベイラビリティーゾーンを 2 つ使用してデプロイメントを行い、アプリケーションを個別の階層に分離することで、アプリケーションの可用性を向上させることができます。AWS Key Management Service を介して暗号化されたストレージを備えた Elastic Load Balancing、Auto Scaling、Amazon RDS マルチ AZ などの複数のアベイラビリティーゾーンで動作するサービスを使

用します。これにより、リソースレベルおよびアベイラビリティーゾーンレベルの耐障害性を持つことができます。

ロードバランサーは、正常なアプリケーションインスタンスにのみトラフィックをルーティングします。ヘルスチェックは、インスタンス上のアプリケーション機能を示すデータプレーン / アプリケーションレイヤーで行う必要があります。制御プレーンに対してこのチェックを行うことはできません。ウェブアプリケーションのヘルスチェック URL は、ロードバランサーと Auto Scaling による使用のために存在し設定されます。これにより障害が発生したインスタンスが削除および置き換えられます。Amazon RDS は、プライマリアベイラビリティーゾーンでインスタンスに障害が発生した場合に 2 番目のアベイラビリティーゾーンで使用できるアクティブなデータベースエンジンを管理し、修復して同じ弾力性に復元します。

サービスの階層を分離し、分散システムの回復パターンを適用します。例えばアベイラビリティーゾーンのフェイルオーバーでデータベースが一時的に使用不能になった場合でもアプリケーションを使用できるようにします。これにより、アプリケーション全体の信頼性を向上させます。

回復力をテストする方法

前のシナリオと同じように、機能テストを行います。ELB、Auto Scaling 、RDS フェイルオーバーの自己修復機能はテストしません。

私たちは、一般的なデータベースの問題、セキュリティ関連のインシデント、失敗したデプロイについてのプレイブックを持つことになります。

災害対策 (DR) を計画する

ランブックは、ワークロード全体の回復と共通レポートのためにあります。復旧では、ワークロードと同じリージョンに保存されているバックアップを使用します。

可用性の設計目標

私たちは、障害のなかには復旧作業を手動で行わざるを得ないケースもあると考えています。ただしこのシナリオでは自動化が進んでいるため、手動の作業が必要なイベントは年間に 2 回のみと想定しています。当社の推定では、復旧の実行を決定するまでに 30 分、復旧自体が 30 分以内に完了するとしています。この場合は障害から復旧するまで 60 分かかることになります。年間で障害が 2 件発生すると仮定すると、その影響時間は年間 120 分です。

つまり、可用性の上限は 99.95% です。実際の可用性は、実際の障害発生率、障害の持続期間、各障害からの実際の復旧速度によっても異なります。このアーキテクチャでは、プログラム更新のために

アプリケーションを一時的にオフラインにする必要がありますが、この更新作業は自動化されています。これについては年間 150 分、更新作業ごとに 15 分、年間 10 回と推定します。これによってサービスが利用できない時間は年間 270 分であるため 可用性の設計目標 は 99.9% です。

要約

トピック	導入
リソースをモニタリングする	サイトのヘルスチェックのみ、ダウンのアラート送信。
需要の変化に対する適応方法	ウェブと Auto Scaling アプリケーション層の ELB、マルチ AZ RDS のサイズ変更。
変更の実装	一括自動デプロイ、ロールバックに関するランブック。
データのバックアップ方法	RPO の要件を満たすための RDS を使用した自動バックアップ、復元に関するランブック。
弾力性のためのアーキテクト	Auto Scaling による自己修復可能なウェブおよびアプリケーション層の提供、マルチ AZ の RDS。
回復力をテストする方法	ELB、自己修復可能なアプリケーション、Multi-AZ の RDS、明示的なテストなし
災害対策 (DR) を計画する	同じ AWS リージョンへ RDS 経由で暗号化バックアップ

99.99% のシナリオ

このアプリケーションの可用性目標を達成するには、アプリケーションはコンポーネント障害に対する耐性を持つ必要があります。アプリケーションは、追加のリソースを取得することなく障害を吸収できなければなりません。この可用性目標のレベルは、E コマースサイト、B to B ウェブサービス、または高トラフィックのコンテンツ/メディアサイトなど、企業にとって主要または重要な収益源となるミッションクリティカルなアプリケーションが対象です。

リージョン内で静的に安定しているアーキテクチャを使用することで、可用性をさらに 向上させることができます。この可用性目標のレベルでは、障害に耐えるためにワークロードの動作をコントロールプレーンで変更する必要はありません。たとえば、アベイラビリティーゾーンが 1 つ使用不可になっても耐えられるだけの十分な容量が必要です。Amazon Route 53 DNS の更新は必要ありません。S3 バケットの作成および変更、新しい IAM ポリシーの作成 (ポリシーの変更)、Amazon ECS タスク設定の変更などを問わず、新しいインフラストラクチャを作成する必要はありません。

リソースをモニタリングする

モニタリングは、問題発生時のアラートだけでなく、オペレーションが成功したときのメトリクスも対象です。さらに、障害が発生したウェブサーバーがリプレイスしたときやデータベースがフェイルオーバーしたとき、またアベイラビリティゾーンに障害が発生したときには、アラートが出るようにします。

需要の変化に対する適応方法

Amazon Aurora を RDS として使用して、リードレプリカの Auto Scaling を有効にします。これらのアプリケーションでは、プライマリコンテンツの書き込み可用性よりも読み取り可用性を優先して設計することも重要なアーキテクチャの判断となります。また、Aurora は、必要に応じて 10 GB 単位で最大 64 TB までストレージを自動的に拡張することもできます。

変更の実装

ここでは、Canary デプロイまたはブルーグリーンデプロイを用いて、分離した各ゾーンにそれぞれアップデートを展開します。このデプロイは、KPI に問題がある場合のロールバックをはじめ、完全に自動化されています。

ランブックは、厳密なレポート要件とパフォーマンス追跡のためのものです。成功したオペレーションが、パフォーマンスまたは可用性の目標を達成できない傾向がある場合、プレイブックを使用してその傾向の原因を特定します。プレイブックは、未発見の障害モードやセキュリティインシデントのためのものです。また、障害の根本原因を明らかにするためのプレイブックもあります。さらに、Infrastructure Event Management サービスの AWS Support とも連携しています。

ウェブサイトを構築および運用するチームは、想定外の障害が発生した場合にその対応方法を決定し、実装後にデプロイする修正の優先順位付けをします。

データのバックアップ方法

データのバックアップと復元には、Amazon RDS を使います。復旧要件を確実に満たすため、これはランブックを使用して定期的に実行されます。

弾力性のためのアーキテクト

このアプローチにはアベイラビリティーゾーンを 3 つ使用することを推奨します。3 つのアベイラビリティーゾーンをデプロイした場合、各ゾーンの静的キャパシティはピーク時の 50 %になります。アベイラビリティーゾーンを 2 つにすることも可能ですが、静的に安定した容量のコストが高くなります。これは両方のアベイラビリティーゾーンがピーク時と同じ 100% のキャパシティーである必要があるためです。ここでは Amazon CloudFront を追加して、地理的なキャッシュとアプリケーションのデータプレーン上のリクエストを削減します。

RDS として Amazon Aurora を使用し、3 つのゾーンすべてにリードレプリカをデプロイします。

アプリケーションは、すべてのレイヤーでソフトウェア / アプリケーションの回復パターンを使用して構築されます。

回復力をテストする方法

デプロイパイプラインには、パフォーマンス、負荷、障害注入テストなどの一連の完全なテストが含まれます。

私たちは、手順を逸脱することなくタスクを実行できるように、ランブックを使用しながら、ゲームデーを通して障害復旧手順を定期的にテストします。ウェブサイトを構築するチームは、ウェブサイトの運用も行っています。

災害対策 (DR) を計画する

ランブックは、ワークロード全体の回復と共通レポートのためにあります。復旧では、ワークロードと同じリージョンに保存されているバックアップを使用します。復元手順は、ゲームデーの一環として定期的に実施されます。

可用性の設計目標

復旧を行うには、少なくとも一部の障害では人間の判断が必要になると想定していますが、このシナリオでは自動化が進んでいるため、この判断が必要となるイベントは年間 2 回であり、復旧対応は迅速に行うことができると想定します。当社の推定では、復旧の実行を決定するまでに 10 分、復旧自体が 5 分以内に完了するとしています。この場合は障害から復旧するまで 15 分かかることになります。年間で障害が 2 件発生すると仮定すると、その影響時間は年間 30 分と推定できます。

つまり、可用性の上限は 99.99% です。実際の可用性は、実際の障害発生率、障害の持続期間、各障害からの実際の復旧速度によっても異なります。このアーキテクチャでは、アプリケーションはオン

ラインで常に更新されていると想定しています。これに基づくと、 可用性の設計目標 は 99.99% です。

要約

トピック	導入
リソースをモニタリングする	すべてのレイヤーと KPI に関するヘルスチェック、設定されたアラーム作動時にアラート送信、すべての障害時にアラート通知。傾向を検知し、目標設計を管理するために、運用ミーティングを厳格に実施。
需要の変化に対する適応方法	ウェブと自動スケーリングアプリケーション層の ELB; Aurora RDSの複数のゾーンでのストレージとリードレプリカの自動スケーリング。
変更の実装	KPI またはアラートがアプリケーション内の未検出の問題を示しているときは、自動デプロイ (カナリアまたはブルーグリーン) と自動ロールバックを実施。分離ゾーンでデプロイ。
データのバックアップ方法	RPO の要件を満たすための RDS を使用した自動バックアップ、ゲームデ–に定期的に自動復元を実践。
弾力性のためのアーキテクト	アプリケーションの障害切り離しゾーンを実装。Auto Scaling による自己修復可能なウェブおよびアプリケーション層の提供。Multi-AZ の RDS。
回復力をテストする方法	コンポーネントと分離ゾーンの障害のテストをゲームデーに定期的に運用スタッフと一緒にパイプラインで実践。不明な問題の診断のためのプレイブックと根本原因分析プロセスが存在。
災害対策 (DR) を計画する	ゲームデーで実践しているのと同じ AWS に RDS 経由で暗号化バックアップ。

複数リージョンのシナリオ

アプリケーションを複数の AWS リージョンで実装すると、リージョンの自律性を保つためにリージョンを分離していることもあり、運用コストが高くなります。この方法を採用するには十分に検討する必要があります。とは言え、各リージョン間にはお互いを断絶するための強力な境界が存在しており、私たちは複数リージョンにまたがって関連する障害が発生することを避けるために多大な努力を払っています。複数リージョンを使用すると、 1 つのリージョンの AWS のサービスでハード依存の障害が発生した場合に、復旧時間をより細かくコントロールできるようになります。このセクションでは、さまざまな実装パターンとその可用性の例について説明します。

トピック

可用性がスリーアンドハーフナイン (99.95%) で、復旧時間が 5 ~ 30 分

このアプリケーションの可用性目標を達成するには、非常に短いダウンタイムと、一定時間内のデータ消失を最小限に抑える必要があります。この可用性目標を持つアプリケーションには、銀行、投資サービス、緊急サービス、データキャプチャなどの分野のアプリケーションがあります。こういったアプリケーションの目標復旧時間および復旧時点は非常に短いです。

復旧時間をさらに短縮するには ウォームスタンバイ を 2 つの AWS リージョンにまたがるアプローチで使用します。パッシブサイトをスケールダウンし、すべてのデータの結果整合を保ちながら、ワークロード全体を両方のリージョンにデプロイします。両方のデプロイは リージョン内で 静的に安定した状態になります。このアプリケーションは、分散システムの回復パターンを使用して構築する必要があります。軽量の ルーティング コンポーネントを作成してワークロードの状態をモニタリングする必要があります。これは、必要に応じてトラフィックをパッシブリージョンにルーティングするように設定できます。

リソースをモニタリングする

ウェブサーバーの交換、データベースやリージョンのフェイルオーバーが発生した際には毎回アラートを発生させます。また、Amazon S3 で静的コンテンツの可用性をモニタリングし、利用不可になったときにアラートを出します。ログ記録の集約は、管理業務を容易にし、また各リージョンの根本原因の分析に役立ちます。

ルーティングコンポーネントは、アプリケーションの状態と、リージョンの強い依存関係の両方をモニタリングします。

需要の変化に対する適応方法

フォーナインのシナリオと同じ。

変更の実装

新しいソフトウェアの提供は、2～4 週間ごとの定期スケジュールで行われます。ソフトウェアの更新は、Canary デプロイやブルーグリーンデプロイパターンにより自動化されます。

ランブックは、リージョンのフェイルオーバーが発生した場合、これらのイベント中に発生した一般的な顧客の問題、通常のレポーティングのためにあります。

一般的なデータベースの問題、セキュリティ関連のインシデント、デプロイの失敗、リージョンのフェイルオーバーによる想定外の顧客の問題、問題の根本原因の究明に関するプレイブックもあります。根本原因がわかると、運用チームと開発チームが一体となって エラーの対応方法を決定し、その修正プログラムが完成したらデプロイを行います。

私たちは、Infrastructure Event Management の AWS Support とも連携しています。

データのバックアップ方法

99.99% のシナリオと同様に、RDS の自動バックアップを使用し、S3のバージョニングを使用しています。データは、アクティブリージョン内の Aurora RDS クラスターからパッシブリージョン内のクロスリージョンリードレプリカに自動的かつ非同期にレプリケートされます。S3 クロスリージョンレプリケーションは、データをアクティブリージョンからパッシブリージョンに自動的かつ非同期に移動するために使用されます。

弾力性のためのアーキテクト

フォーナインのシナリオと同じですが、リージョン間フェイルオーバーもできます。これは手動で管理されます。フェイルオーバーの間は、DNS フェイルオーバーによりリクエストを静的なウェブサイトにルーティングし、2 つ目のリージョンで復旧を行います。

回復力をテストする方法

フォーナインのシナリオと同じです。さらに、ランブックを使用してゲームデーを行い、アーキテクチャを検証します。即時の実装とデプロイのために、RCA の修正が機能リリースよりも優先されます。

災害対策 (DR) を計画する

リージョン間フェイルオーバーは手動で管理されます。すべてのデータは非同期にレプリケートされます。ウォームスタンバイの インフラストラクチャ はスケールアウトされます。これは、AWS Step Functions で実行されるワークフローを使用して自動化できます。Auto Scaling グループを更新してインスタンスのサイズを変更する SSM ドキュメントを作成できるため、AWSSystems Manager (SSM) もこの自動化に役立ちます。

可用性の設計目標

復旧を行うためには、少なくとも一部の障害では人間の判断が必要になると想定していますが、このシナリオでは自動化が進んでいるため、この判断が必要となるイベントは年間 2 回であると想定します。当社の推定では、復旧の実行を決定するまでに 20 分、復旧自体が 10 分以内に完了するとしています。この場合は障害から復旧するまでに 30 分かかることになります。年間で障害が 2 件発生すると仮定すると、その影響時間は年間 60 分と推定できます。

つまり、可用性の上限は 99.95% です。実際の可用性は、実際の障害発生率、障害の持続期間、各障害からの実際の復旧速度によっても異なります。このアーキテクチャでは、アプリケーションはオンラインで常に更新されていると想定しています。これに基づくと、 可用性の設計目標 は 99.95% です。

要約

トピック	導入
リソースをモニタリングする	AWS リージョンレベルでのDNSのヘルスチェックやKPIを含む全レイヤーでのヘルスチェック、設定したアラームが作動した場合のアラート送信、すべての障害に対するアラート通知。傾向を検知し、目標設計を管理するために、運用ミーティングを厳格に実施。
需要の変化に対する適応方法	ウェブと 自動スケーリング アプリケーション層の ELB; Aurora RDS のアクティブまたはパッシブリージョンでの、複数ゾーンのストレージとリードレプリカの自動スケーリング。静的安定性を実現するために AWS リージョン間で同期されたデータとインフラストラクチャ。

トピック	導入
変更の実装	Canary またはブルー/グリーンによる自動デプロイと、KPI またはアラートによってアプリケーションに未検出の問題があることが示された場合の自動ロールバック。デプロイは、一度に 1 つの AWS リージョンの 1 つの分離ゾーンに対して行われます。
データのバックアップ方法	RPO の要件を満たすための RDS を使用した AWS リージョンごとの自動バックアップと、ゲームデ–で定期的に実践している自動復元。Aurora RDS および S3 データは、アクティブリージョンからパッシブリージョンに自動的かつ非同期的にレプリケートされます。
弾力性のためのアーキテクト	自動スケーリングによる自己修復可能なウェブおよびアプリケーション層の提供。Multi-AZ の RDS。フェイルオーバー時に提示された静的サイトを使用してリージョンのフェイルオーバーを実施。
回復力をテストする方法	コンポーネントと分離ゾーンの障害のテストをゲームデーに定期的に運用スタッフと一緒にパイプラインで実践。不明な問題の診断のためのプレイブックと根本原因分析プロセスが存在。問題の内容とその修正方法または予防方法の通信経路。即時の実装とデプロイのために、RCA の修正を機能リリースよりも優先。

トピック	導入
災害対策 (DR) を計画する	別のリージョンにデプロイされたウォームスタンバイ。インフラストラクチャは、 AWS Step Functions または AWS Systems Manager ドキュメントを使用して実行されるワークフローを使用してスケールアウトされます。RDS 経由の暗号化されたバックアップ。2 つの AWS リージョン間のクロスリージョンリードレプリカ。Amazon S3 での静的アセットのクロスリージョンレプリケーション。現在の有効な AWS リージョンへの復元を AWS と連携してゲームデ–に実践。

ファイブナイン (99.999%) 以上のシナリオで、復旧時間が 1 分未満

このアプリケーションの可用性目標を達成するには、ダウンタイムや一定時間内のデータロスがほぼゼロである必要があります。この可用性目標を持つアプリケーションには、非常に大きな収益を生み出すビジネスの中核となる、一部の銀行、投資サービス、ファイナンス、政府機関、その他の重要なビジネスアプリケーションなどがあります。求められているのは、極めて一貫性が高いデータストアと、全レイヤーにおける完全な冗長性です。当社では、SQL ベースのデータストアを選択しています。しかし、RPO を極端に短くすることが困難となるシナリオもあります。データをパーティションに分割すれば、データロスをなくすことができるかもしれません。そのためには、地理的に離れたロケーション間のデータの整合性を保つためにパーティション間でデータの移動またはコピーする機能を加える必要が出てくると共に、アプリケーションロジックとレイテンシーを加える必要が出てくるかもしれません。NoSQL データベースを使用した方が、このパーティション化は容易に行えるかもしれません。

さらに可用性を向上させるには、 Active-Active 複数の AWS リージョンにまたがるアプローチ。ワークロードは、 リージョン内で 静的に安定しているすべての希望するリージョンにデプロイされます (したがって、残りのリージョンは 1 つのリージョンが失われても負荷を処理できます)。A ルーティング 層は、トラフィックを正常な状態の地理的ロケーションに向け、そのロケーションに異常があれば自動的に宛先を変更したり、データレプリケーション層を一時的に停止させたりします。Amazon Route 53 では 10 秒間隔でヘルスチェックを行っており、TTL を最低 1 秒に設定することも可能です。

リソースをモニタリングする

スリーアンドハーフナインのシナリオと同じです。さらに、リージョンが異常であると検出され、トラフィックがそのリージョンからルーティングされた場合にアラームが送信されます。

需要の変化に対する適応方法

スリーアンドハーフナインのシナリオと同じです。

変更の実装

デプロイメントパイプラインには、パフォーマンス、負荷、障害注入テストなどの一連の完全なテストが含まれます。アップデート時には、Canary デプロイまたはブルーグリーンデプロイを用いて、分離された各ゾーンに対し1箇所ずつ順番にアップデートをデプロイし、1つのリージョンが完了してから別のリージョンで開始します。このデプロイを行う間は、ロールバックを高速化するために、旧バージョンが引き続きインスタンスで実行されます。これらの作業は、KPI に問題があった場合のロールバックを含め、完全に自動化されています。モニタリングは、問題発生時のアラートだけでなく、オペレーションが成功したときのメトリクスも対象です。

ランブックは、厳密なレポート要件とパフォーマンス追跡のためのものです。成功したオペレーションが、パフォーマンスまたは可用性の目標を達成できない傾向がある場合は、プレイブックを使用してトレンドの原因を特定します。プレイブックは、未発見の障害モードやセキュリティインシデントのためのものです。また、障害の根本原因を明らかにするためのプレイブックもあります。

ウェブサイトを構築するチームは、ウェブサイトの運用も行っています。このチームは、想定外の障害が発生した場合にその対応方法を決定し、実装後に適用する修正の優先順位付けをします。私たちは、Infrastructure Event Management の AWS Support とも連携しています。

データのバックアップ方法

スリーアンドハーフナインのシナリオと同じです。

弾力性のためのアーキテクト

このアプリケーションは、ソフトウェア / アプリケーションの回復パターンを使用して構築する必要があります。求められる可用性を実現するため、さらに多くのルーティングレイヤーが必要となる可能性があります。この実装の追加による複雑性は、過小評価してはいけません。このアプリケーションは障害が伝搬しないよう分離されデプロイされた各ゾーンに実装され、パーティション化してデプロイされ、顧客にリージョン規模の障害からの影響が顧客に及ばないようにしています。

回復力をテストする方法

私たちは、手順を逸脱することなくタスクを実行できるように、ランブックを使用しながら、ゲームデーを通してアーキテクチャを検証します。

災害対策 (DR) を計画する

Active-Active 完全なワークロードインフラストラクチャとデータが複数のリージョンにある、アクティブ/アクティブマルチリージョンデプロイ。ローカル読み取り、グローバル書き込みの戦略を使用して、1 つのリージョンがすべての書き込みのプライマリデータベースになり、他のリージョンへの読み取り用にデータがレプリケートされます。プライマリ DB リージョンに障害が発生した場合、新しい DB を昇格させる必要があります。ローカル読み取り、グローバル書き込みには、DB 書き込みが処理されるホームリージョンに割り当てられたユーザーがいます。これにより、ユーザーは任意のリージョンから読み書きできますが、異なるリージョンでの書き込み間で発生する可能性のあるデータの競合を管理するには、複雑なロジックが必要です。

リージョンが異常ありと検出された場合、ルーティングレイヤーはトラフィックを残りの正常なリージョンに自動的にルーティングします。手動による介入は必要ありません。

データストアは、潜在的な競合を解決できる方法でリージョン間のレプリケートを行う必要があります。レイテンシーの理由から、パーティション間でデータをコピーまたは移動して各パーティション内のリクエストまたはデータ量のバランスをとるために、ツールおよび自動化プロセスを作成する必要があります。データ競合解決のための修正には、運用のためのランブックも追加する必要があります。

可用性の設計目標

すべての復旧作業の自動化に多額の投資をすると、復旧は 1 分以内に完了すると想定しています。手動による復旧は想定しておらず、四半期ごとに最大 1 回の自動復旧があると想定しています。この場合は障害から復旧するまで 4 分かかることになります。アプリケーションはオンラインで常にアップデートされていると想定しています。これに基づくと、 可用性の設計目標 は 99.999% です。

要約

トピック	導入
リソースをモニタリングする	AWS リージョンレベルでのDNSのヘルスチェックやKPIを含む全レイヤーでのヘルス

トピック	導入
	チェック、設定したアラームが作動した場合のアラート送信、すべての障害に対するアラート通知。傾向を検知し、目標設計を管理するために、運用ミーティングを厳格に実施。
需要の変化に対する適応方法	ウェブと自動スケーリングアプリケーション層の ELB。 Aurora RDSのアクティブまたはパッシブリージョンでの、複数のゾーンでのストレージとリードレプリカの自動スケーリング。静的安定性を実現するために AWS リージョン間で同期されたデータとインフラストラクチャ。
変更の実装	Canary またはブルー/グリーンによる自動デプロイと、KPI またはアラートによってアプリケーションに未検出の問題があることが示された場合の自動ロールバック。デプロイは、一度に 1 つの AWS リージョンの 1 つの分離ゾーンに対して行われます。
データのバックアップ方法	RPO の要件を満たすための RDS を使用した AWS リージョンごとの自動バックアップと、ゲームデ–で定期的に実践している自動復元。Aurora RDS および S3 データは、アクティブリージョンからパッシブリージョンに自動的かつ非同期的にレプリケートされます。
弾力性のためのアーキテクト	アプリケーションの障害切り離しゾーンを実装。Auto Scaling による自己修復可能なウェブとアプリケーション層の提供。Multi-AZ の RDS。リージョンのフェイルオーバーの自動化。

トピック	導入
回復力をテストする方法	コンポーネントと分離ゾーンの障害のテストをゲームデーに定期的に運用スタッフと一緒にパイプラインで実践。不明な問題の診断のためのプレイブックと根本原因分析プロセスが存在。問題の内容とその修正方法または予防方法の通信経路。即時の実装とデプロイのために、RCA の修正を機能リリースよりも優先。
災害対策 (DR) を計画する	少なくとも 2 つのリージョンにデプロイされたアクティブ/アクティブ。インフラストラクチャは完全にスケーリングされ、リージョン間で静的に安定しています。データはリージョン間でパーティション分割され、同期されます。RDS 経由の暗号化されたバックアップ。リージョンの障害はゲームデーで実施され、AWS と連携します。復元中に、新しいプライマリデータベースへの昇格が必要になる場合があります。

リソース

ドキュメント

- Amazon Builders' Library - Amazon がソフトウェアを構築および運用する方法
- AWS アーキテクチャセンター

ラボ

- AWS Well-Architected 信頼性ラボ

外部リンク

- アダプティブキューイングパターン: Fail at Scale

- Availability and Beyond: Understanding and improving the resilience of distributed systems on AWS (可用性を超えて: AWS での分散システムの回復力の理解と向上)

本

- Robert S.Hammer「Patterns for Fault Tolerant Software」
- Andrew Tanenbaum と Marten van Steen「分散システム: 原則とパラダイム」

まとめ

可用性と信頼性のトピックに詳しくない方も、ミッションクリティカルなワークロードの可用性を最大化するインサイトを求めている経験豊富な方も、このホワイトペーパーによってお客様の考えを深めたり、新しいアイデアを導き出したり、新しい質問につなげたりしていただければ光栄です。これにより、ビジネスのニーズに基づいた適切な可用性のレベルと、それを実現するための信頼性を設計する方法について理解が深まることを願います。ここで提供されている設計、運用、復旧に関するレコメンデーションや、AWS ソリューションアーキテクトの知識と経験を活用することを推奨します。特に AWS で高レベルの可用性を達成したお客様の成功事例についてなど、ご意見やご感想をお待ちしております。アカウントチームにお問い合わせいただくか、 当社のウェブサイトからお問い合わせください。 。

寄稿者

本書の寄稿者は次のとおりです。

- Seth Eliot、プリンシパルデベロッパーアドボケイト、Amazon Web Services
- Mahanth Jayadeva、Well-Architected ソリューションアーキテクト、Amazon Web Services
- Amulya Sharma、プリンシパルソリューションアーキテクト、Amazon Web Services
- Jason DiDomenico、Cloud Foundations シニアソリューションアーキテクト、Amazon Web Services
- Marcin Bednarz、プリンシパルソリューションアーキテクト、Amazon Web Services
- Tyler Applebaum、シニアソリューションアーキテクト、Amazon Web Services
- Rodney Lester、App Modernization プリンシパルソリューションアーキテクト、Amazon Web Services
- Joe Chapman、シニアソリューションアーキテクト、Amazon Web Services
- Adrian Hornsby、プリンシパルシステム開発エンジニア、Amazon Web Services
- Kevin Miller、S3 バイスプレジデント、Amazon Web Services
- Shannon Richards、プリンシパルテクニカルプログラムマネージャー、Amazon Web Services
- Laurent Domb、チーフテクノロジスト - Fed Fin、Amazon Web Services
- Kevin Schwarz、シニアソリューションアーキテクト、Amazon Web Services
- Rob Martell、プリンシパルクラウドレジリエンスアーキテクト、Amazon Web Services
- Amazon Web Services、Senior Solutions Architect Manager DR、Priyam Reddy
- Amazon Web Services、プリンシパルテクノロジスト、Jeff Ferris
- Amazon Web Services、シニアソリューションアーキテクト、Matias Battaglia

その他の資料

詳細については、次の資料を参照してください。

- AWS Well-Architected Framework
- AWS アーキテクチャセンター

改訂履歴

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 フィードを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	新しい実装ガイダンスを使用してベストプラクティスを更新。	June 27, 2024
ベストプラクティスガイダンスの更新	ベストプラクティスを更新し、以下の領域に関する新しいガイダンスを追加。障害を防ぐように分散システムでの操作を設計する、障害を軽減または障害に耐えるように分散システムでの操作を設計する、ワークロードリソースをモニタリングする、需要の変化に適応するようにワークロードを設計する、変更の実装、テストの信頼性。	December 6, 2023
ベストプラクティスガイダンスの更新	ベストプラクティスを更新し、以下の領域に関する新しいガイダンスを追加。ワークロードリソースをモニタリングする、コンポーネントの障害に耐えられるようにワークロードを設計する。	October 3, 2023
ベストプラクティスガイダンスの更新	ベストプラクティスを更新し、以下の領域に関する新しいガイダンスを追加。ワークロードサービスアーキテクチャを設計する、障害を軽減	July 13, 2023

	または障害に耐えるように分散システムでの操作を設計する、ワークロードリソースをモニタリングする。	
マイナーな更新	インクルーシブでない表現を削除。	April 13, 2023
新しいフレームワークの更新	規範ガイダンスを使用してベストプラクティスを更新、および新しいベストプラクティスを追加。	April 10, 2023
ホワイトペーパーの更新	新しい実装ガイダンスを使用してベストプラクティスを更新。	December 15, 2022
マイナーな更新	図の番号を修正し、全体を通してマイナーな変更。	November 17, 2022
ホワイトペーパーの更新	ベストプラクティスに加筆し、改善計画を追加。	October 20, 2022
ホワイトペーパーの更新	「障害部分を切り離してワークロードを保護する」および「コンポーネントの障害に耐えられるようにワークロードを設計する」セクションの信頼性の柱に 2 つの新しいベストプラクティスを追加。	May 5, 2022
マイナーな更新	イントロダクションに持続可能性の柱を追加。	December 2, 2021

ホワイトペーパーの更新	ディザスタリカバリのガイダンスを更新して、Route 53 Application Recovery Controller を追加DevOps Guru への参照を追加。いくつかのリソースリンクの更新と編集上のマイナー変更。	October 26, 2021
マイナーな更新	AWS Fault Injection Service (AWS FIS) に関する情報を追加。	March 15, 2021
マイナーな更新	マイナーなテキストの更新。	January 4, 2021
ホワイトペーパーの更新	付録 A を更新して、Amazon SQS、Amazon SNS、Amazon MQ の可用性設計目標を更新。 テーブルの行を見やすく並べなおし。可用性とディザスタリカバリの違いと、それらの回復力への貢献の説明を改善。マルチリージョンアーキテクチャ (可用性) とマルチリージョン戦略 (ディザスタリカバリ) の範囲を拡大。参照書籍を最新版に更新。可用性の計算を拡張して、リクエストベースの計算とショートカット計算を加筆。ゲームデーの説明を改善。	December 7, 2020
マイナーな更新	付録 A を更新し、AWS Lambda の可用性設計の目標を更新	October 27, 2020

マイナーな更新	付録 A を更新し、AWS Global Accelerator の可用性設計の目標を追加	July 24, 2020
新しいフレームワークの更新	大幅な更新とコンテンツの新規追加/改訂を次のとおり実施:「ワークロードアーキテクチャ」のベストプラクティスセクションを追加、ベストプラクティスを「変更管理」セクションと「障害管理」セクションに再編成、リソースを更新、最新の AWS リソースおよびサービス (AWS Global Accelerator、AWS Service Quotas、AWS Transit Gateway など) が盛り込まれるように更新、信頼性、可用性、回復力の定義の追加/更新、Well-Architected レビューに使用される AWS Well-Architected Tool (質問とベストプラクティス) に沿うようホワイトペーパーを調整、設計原則を再整理、障害から自動的に復旧するを復旧手順をテストするの前に移動、図と等式のフォーマットを更新、「主なサービス」セクションを削除して、主な AWS のサービスの参照先をベストプラクティスに統合。	July 8, 2020
マイナーな更新	壊れたリンクを修正	October 1, 2019
ホワイトペーパーの更新	付録 A を更新	April 1, 2019

ホワイトペーパーの更新	具体的な AWS Direct Connect ネットワーク推奨事項とサービス設計目標を追加	September 1, 2018
ホワイトペーパーの更新	設計の原則と制限管理のセクションを追加。リンク更新、アップストリーム/ダウンストリームの不明瞭な用語を削除、信頼性の柱の残りのトピックの可用性のシナリオに明示的な参照を追加。	June 1, 2018
ホワイトペーパーの更新	DynamoDB クロスリージョンソリューションを DynamoDB Global Tables に変更しました。サービス設計目標を追加	March 1, 2018
マイナーな更新	可用性の計算を微修正してアプリケーションの可用性を追加	December 1, 2017
ホワイトペーパーの更新	高可用性設計に関するガイダンスを更新し、概念、ベストプラクティス、実装例を追加。	November 1, 2017
初版発行	信頼性の柱 - AWS Well-Architected フレームワークを発行しました。	November 1, 2016