信頼性の柱

AWS Well-Architected フレームワーク

2020 年 4 月

Archived

This paper has been archived.

The latest version is now available at:

https://docs.aws.amazon.com/ja_jp/wellarchitected/latest/reliability-pillar/welcome.html

注意

お客様は、この文書に記載されている情報を独自に評価する責任を負うものとします。本書は、(a) 情報提供のみを目的としており、(b) AWS の現行製品と慣行ついて説明していますが、予告なしに変更されることがあり、(c) AWS およびその関連会社、サプライヤーまたはライセンサーからの契約上の義務や保証をもたらすものではありません。AWS の製品やサービスは、明示または暗示を問わず、一切の保証、表明、条件なしに「現状のまま」提供されます。お客様に対する AWS の責任は、AWS 契約により規定されます。本書は、AWS とお客様の間で締結される一切の契約の一部ではなく、その内容を修正することもありません。

© 2020 Amazon Web Services, Inc. or its affiliates.All rights reserved.

Archived

目次

はじめに 1

信頼性 2

設計の原則 2

定義 3

可用性のニーズを理解する 9

基盤 10

サービスクォータの管理と制約 11

ネットワークトポロジを計画する 14

ワークロードのアーキテクチャ 21

お客様のワークロードサービスアーキテクチャを設計する 22

障害を防ぐために分散システムでの操作を設計する 26

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 31

変更管理 39

ワークロードリソースをモニタリングする 39

需要の変化に適応するようにワークロードを設計する 46

変更の実装 50

障害の管理 56

データのバックアップ方法 57

障害部分を切り離してワークロードを保護する 60

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 68

Archived

テストの信頼性 75
災害対策 (DR) を計画する 80
可用性目標の実装例 84
依存関係の選択 85
単一リージョンのシナリオ 86
複数リージョンのシナリオ 97
まとめ 109
寄稿者 109
その他の資料 110
ドキュメント改訂履歴 110
付録 A: 一部の AWS のサービスの可用性設計 112

Archived

要約

本書は、AWS Well-Architected フレームワークの信頼性の柱に焦点を当てています。本書は、お客様がアマゾン ウェブ サービス (AWS) の環境の設計、配信、メンテナンスにベストプラクティスを適用するうえで役立つガイダンスを提供します。

Archived

はじめに

AWS Well-Architected フレームワークは、AWS でワークロードを構築する際に行う判断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用いることで、クラウド上で信頼性が高く、安全で、効率的で、費用対効果の高いワークロードを設計および運用するための、アーキテクチャに関する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 フレームワークは下記の 5 つの柱を基本としています。

- 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

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。

本書は、最高技術責任者 (CTO)、アーキテクト、開発者、および運用チームメンバーなどの技術担当者を対象としており、これを読むことにより、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

Archived

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

信頼性の柱は、ワークロードが意図した機能を期待どおりに正しく一貫して実行する能力を包含します。これには、そのライフサイクル全体を通じてワークロードを稼働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、詳細なベストプラクティスのガイダンスを提供します。

設計の原則

クラウドでは、信頼性の向上に役立つ多くの原則があります。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べきです。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、テストが実施されます。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。自動化により、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。このアプローチは、実際の障害シナリオが発生する前にテストして修正できるので、障害経路が明らかになり、リスクが軽減されます。

Archived

- **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高める**: 1 つ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させることで、共通の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 (よくサービス妨害攻撃の目標となります)。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。制限はまだありますが、いくつかのクオータは制御でき、そのほかのクオーターも管理できます (サービスクォータと制約の管理を参照)。
- **自動化で変更を管理する:** インフラストラクチャの変更は自動化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
- ワークロードのアーキテクチャ
- 変更の管理
- 障害の管理

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

Archived

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。

弾力性、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、その主なものは弾力性です。

- **弾力性**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、悪意のある行為者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を消費するか、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。

弾力性は、このホワイトペーパーにおける主な焦点です。

Archived

他の 4 つの要素も重要であり、AWS Well-Architected フレームワークのそれぞれの柱によってカバーされています。ベストプラクティスでそれらに触れますが、本書の焦点はあくまで弾力性です。

可用性

可用性 (サービスの可用性とも呼ばれます) は、信頼性を定量的に測定するために一般的に使用されるメトリクスです。

- **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。AWS では可用性を以下のように定義します。

- $Availability = \frac{Available\ for\ Use\ Time}{Total\ Time}$
- 一定期間 (通常は 1 年) の稼働時間の割合 (例: 99.9%)
- 一般的には「9 の個数」で省略して表現され、たとえば「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、最初の箇条書きの定義の式にある「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) を除外するケースもあります。ただし、実際にはこのダウンタイム中にサービスを使用したいユーザーがいる可能性もあるため、これは誤った解釈です。

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。

Archived

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、データ抽出、転送、ジョブのロード
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	動画配信、ブロードキャストのワークロード
99.999%	5 分	ATM トランザクション、通信のワークロード

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。

$$Avail_{workload} = Avail_{invok} \times Avail_{dep1} \times Avail_{dep2}$$

$$99.99\% \times 99.99\% \times 99.99\% = 99.97\%$$

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 (冗長性の高いアベイラビリティーゾーンなど) 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製品の障害率を引いたものになります。たとえば、あるシステムが 2 つの独立

Archived

したコンポーネントから構成されており、それぞれ 99.9% の可用性を持つ場合、システム全体の可用性は 99.999% になります。

$$Avail_{workload} = Avail_{MAX} - \left(\left(100\% - Avail_{dependency}\right) \times \left(100\% - Avail_{dependency}\right)\right)$$

$$100\% - (0.1\% \times 0.1\%) = 99.9999\%$$

しかし、依存するシステムの可用性が不明である場合はどうしたらよいでしょうか?

依存するシステムの可用性を計算する。 依存関係のあるシステムには、多くの AWS のサービスにおける可用性の設計目標など、可用性についてのガイダンスを開示しているものがあります。(付録 A:「一部の AWS のサービスの可用性設計」を参照してください)。しかしここに情報がない場合 (たとえば、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。

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

たとえば、MTBF が 150 日で MTTR が 1 時間の場合、可用性の推測値は 99.97% です。

詳細については、可用性の計算に役立つこのドキュメント (システム全体の可用性の計算)を参照してください。

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

Archived

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。

目標復旧時間 (RTO) と目標復旧時点 (RTO)。

これらの用語は災害対策 (DR) に関連していることがほとんどで、災害時にワークロードの可用性を回復するための目標と戦略がセットになっています。

目標復旧時間 (RTO) は、組織のミッションまたはミッション/ビジネスプロセスに悪影響が及ぶ前の、ワークロードのコンポーネントが復旧フェーズにあるため利用できない全体的な時間の長さを表します。

目標復旧時点 (RPO) は、組織のミッションまたはミッション/ビジネスプロセスに悪影響が及ぶ前の、ワークロードのデータが利用できなくなる可能性がある全体的な時間の長さを表します。

可用性がワークロードデータに左右される最も一般的なケースでは、RPO は必ず RTO より短くなります。

Archived

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。

可用性のニーズを理解する

アプリケーションの可用性を、アプリケーション全体の単一ターゲットと始めに考えてしまうことがよくあります。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。

推奨事項
各アプリケーションに固有の局面を厳密に評価し、必要に応じてビジネスのニーズに合わせて可用性の設計目標を変えてください。

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。たとえば、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/書き込み操作はすべてデータプレーンによる操作です。逆

Archived

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。

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の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ができるアプリケーション開発の経験が豊富にあります。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 を導入すれば、私たちが可用性の目標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

基盤に必要な条件は、その範囲が単一のワークロードまたはプロジェクトを超えて広がります。システムを設計する前に、信頼性に影響を与える基盤となる要件を満たしておく必要があります。たとえば、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。

Archived

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。けれども AWS では、このようは基盤となる要件のほとんどがすで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。

- サービスクォータの管理と制約
- ネットワークトポロジのプロビジョニング

サービスクォータの管理と制約

クラウドベースのワークロードアーキテクチャには、 サービスクォータというものがあります (サービスの制限とも言います)。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ために 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。

サービスクォータと制約を把握する: ワークロードアーキテクチャのデフォルトのクォータとクォータ引き上げリクエストを把握しておきましょう。さらに、ディスクやネットワークなど、どの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。

Archived

Service Quotas は、100 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 コンソールまたは AWS SDK からクォータの引き上げをリクエストしたり、追跡したりできます。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オータを表示する、サービスクオータチェックを提供しています。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。たとえば、「Amazon VPC クォータ」を参照してください。スロットルされた API のレート制限は、API Gateway 内で使用量プランを変更することで設定できます。各サービスの設定で変更できるその他の制限には、プロビジョンド IOPS、割り当てられた RDS ストレージ、および EBS ボリューム割り当てなどがあります。Amazon Elastic Compute Cloud (Amazon EC2) には、インスタンス、Amazon Elastic Block Store (Amazon EBS)、Elastic IP アドレス制限の管理に役立つ独自のサービス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 サポートに連絡し、緩和策の有無についてお問い合わせください。

アカウントとリージョンをまたいでクォータを管理する: 複数の AWS アカウントまたは AWS リージョンをご利用の場合は、必ず本番ワークロードを実行するすべての環境で必要十分なクォータをリクエストしてください。

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。

設計段階で不変のサービスクォータと制約に適応させる：サービスクォータと物理リソースには変更できないものもあることに注意し、これらが信頼性に影響を及ぼさないように設計しましょう。

Archived

たとえば、ネットワーク帯域幅、AWS Lambda ペイロードサイズ、API Gateway のスロットルバーストレート、Amazon Redshift クラスターへの同時ユーザー接続数などは変更できません。

クォータを監視、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しょう。

サポートされているサービスの場合、CloudWatch アラームを設定して使用量をモニタリングし、クォータのしきい値に近づいていることのアラームを受けることで、クォータを管理できます。このアラームは、サービスクォータまたは Trusted Advisor からトリガーできます。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および抽出し、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。

クォータの管理を自動化する: しきい値に近づいているときにアラート通知するツールを実装しましょう。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的に行うことができます。

お使いの Configuration Management Database (CMDB) またはチケット発行システムをサービスクォータと統合すると、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 に加えて、サービスクォータの引き上げは AWS コマンドラインツールを使って自動化できます。

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余裕があることを確認する: リソースにエラーが発生しても、リソースが正常に終了するまで、クォータに基づいて計算される可能性があります。エラーが生じたリソースが停止されるまで、エラーが生じたすべてのリソースと代替リソースの合計リソース数がクォータ内に収まるようにします。この開きを計算するときは、アベイラビリティーゾーンの不具合を考慮する必要があります。

Archived

リソース

動画

- AWS Live re:Inforce 2019 - サービスクォータ

ドキュメント

- サービスクォータとは?
- AWS サービスクォータ (以前は「サービスの制限」と呼ばれていました)
- Amazon EC2 のサービスの制限
- AWS Trusted Advisor による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 Answers の AWS Limit Monitor
- AWS Marketplace: 制限の追跡に役立つ CMDB 製品
- APN パートナー: 設定管理を支援できるパートナー

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。

Archived

Amazon Virtual Private Cloud (Amazon VPC) を使用すると、AWS クラウドに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し、仮想ネットワークで AWS リソースを起動することが可能になります。

ワークロードのパブリックエンドポイントには可用性の高いネットワーク接続を使用する: このようなエンドポイントとそれへのルーティングは、可用性が高いネットワークを用いる必要があります。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 (CDN)、API ゲートウェイ、負荷分散やリバースプロキシを使用します。

Amazon Route 53、AWS Global Accelerator、Amazon CloudFront、Amazon API Gateway、Elastic Load Balancing (ELB) はすべて、可用性の高いパブリックエンドポイントを提供します。また、負荷分散とプロキシについて、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。

ワークロードで提供されるサービスの消費者は、エンドユーザーであろうと他のサービスのユーザーであろうと、これらのサービスエンドポイントでリクエストを行います。可用性の高いエンドポイントを提供するために利用できる AWS のリソースがいくつかあります。

Elastic Load Balancing は、アベイラビリティーゾーン全体の負荷分散を提供しています。これはレイヤー 4 (TCP) またはレイヤー 7 (http/https) のルーティングを行うことができ、AWS WAF と統合しています。また、AWS Auto Scaling との統合によって、自己修復可能なインフラストラクチャを構築することが可能で、トラフィックが増加したときはその状態を緩和し、トラフィックが減少したときはリソースを解放します。

Amazon Route 53 は、スケーラブルで可用性の高いドメインネームシステム (DNS) サービスで、Amazon EC2 インスタンス、Elastic Load Balancing ロードバランサー、Amazon S3 バケットなどの AWS で実行するインフラストラクチャとユーザーリクエストを効率的に接続します。これはユーザーを AWS の外部のインフラストラクチャにルーティングするためにも使用できます。

Archived

AWS Global Accelerator は、AWS グローバルネットワーク上で最適なエンドポイントにトラフィックを送信するために使用できるネットワークレイヤーサービスです。

分散型サービス妨害 (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。AWS Shield は、追加費用なしでワークロード上の AWS のサービスエンドポイントをこのような攻撃から自動的に保護します。これらの機能は、APN Partners や AWS Marketplace の仮想アプライアンスを使って、ニーズに合わせて拡張することができます。

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: 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 (DX) 接続または VPN トンネルを使用しましょう。高可用性を実現するには、複数の DX ロケーションを使用します。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てください。VPN を終端する AWS Marketplace アプライアンスを評価したい場合があります。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。

Direct Connect は、オンプレミス環境から AWS への専用ネットワーク接続を簡単に確立できるようにするクラウドサービスです。Direct Connect ゲートウェイを使用すると、オンプレミスのデータセンターを複数の AWS リージョンにまたがる複数の AWS VPC に接続できます。

この冗長性は、接続の回復力に影響を与える可能性のある障害に対処します。

- トポロジの障害に対する耐障害性を備えるにはどうしたらよいですか?
- 何かを誤設定して接続を削除するとどうなりますか?
- トラフィックとサービス利用量が予想外に増加した場合に対応できますか?
- 分散型サービス拒否 (DDoS) 攻撃の影響を緩和できますか?

VPC を VPN 経由でオンプレミスのデータセンターに接続するときは、ベンダーとそのアプライアンスを実行する際に必要となるインスタンスサイズを選定する際に、弾力性と必要とする帯域幅の要件を考慮する必要があります。使用する VPN アプライアンスがその実装において十

Archived

分な弾力性がない場合、2 つ目のアプライアンスを通じて冗長接続を設定する必要があります。すべてのシナリオにおいて、許容可能な復旧時間を定義し、その要件を満たすかどうかをテストする必要があります。

Direct Connect 接続を使用して VPC をデータセンターに接続することにし、この接続の可用性を高める必要がある場合は、各データセンターごとに冗長化した DX 接続を使用します。冗長接続では、最初の接続とは異なる場所から 2 番目の DX 接続を使用する必要があります。複数のデータセンターがある場合は、接続が異なる場所で終端するようにしてください。これをセットアップするには、Direct Connect Resiliency Toolkit をご利用ください。

AWS VPN を使用してインターネット経由の VPN にフェイルオーバーする場合、VPN トンネルごとに最大 1.25Gbps のスループットはサポートしますが、同じ VGW 上で終端する AWS VPN トンネルが複数ある場合、アウトバウンドトラフィック用の Equal Cost Multi Path (ECMP) はサポートしないということを理解しておくことが重要です。フェイルオーバー中に 1 Gbps 未満の速度を許容できないのであれば、Direct Connect 接続のバックアップとして AWS Managed VPN を使用することはお勧めしません。

VPC エンドポイントを使用して、VPC を、パブリックインターネットを経由せずに、サポートされている AWS のサービスおよび AWS PrivateLink が提供する VPC エンドポイントサービスにプライベートに接続することもできます。エンドポイントは仮想デバイスです。これは、水平にスケーリングされ、冗長性と可用性に優れた VPC コンポーネントです。仮想デバイスにより、ネットワークトラフィックに可用性のリスクや帯域幅の制約を課すことなく、VPC 内のインスタンスとサービス間の通信が可能になります。

IP サブネットを割り当てる際は拡張性と可用性を考慮する：個別の Amazon VPC の IP アドレス範囲は、将来の拡張やアベイラビリティーゾーン間でのサブネットへの IP アドレスの割り当てを考慮し、ワークロードの要件を満たすために十分な大きさが必要です。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。

Archived

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。

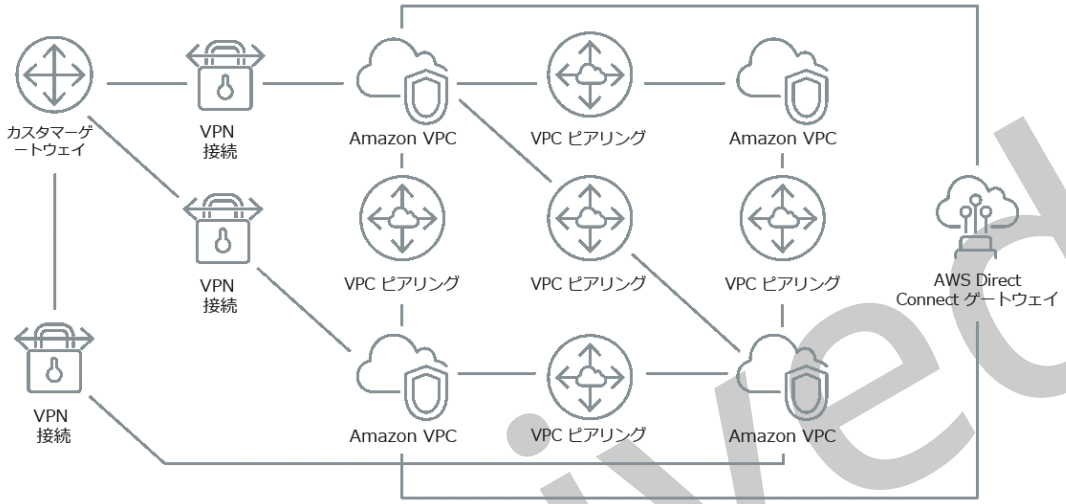
- リージョンごとに複数の VPC 用の IP アドレス空間を割り当てます。
- VPC 内で、複数のアベイラビリティーゾーンにまたがる複数のサブネット用の空間を割り当てます。
- 将来の拡張のために、常に未使用の CIDR ブロック空間を VPC 内に残しておきます。
- 機械学習用のスポットフリート、Amazon EMR クラスター、Amazon Redshift クラスターなど、使用する可能性のある EC2 インスタンスの一時的なフリートのニーズを満たす IP アドレス空間があることを確認します。
- 各サブネット CIDR ブロックの最初の 4 つの IP アドレスと最後の IP アドレスは予約されており、お客様はご使用いただけません。

大きな VPC CIDR ブロックのデプロイを計画する必要があります。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。最大規模の VPC (/16) をデプロイする場合、65,000 を超える IP アドレスが割り当てられることになります。ベース 10.x.x.x IP アドレス空間だけで、そのような VPC を 255 個プロビジョニングできます。したがって、VPC の管理を容易にするためには、小さすぎて失敗するよりも、大きすぎて失敗するほうが良いでしょう。

多対多メッシュよりもハブアンドスポークトポロジーを優先する: 3 つ以上のネットワークアドレス空間 (たとえば、VPC やオンプレミスネットワーク) が VPC ピア接続、AWS Direct Connect、または VPN を介して接続されている場合は、AWS Transit Gateway が提供するようなハブアンドスポークモデルを使用します。

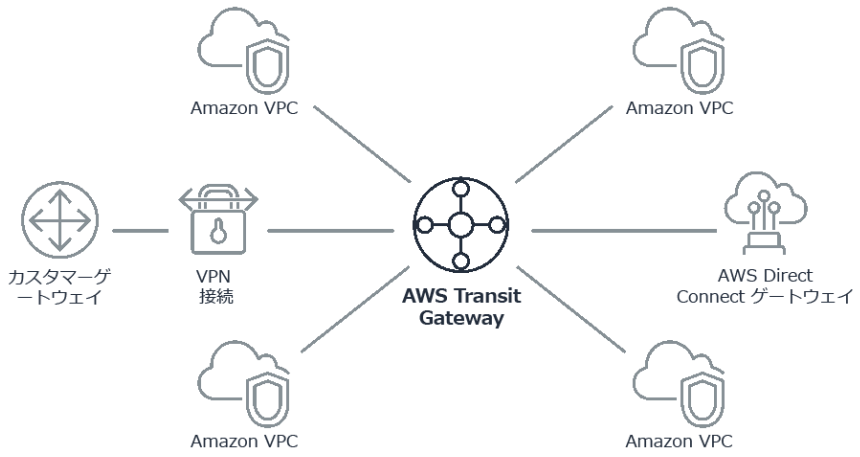
Archived

そのようなネットワークが 2 つしかない場合は、それらを互いに接続するだけで済みますが、ネットワークの数が増えるにつれて、そのようなメッシュ接続の複雑さは受容できないものになります。AWS Transit Gateway は、維持しやすいハブアンドスポークモデルを提供し、複数のネットワーク間でトラフィックをルーティングできるようにします。



AWS Transit Gateway なしの場合: *VPN* 接続を使用して、各 *Amazon VPC* を相互にピア接続し、オンサイトの各ロケーションをピア接続する必要があります。これは、拡張するにつれ複雑さが増す可能性があります。

Archived



AWS Transit Gateway ありの場合: 各 *Amazon VPC* または *VPN* を *AWS Transit Gateway* に接続するだけで、各 *VPC* または *VPN* との間でトラフィックをルーティングできます。

接続されているすべてのプライベートアドレス空間で、重複しないプライベート IP アドレス範囲を適用する：ピア接続または VPN 経由での接続の場合、各 VPC の IP アドレス範囲が重複しないようにしなければなりません。同様に、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する必要があります。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。

これには、IP アドレス管理 (IPAM) システムが役立ちます。いくつかの IPAM は AWS Marketplace で入手できます。

リソース

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)

Archived

ドキュメント

- What Is a Transit Gateway?
- What Is Amazon VPC?
- Direct Connect ゲートウェイの操作
- Direct Connect Resiliency Toolkit で使用を開始する方法
- Multiple data center HA network connectivity
- What Is AWS Global Accelerator?
- Using redundant Site-to-Site VPN connections to provide failover
- VPC Endpoints and VPC Endpoint Services (AWS PrivateLink)
- Amazon Virtual Private Cloud Connectivity Options Whitepaper
- AWS Marketplace for Network Infrastructure
- APN Partner: partners that can help plan your networking

ワークロードのアーキテクチャ

信頼性の高いワークロードは、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、5 つの Well-Architected の柱すべてにかけ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。

- 適切なサービスアーキテクチャを実装する
- 障害を防ぐために分散システムでソフトウェアを設計する
- 障害を軽減するために分散システムでソフトウェアを設計する

Archived

お客様の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。

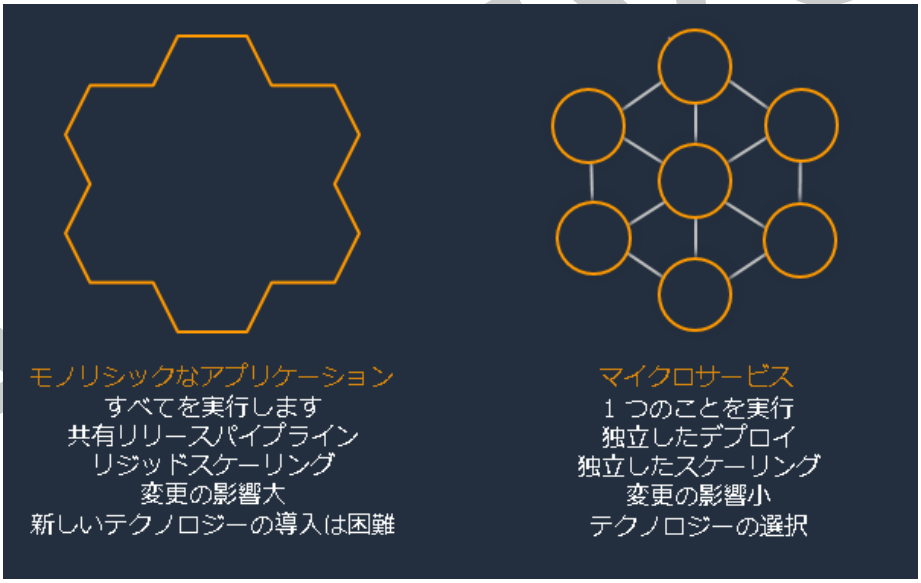
AWS では長く SOA を使用してきましたが、現在はマイクロサービスを使用してシステムを構築することを受け入れています。マイクロサービスには多くの魅力がありますが、可用性という観点から重要なのは、マイクロサービスが小さくてシンプルであることです。マイクロサービスでは、さまざまなサービスに求められる可用性を区別して、最も高い可用性ニーズを持つマイクロサービスに特化して投資を行うことができます。たとえば、Amazon.com で製品情報ページ (詳細ページ) を配信するには、ページの個別の部分を作成するために何百ものマイクロサービスが呼び出されます。製品と料金の詳細を表示するために不可欠なサービ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。

ワークロードの分割方法を選択する: モノリシックアーキテクチャは避けてください。モノリシックアーキテクチャではなく、SOA とマイクロサービスのどちらかを選択する必要があります。どちらかの選択を行うときは、複雑さに比例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。小さなセグメントを使用する利点には、俊敏性、組織の柔軟

Archived

性、スケーラビリティが高い点などがあります。複雑さにより、潜在的にレイテンシーが増加したり、デバッグが複雑化したり、運用上の負担が増加したりします。

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化を行います。これは、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が、特にマイクロサービスアーキテクチャをデプロイする場合は、特に考慮すべきトレードオフがあります。1 つ目は、分散コンピューティングアーキテクチャによって、ユーザーのレイテンシー要件の達成が困難になり、ユーザーの操作のデバッグとトレースが複雑化する可能性がある点です。AWS X-Ray を使ってこの問題の解決に役立てることもできます。考慮するべきもう 1 つの点は、管理するアプリケーションの数が増えるにつれて運用が複雑化することです。これには複数の独立コンポーネントのデプロイが伴います。

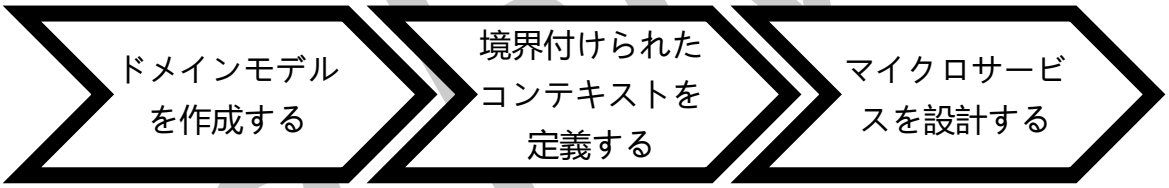


モノリシックアーキテクチャ対マイクロサービスアーキテクチャ

特定のビジネスドメインと機能に焦点を当てたサービスを構築する: SOA では、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構築できます。マイクロサービスはドメインモデルと境界付けられたコンテキストを使用してこれをさらに制限し、各サービスが *1*

つのことだけを実行するようにします。特定の機能に焦点を当てることで、さまざまなサービスの信頼性要件を差別化し、より具体的に的を絞って投資することができます。簡潔なビジネス上の問題と各サービスに関連付けられた小さなチームにより、組織のスケーリングも容易になります。

マイクロサービスアーキテクチャを設計する際は、ドメイン駆動設計 (DDD) を使用して、エンティティでビジネス上の問題をモデル化すると便利です。たとえば、Amazon.com のエンティティには、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。次に、モデルは、境界付けられたコンテキストを使ってさらに小さなモデルに分割され、類似の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。したがって、Amazon のサンプルパッケージを例として用いると、配送とスケジュールは発送コンテキストの一部となり、料金、割引、通貨は料金のコンテキストの一部となります。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。



API ごとにサービス契約を提供する: サービス契約は、サービス統合に関するチーム間で文書化した合意で、機械で読み取ることができる API 定義、レート制限、パフォーマンスの期待値が含まれます。バージョニング戦略により、クライアントは既存の 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。契約に違反しない限り、デプロイはいつでも行うことができます。サービスプロバイダーチームは、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。同様に、サービスコンシューマーは独自のテクノロジーを使用できます。

マイクロサービスは、SOA の概念を取り入れて、最小限の機能セットを備えたサービスを構築します。各サービスでは、サービスを使用するための API、設計目標、制限、その他の考慮事

Archived

項が公開されています。これがアプリケーションの呼び出しに関する「契約」となります。これには次のような 3 つのメリットがあります。

- 各マイクロサービスが対応すべきビジネス上の課題は簡潔なものであり、小さなチームでその課題に対処できる。これにより組織の拡大が可能となります。
- API の要件およびその他の「契約」の条件を満たしている限り、チームはいつでもデプロイできる。
- API の要件およびその他の「契約」の条件を満たしている限り、チームはあらゆる技術スタックを使うことができる。

Amazon API Gateway は、開発者があらゆる規模の API を簡単に作成、公開、維持、モニタリング、保護できるフルマネージドサービスです。Amazon API Gateway は、トラフィック管理、認可とアクセスコントロール、モニタリング、API バージョン管理など、最大数十万個の同時 API 呼び出しの受け入れと処理に伴うすべてのタスクを取り扱います。以前は Swagger 仕様として知られていた OpenAPI 仕様 (OAS) を使用して、API 契約を定義し、API Gateway にインポートできます。API Gateway を使用すると、API をバージョン管理してデプロイできます。

リソース

ドキュメント

- Amazon API Gateway:OpenAPI を使用して REST API を設定する
- Implementing Microservices on AWS
- AWS でのマイクロサービス

外部リンク

- Microservices - a definition of this new architectural term
- Microservice Trade-Offs
- 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)

Archived

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。

必要な分散システムの種類を特定する：ハードなリアルタイム分散システムでは、応答を同期的かつ迅速に行えるようにする必要がありますが、ソフトなリアルタイムシステムでは、応答に数分以上の寛大な時間枠があります。オフラインシステムは、バッチ処理または非同期処理を通じて応答を処理します。ハードなリアルタイム分散システムは、最も厳格な信頼性要件を持っています。

分散システムの最も困難な課題は、リクエスト/応答サービスとしても知られているハードなリアルタイム分散システムにあります。それを困難にしているのは、リクエストが前触れもなく送信され、直ちに応答しなくてはならないという点です (お客様がレスポンスを待っているなど)。この例には、フロントエンドウェブサーバー、オーダーパイプライン、クレジットカードトランザクション、すべての AWS API やテレフォニーなどがあります。

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疎結合です。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。

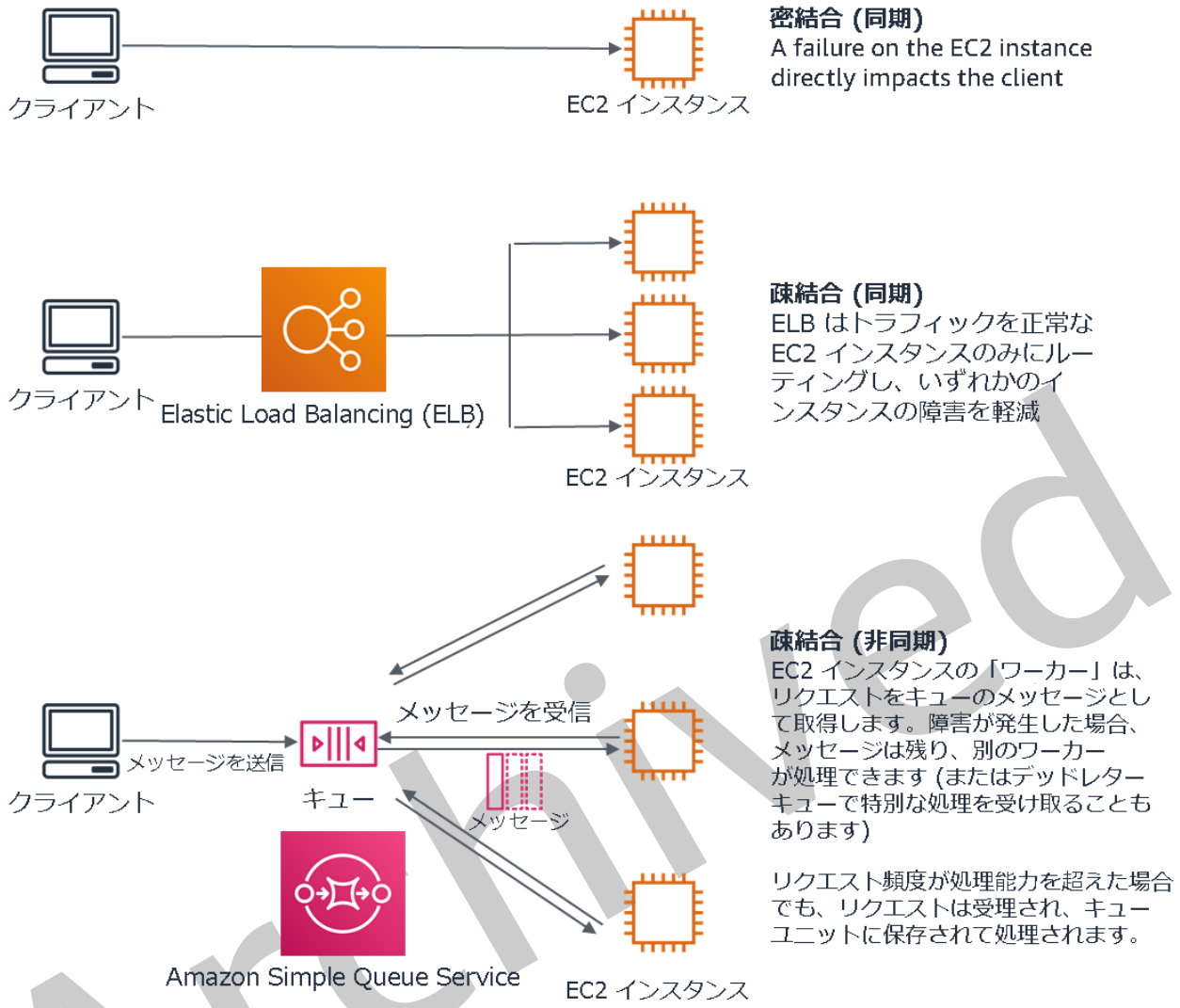
1 つのコンポーネントを変更すると、それに依存する他のコンポーネントも強制的に変更される場合、それらは密結合されています。疎結合はこの依存関係を壊すため、依存コンポーネントが知る必要があるのは、バージョン管理されて公開されたインターフェイスのみです。依存関係があるコンポーネント間に疎結合を実装すると、あるコンポーネントの障害が別のコンポーネントに影響を及ぼさないようにすることができます。

Archived

疎結合により、そのコンポーネントに依存する他のコンポーネントのリスクを最小限に抑えながら、コンポーネントにコードまたは機能を自由に追加できます。また、スケールアウトしたり、依存関係の基盤となる実装を変更したりできるため、スケーラビリティが向上します。

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、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、AWS Step Functions を通して統合されます。

Archived



Archived

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、クライアント (イベントプロデューサー) を、依存しているサービス (イベントコンシューマー) から抽象化できます。Amazon Simple Notification Service は、高スループットで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、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 (クライアントは停止し、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。

すべての応答にべき等性を持たせる: べき等なサービスは、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。べき等な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懸念がなくなるため、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のは困難です。つまり、アクションは 1 回だけ実行されるため、同一のリクエストを複数回行っても、リクエストを 1 回行ったのと効果は同じです。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。

動作を継続的に行う: 負荷が急激に変化すると、システム障害が発生することがあります。たとえば、何千台ものサーバーの状態をモニタリングするヘルスチェックシステムは、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であれば、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

Archived

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、またはその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常に作業を行っており、大きな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。実際、Amazon Route 53 ヘルスチェックのコントロールプレーンはこのように設計されています。

リソース

動画

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

ドキュメント

- CloudWatch メトリクスを発行する AWS のサービス
- Amazon Simple Queue Service とは?
- Amazon EC2: べき等性を保証する
- Amazon Builders' Library: 分散システムの課題
- 一元的ログ記録ソリューション
- AWS Marketplace: モニタリングとアラートに使用できる製品
- APN パートナー: モニタリングとログ記録を支援できるパートナー

Archived

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。

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: コンポーネントの依存関係が正常でない場合でも、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。たとえば、依存関係の呼び出しが失敗した場合は、代わりに事前定義された静的応答を使用します。

サービス A によって呼び出されたサービス B が、次にサービス C を呼び出すとします。

サービス *C* は、サービス *B* から呼び出されると失敗します。サービス *B* は、低下した応答をサービス *A* に返します。

サービス B がサービス C を呼び出すと、エラーまたはタイムアウトを受け取ります。サービス B は、サービス C (およびそれに含まれるデータ) からの応答がないため、返せるものを返します。これは、最後にキャッシュされた適切な値であるかもしれません。または、サービス B は、サービス C から受け取るはずだったものを所定の静的応答に置き換える可能性もありま

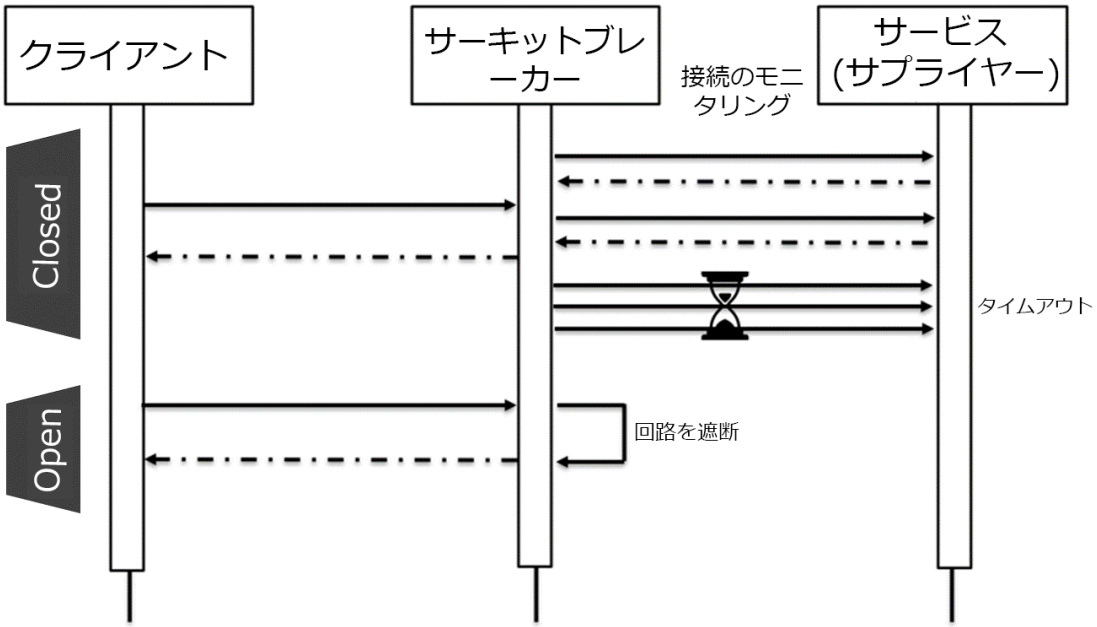
Archived

す。次に、低下した応答を呼び出し元のサービス A に返すことでしょう。この静的応答がない場合、サービス C で障害が発生すると、サービス B を介してサービス A にカスケードされ、可用性が失われます。

ハードな依存関係の可用性方程式の乗法的因子 (ハードな依存関係を使用した可用性の計算を参照) に従って、C の可用性が低下すると、B の有効な可用性に深刻な影響を与えます。静的応答を返すことにより、サービス B はサービス C の障害を軽減します (パフォーマンスは低下しますが)。これにより、サービス C の可用性が 100% であるかのように見せます (エラーがある状態で静的応答を確実に返すことを前提にすると)。静的応答は単純にエラーを返す代わりに行う手段で、別の手段を使って応答を再計算する試みではないことに注意してください。まったく異なるメカニズムで同じ結果を達成しようとする試みは、フォールバック動作と呼ばれ、回避すべきアンチパターンです。

グレイスフルデグラデーションの例は他にも、 サーキットブレーカーパターンがあります。障害が一時的な場合は、再試行戦略を用いるのがよいでしょう。障害が一時的ではなく、操作が失敗する可能性が高い場合、サーキットブレーカーパターンは、失敗する可能性が高いリクエストをクライアントが実行できないようにします。リクエストが正常に処理されると、サーキットブレーカーは閉じられ、リクエストは正常に流されます。リモートシステムがエラーを返し始めるか、レイテンシーが高くなると、サーキットブレーカーが開かれ、依存関係が無視されるか、結果的に返される応答は単純に取得されたが包括的ではない応答 (単なる応答キャッシュである場合もあります) に置き換えられます。システムは、依存性が回復したかどうかを判断するために、依存関係を定期的に呼び出そうとします。依存関係が確認できたら、サーキットブレーカーは閉じられます。

Archived



サーキットブレーカーが閉じた状態と開いた状態を示した図。

図に示されている閉じた状態と開いた状態に加えて、開いた状態で設定可能な期間が経過すると、サーキットブレーカーは半分開いた状態に移行することもあります。この状態では、通常よりはるかに低いレートで定期的にサービスを呼び出そうとします。このプローブは、サービスの状態を確認するために使用します。半開状態で何度か成功すると、サーキットブレーカーは閉じた状態に移行し、通常のリクエストフローが再開されます。

スロットルリクエスト: これは、予想外の需要の増加に対応するための軽減パターンです。一部のリクエストは受け入れられますが、定義された制限を超えるリクエストは拒否され、スロットルされたことを示すメッセージが返されます。クライアントの期待は、リクエストが戻されて放棄されるか、遅い速度で再試行することです。

お客様のサービスは、各ノードまたはセルが処理できる既知のリクエスト容量に合わせて設計する必要があります。これは負荷テストによって設定できます。リクエストの到着率をトラッキングし、到着率が一時的に制限を超えると、リクエストが適切にスロットリングされたことを示す応答があります。これによってユーザーは、適切な容量を持つ別のノードおよびセルで

Archived

再試行を行うことができます。Amazon API Gateway にはリクエストのスロットリングに対応したメソッドがあります。Amazon SQS と Amazon Kinesis は、リクエストをバッファリングし、リクエスト頻度を平準化し、非同期で対処できるリクエストのスロットリングの必要性を軽減します。

再試行呼び出しの制御と制限: 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。これらの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。

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。操作中に障害が発生すると、これらのコンポーネントのいずれかにエラーが発生し始める可能性があります。エラーを処理するデフォルトの手法は、クライアント側で再試行を行うことです。この手法により、アプリケーションの信頼性と可用性が向上します。ただし、再試行が大規模に行われた場合 (またエラーが発生してからすぐにクライアントが失敗した操作を再試行しようとすると)、ネットワークは、新しいリクエストと再試行されたリクエストですぐに飽和状態になり、それぞれがネットワーク帯域幅を奪い合うことになる可能性があります。これにより、 再試行の嵐が発生し、サービスの可用性が低下する可能性があります。このパターンは、システムが完全にダウンするまで続くかもしれません。

このようなシナリオを回避するには、一般的な**エクスポネンシャルバックオフ**などのバックオフアルゴリズムを使用する必要があります。エクスポネンシャルバックオフアルゴリズムは、再試行が行われる速度を徐々に下げて、ネットワークの輻輳を回避します。

多くの SDK およびソフトウェアライブラリ (AWS のものを含む) は、このようなアルゴリズムを持ったバージョンを実装しています。ただし、バックオフアルゴリズムが存在すると想定することはしないでください。必ずこれをテストして検証してください。

分散システムでは、すべてのクライアントが同時にバックオフし、再試行呼び出しのクラスターが発生する可能性があるため、単にバックオフするだけでは不十分です。Marc Brooker はそ

Archived

のブログ記事 Exponential Backoff And Jitter では、再試行呼び出しのクラスターを防ぐためにエクスポネンシャルバックオフの wait() 関数をどのように変更したらよいかを説明しています。wait() 関数に**ジッター**を加えることでこれを解決します。時間がかかり過ぎる再試行を行わないようにするには、実装ではバックオフを最大値に制限する必要があります。

最後に、再試行が失敗するまでの最大再試行回数または最長経過時間を設定することが重要です。AWS SDK はデフォルトでこれを実装しており、設定を変更することもできます。スタックの下位にあるサービスの場合、再試行の上限を 0 または 1 にするとリスクが緩和されますが、スタックの上位にあるサービスに再試行が委任されるため、効果的な方法です。

高速に失敗してキューを抑制する: ワークロードがリクエストに正常に応答できない場合は、すぐに失敗するようにします。これにより、リクエストに関連付けられたリソースを解放でき、リソースが不足した場合にサービスを復旧できます。ワークロードは正常に応答できるが、リクエスト頻度が高すぎる場合は、代わりにキューを使用してリクエストをバッファします。ただし、長いキューは許可しないでください。クライアントがすで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。

このベストプラクティスは、サーバー側、つまりリクエストのレシーバーに当てはまります。

キューはシステムの複数のレベルで発生する可能性があるため、応答が必要な新しいリクエストの前に (もはや応答を必要としない) 古い応答が処理されると、迅速に復旧する能力が著しく阻害される可能性があることに注意してください。キューがどこに存在するかに注意を払ってください。ワークフローや、データベースに記録された作業の中に隠れていることもよくあります。

クライアントタイムアウトを設定する: タイムアウトを適切に設定し、体系的に検証しましょう。デフォルト値は通常高く設定されているため、デフォルト値のままにしないでください

このベストプラクティスは、クライアント側、つまりリクエストの送信者に当てはまります。

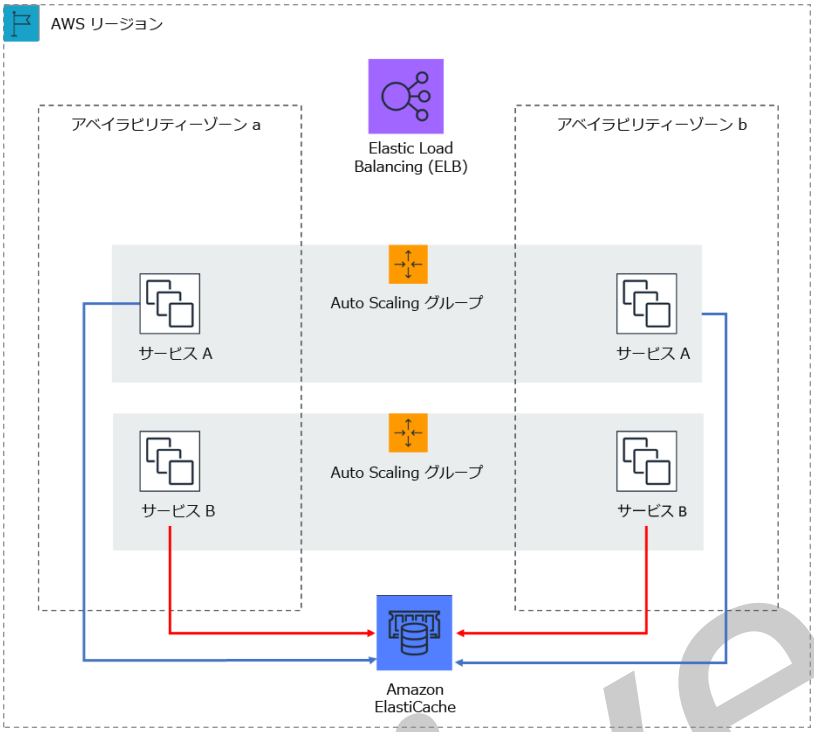
Archived

リモート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、プロセス全体のすべての呼び出しに一般的に行います。多くのフレームワークには組み込みのタイムアウト機能がありますが、その多くのデフォルト値は無限または高すぎるため、注意が必要です。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。

Amazon がタイムアウト、再試行、ジッターを伴うバックオフをどのように扱っているかの詳細については、「Builders' Library: ジッターを伴うタイムアウト、再試行、およびバックオフ」を参照してください。

可能な場合はサービスをステートレスにする: サービスは状態を必要としないか、状態をオフロードして、異なるクライアントリクエスト間でローカルにディスクまたはメモリに保存されたデータに依存しないようにする手もあります。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。Amazon ElastiCache または Amazon DynamoDB は、オフロードした状態に適した宛先です。

Archived



このステートレスウェブアプリケーションでは、セッション状態は *Amazon ElastiCache* にオフロードされます。

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。

ステートレスになるように設計したら、AWS Lambda や AWS Fargate などのサーバーレスコンピューティングプラットフォームを利用できます。

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。

緊急レバーを実装する: これは、ワークロードの可用性に対する影響を軽減できる可能性がある迅速なプロセスです。根本原因がなくても操作できます。緊急レバーは、完全に決定的なアク

Archived

ティブ化と非アクティブ化の基準を提供することにより、リゾルバーの認知負荷をゼロに減らせるものが理想的です。緊急レバーの例には、すべてのロボットトラフィックをブロックしたり、静的応答を行ったりすることが含まれます。緊急レバーは多くの場合手動ですが、自動化することもできます。

緊急レバーを実装して使用するためのヒント:

- 緊急レバーがアクティブになったら、実行数を増やすのではなく、減らす
- シンプルに保ち、バイモーダルな行動は避ける
- 緊急レバーを定期的にテストする

緊急レバーではないアクションの例を次に示します。

- 容量を追加する
- サービスに依存するクライアントのサービス所有者を呼び出して、呼び出しを減らすよう依頼する
- コードを変更してリリースする

リソース

動画

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

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
- 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
- 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 Builders' Library:分散システムでのフォールバックの回避

Archived

- Amazon Builders' Library:乗り越えられないキューバックログの回避
- Amazon Builders' Library:キャッシュの課題と戦略

ラボ

- Well-Architected ラボ: Level 300: Implementing Health Checks and Managing Dependencies to Improve Reliability

外部リンク

- CircuitBreaker(「Release It!」の本のサーキットブレーカーをまとめたもの)

本

- Michael Nygard「Release It! 本番環境対応ソフトウェアを設計およびデプロイする」

変更管理

ワークロードを信頼できる形で実行するには、ワークロードやその環境の変化は予測して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。

- 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

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。ログとメトリクスをモニタリングしたり、しきい値を超えた場合や重大なイベントが発生した場合に通知を送信したりするようにワークロードを設定できます。モニタリングにより、ワーク

Archived

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 つの明確な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

Archived

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。Amazon CloudWatch またはサードパーティ製ツールを使用して、ワークロードのコンポーネントをモニタリングします。AWS のサービスは、Personal Health Dashboard を使って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に作成しきい値を設定します。

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、ワークロードの複数のレイヤーに対する洞察を得てきました。

AWS では、豊富なモニタリング情報とログ情報を利用できるので、これを使って需要の変化プロセスを定義できます。以下は、ログとメトリクスデータを生成するサービスと機能の一部のリストです。

- Amazon ECS、Amazon EC2、Elastic Load Balancing、AWS Auto Scaling、および Amazon EMR は、CPU、ネットワーク I/O、およびディスク I/O の平均のメトリクスを公開しています。
- Amazon CloudWatch Logs は、Amazon Simple Storage Service (Amazon S3)、Classic Load Balancers、Application Load Balancers で有効にできます。
- VPC フローログを有効にして、VPC に出入りするネットワークトラフィックを分析できます。
- AWS CloudTrail は、AWS マネジメントコンソール、AWS SDK、コマンドラインツールを通じて実行されたアクションを含む、AWS アカウントアクティビティをログに記録します。
- Amazon EventBridge は、AWS のサービスの変更を示すシステムイベントのリアルタイムのストリームを提供します。

Archived

- AWS には、オペレーティングシステムレベルのログを収集し、CloudWatch Logs にストリームできるツールがあります。
- カスタム Amazon CloudWatch メトリクスは、あらゆるディメンションのメトリクスに使用できます。
- Amazon ECS と AWS Lambda はログデータを CloudWatch Logs にストリーミングします。
- Amazon Machine Learning (Amazon ML)、Amazon Rekognition、Amazon Lex、Amazon Polly は、成功または失敗したリクエストのメトリクスを提供しています。
- AWS IoT は、ルールの実行数のメトリクスと、ルールの成功と失敗に関するメトリクスを提供しています。
- Amazon API Gateway は、リクエストの数、誤ったリクエスト、API のレイテンシーに関するメトリクスを提供しています。
- Personal Health Dashboard はお客様が使用している 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や可用性について、各ユーザーに適した情報を表示します。

さらに、リモートのロケーションからすべての外部エンドポイントをモニタリングし、それらが基本の実装から独立していることを確認します。このアクティブなモニタリングは、合成トランザクション (「ユーザー Canary」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) で実行できます。合成トランザクションは、アプリケーションのコンシューマーが実行する一般的なタスクを定期的に実行します。合成トランザクションは短期間に保ち、テスト中にワークフローに負荷をかけすぎないようにしてください。Amazon CloudWatch Synthetics では、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。

Archived

集計 – メトリクスを定義して計算する: 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、主要な集約と保存のレイヤーとして機能します。AWS Auto Scaling や ELB などの一部のサービスでは、CPU 負荷やクラスターまたはインスタンス全体の平均リクエストレイテンシーについて、「そのまま」使用できるデフォルトメトリクスを提供しています。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、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。

リアルタイムの処理と警告 – 通知を送信する: 重大なイベントが発生したときに、それを把握しておく必要がある組織が通知を受信します。

アラートは Amazon Simple Notification Service (Amazon SNS) トピックにも送信することが可能で、任意の数のサブスクライバーにプッシュ送信できます。たとえば Amazon SNS では、E メールエイリアスにアラートを転送して、技術スタッフが対応できるようにしています。

リアルタイムの処理と警告 — 自動応答: 自動化を使用して、イベントが検出されたときに (故障したコンポーネントを交換するなどの) アクションを実行します。

アラートは、クラスターが需要の変化に対応できるように、AWS Auto Scaling イベントをトリガーします。アラートは Amazon Simple Queue Service (Amazon SQS) にも送信が可能で、これがサードパーティ製のチケットシステムとの統合ポイントとして機能します。アラートは AWS Lambda からも登録が可能で、ユーザーは変化に動的に反応できる非同期のサーバーレスモデルを使用できます。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、AWS Systems Manager Automation をトリガーして問題を修正できます。

Archived

ストレージと分析: 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、ログデータの分析に使用できるシンプルかつ強力なクエリ言語をサポートしています。Amazon CloudWatch Logs では、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。詳細については、Amazon Athena ユーザーガイドの「サポート対象の SerDes およびデータ形式」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。

データの集計、処理、保存、分析を実行できるパートナーやサードパーティ製のツールは数多くあります。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。

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行されているかを頻繁に確認し、重大なイベントと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。

Archived

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 では、インシデント発生時の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 の 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、運用パフォーマンスをレビューし、チーム間で学んだことを共有することを目的として、毎週ミーティングを開いています。AWS には非常に多くのチームが存在するため、レビューするワークロードをランダムに選択する The Wheel を作成しました。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。

システムを介して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: AWS X-Ray またはサードパーティ製のツールを使用することで、開発者は分散システムの分析とデバッグをより簡単に行い、アプリケーションとその基盤となるサービスのパフォーマンスを把握できます。

Archived

リソース

ドキュメント

- Amazon CloudWatch メトリクスの使用
- Canary の使用 (Amazon CloudWatch Synthetics)
- Amazon CloudWatch Logs Insights のサンプルクエリ
- AWS Systems Manager Automation
- AWS X-Ray とは?
- Amazon CloudWatch Synthetics と AWS X-Ray を使用したデバッグ
- Amazon Builders' Library:運用の可視性を高めるために分散システムを装備する

需要の変化に適応するようにワークロードを設計する

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、リソースは常に、現行の需要に厳密に適合します。

リソースを取得またはスケーリングするとき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。サードパーティ製のツールや AWS SDK を使っ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 のサービスには、Amazon S3、Amazon CloudFront、AWS Auto Scaling、AWS Lambda、Amazon DynamoDB、AWS Fargate、Amazon Route 53 などがあります。

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。AWS Auto Scaling は他にも、Amazon EC2 インスタンスおよびスポットフリート、Amazon ECS タ

Archived

スク、Amazon DynamoDB テーブルとインデックス、Amazon Aurora レプリカなどのリソースのスケーリングプランを作成することもできます。

EC2 インスタンスまたは EC2 インスタンスでホストされている Amazon ECS コンテナをスケーリングする場合は、複数のアベイラビリティーゾーン (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ーゾーン間のバランスを維持します。

AWS Lambda を使用すると、自動的にスケーリングが行われます。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 とサービスクォータで、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。たとえば、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を設定して使用します。コンテンツ配信ネットワーク (CDN) 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出すことができます。

ワークロードの障害を検出したらリソースを取得する: 可用性が影響を受ける場合は、必要に応じてリソースを事後対応的にスケーリングし、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、自動操作をトリガーしてリソースを自動的にスケーリングします。

Archived

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。たとえば、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、コンソールまたは AWS CLI で行えます。ただし、可能な限り自動操作を使用する必要があります (「**ワークロードをスケールアップまたはスケールダウンするときに自動操作を使用する**」を参照)。

ワークロードにさらにリソースが必要であることを検出したら、リソースを取得する: リソースを積極的にスケーリングして需要を満たし、可用性に影響を与えるのを回避します。

多くの AWS のサービスは、需要に合わせて自動的にスケーリングします (「**ワークロードをスケールアップまたはスケールダウンするときに自動操作を使用する**」を参照)。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指標に基づいて 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 (たとえば、CPU 使用率の 70%) を維持するためのリソースを追加または削除します。

AWS Auto Scaling は、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。

$$L = \lambda W$$

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)

Archived

たとえば、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。

ワークロードの負荷テスト: 負荷テスト手法を採用して、スケーリングによりワークロード要件を満たせる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。負荷テストでは、ワークロードのブレークポイントとテストパフォーマンスを見極める必要があります。AWS では、本番ワークロードの規模をモデル化する一時的なテスト環境を簡単に設定できます。クラウドでは、オンデマンドで本稼働規模のテスト環境を作成し、テストを完了したらリソースを解放できます。テスト環境の支払いは実行時にのみ発生するため、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。

リソース

ドキュメント

- AWS Auto Scaling: スケーリングプランの仕組み
- Amazon EC2 Auto Scaling とは?
- DynamoDB Auto Scaling によるスループットキャパシティーの自動管理
- Amazon CloudFront とは?
- 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
- AWS Marketplace: Auto Scaling で使用できる製品

Archived

- APN パートナー: 自動化されたコンピューティングソリューションの作成を支援できるパートナー

外部リンク

- リトルの法則について語る

変更の実装

新しい機能を導入し、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。

デプロイなどの標準的なアクティビティにはランブックを使用する: ランブックは、特定の結果を達成するために事前定義されたステップです。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。標準的なアクティビティには、ワークロードのデプロイ、パッチの適用、DNS の変更などがあります。

たとえば、デプロイ中のロールバックの安全性を確保するにはプロセスを配置します。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、自動実行をトリガーします。

高度に自動化された高度なワークロードであっても、ランブックはゲームデーを実行したり、厳しい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。

Archived

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。

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。

デプロイの一部として弾力性テストを統合する: (カオスエンジニアリングの一環としての) 弾力性テストは、本番稼働前環境で自動デプロイパイプラインの一部として実行します。

このようなテストはステージングされ、本番稼働前にパイプラインで実行します。これらも本番環境で実行する必要がありますが、ゲームデーの一部として実行します。

イミュータブルインフラストラクチャを使用してデプロイ: これは、本番システムで更新、セキュリティパッチ、または設定の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。

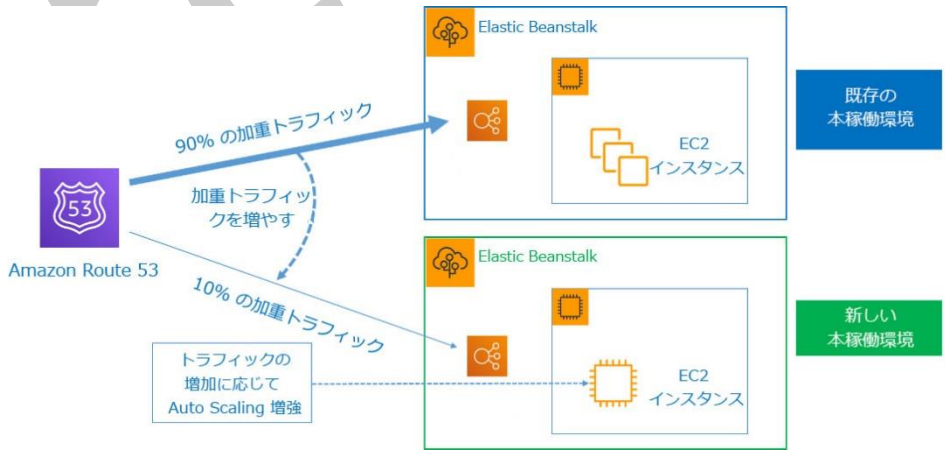
イミュータブルインフラストラクチャパラダイムを実装したものとして最も一般的なのが、イミュータブルサーバーです。つまり、サーバーに更新または修正が必要な場合、既存のサーバーを更新するのではなく、新しいサーバーをデプロイします。そのため、アプリケーションのすべての変更は、SSH 経由でサーバーにログインしてソフトウェアバージョンを更新するのではなく、コードリポジトリにソフトウェアをプッシュすることから始まります (たとえば git push)。イミュータブルインフラストラクチャでは変更が許可されていないため、デプロイされたシステムの状態をしっかりと把握します。イミュータブルインフラストラクチャは、本質的に一貫性があり、信頼性が高く、予測可能であり、ソフトウェアの開発と運用の多くの側面を簡素化します。

Archived

イミュータブルインフラストラクチャにアプリケーションをデプロイする場合は、Canary デプロイまたはブルー/グリーンデプロイを使用します。

Canary デプロイは、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、変更やエラーをモニタリングしながら、希望の速度で完全にデプロイされるまでデプロイを続行できます。AWS CodeDeploy は、Canary デプロイを実行できるようにするデプロイ設定を行うことができます。

ブルーグリーンデプロイは 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、すべてのトラフィックが一度に切り替えられますが、各バージョンへのトラフィックの一部を使用して、Amazon Route 53 の加重 DNS ルーティング機能を使用して新しいバージョンの採用をダイヤルアップすることもできます。AWS CodeDeploy と AWS Elastic Beanstalk は、ブルー/グリーンデプロイを実行できるようにデプロイ設定を行うことができます。



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

イミュータブルインフラストラクチャの利点:

Archived

- 設定ドリフトの低減: バージョンで管理された既知の基本設定からサーバーを頻繁に置き換えることにより、インフラストラクチャが既知の状態にリセットされ、設定のドリフトを回避できます。
- 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。
- 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、何も変更されません。デプロイプロセスの信頼性が高まります。
- 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: 以前の作業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。
- 一貫したテストおよびデバッグ環境: すべてのサーバーが同じイメージを使用するため、環境間で違いはありません。1 つのビルドが複数の環境にデプロイされます。また、環境の整合性が失われるのを防ぎ、テストとデバッグが簡素化されます。
- スケーラビリティの向上: サーバーはベースイメージを使用し、一貫性があり、再現性があるため、Auto Scaling はとても簡単です。
- 簡素化されたツールチェーン: 本番用ソフトウェアのアップグレードを管理する設定管理ツールが不要になるため、ツールチェーンが簡素化されます。サーバーにツールやエージェントが追加でインストールされることはありません。変更はベースイメージに加えられ、テストされてロールアウトされます。
- セキュリティの強化: サーバーへのすべての変更を拒否することで、インスタンスの SSH 接続を無効にし、キーを削除できます。これにより攻撃経路が減少し、組織のセキュリティ体制が向上します。

自動化による変更の導入: 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。

Archived

本番システムに変更を加えることは、多くの組織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。AWS CodePipeline では、ワークロードを解放するために必要な手順を管理できます。これには、AWS CodeDeploy を使用してデプロイされた状態が含まれ、Amazon EC2 インスタンス、オンプレミスインスタンス、サーバーレス Lambda 関数、または Amazon ECS サービスへのアプリケーションコードのデプロイを自動化します。

推奨事項
運用上の最も難しい手順には人の手を借り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、まさにこの同じ理由から自動化が推奨されます。

リスクを最小限に抑えるための別のデプロイパターン:

機能フラグ (機能トグルとも呼ばれます) は、アプリケーションの設定オプションです。ソフトウェアのデプロイ時に機能をオフにすると、その機能は顧客に表示されなくなります。Canary デプロイと同様、この機能をオンにすることも、変更のペースを 100% に設定して影響を確認することもできます。デプロイに問題が発生した場合は、ロールバックしないで単純に機能をオフに戻すことができます。

障害部分を切り離すゾーンでデプロイする: AWS が自社のデプロイのために確立した最重要ルールの 1 つは、リージョン内で複数のアベイラビリティーゾーンに同時アクセスしないことです。これは、アベイラビリティーゾーンを独立させて可用性を正しく計算するために重要となります。デプロイにあたり、このような考慮事項を念頭に置くことを推奨します。

Archived

運用準備状況レビュー (ORR)

AWS では、運用準備状況の確認を実施して、テストの完全性、モニタリング能力、さらには SLA に対するアプリケーションパフォーマンスの監査を評価し、障害時や運用における異常があったときにデータを提供することが有効であると考えています。正式な ORR は、初回の本番デプロイの前に実施されます。AWS では、定期的に ORR を繰り返し (年 1 回または重要なパフォーマンス期間の前)、運用に対する期待事項から「ドリフトする」ことがないようにします。運用準備の詳細については、AWS Well-Architected フレームワークの運用上の優秀性の柱を参照してください。

推奨事項
アプリケーションの運用準備状況レビュー (ORR) を初回の本番稼働の前に実施し、その後も定期的に行ってください。

リソース

動画

- AWS Summit 2019: AWS の CI/CD

ドキュメント

- AWS CodePipeline とは?
- CodeDeploy とは?
- ブルー/グリーンデプロイの概要
- サーバーレスアプリケーションの段階的なデプロイ
- 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
- Amazon Builders' Library: 継続的デリバリーによる高速化
- AWS Marketplace: デプロイの自動化に使用できる製品

Archived

- 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

ラボ

- Well-Architected ラボ:レベル 300: EC2 RDS と S3 の弾力性のテスト

外部リンク

- CanaryRelease

障害の管理

> 障害は発生するもので、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、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。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。たとえば、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に設計されています。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、弾力性を持たせるようにするための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。

- 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
- 弾力性をテストする方法
- 災害対策 (DR) の計画

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。

バックアップが必要なすべてのデータを特定してバックアップするか、ソースからデータを再現する: Amazon S3 は、複数の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 のサービスには、バックアップを作成する機能が組み込まれています。または、サードパーティ製のバックアップソフトウェアを使用できます。また、データを他のソースから複製して RPO を満たせる場合は、バックアップが必要ない場合もあります。

オンプレミスのデータは、Amazon S3 バケットと AWS Storage Gateway を使用して AWS クラウドにバックアップできます。Amazon S3 Glacier または S3 Glacier Deep Archive を使用してバックアップデータをアーカイブすると、手頃な料金で時間的制約のないクラウドストレージを実現できます。

Archived

データを 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) にロードしてそれを分析した場合、これは他のソースから再現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 リードレプリカがあります。

バックアップの保護と暗号化: AWS Identity and Access Management (IAM) などの認証と承認によってアクセスを検出し、暗号化を使用してデータの整合性の侵害を検出します。

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、それを暗号化した後に永続化します。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が S3 に送信される前にそのデータを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) を使用してデータキーを作成して保存するか、独自のキーを提供することができます (提供した場合、責任を負うのはお客様です)。AWS KMS を使用すると、AWS 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 バックアップは常に暗号化されます。

データのバックアップを自動的に実行する: 定期的なスケジュールに基づいて、またはデータセット内の変更に基づいて自動的にバックアップが作成されるように設定します。RDS インスタンス、EBS ボリューム、DynamoDB テーブル、および S3 オブジェクトはすべて、自動的にバックアップされるように設定できます。AWS Marketplace ソリューションまたはサードパーティのソリューションも使用できます。

Archived

Amazon Data Lifecycle Manager を使用して、EBS スナップショットを自動化できます。Amazon RDS と Amazon DynamoDB では、特定時点への復元による継続的なバックアップが行えます。Amazon S3 のバージョン管理は、一度有効にすると、自動的に行われます。

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 バケット全体を別の 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。

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。

AWS を利用して、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 と Amazon DynamoDB では、ポイントインタイムリカバリ (PITR) が可能です。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。

リソース

動画

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

ドキュメント

- AWS Backup とは?
- Amazon S3: 暗号化によるデータの保護

Archived

- AWS でバックアップを暗号化する
- DynamoDB のオンデマンドバックアップと復元
- EFS-to-EFS バックアップ
- AWS Marketplace: バックアップに使用できる製品
- APN パートナー: バックアップを支援できるパートナー

ラボ

- Well-Architected ラボ: Level 200: Testing Backup and Restore of Data

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。

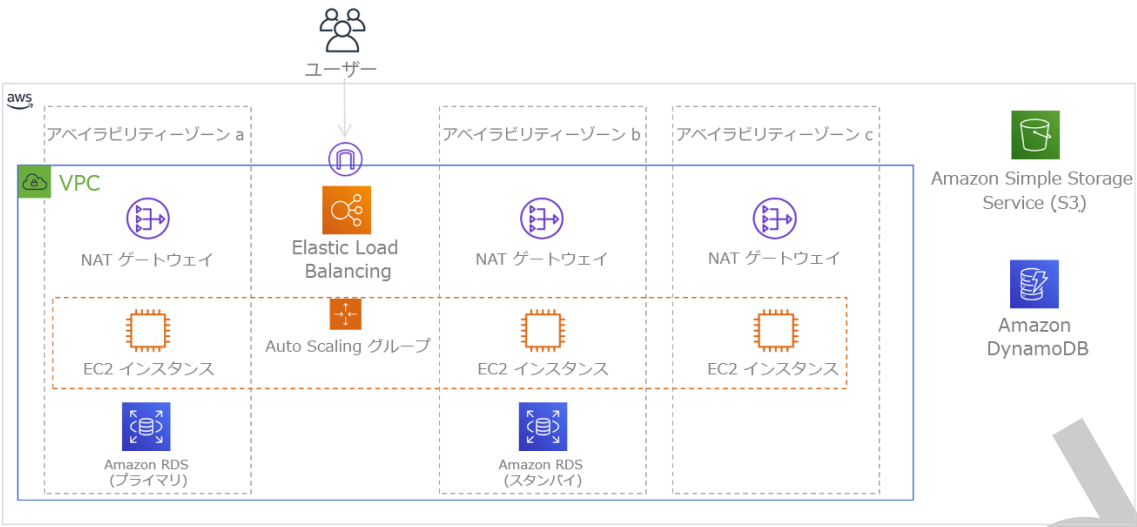
ワークロードを複数のロケーションにデプロイする: ワークロードデータとリソースを複数のアベイラビリティーゾーンに分散するか、必要に応じて複数の AWS リージョンにかけて分散します。これらのロケーションは、必要に応じて多様化できます。

AWS のサービス設計の基本原則の 1 つに、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があります。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。冗長化されたコンポーネントに依存するシステムを構築する場合、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、複数の AWS リージョンを使用する場合は、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。

Archived

アベイラビリティーゾーン: AWS リージョンは 2 つ以上のアベイラビリティーゾーンで構成され、それらのアベイラビリティーゾーンはそれぞれ独立するように設計されています。各アベイラビリティーゾーンは、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ーゾーンは、ユーティリティ電源への専用接続、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、アベイラビリティーゾーン内外の独立したネットワーク接続などの独立した物理インフラストラクチャを持っています。地理的には分離されていても、アベイラビリティーゾーンは、同じリージョンのエリアに配置されています。これにより、データベース間などで行う同期データレプリケーションで、アプリケーションのレイテンシーに過度な影響を与えることがなくなります。このようにして、アクティブ/アクティブの設定またはアクティブ/スタンバイの設定でアベイラビリティーゾーンを使用することができます。アベイラビリティーゾーンは独立しているため、複数のゾーンを使用すると、アプリケーションの可用性が向上します。AWS のサービス (EC2 インスタンスのデータプレーンなど) には、アベイラビリティーゾーン全体が運命共同体となり、厳密なゾーンサービスとしてデプロイされるものもあります。このようなサービスを使用すると、特定のアベイラビリティーゾーン内のリソース (インスタンス、データベース、その他インフラストラクチャ) を独立して操作することができます。AWS は以前からリージョンに複数のアベイラビリティーゾーンを配置してきました。

Archived



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 です。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ーゾーン) 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ー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。リージョンにおける AWS のサービスは、設定された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/アクティブの設定で複数のアベイラビリティーゾーンを使用しています。

推奨事項
ある**アベイラビリティーゾーン**のサービス中断時にコントロールプレーン API の可用性にアプリケーションが依存する場合は、API フィルターを使用して API リクエスト (例えば DescribeInstances) ごとに 1 つのアベイラビリティーゾーンの結果をリクエストします。

Archived

AWS ローカルゾーンは、サブネットや EC2 インスタンスなどのゾーン内の 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 リージョン内のアベイラビリティーゾーンと同じ様に機能します。それらが特別なのは、関連 AWS リージョンにあるのではなく、現在ご利用の 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には、高帯域幅で安全な接続を維持しています。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 グローバルエッジネットワークは、世界中の都市のエッジロケーションで構成されています。Amazon CloudFront は、このネットワークを使用して、コンテンツをより低いレイテンシーでエンドユーザーに配信しています。AWS Global Accelerator を使用すると、ワークロードエンドポイントをこのようなエッジロケーションに作成して、ユーザーに近い場所で AWS グローバルネットワークへのオンボーディングが行えます。Amazon API Gateway は、CloudFront ディストリビューションを使用してエッジに最適化された API エンドポイントを有効化して、最も近いエッジロケーションからクライアントがアクセスできるようにします。

AWS リージョン: 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。

推奨事項
ワークロードの信頼性目標はほとんど、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。マルチリージョンである必要があるワークロードの場合のみ、マルチリージョンアーキテクチャをご検討ください。

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。たとえば、Amazon Aurora Global Database、Amazon DynamoDB グローバルテーブル、Amazon S3 のク

Archived

ロスリージョンレプリケーション、Amazon RDS を使用したクロスリージョンリードレプリカ、さまざまなスナップショットや Amazon マシンイメージ (AMI) を他のリージョンにコピーできる機能などがあります。ただし、リージョンの自律性を維持する方法でこれを行っています。このアプローチは、AWS Identity and Access Management (IAM) サービスのコントロールプレーンや、グローバルエッジデリバリー (Amazon CloudFront や Amazon Route 53) を提供する当社のほぼすべてのサービスに適用されています。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ついては、可能な場合はハイブリッドエクスペリエンスを得られるように設計します。AWS Direct Connect により、お客様の設備から AWS への専用ネットワーク接続が行え、オンプレミスと併せて実行できるようにしています。

別のオプションは、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フルマネージドサービスです。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、Outposts は最寄りの AWS リージョンに接続されます。すると、Outpost を使用して、低レイテンシーまたはローカルデータ処理要件を持つワークロードをサポートできます。

単一の場所に制限されているコンポーネントの自動復旧: ワークロードのコンポーネントが 1 つ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実行できる場合は、定義され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。

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、弾力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

Archived

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。

たとえば、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弾力性を確保する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。複数のマスターノードをプロビジョニングできます。EMR ファイルシステム (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 リージョンにかけてレプリケートできます。

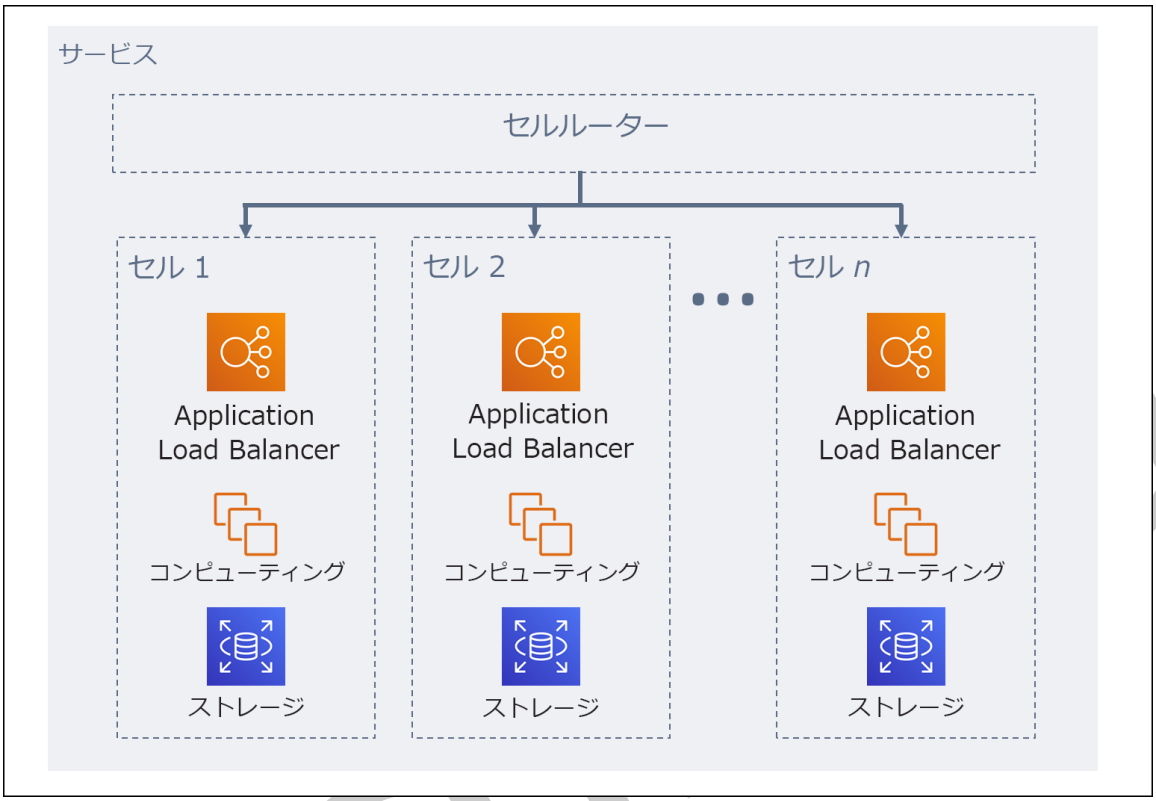
Amazon Redshift もそれと同様に、デフォルトでは、選択した AWS リージョン内のランダムに選択されたアベイラビリティーゾーンにクラスターをプロビジョニングします。すべてのクラスターノードが同じゾーンにプロビジョニングされます。

バルクヘッドアーキテクチャを使用する: 船の隔壁のように、このパターンは、障害がリクエスト/ユーザーの小さなサブセットに確実にとどまるようにすることで、障害のあるリクエストの数が制限され、ほとんどがエラーなしで続行できるようにします。データの隔壁は通常パーティションまたはシャードと呼ばれ 、サービスの隔壁はセルと呼ばれます。

セルベースのアーキテクチャでは 、各セルは独立した完全なサービスのインスタンスで、最大サイズは固定されています。負荷が増加すると、セルが追加されることでワークロードが増大します。着信トラフィックでパーティションキーを使用して、リクエストを処理するセルを決定します。障害は発生した単一のセルに限定され、他のセルがエラーなしで継続するため、障害のあるリクエストの数が制限されます。セル間の相互作用を最小限に抑え、各リクエストに複雑なマッピングサービスを含める必要がないように、適切なパーティションキーを特定することが重要です。複雑なマッピングを必要とするサービスは、問題をマッピングサービスにシ

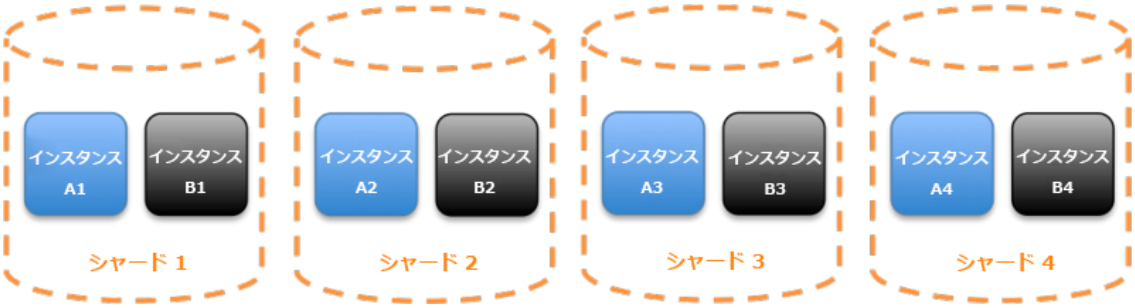
Archived

フトするだけですが、セル間の相互作用を必要とするサービスは、セルの独立性が低下します (そのため、これにより可用性の改善が想定されます)。



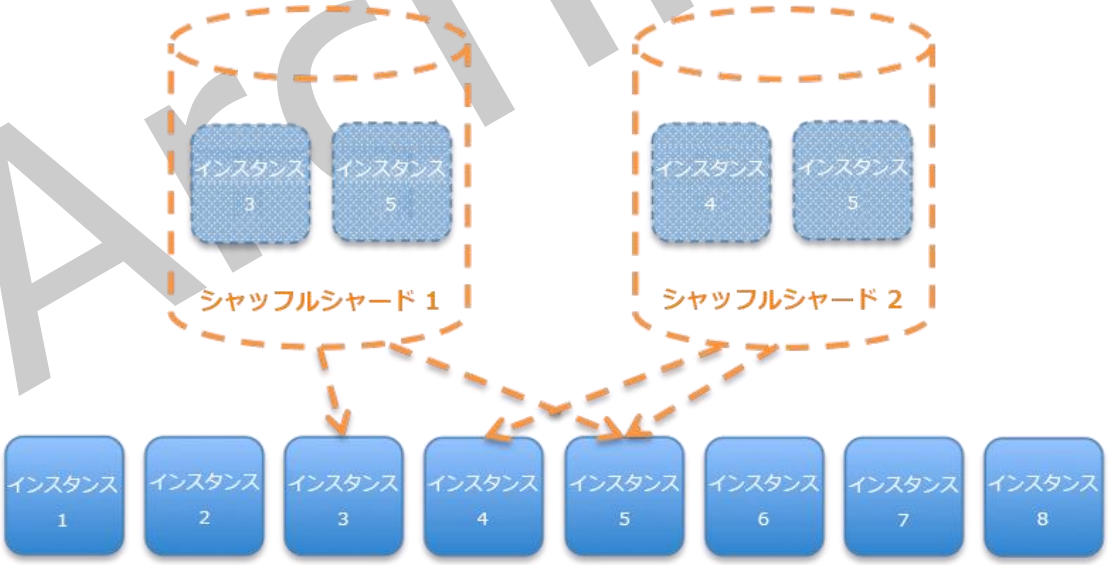
セルベースのアーキテクチャ

Colm MacCarthaigh は、 AWS ブログの記事で、Amazon Route 53 がシャッフルシャーディングの概念を用いて顧客のリクエストをシャードに分離する方法を説明しています。この場合、シャードは 2 つ以上のセルで構成されます。パーティションキーに基づいて、顧客 (またはリソース、または分離対象) からのトラフィックは、割り当てられたシャードにルーティングされます。シャードごとに 2 つのセルを持つ 8 つのセルがあり、顧客が 4 つのシャードに分割された場合、問題が発生した場合に影響を受ける顧客は全体の 25% です。



それぞれ *2* つのセルを持つ *4* つの従来のシャードに分割されたサービス

シャッフルシャーディングでは、それぞれ 2 つのセルを持つ仮想シャードを作成し、顧客をそれらの仮想シャードの 1 つに割り当てます。問題が発生した場合でも、サービス全体の 4 分の 1 が失われる可能性がありますが、顧客またはリソースが割り当てられる方法から、シャッフルシャーディングでは影響を受ける範囲が 25% を大幅に下回ることになります。8 つのセル中の 2 つのセルには 28 のユニークな組み合わせがあります。つまり、シャッフルシャード (仮想シャード) が 28 通りあります。数百または数千の顧客がいて、各顧客を 1 つのシャッフルシャードに割り当てた場合、問題が発生したことで影響を受ける範囲はわずか 1/28 です。これは通常のシャーディングより 7 倍優れています。



それぞれ *2* つのセルを持つ *28* のシャッフルシャード *(*仮想共有*)* に分割されたサービス *(28* 通りのうち *2* つのシャッフルシャードのみを表示*)*

Archived

シャードは、セルだけでなく、サーバー、キュー、またはその他のリソースにも使用できます。

リソース

動画

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)
- シャッフルシャーディング: AWS re:Invent 2019: Introducing The Amazon Builders' Library (DOP328)
- AWS re:Invent 2018: How AWS Minimizes the Blast Radius of Failures (ARC338)
- AWS re:Invent 2019: Innovation and operation of the AWS global network infrastructure (NET339)

ドキュメント

- What is AWS Outposts?
- 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
- AWS Local Zones のよくある質問
- AWS グローバルインフラストラクチャ
- リージョン、アベイラビリティーゾーン、Local Zones
- Amazon Builders' Library: シャッフルシャーディングを使ったワークロードの分離

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、高い弾力性があるように設計する必要があります。

Archived

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、障害を検出する：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、ワークロードのパフォーマンスが低下したか、または完全な障害が発生したらすぐにお客様と自動化システムが把握できるようにします。ビジネス価値に基づいて重要業績評価指標 (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、その能力が検出および修正戦略の主要な指標である必要があります。

障害の発生していない場所にある正常なリソースへのフェイルオーバー: あるロケーションに障害が発生した場合でも、正常なロケーションのデータとリソースが引き続きリクエストに対応できるようにしましょう。Elastic Load Balancing や AWS Auto Scaling などの AWS のサービスは、アベイラビリティーゾーン間で負荷を分散できるため、マルチゾーンのワークロードの方が簡単です。マルチリージョンのワークロードの場合、状況はさらに複雑です。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が、プライマリロケーションに障害が発生した場合は、リードレプリカをマスターに昇格させ、そこへトラフィックを向ける必要があります。Amazon Route 53 と AWS Global Accelerator は、AWS リージョン間のトラフィックのルーティングにも役立ちます。

ワークロードが Amazon S3 や Amazon DynamoDB などの AWS のサービスを使用している場合、自動的に複数のアベイラビリティーゾーンにデプロイされます。障害が発生した場合、AWS コントロールプレーンはトラフィックを正常な場所に自動的にルーティングします。Amazon RDS の場合、設定オプションとしてマルチ AZ を選択する必要があります。そして障害が発生すると、AWS はトラフィックを正常なインスタンスに自動的にルーティングします。Amazon EC2 インスタンスまたは Amazon ECS タスクの場合、デプロイ先のアベイラビリティーゾーンを選択します。次に Elastic Load Balancing は、異常なゾーンのインスタンスを検出し、トラフィックを正常なゾーンにルーティングするソリューションを提供します。Elastic

Archived

Load Balancing は、オンプレミスのデータセンター内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。

マルチリージョンのアプローチ (オンプレミスのデータセンターも含まれる場合があります) の場合、Amazon Route 53 はインターネットドメインを定義し、ヘルスチェックを含むルーティングポリシーを割り当てて、トラフィックが正常なリージョンにルーティングされるようにします。または、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 アドレスを提供します。そして、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、選択した AWS リージョンのエンドポイントにルーティングして、パフォーマンスと信頼性を向上させます。

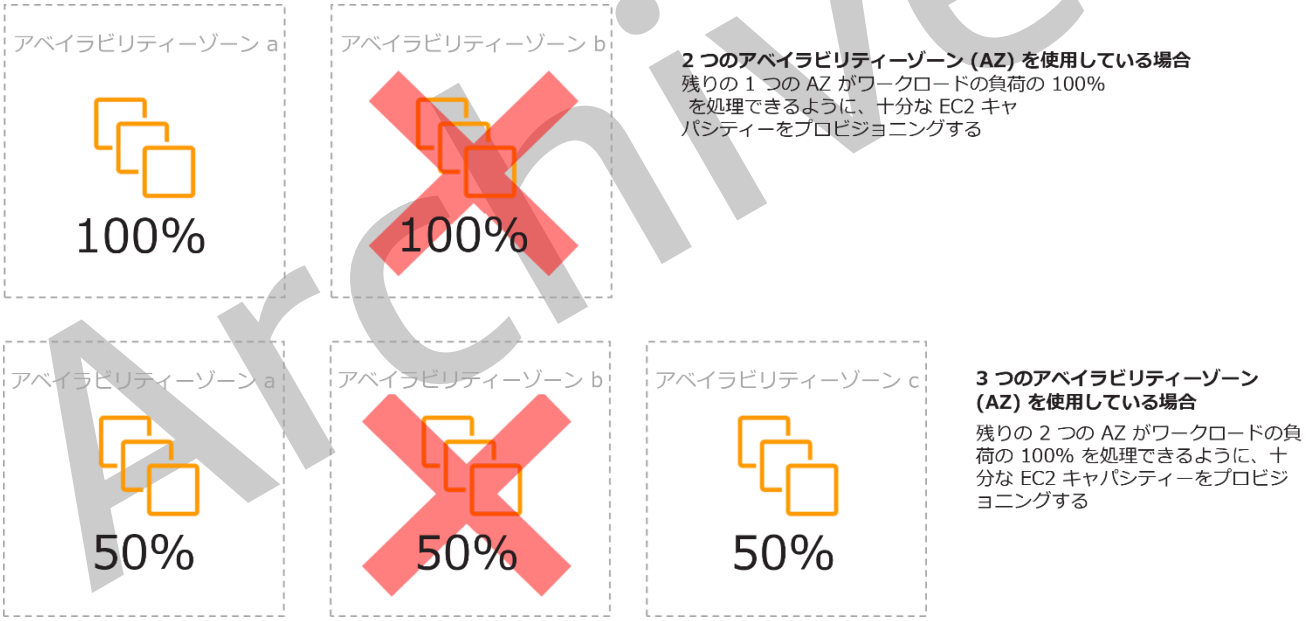
AWS は、障害からの復旧を念頭に置いてサービスの設計に取り組んでいます。当社は、障害からの復旧時間とデータへの影響を最小限に抑えるサービスを設計しています。当社のサービスは主にデータストアを使用しており、リクエストが認識されるのは複数のレプリカにわたりデータが永続的に保存された後です。そのようなサービスおよびリソースには、Amazon Aurora、Amazon Relational Database Service (Amazon RDS) マルチ AZ DB インスタンス、Amazon S3、Amazon DynamoDB、Amazon Simple Queue Service (Amazon SQS)、Amazon Elastic File System (Amazon EFS) などがあります。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の独立性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。

静的安定性を使用してバイモーダル動作を防止する: バイモーダル動作とは、たとえばアベイラビリティーゾーン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システムを構築する必要があります。この場合、1 つのゾーン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を各ゾーンにプロビジョニングしてから、Elastic Load Balancing または

Archived

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。

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。これは、コストがかかる懸念と比較検討する必要があります。プロビジョニングするコンピューティングキャパシティーを減らし、障害が発生した場合は新しいインスタンスを起動する方が、コストは低くなります。ただし、大規模な障害 (アベイラビリティーゾーンの障害など) が発生した場合には、効果が低くなります。このアプローチは障害が発生する前に準備するのではなく、障害が発生したときに事後的に対処することになるためです。ソリューションを考える際は、信頼性とワークロードのコストのニーズを比較検討する必要があります。より多くのアベイラビリティーゾーンを使用することで、静的安定性に必要なコンピューティングキャパシティーが減少します。



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。

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

Archived

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することになるでしょう。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、アラームをトリガーします。

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、ワークロードのリクエストを大幅に変更し、障害が発生する可能性が高いため、許可すべきではありません。

すべてのレイヤーの修復を自動化: 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。

再起動する機能は、障害を修復するための重要なツールです。分散システムについて前述したように、ベストプラクティスは、可能な場合はサービスをステートレスにすることです。これにより、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 (EC2 インスタンス、Lambda 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。さまざまなタイプの障害をそれぞれ捕捉、特定、修正するための新しいメカニズムを構築するのではなく、さまざまなカテゴリの障害を同じ復旧戦略にマッピングします。ハードウェアの障害、オペレーティングシステムのバグ、メモリリーク、その他の原因で、インスタンスが機能しなくなることがあります。状況ごとにカスタム修復を構築するのではなく、そのいずれかをインスタンスの障害として扱います。インスタンスを終了し、AWS Auto Scaling がそのインスタンスを置き換えられるようにします。その後、障害が発生したリソースの分析を帯域外で実行します。

Archived

もう 1 つの例は、ネットワークリクエストを再起動する機能です。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 (ROC) 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。

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 (または他のターゲット) をトリガーして、ワークロードでカスタム修正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。AWS OpsWorks を使用している場合は、レイヤーレベルで EC2 インスタンスの自動ヒーリングを設定できます。

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、複数の新しいリソースを一度に取得するのではなく、静的安定性が高可用性のために優先されます。

イベントが可用性に影響を与えるときに通知を送信する: 重大なイベントが検出されると、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。Amazon CloudWatch アラームは、発生した障害に基づいてトリガーできます。また、実行された自動ヒーリングアクションに基づいてトリ

Archived

ガーすることもできます。CloudWatch アラームは、Amazon SNS 統合を使用して、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。

リソース

動画

- Static stability in AWS: AWS re:Invent 2019: Introducing The Amazon Builders' Library (DOP328)

ドキュメント

- AWS OpsWorks:自動ヒーリングを使用した、失敗したインスタンスの置き換え
- Amazon EventBridge とは?
- Amazon Route 53: ルーティングポリシーの選択
- What Is AWS Global Accelerator?
- 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
- Amazon Builders' Library:ヘルスチェックの実装
- AWS Marketplace: 耐障害性に活用できる製品
- APN パートナー: 耐障害性の自動化を支援できるパートナー

ラボ

- Well-Architected ラボ: Level 300: Implementing Health Checks and Managing Dependencies to Improve Reliability

外部リンク

- The Berkeley/Stanford Recovery-Oriented Computing (ROC) Project

Archived

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機能以外の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。ワークロードの弾力性をテストして、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。

プレイブックを使用して障害を調査する: プレイブックに調査プロセスを記載して文書化することで、十分に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できるようになります。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。

このプレイブックは、事後的な行動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予防的な計画です。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。

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、寄与する原因と予防措置項目を特定します。この情報を使用して、繰り返し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、寄与因子と是正措置を必要に応じて伝えます。

Archived

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。

テスト機能要件: 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。

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。たとえば、開発者は 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。

さらに、顧客の行動を実行およびシミュレートできる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 デプロイと混同しないでください) は、最も重要なテストプロセスの 1 つであることが経験からわかっています。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 では、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。

テストのスケーリングとパフォーマンスの要件: これには、ワークロードがスケーリングと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が含まれます。

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。

Archived

カオスエンジニアリングを使用した弾力性のテスト: 本番稼働前および運用環境に定期的に障害を挿入してテストを実行します。ワークロードが障害にどのように反応するかの仮説を立て、その仮説をテスト結果と比較して、一致しない場合はプロセスを繰り返します。本番稼働テストがユーザーに影響を与えないようにします。

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。自動化により、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。これにより、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明らかになるため、リスクが軽減されます。

> カオスエンジニアリングは、システムの生産において乱れ条件に耐える能力があるという確証を高めるために、システム上で実験する規律です。– カオスエンジニアリングの原則

本番稼働前の環境とテスト環境では、カオスエンジニアリングを定期的に実行し、CI/CD サイクルの一部にする必要があります。本番稼働時、チームは可用性を妨げないように注意する必要があります。本番稼働中にカオスエンジニアリングのリスクを制御する方法の 1 つとしてゲームデーを使用してください。

テストの取り組みは可用性の目標に見合ったものにする必要があります。可用性の目標を達成するためにテストを実施することこそが、目標達成の信頼性を高める唯一の方法です。

ワークロードが障害に対する弾力性を持てるように設計したコンポーネントの障害をテストします。これには、EC2 インスタンスの損失、プライマリ Amazon RDS データベースインスタンスの障害、アベイラビリティーゾーンの停止などがあります。

外部依存関係が利用できないかテストします。依存関係にあるシステムの一時的な障害に対するワークロードの弾力性を、1 秒未満から数時間までの継続時間でテストする必要があります。

Archived

その他のサービス品質劣化のパターンでは、機能が低下して応答が遅くなり、サービスが停止す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。ネットワークへの影響 (遅延やメッセージのドロップなど)、DNS 障害 (名前を解決できない、依存サービスへの接続を確立できないなど) などの障害をシステムに挿入する機能を活用することも可能です。

障害を挿入するためのサードパーティオプションがいくつかあります。これには、Netflix Simian Army、The Chaos ToolKit、Shopify Toxiproxy などのオープンソースオプション、それに Gremlin などの商用オプションがあります。カオスエンジニアリングの実装方法の初期調査では、自己作成のスクリプトを使用することをお勧めします。これにより、エンジニアリングチームはワークロードにどのようにしてカオスが導入されるかを理解できます。これらの例については、Bash、Python、Java、PowerShell などの複数の言語を使用した「EC2 RDS および S3 の弾力性のテスト」を参照してください。また、AWS Systems Manager を使用して Amazon EC2 にカオスを挿入するを実装する必要があります。これにより、AWS Systems Manager ドキュメントを使用して電圧低下や高 CPU 状態をシミュレートできます。

ゲームデーを定期的に実施する: ゲームデーを使用して、実際の障害シナリオに関わる人と一緒に、可能な限り本番環境に近い環境 (本番環境を含む) で障害対処手順を実行します。ゲームデーでは、本番環境のテス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。

ゲームデーを定期的にスケジュールして、本番環境のイベントをシミュレートすることで、アーキテクチャとプロセスのパフォーマンスをテストします。これは、改善可能な箇所を把握したり、イベントに対応する体験を組織的に醸成するのに役立ちます。

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。プレイブックを用意します。次に、障害が所定の方法で本番システムに注入され、影響を評価します。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

Archived

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は必要に応じて (プレイブックを参照して) 手動で修正します。ゲームデーは本番環境で行われるため、顧客の可用性に影響を与えないように期すために、あらゆる予防策を講じる必要があります。

リソース

動画

- AWS re:Invent 2019: Improving resiliency with chaos engineering (DOP309-R1)

ドキュメント

- 継続的デリバリーと継続的インテグレーション
- Canary の使用(Amazon CloudWatch Synthetics)
- AWS CodeBuild の CodePipeline を使用してコードをテストし、ビルドを実行する
- AWS Systems Manager で運用プレイブックを自動化する
- AWS Marketplace: 継続的インテグレーションに利用できる製品
- 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

ラボ

- Well-Architected ラボ: Level 300: Testing for Resiliency of EC2 RDS and S3

外部リンク

- Principles of Chaos Engineering
- Resilience Engineering: Learning to Embrace Failure
- Apache JMeter

Archived

本

- Casey Rosenthal、Lorin Hochstein、Aaron Blohowiak、Nora Jones、Ali Basiri。"Chaos Engineering" (August 2017)

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、可用性を回復するための目標です。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。

ダウンタイムとデータ損失の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) があります。

目標復旧時間 (RTO) は、組織のミッションまたはミッション/ビジネスプロセスに悪影響が及ぶ前の、ワークロードのコンポーネントが復旧フェーズにあるため利用できない全体的な時間の長さを表します。

目標復旧時点 (RPO) は、組織のミッションまたはミッション/ビジネスプロセスに悪影響が及ぶ前の、ワークロードのデータが利用できなくなる可能性がある全体的な時間の長さを表します。

重要なデータでは可用性を復元できないため、RPO は必ず RTO より短くなります。

復旧目標を達成するため、定義された復旧戦略を使用する: 災害対策 (DR) 戦略は、ワークロードの目標を満たすように定義されています。

マルチリージョン戦略が必要でない限り、AWS リージョン内の複数のアベイラビリティーゾーンを使用して、AWS の復旧目標を達成することをお勧めします。

必要に応じて、ワークロードのマルチリージョン戦略を設計するときは、次の戦略のいずれかを選択してください。戦略は、複雑さの昇順、および RTO と RPO の降順でリストされていま

Archived

す。*DR* リージョンは、ワークロードに使用されるリージョン以外の AWS リージョン (または、ワークロードがオンプレミスの場合は AWS リージョン) を指します。

- **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 DR リージョンにバックアップします。災害からの復旧に必要な場合は、このデータを復元します。
- **パイロットライト** (RPO は分単位、RTO は時間単位): DR リージョンでシステムの最も重要なコア要素を常に実行している環境の最小バージョンを維持します。復旧の必要が生じたときに、重要なコアを中心として完全な本番環境をすばやくプロビジョニングすることができます。
- **ウォームスタンバイ** (秒単位の RPO、分単位の RTO): 常に DR リージョンで実行されている完全に機能する環境の縮小バージョンを維持します。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。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。
- **マルチリージョンのアクティブ/アクティブ** (RPO はゼロまたはおそらく秒単位、RTO は秒単位): 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処します。この戦略では、使用しているリージョン間でユーザーとデータを同期する必要があります。復旧時には、Amazon Route 53 や AWS Global Accelerator などのサービスを使用して、ワークロードが正常な場所にユーザートラフィックをルーティングします。

Archived

推奨事項
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。どちらにも、DR リージョンで実行されている環境が含まれています。これらの間で、DR 戦略に追加のインフラストラクチャのデプロイが必要な場合は、パイロットライトを使用します。既存のインフラストラクチャのスケールアップとスケールアウトのみが必要な場合は、ウォームスタンバイを使用します。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。

災害対策の実装をテストして、実装を検証する。DR へのフェイルオーバーを定期的にテストして、RTO と RPO が満たされていることを確認します。

回避すべきなのは、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。復旧経路が複雑または重大な場合に復旧経路が正常に機能するという信頼性を高めるには、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。

DR サイトまたはリージョンで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に必要に応じた状態であることを確認します。たとえば、AMI とサービスクォータが最新であることを確認します。

Archived

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。ドリフトを検出して AWS Systems Manager Automation をトリガーして修正し、アラームを発生させることができます。AWS CloudFormation は、デプロイしたスタックのドリフトを追加で検出できます。

復旧の自動化: 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、DR サイトまたはリージョンにトラフィックを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 のサービスは正常なアベイラビリティーゾーンに負荷を分散できる一方、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。

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では、AWS Marketplace から入手可能な CloudEndure Disaster Recovery により、組織は AWS に自動化された災害対策戦略を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 災害対策もサポートしています。

リソース

動画

- AWS re:Invent 2019: Backup-and-restore and disaster-recovery solutions with AWS (STG208)

ドキュメント

- AWS Backup とは?
- AWS Config ルールによる非準拠 AWS リソースの修復
- AWS Systems Manager Automation
- AWS CloudFormation: CloudFormation スタック全体でドリフトを検出する
- Amazon RDS: クロスリージョンバックアップコピー

Archived

- RDS: リージョン間でリードレプリカをレプリケートする
- S3: クロスリージョンレプリケーション
- Route 53: DNS フェイルオーバーの設定
- CloudEndure Disaster Recovery
- 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 への CloudEndure Disaster Recovery
- AWS Marketplace: 災害対策に使用できる製品
- APN パートナー: 災害対策を支援できるパートナー

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。

Archived

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
リソースのモニタリング	ワークロードリソースのモニタリング
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
弾力性をテストする方法	テストの信頼性
災害対策 (DR) の計画	災害対策 (DR) の計画

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。各 AWS サービスの設計目標については、「付録 A: 一部の AWS サービスの可用性のために設計」を参照してください。

Archived

単一リージョンのシナリオ

99%(ツーナイン)シナリオ

このようなワークロードはビジネスに役立ちますが、利用できない場合は不便です。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。

リソースのモニタリング

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、障害発生時の復旧を高速化させます。

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。

Archived

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。

弾力性をテストする方法

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) の計画

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

Archived

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、前述の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。

概要

トピック	**実装例**
リソースのモニタリング	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
弾力性をテストする方法	全面的な再構築、バックアップから復元
災害対策 (DR) の計画	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

Archived

99.9%(スリーナイン)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できます。

リソースのモニタリング

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。

Archived

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は、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し、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できます。本シナリオでは、Elastic Load Balancing、Auto Scaling、AWS Key Management Service による暗号化ストレージを備えた Amazon RDS マルチ AZ など、複数のアベイラビリティーゾーンにまたがって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の性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 が表示され、ロードバランサーおよび Auto Scaling で使用できるように設定されるので、失敗したインスタンスは削除されて置き換えられます。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は、アクティブなデータベースエンジンをセカンダリアベイラビリティーゾーンで利用可能な状態にし、その後、修復して同じ弾力性を持った状態に復元します。

Archived

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。

弾力性をテストする方法

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) の計画

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

当社は、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。

Archived

概要

トピック	実装例
リソースのモニタリング	サイトのヘルスチェックのみ、ダウン時にアラート送信
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
弾力性をテストする方法	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) の計画	同じ AWS リージョンへの暗号化されたバックアップ (RDS を使用)

Archived

99.99%(フォーナイン)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、アプリケーションが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するアーキテクチャを使用すると、さらに可用性を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 に対しては更新を行うべきではありません。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、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。

リソースのモニタリング

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。このような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。Aurora では、必要に応じてストレージを 10 GB 単位で最大 64 TB まで自動的に拡張することもできます。

Archived

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は、プレイブックを使用して、何がそういった傾向を引き起こしているのかを特定します。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、AWS Support for Infrastructure Event Management のサービス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、実装後に適用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。

Archived

弾力性をテストする方法

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) の計画

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、この**可用性設計目標**は 99.99% です。

Archived

概要

トピック	実装例
リソースのモニタリング	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 (Canary デプロイまたはブルーグリーンデプロイ) と自動ロールバックを実施。分離したゾーン毎のデプロイ。
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、ゲームデー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	アプリケーションに対し、お互いに障害が伝搬しないよう分離された複数のゾーンを実装。Auto Scaling による自己修復

Archived

トピック	実装例
	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。
弾力性をテストする方法	一連のコンポーネントと障害テストおよび分離ゾーン障害テストを定期的にゲームデーに運用スタッフと一緒に実施。不明な問題の診断のためのプレイブックと根本原因の分析プロセスが存在。
災害対策 (DR) の計画	RDS を使用して同じ AWS リージョンへの暗号化されたバックアップをゲームデーに実施。

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運用コストが増大しますが、その理由の 1 つは、独立性を保つため各リージョンは他のリージョンから完全に分離されていることが挙げられます。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 のサービスにおいて強い依存性のある故障が発生した場合に、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。

Archived

可用性が 99.95% で、復旧時間が 1～30 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。

2 つの AWS のリージョンにわたり「ウォームスタンバイ」アプローチを使うことで、復旧時間を改善することができます。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、それぞれのリージョン内で静的に安定します。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。ワークロードの状態をモニタリングする軽量のルーティングコンポーネントを作成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。

リソースのモニタリング

ウェブサーバーのリプレース、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、リージョンのハードな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

99.99%(フォーナイン)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。

Archived

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、AWS Support for Infrastructure Event Management のサービスとも連携しています。

データのバックアップ方法

99.99%(フォーナイン)のシナリオと同様に、RDS バックアップを自動化し、S3 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

99.99%(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。

弾力性をテストする方法

99.99%(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) の計画

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

Archived

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、AWS Systems 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、この**可用性設計目標**は 99.95% です。

まとめ

トピック	実装例
リソースのモニタリング	すべてのレイヤーに関するヘルスチェック (AWS リージョンレベルの DNS ヘルス、KPI を含む)、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 のアクティ

Archived

	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 リージョン間でデータとインフラストラクチャを同期。
変更の実装	Canary デプロイまたはブルー/グリーンデプロイを介した自動デプロイと、KPI またはアラートがアプリケーションで未検出の問題を示し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 を使用した自動バックアップ (AWS リージョンごと)、ゲームデーに定期的に自動復元を実践。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	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。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。

Archived

弾力性をテストする方法	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の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) の計画	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。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ーに実践。

Archived

99.999%(ファイブナイン)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。

複数の AWS リージョンにわたりアクティブ/アクティブアプローチまたはマルチマスターアプローチを使用することで、可用性をさらに向上させることができます。ワークロードは、リージョン間で静的に安定しており、必要なすべての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)。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。

リソースのモニタリング

99.95%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

99.95%のシナリオと同じです。

変更の実装

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。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

Archived

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、何がそういった傾向を引き起こしているのかを特定します。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、AWS Support for Infrastructure Event Management のサービスとも連携しています。

データのバックアップ方法

99.95%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。

Archived

弾力性をテストする方法

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) の計画

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マスター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。マスター 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

全ての復旧作業を自動化するためにかなりの投資が行われ、復旧作業は 1 分以内に完了することを想定しています。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、この**可用性設計目標**は 99.999% です。

Archived

まとめ

トピック	実装例
リソースのモニタリング	すべてのレイヤーに関するヘルスチェック (AWS リージョンレベルの DNS ヘルス、KPI を含む)、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 リージョン間でデータとインフラストラクチャを同期。
変更の実装	Canary デプロイまたはブルー/グリーンデプロイを介した自動デプロイと、KPI またはアラートがアプリケーションで未検出の問題を示し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。

Archived

トピック	実装例
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 (AWS リージョンごと)、ゲームデーに定期的に自動復元を実践。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	アプリケーションに対し、お互いに障害が伝搬しないよう分離された複数のゾーンを実装。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。
弾力性をテストする方法	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の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) の計画	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

Archived

トピック	実装例
	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、新しいデータベースマスター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して運用する方法
- 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: 大規模な失敗
- システム全体の可用性の計算

本

- Robert S. Hammer「Patterns for Fault Tolerant Software」

Archived

- Andrew Tanenbaum、Marten van Steen「Distributed Systems: Principles and Paradigms」

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問い合わせるか、ウェブサイトからお問い合わせください。

寄稿者

本ドキュメントの寄稿者は次のとおりです。

- Seth Eliot、プリンシパルリライアビリティソリューションアーキテクト、Well-Architected、アマゾン ウェブ サービス
- Adrian Hornsby、プリンシパルテクニカルエバンジェリスト、アーキテクチャ、アマゾン ウェブ サービス
- Philip Fitzsimons、シニアマネージャー Well-Architected、アマゾン ウェブ サービス
- Rodney Lester、リライアビリティリード、Well-Architected アマゾン ウェブ サービス
- Kevin Miller、ソフトウェア開発ディレクター、アマゾン ウェブ サービス

Archived

- Shannon Richards、シニアテクニカルプログラムマネージャー、アマゾン ウェブ サービス

その他の資料

詳細については、以下を参照してください。

- AWS Well-Architected フレームワーク

ドキュメント改訂履歴

日付	説明
2020 年 4 月	以下を含む、大幅な更新と新規コンテンツ/改訂されたコンテンツ。 • 「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 • リソースの更新 • 最新の AWS リソースとサービス (AWS Global Accelerator、AWS Service Quotas、AWS Transit Gateway など) を含むように更新 • 信頼性、可用性、弾力性の定義を追加/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) に合わせて調整されたホワイトペーパー

Archived

日付	説明
	• 設計原則の順序変更、**障害から自動的に復旧**を**テスト復旧手順**の前に移動 • 方程式の図と形式を更新 • 主要なサービスのセクションを削除し、主要な AWS のサービスへの参照をベストプラクティスに統合
2019 年 10 月	壊れたリンクを修正
2019 年 4 月	付録 A を更新
2018 年 9 月	AWS Direct Connect の具体的なネットワーク推奨事項とサービス設計目標を追加
2018 年 6 月	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。
2018 年 3 月	DynamoDB クロスリージョンソリューションを DynamoDB グローバルテーブルに変更 サービス設計目標を追加
2017 年 12 月	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
2017 年 11 月	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。
2016 年 11 月	初版発行

Archived

付録 A: 一部の AWS のサービスの可用性設計

一部の AWS のサービスが達成目標とする可用性について下記にまとめます。これらの値は、サービスレベルアグリーメント (SLA) または保証を表すものではなく、各サービスの設計目標に対するインサイトとなるものです。場合によっては、可用性の設計目標に意味のある差異が存在するサービス部分を区別します。このリストは、すべての AWS のサービスを包括するものではありません。また、追加サービスに関する情報で定期的に更新する予定です。Amazon CloudFront、Amazon Route 53、Identity and Access Management コントロールプレーンはグローバルサービスであり、それに応じてコンポーネントの可用性の目標が定められています。他のサービスは AWS リージョン内のサービスを提供し、それに応じて可用性の目標が定められています。多くのサービスがアベイラビリティーゾーン間で独立しています。このような場合、単一のアベイラビリティーゾーンに対する場合と、2 つまたはそれ以上のアベイラビリティーゾーンが使用される場合の可用性の設計目標を設定します。

注意: 以下の表の数値は耐久性 (データの長期保存) ではなく、可用性の数値 (データまたは関数へのアクセス) です。

サービス	コンポーネント	可用性の設計目標
Amazon API Gateway	コントロールプレーン	99.950%
	データプレーン	99.990%
Amazon Aurora	コントロールプレーン	99.950%
	シングル AZ データプレーン	99.950%
	マルチ AZ データプレーン	99.990%
AWS CloudFormation	サービス	99.950%

Archived

サービス	コンポーネント	可用性の設計目標
Amazon CloudFront	コントロールプレーン	99.900%
	データプレーン (コンテンツ配信)	99.990%
Amazon CloudSearch	コントロールプレーン	99.950%
	データプレーン	99.950%
Amazon CloudWatch	CW メトリクス (サービス)	99.990%
	CW イベント (サービス)	99.990%
	CW ログ (サービス)	99.950%
AWS Database Migration Service	コントロールプレーン	99.900%
	データプレーン	99.950%
AWS Data Pipeline	サービス	99.990%
Amazon DynamoDB	サービス (標準)	99.990%
	サービス (グローバルテーブル)	99.999%
Amazon EC2	コントロールプレーン	99.950%
	シングル AZ データプレーン	99.950%
	マルチ AZ データプレーン	99.990%
Amazon ElastiCache	サービス	99.990%

Archived

サービス	コンポーネント	可用性の設計目標
Amazon Elastic Block Store	コントロールプレーン	99.950%
	データプレーン (容量の可用性)	99.999%
Amazon Elasticsearch	コントロールプレーン	99.950%
	データプレーン	99.950%
Amazon EMR	コントロールプレーン	99.950%
Amazon S3 Glacier	サービス	99.900%
AWS Glue	サービス	99.990%
Amazon Kinesis Data Streams	サービス	99.990%
Amazon Kinesis Data Firehose	サービス	99.900%
Amazon Kinesis Video Streams	サービス	99.900%
Amazon Neptune	サービス	99.900%
Amazon RDS	コントロールプレーン	99.950%
	シングル AZ データプレーン	99.950%
	マルチ AZ データプレーン	99.990%
Amazon Rekognition	サービス	99.980%
Amazon Redshift	コントロールプレーン	99.950%

Archived

サービス	コンポーネント	可用性の設計目標
	データプレーン	99.950%
Amazon Route 53	コントロールプレーン	99.950%
	データプレーン (クエリ解決)	100.000%
Amazon SageMaker	データプレーン (モデルホスティング)	99.990%
	コントロールプレーン	99.950%
Amazon S3	サービス (標準)	99.990%
AWS Auto Scaling	コントロールプレーン	99.900%
	データプレーン	99.990%
AWS Batch	コントロールプレーン	99.900%
	データプレーン	99.950%
AWS CloudHSM	コントロールプレーン	99.900%
	シングル AZ データプレーン	99.900%
	マルチ AZ データプレーン	99.990%
AWS CloudTrail	コントロールプレーン (設定)	99.900%
	データプレーン (データイベント)	99.990%

Archived

サービス	コンポーネント	可用性の設計目標
	データプレーン (管理イベント)	99.999%
AWS Config	サービス	99.950%
AWS Direct Connect	コントロールプレーン	99.900%
	単一ロケーションデータプレーン	99.900%
	マルチロケーションデータプレーン	99.990%
Amazon Elastic File System	コントロールプレーン	99.950%
	データプレーン	99.990%
AWS Identity and Access Management	コントロールプレーン	99.900%
	データプレーン (認証)	99.995%
AWS IoT Core	サービス	99.900%
AWS IoT Device Management	サービス	99.900%
AWS IoT Greengrass	サービス	99.900%
AWS Lambda	関数の呼び出し	99.950%
AWS Secrets Manager	サービス	99.900%
AWS Shield	コントロールプレーン	99.500%

Archived

サービス	コンポーネント	可用性の設計目標
	データプレーン (検出)	99.000%
	データプレーン (軽減)	99.900%
AWS Storage Gateway	コントロールプレーン	99.950%
	データプレーン	99.950%
AWS X-Ray	コントロールプレーン (コンソール)	99.900%
	データプレーン	99.950%
EC2 Container Service	コントロールプレーン	99.900%
	EC2 Container Registry	99.990%
	EC2 Container Service	99.990%
Elastic Load Balancing	コントロールプレーン	99.950%
	データプレーン	99.990%
キーマネジメントシステム (KMS)	コントロールプレーン	99.990%
	データプレーン	99.995%

Archived