AWS Well-Architected Framework

持続可能性の柱

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

持続可能性の柱: AWS Well-Architected Framework

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

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

要約とイントロダクション i
はじめに 1
クラウド持続可能性 2
責任共有モデル 3
クラウド上での持続可能性 3
クラウド上での持続可能性 4
クラウドによる持続可能性 4
クラウドでの持続可能性の設計原則 4
改善プロセス 6
サンプルシナリオ 7
改善の目標を特定する 7
リソース 7
具体的な改善点を評価する 7
プロキシメトリクス 8
ビジネスメトリクス 8
主要業績評価指標 9
改善を推定する 9
改善点を評価する 10
改善点の優先順位を付けて計画する 11
改善点をテストおよび検証する 12
変更を本稼働環境にデプロイする 13
成果を測定し、成功を再現する 13
非機能的な要件としての持続可能性 16
クラウドでの持続可能性のベストプラクティス 18
リージョンの選択 18
SUS01-BP01 ビジネス要件と持続可能性の目標の両方に基づいてリージョンを選択する 18
需要に合わせた調整 20
SUS02-BP01 ワークロードインフラストラクチャを動的にスケールする 20
SUS02-BP02 SLA を持続可能性の目標に合わせる 24
SUS02-BP03 未使用アセットの創出と維持の停止 25
SUS02-BP04 ネットワーク要件に基づいてワークロードの地理的配置を最適化する 27
SUS02-BP05 実行されるアクティビティに応じてチームメンバーのリソースを最適化する 30

SUS02-BP06 需要曲線を平坦化するためにバッファリングまたはスロットリングを実装する 31
ソフトウェアとアーキテクチャ 33
SUS03-BP01 非同期のジョブおよびスケジュールされたジョブ向けにソフトウェアとアーキテクチャを最適化する 34
SUS03-BP02 使用率が低い、またはまったく使用しないワークロードのコンポーネントを削除またはリファクタリングする 37
SUS03-BP03 時間やリソースを最も多く消費するコード領域を最適化する 38
SUS03-BP04 デバイスや機器への影響を最適化する 40
SUS03-BP05 データアクセスとストレージパターンのサポートが最も優れたソフトウェアパターンとアーキテクチャを使用する 42
データ管理 44
SUS04-BP01 データ分類ポリシーを実装する 45
SUS04-BP02 データのアクセスパターンとストレージパターンをサポートするテクノロジーを使用する 46
SUS04-BP03 ポリシーを使用してデータセットのライフサイクルを管理する 51
SUS04-BP04 伸縮性とオートメーションを使用してブロックストレージまたはファイルシステムを拡張する 54
SUS04-BP05 不要なデータや重複するデータを削除する 55
SUS04-BP06 共有ファイルシステムまたはストレージを使用して共通データにアクセスする 58
SUS04-BP07 ネットワーク間でのデータ移動を最小限に抑える 60
SUS04-BP08 データは再作成が難しい場合にのみバックアップする 62
ハードウェアとサービス 64
SUS05-BP01 ニーズに合わせて最小限のハードウェアを使用する 64
SUS05-BP02 影響が最も少ないインスタンスタイプを使用する 66
SUS05-BP03 マネージドサービスを使用する 70
SUS05-BP04 ハードウェアベースのコンピューティングアクセラレーターの使用を最適化する 72
プロセスと文化 74
SUS06-BP01 持続可能性の改善を迅速に導入できる方法を採用する 74
SUS06-BP02 ワークロードを最新に保つ 76
SUS06-BP03 ビルド環境の利用率を高める 78
SUS06-BP04 マネージド型 Device Farm を使用してテストする 79
まとめ 82
寄稿者 83

その他の資料 84
改訂履歴 85
注意 86
AWS Glossary 87

持続可能性の柱 - AWS Well Architected フレームワーク

公開日: 2023 年 10 月 3 日 (改訂履歴)

このホワイトペーパーでは、アマゾン ウェブ サービス (AWS) Well-Architected フレームワークの持続可能性の柱に焦点を当てます。AWS ワークロードの持続可能性目標を達成するために使用できる設計原則、運用ガイダンス、ベストプラクティス、潜在的なトレードオフ、および改善計画を提供します。

はじめに

AWS Well-Architected フレームワークは、お客様が AWS でワークロードを構築する際に選択技のメリットとデメリットを理解できるようにするためのものです。このフレームワークを利用すると、安全で信頼性および有効性が高く、コスト効率に優れた持続可能なワークロードを AWS クラウド で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。フレームワークにより、アーキテクチャをベストプラクティスに照らし合わせて一貫的に測定し、改善すべき点を特定する手段を提供します。Well-Architected ワークロードにより、ビジネス成果をサポートする能力が大幅に向上します。

このフレームワークは次の 6 つの柱に基づいています。

- 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
- サステナビリティ

本書は持続可能性の柱に焦点を合わせており、持続可能性のスコープ内では、環境的持続可能性に重点を置いています。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、技術面での役割を担う人物を対象としたツールです。

本書を読むことで、持続可能性を念頭に置いてクラウドアーキテクチャを設計するための AWS の最新の推奨事項と戦略を理解できます。本書の実践を採用することにより、効率を最大化して廃棄物を減少させるアーキテクチャを構築できます。

クラウド持続可能性

持続可能性を守ることで、お客さまのビジネスが環境、経済、社会に与える長期的な影響を解決します。それらの 国連の「環境と開発に関する世界委員会」は、 持続可能な開発を「将来の世代のニーズを満たす能力を損なうことなく、現在のニーズを満たす開発」と定義しています。 お客様の企業または組織が、環境によくない影響を与える可能性があります。直接的または間接的な炭素排出量、再利用できない廃棄物、清浄水などの共有資源に対するダメージなどです。

クラウドワークロードの構築において、持続可能性の実践とは、使用しているサービスの影響の理解、ワークロードのライフサイクル全体における影響の数値化、および設計原則とベストプラクティスの適用によるそれら影響の軽減化です。このドキュメントは、環境に対する影響、特にエネルギーの消費と効率性に焦点を当てています。アーキテクトにとって、リソースの使用量を削減するための直接的な対応がわかる重要な手段であるためです。

環境への影響を重視する場合、これらの影響が一般的にどのように計上されているか、また、組織自身の排出量計算への後続の影響について理解する必要があります。それらの 温室効果ガス (GHG) プロトコル は、炭素排出量を以下のスコープに分類し、AWS など、それぞれのスコープにおけるクラウドプロバイダーの関連する排出量の例を示しています。

- スコープ 1: 組織の活動から、または管理下にあるすべての直接排出量。例えば、データセンター用バックアップジェネレーターの燃料燃焼です。
- スコープ 2: データセンターなどで購入し使用する電力による間接的な排出量。例えば、商用発電からの排出量などです。
- スコープ 3: 管理できないソースからの組織の活動によるその他すべての間接的排出量。AWS の例には、データセンターの建設や、データセンターに設置される IT ハードウェアの製造や輸送に関連する排出量が含まれます。

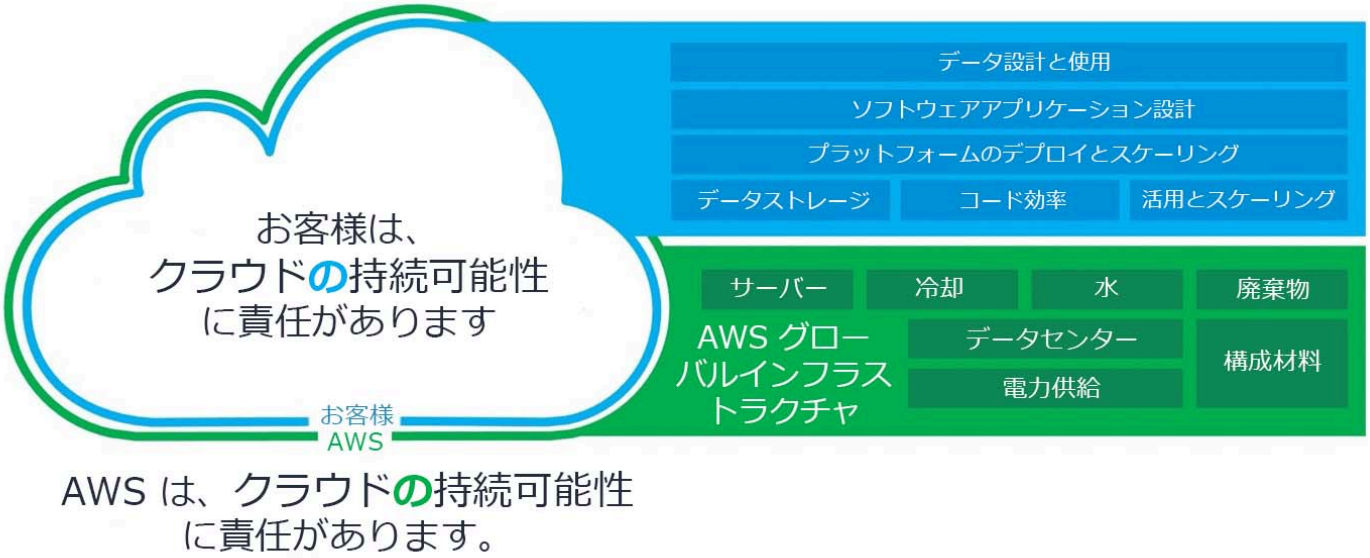
AWS のお客様の観点からみると、AWS で実行されるワークロードからの排出量は、間接的排出量、そしてスコープ 3 の排出量の一部とみなされます。展開された各ワークロードは、前のスコープそれぞれから合計した AWS 排出量の一部を生成します。ワークロード当たりの実際の量は、使用する AWS サービス、それらのサービスが使用するエネルギー量、AWS データセンターが稼働する際の電力網の二酸化炭素排出量、および再生可能エネルギーの AWS による調達などいくつかの要素によって異なります。

本書では最初に環境持続可能性の責任共有モデルについて説明し、次にアーキテクチャ上のベストプラクティスを提供します。そうすることで、AWS データセンターでそれらに必要な総リソースを減らすことになり、ワークロードの影響を最小限に抑えることができます。

責任共有モデル

環境持続可能性は、お客様と AWS の共有責任です。

- AWS は、 クラウドの持続可能性の最適化に責任があり、効率的で共有されたインフラストラクチャ、水資源管理、再生可能エネルギーの調達などを提供しています。
- お客様は、 クラウドにおける持続可能性に責任があり、ワークロードとリソース利用率を最適化し、ワークロードにデプロイする必要がある総リソースを最小限に抑えるようにします。



責任共有モデル

クラウド上での持続可能性

クラウドプロバイダーは、効率的な電力と冷却技術に投資し、エネルギー効率の高いサーバー群を運用し、高いサーバー利用率を実現しているため、一般的なオンプレミス型と比べて二酸化炭素排出量が少なく、エネルギー効率に優れています。クラウドワークロードは、ネットワーク、電力、冷却、物理設備などの共有リソースを利用することにより、影響を減らしています。より効率的なテクノロジーが利用可能になり次第、そこにクラウドワークロードを移行し、クラウドベースのサービスを使用してワークロードを変革し、持続可能性を高めることができます。

リソース

- Amazon Web Services へ移行することによる炭素削減の可能性

- AWS がサステナビリティソリューションを実現

クラウド上での持続可能性

クラウドでの持続可能性は、プロビジョニングされたリソースを最大限に活用し、必要な合計リソースを最小化することによって、ワークロードのすべてのコンポーネントにわたってエネルギーの削減と効率を主な主眼とした継続的取り組みです。この取り組みは、効率的なプログラミング言語の最初の選択から、最新のアルゴリズムの採用、効率的なデータストレージ技法の使用、適切にサイズ設定された効率的なコンピューティングインフラストラクチャへのデプロイ、高出力のエンドユーザーハードウェアの要件の最小化まで多岐にわたります。

クラウドによる持続可能性

デプロイしたワークロードの影響を最小化するだけではなく、AWS クラウド を使ってより幅広い持続可能性の課題に対応するよう設計されたワークロードを実行できます。これらの課題の例としては、二酸化炭素排出量の削減、エネルギー消費量の削減、水のリサイクル、ビジネスや組織の他の分野での廃棄物の削減などが挙げられます。

サステナビリティ から クラウド経由の持続可能性は、AWS テクノロジーを使ってより幅広い持続可能性の課題を解決するときに実現します。例えば、 Amazon Monitron などの機械学習ツールを使って、工業用機械で異常な動作を検出できます。この検出データを使って予防メンテナンスを行うことで、予期せぬ機器の故障による環境事故のリスクを低減し、機械が最高の効率で確実に稼働し続けることができます。

クラウドでの持続可能性の設計原則

クラウドワークロードを構築する際には、これらの設計原則を適用して、持続可能性を最大化して、影響を最小限に抑えます。

- 影響を理解する: クラウドワークロードの影響を計測し、ワークロードの将来の影響をモデル化します。顧客がお客様の製品を使用することによる影響、および最終的に製品を廃止および使用停止する際の影響などを含む、すべての影響源を含めます。作業単位ごとに必要なリソースと排出量を確認し、生産量と、クラウドワークロードの全影響を比較します。このデータを利用して重要業績評価指標 (KPI) を作成し、影響を抑えながら生産性を向上させる方法を評価して、提案された変更による影響を経時的に見積もります。
- 持続可能性の目標を設定する: クラウドワークロードごとに、持続可能性の長期目標を立てます。トランザクションごとのコンピューティングリソースやストレージリソースの削減などです。既存

のワークロードに対する持続可能性向上のための投資のリターンをモデル化し、持続可能性目標に必要な投資のリソースを所有者に与えます。成長計画を立て、その成長により、ユーザー単位やトランザクション単位など適した単位に対して計測される影響の大きさが結果的に削減できるようにワークロードを構築します。目標により、ビジネスや組織のより大きな持続可能性目標の支援、回帰の特定、改善できる可能性のある分野の優先順位付けが可能になります。

- 使用率を最大化する: ワークロードのサイズを適正化し効率的な設計を実装して、使用率を高く保ち、基盤となるハードウェアのエネルギー効率を最大化します。ホスト単位のベースライン電力消費量があるため、使用率 30% のホスト 2 つは、使用率 60% のホスト 1 つよりも効率が悪くなります。同時に、アイドル状態のリソース、処理、ストレージをなくすか、最小化して、ワークロードに必要な合計エネルギー量を削減します。
- より効率的なハードウェアやソフトウェアの新製品を予測して採用する: パートナーやサプライヤーが行っているアップストリームの改善をサポートし、お客様のクラウドワークロードへの影響の軽減に役立てます。より効率的なハードウェアやソフトウェアの新製品を継続的にモニターし評価します。新しい効率的なテクノロジーを迅速に採用できるように、設計に柔軟性を持たせます。
- マネージドサービスを使用する: 広範な顧客ベースでサービスを共有することで、リソースの使用率を最大化できます。こうすることで、クラウドワークロードをサポートするために必要なインフラストラクチャ数を削減できます。例えば、ワークロードを AWS クラウド に移行し、サーバーレスコンテナに AWS Fargate などのマネージドサービスを採用することで、電力やネットワークなど、データセンターに共通するコンポーネントの影響を顧客間で共有できます。マネージドサービスは、AWS が大規模に運用し、効率的なオペレーションについて責任を持ちます。お客様の影響を最小化できるマネージドサービスを使用します。例えば、Amazon S3 ライフサイクル設定を使用して、あまり頻繁にアクセスされていないデータを自動的にコールドストレージに移動したり、Amazon EC2 Auto Scaling を使用して容量を需要に合わせたりできます。
- クラウドワークロードのダウンストリームの影響を軽減する: お客様のサービスを使用するために必要なエネルギーやリソースの量を削減します。お客様のサービスを使用するために顧客がデバイスをアップグレードしなければならない必要性を削減または排除します。Device Farm を使用したテストで予想される影響を理解し、顧客とともにテストしてお客様のサービスを使用することによる実際の影響を理解します。

改善プロセス

アーキテクチャの改善プロセスには、現状と改善策の把握、改善対象の選択、改善のテスト、優れた改善策の採用、成功例の定量化、他でも再現できるように得られた知見の共有、サイクルの繰り返しがあります。

改善の目標には次のようなものがあります。

- 無駄、低利用率、アイドルまたは未使用のリソースを排除すること
- 消費するリソースから生まれる価値を最大化する

> **Note**
> プロビジョニングしたリソースをすべて使用し、最小限のリソースで同じ作業を完了する。

最適化の初期段階では、まず無駄が多く使用率が低い分野を排除し、次に特定のワークロードに合った、ターゲットを絞った最適化に移行します。

リソースの消費量の変化を経時的にモニターします。変化の蓄積によりリソース消費が非効率的または著しく増加している箇所を特定します。消費の変化を解決するために改善が必要かどうかを検討し、優先順位の高い改善を実装します。

次のステップは、クラウドワークロードの持続可能性に焦点を当てた改善点を評価し、優先順位を付け、テストし、デプロイする反復プロセスとして設計されています。

1. 改善の目標を特定する: この文書に記載されている持続可能性のためのベストプラクティスに照らし合わせて、ワークロードを見直し、改善目標を特定します。
2. 具体的な改善点を評価する: 潜在的な改善、予想されるコスト、ビジネスリスクに対する特定の変更を評価します。
3. 改善点の優先順位を付けて計画する: 最小のコストとリスクで最大の改善をもたらす変更に優先順位を付け、テストと実装のための計画を立てます。
4. 改善点をテストおよび検証する:: テスト環境での変更を実施し、改善の可能性を検証します。
5. 変更を本稼働環境にデプロイする: 本稼働環境全体に変更を実装します。
6. 成果を測定し、成功を再現する: ワークロード間で成功を再現する機会を探し、受け入れがたい結果が出た場合は変更を元に戻します。

サンプルシナリオ

この文書では、後述するシナリオ例を参照し、改善プロセスの各ステップを説明します。

貴社には Amazon EC2 インスタンスで複雑な画像操作を行い、ユーザーがアクセスできるように変更されたファイルと元のファイルを保存するワークロードがあります。アクティビティの処理は CPU に負荷がかかり、出力ファイルは非常に大きくなります。

改善の目標を特定する

持続可能性の目標を達成するためのベストプラクティスを理解します。これらのベストプラクティスと 改善のための推奨事項の詳しい説明は この文書の後半にあります。

ワークロードと使用するリソースをレビューします。識別 大規模なデプロイとよく使うリソースなどの ホットスポットを特定します。これらのホットスポットを評価することで、リソースの有効活用を改善し、ビジネス成果を達成するために必要な総リソースを削減する機会を得ることができます。

ワークロードをベストプラクティスに対してレビューし、改善点の候補を特定します。

このステップを サンプルシナリオに適用することで、改善点に考えられる目標として次のようなベストプラクティスを特定します。

- ニーズに合わせて最小限のハードウェアを使用する
- データアクセスとストレージパターンを万全にサポートするテクノロジーを使用する

リソース

- 持続可能性のために AWS インフラストラクチャを最適化する、パート I: コンピューティング
- 持続可能性のために AWS インフラストラクチャを最適化する、パート II: ストレージ
- 持続可能性のために AWS インフラストラクチャを最適化する、パート III: ネットワーキング

具体的な改善点を評価する

ワークロードによってプロビジョニングされたリソースを理解して、作業の単位を完了します。潜在的な改善点を評価して、潜在的な影響、実装のコスト、関連付けられたリスクを見積もります。

経時的な改善点を測定するには、AWS でプロビジョニングした内容と、それらのリソースがどのように消費されるかをまず理解します。

AWS の使用状況の全体的な概要を把握してから、AWS コストと使用状況レポートを使って、ホットスポットを特定します。この AWS サンプルコードを使って Amazon Athena を利用しながらレポートをレビューおよび分析します。

プロキシメトリクス

特定の変更を評価する際、その変更が関連リソースに及ぼす影響を最も定量的に把握できるのはどのメトリクスなのかを評価する必要もあります。これらのメトリクスは プロキシメトリクスと呼ばれています。.評価する改善点のタイプに最も当てはまるプロキシメトリクスと、改善の対象となるリソースを選択します。これらのメトリクスは経時的に進化します。

ワークロードをサポートするためにプロビジョニングされたリソースには、コンピューティング、ストレージ、およびネットワークリソースが含まれます。プロキシメトリクスを使ってプロビジョニングされたリソースを評価して、それらのリソースがどのように消費されるかを確認します。

プロキシメトリクスを使って、ビジネス成果を達成するためにプロビジョニングされたリソースを測定します。

リソース	プロキシメトリクスの例	改善目標
コンピューティング	vCPU 分	プロビジョニングされたリソースの使用率を最大化する
ストレージ	プロビジョニングされた GB	プロビジョニング合計を減らす
ネットワーク	転送された GB または転送されたパケット数	転送合計および転送距離を減らす

ビジネスメトリクス

ビジネス成果の達成を定量化するビジネスメトリクスを選択します。ビジネスメトリクスは、ワークロードが提供する価値、たとえば、同時アクティブユーザー数、提供された API コール数、完了したトランザクション数などを反映するものである必要があります。これらのメトリクスは経時的に進化する可能性があります。財務ベースのビジネスメトリクスを評価する際は注意してください。トランザクションの価値に一貫性がないと比較が無効になります。

主要業績評価指標

以下の式を用いて、プロビジョニングされたリソースを達成したビジネス成果で割ることで、単位作業あたりのプロビジョニングされたリソースを決定します。

$$\text{作業単位当たりのプロビジョニングされたリソース} = \frac{\text{プロビジョニングされたリソースのプロキシメトリックス}}{\text{結果のビジネスメトリックス}}$$

KPI 式

作業単位当たりのリソースを KPI として使用します。比較の基準としてプロビジョニングされたリソースに基づいてベースラインを確立します。

リソース	KPI 例	改善目標
コンピューティング	トランザクション当たりの vCPU 分数	プロビジョニングされたリソースの使用率を最大化する
ストレージ	トランザクション当たりの GB	プロビジョニング合計を減らす
ネットワーク	トランザクション当たりの転送 GB またはトランザクション当たりの転送パケット数	転送合計および転送距離を減らす

改善を推定する

改善は、プロビジョニングされたリソースの量的削減 (プロキシメトリクスによって示される) と、作業単位当たりプロビジョニングされたリソースの基準値からの変化の割合の両方で見積もられます。

リソース	KPI 例	改善目標
コンピューティング	トランザクション当たりの vCPU 分の削減 %	使用率を最大化する

リソース	KPI 例	改善目標
ストレージ	トランザクション当たりの GB の削減 %	プロビジョニング合計を減らす
ネットワーク	トランザクション当たりの転送 GB またはトランザクション当たりの転送パケット数の削減 %	転送合計および転送距離を減らす

改善点を評価する

予想される実質的な利益に対する潜在的な改善点を評価します。実装と維持の時間、コスト、工数レベル、そして想定外の影響などのビジネスリスクを評価します。

目標とする改善は、多くの場合、消費するリソースのタイプとのトレードオフを示します。例えば、コンピューティングの消費を抑えるために、結果を保存したり、転送されるデータを制限するために、結果をクライアントに送る前にデータを処理したりすることができます。ただし、 トレードオフについては 後にさらに説明します。

ワークロードのリスクを評価する際には、セキュリティ、信頼性、パフォーマンス効率、コストの最適化、改善によるワークロードの運用能力への影響など、非機能的要件も含めます。

このステップを サンプルシナリオ適用することで、次の結果を伴う目標の改善点を評価します。

ベストプラクティス	目標とする改善点	潜在的な	コスト	リスク
ニーズに合わせて最小限のハードウェアを使用する	予測スケーリングを実装して、使用率が低い期間を減らす	ミディアム	低	低
データアクセスとストレージパターンを万全にサポートするテ	より効果的な圧縮メカニズムを実装して、合計ストレージと達	高	低	低

ベストプラクティス	目標とする改善点	潜在的な	コスト	リスク
クノロジーを使用する	成までの時間を減らす			

予測スケジューリングを導入することにより、使用率の低いインスタンスや未使用のインスタンスが消費する vCPU 時間を削減し、既存のスケーリングメカニズムと比較して、消費するリソースを約 11% 削減する中程度の効果が得られました。関与するコストは低く、クラウドリソースの設定と Amazon EC2 Auto Scaling の予測スケーリングの運用が含まれます。リスクは、予測を超える需要に対して反応的にスケールアウトを行うと、パフォーマンスが制約されることです。

より効果的な圧縮を導入することで、オリジナル画像と加工したすべての画像でファイルサイズを大幅に削減し、制作に必要なストレージ容量を約 25% 削減できます。新しいアルゴリズムを実装することは、リスクが少なく労力の少ない代替案です。

改善点の優先順位を付けて計画する

最も低いコストと許容できるリスクで、最も大きな効果が期待できるものに基づいて、特定した改善点の優先順位を決定します。

最初にどの改善に重点を置くかを決め、リソース計画や開発ロードマップに盛り込みます。

このステップを サンプルシナリオに適用して、目標の改善点を次のように優先します。

優先度	改善	潜在的な	コスト	リスク
1	より効果的な圧縮メカニズムを実装する	高	低	低
2	予測スケーリングを実装する	ミディアム	低	低

ファイル圧縮の更新は、可能性が高く低コスト、そしてリスクを伴うため、貴社にとって価値の高いターゲットであり、予測スケーリングの実装よりも優先されます。ファイル圧縮が完了したら、潜在

的影響が中程度で低コスト、そしてリスクが低い予測スケーリングを実装することを優先的に改善すべきと判断します。

ファイル圧縮の改善と予測スケーリングをバックログに追加するよう、チームメンバーを任命します。

改善点をテストおよび検証する

最小限の投資で小さなテストを実施し、大規模な取り組みのリスクを減らします。

テストと検証を行うためのコストとリスクを制限するため、テスト環境にワークロードの代表的コピーを導入する。事前定義された一連のテストトランザクションを実行し、プロビジョニングされたリソースを測定し、作業単位あたりの使用リソースを決定し、テストのベースラインを確立します。

目標とする改善策をテスト環境に導入し、同じ条件下で同じ手法でテストを繰り返します。次に、改善した状態で、プロビジョニングされたリソースと作業単位当たりの使用リソースを測定します。

作業単位当たりのプロビジョニングされたリソースのベースラインからの変化率を計算し、本稼働環境でプロビジョニングされたリソースにおいて予想される量的削減を決定します。これらの値を予想される値と比較します。結果が、満足いく改善レベルかどうかを判断します。消費される追加リソースのトレードオフにより、改善による実質的な利益が受け入れられなくなるかどうかを評価します。

改善が成功であるかどうか、また、その変更を本番に導入するためにリソースを投入すべきかどうかを判断します。もし、この時点で変更が失敗と評価された場合は、次の目標のテストと検証にリソースを振り向け、改善サイクルを継続してください。

作業単位当たりのプロビジョニングされたリソースの削減 %	プロビジョニングされたリソースにおける量的削減	アクション
期待通り	期待通り	改善を続行
期待に満たなかった	期待通り	改善を続行
期待通り	期待に満たなかった	他の改善方法を模索
期待に満たなかった	期待に満たなかった	他の改善方法を模索

このステップを サンプルシナリオテストを実行し、成功を検証します。

改善した圧縮アルゴリズムでテストを行った結果、作業単位当たりのプロビジョニングされたリソース (オリジナル画像と修正画像の両方に必要なストレージ) の削減率は平均 30% と期待通りであり、計算負荷の増加は無視できる程度でした。

改善された圧縮アルゴリズムを実運用中の既存ファイルに適用するのに必要な追加コンピューティングリソースは、達成されたストレージの削減と比較して重要ではないと判断します。必要なリソースにおける量的削減の成功 (ストレージの TB) が確認され、この改善は本番環境へのデプロイが承認されました。

変更を本稼働環境にデプロイする

テスト、検証、および承認された改善点を本稼働環境に対して実装します。限定的なデプロイで実装し、ワークロードの機能を確認し、プロビジョニングされたリソースと単位作業あたりの消費リソースが限定的なデプロイの中で実際に削減されているかをテストし、変更による想定外の結果がないかどうかをチェックします。テスト成功後、完全なデプロイに進みます。

テストが失敗したり、変更による想定外の結果が受け入れられないものであったりした場合は、変更を元に戻します。

このステップを サンプルシナリオ以下のアクションを実行してください。

ブルー/グリーンデプロイ方式による限定的デプロイを使って、本稼働環境に変更を実装します。新たにデプロイしたインスタンスに対する機能テストが成功しました。オリジナルと操作された画像ファイルのプロビジョニングされたストレージが平均 26% 削減されていることがわかります。新しいファイルを圧縮しても、コンピューティング負荷が増えるというエビデンスはありません。

画像ファイルの圧縮にかかる時間が予想外に短くなったのは、新しい圧縮アルゴリズムのコードが高度に最適化されたためと思われます。

新しいバージョンの完全デプロイに進みます。

成果を測定し、成功を再現する

次の方法で結果を測定し、成功を再現します:

- 作業単位当たりのプロビジョニングされたリソースに対する初期改善と、プロビジョニングされたリソースにおける量的削減を測定します。
- 最初の推定値とテスト結果を、本番の測定値と比較します。差異を生じさせた可能性のある要因を特定し、必要に応じて推定やテスト方法を更新します。
- 成功、成功の程度を決定し、利害関係者と結果を共有します。

- テストの失敗や変更による想定外の否定的な結果が原因で変更を元に戻す場合、その要因を特定します。実行可能な場合は反復し、変更の目標を達成するための新しいアプローチを評価します。
- 学んだことを標準化し、成功した改善を他のシステムにも適用することで、同じように利益を得ることができます。方法論、関連する成果物、および実際の利益を記録し、チームや組織全体で共有することで、他の人々があなたの基準を採用し、成功を再現できるようにします。
- 作業単位ごとにプロビジョニングされたリソースをモニタリングし、時間の経過に伴う変化と総影響をトラッキングします。ワークロードの変化、あるいはお客様のワークロードの消費方法の変化は、改善の効果に影響を与える可能性があります。改善の効果が短期的に著しく低下した場合、または時間の経過とともに効果が累積的に低下した場合は、改善機会を再評価してください。
- 改善から得られる正味利益を長期にわたって定量化し、改善活動からの投資対効果を示します (可能であれば、その改善を適用した他のチームが受けた利益も含みます)。

このステップを サンプルシナリオに適用することで、次の結果を測定します。

お客様のワークロードでは、既存の画像ファイルに新しい圧縮アルゴリズムを導入、適用した後、ストレージ要件が 23% 削減されるという初期改善が見られました。

測定値は初期推定値 (25%) とおおよそ一致しており、テスト値 (30%) と比較したときの有意差はテストで使用した画像ファイルが本稼働環境の画像ファイルを代表していない結果と判断されます。テスト用の画像セットを修正し、本稼働環境の画像をより適切に反映させることができます。

改善は完全な成功とみなされました。プロビジョニングされたストレージの総削減量は、推定 25% より 2% 少ないですが、23% は持続可能性への影響においてやはり大きな改善であり、同等のコスト削減を伴います。

変更の唯一の想定外の結果は、圧縮を実行するためにかかる時間の有益な削減と、同等の消費 vCPU 削減です。これらの改善は、高度に最適化されたコードに起因するものです。

社内でオープンソースプロジェクトを立ち上げ、コード、関連成果物、変更の実装方法に関するガイダンス、実装結果を共有します。社内でオープンソースプロジェクトを立ち上げることにより、お客様のチームが永続的ファイルストレージのすべてのユースケースにコードを採用することが容易になります。チームはその改善策を標準として採用します。社内オープンソースプロジェクトの二次的な利点は、そのソリューションを採用した全員がそのソリューションの改良の恩恵を受けられること、そして誰でもプロジェクトへの改善に貢献できることです。

成功事例を公開し、オープンソースプロジェクトを組織全体で共有するのです。このソリューションを採用したすべてのチームは、最小限の投資でその利益を再現し、投資から受ける純利益を増やすことができます。このデータを継続的な成功事例として公開します。

経時的な改善の影響を引き続きモニタリングし、必要に応じて社内オープンソースプロジェクトに変更を加えます。

非機能的な要件としての持続可能性

ビジネス要件のリストに持続可能性を加えることで、より費用対効果の高いソリューションが実現できます。使用するリソースからより多くの価値を得ることに注力し、リソース量を減らすことは、使用した分のみ料金を支払うことになるため、AWS のコスト削減に直接つながります。

持続可能性の目標を達成するためには、アップタイム、可用性、応答時間など、他の 1 つまたは複数の従来のメトリクスにおいて同等のトレードオフを必要としないかもしれません。サービスレベルに重大な影響を与えることなく、持続可能性を大幅に高めることができるのです。軽微なトレードオフが必要な場合、このトレードオフによって得られる持続可能性の向上は、サービスの質の変化を上回ります。

機能要件を開発する際には、持続可能性の改善について継続的に実験を行うよう、チームメンバーに働きかけてください。また、チームは、目標を設定する際にプロキシ メトリクスを組み込み、ワークロードを開発する際にリソースの集約度を評価するようにする必要があります。

次は、消費するクラウドリソースを削減するためのトレードオフの例です。

結果の品質を調整する: 近似コンピューティングを利用して、結果の品質 (QoR) とワークロード強度の低減を交換することができます。近似コンピューティングの実践は、顧客が必要とするものと実際に作成するものとの間のギャップを利用する機会を探します。たとえば、データを セット データ構造にした場合、SQL の ORDER BY 演算子を削除して不要な処理を削除し、リソースを節約しながらも、満足の行く回答を得ることができます。

応答時間を調整する: 応答速度が遅い回答は、共有のオーバーヘッドを最小にすることで炭素を削減することができます。アドホックなエフェメラルタスクを処理すると、起動オーバーヘッドが発生する場合があります。タスクを受信するたびにオーバーヘッドを支払う代わりに、タスクをグループ化してバッチ単位で処理を行います。バッチ処理では、インスタンスのスピンアップ、ソースコードのダウンロード、プロセスの実行などの共有オーバーヘッドを削減するために、応答時間の増加をトレードオフします。

可用性を調整する: AWS では、わずか数回のクリックで、簡単に冗長性を追加し、高可用性ターゲットを達成することができます。常に利用率が低下するアイドル状態のリソースをプロビジョニングすることで、静的安定性のような手法で冗長性を高めることができます。目標設定時にビジネスのニーズを評価します。可用性のトレードオフは比較的小さくても、利用率の増加が大幅に向上します。たとえば、静的安定アーキテクチャのパターンでは、コンポーネントに障害が発生した後、直ちに負荷を引き受けるために、アイドル状態のフェイルオーバー能力をプロビジョニングします。可用性の要件を緩和することで、代替リソースをデプロイするためのオートメーション時間を確保し、ア

イドル状態のオンライン性能を不要にすることができます。フェイルオーバーキャパシティーをオンデマンドで追加することで、通常のオペレーション中にビジネスに影響を与えることなく、全体的な使用率を高め、コスト削減という二次的利点が生じます。

クラウドでの持続可能性のベストプラクティス

ワークロードの配置を最適化し、需要、ソフトウェア、データ、ハードウェア、プロセスに合わせてアーキテクチャを最適化して、エネルギー効率を高めます。それぞれの分野で、クラウドワークロードの持続可能性に対する影響を削減するためのベストプラクティスがあります。使用率を最大化し、無駄や、ワークロードをサポートするためにデプロイされ電力を消費するリソース総量を最小化する方法です。

トピック

- リージョンの選択
- 需要に合わせた調整
- ソフトウェアとアーキテクチャ
- データ管理
- ハードウェアとサービス
- プロセスと文化

リージョンの選択

ワークロードのためのリージョンの選択は、パフォーマンス、コスト、カーボンフットプリントなどの KPI に大きく影響します。これらの KPI を効果的に改善するには、ビジネス要件と持続可能性の目標の両方に基づいて、ワークロードのリージョンを選択する必要があります。

ベストプラクティス

- SUS01-BP01 ビジネス要件と持続可能性の目標の両方に基づいてリージョンを選択する

SUS01-BP01 ビジネス要件と持続可能性の目標の両方に基づいてリージョンを選択する

パフォーマンス、コスト、カーボンフットプリントなどの KPI を最適化するために、ビジネス要件と持続可能性の目標の両方に基づいて、ワークロードのリージョンを選択します。

一般的なアンチパターン:

- 自分の場所に基づいてワークロードのリージョンを選択する。

- すべてのワークロードリソースを 1 つの地理的場所に統合する。

このベストプラクティスを確立するメリット: Amazon の再生可能エネルギープロジェクトや公開されている炭素強度の低いリージョンの近くにワークロードを配置することで、クラウドワークロードのカーボンフットプリントを削減できます。

このベストプラクティスを確立しない場合のリスクレベル: 中

実装のガイダンス

AWS クラウド は、リージョンと Point of Presence (PoP) のネットワークを常に拡大し、それらをグローバルなネットワークインフラストラクチャでつないでいます。ワークロードのためのリージョンの選択は、パフォーマンス、コスト、カーボンフットプリントなどの KPI に大きく影響します。これらの KPI を効果的に改善するには、ビジネス要件と持続可能性の目標の両方に基づいて、ワークロードのリージョンを選択する必要があります。

実装手順

- 以下の手順に従って、コンプライアンス、利用可能な機能、コスト、レイテンシーなどのビジネス要件に基づき、ワークロードに適したリージョンを評価し、候補をリストアップします。
  - 必要なリージョンの規制に基づいて、これらのリージョンがコンプライアンスに準拠していることを確認します。
  - AWS リージョンサービスリストを使用して、ワークロードの実行に必要なサービスと機能をリージョンが備えているかどうかを確認します。
  - AWS Pricing Calculator を使用して、各リージョンのワークロードのコストを計算します。
  - エンドユーザーの拠点と各 AWS リージョン 間のネットワークレイテンシーをテストします。
- Amazon の再生可能エネルギープロジェクトに近いリージョンであり、グリッドの公開されている炭素集約度が他の場所 (またはリージョン) よりも低いリージョンを選択します。
  - 関連する持続可能性ガイドラインを特定し、温室効果ガスプロトコル (市場ベースとロケーションベースの方法) に基づいて年間の炭素排出量を追跡および比較します。
  - 炭素排出量の追跡に使用する方法に基づいてリージョンを選択します。持続可能性のガイドラインに基づいてリージョンを選択する方法の詳細については、持続可能性の目標に基づいてワークロードのリージョンを選択する方法を参照してください。

リソース

関連するドキュメント:

- 炭素排出量の推定の理解
- Amazon Around the Globe (世界中の Amazon)
- Renewable Energy Methodology (再生可能エネルギーの方法論)
- What to Consider when Selecting a Region for your Workloads (ワークロードに応じたリージョンを選択する際の注意点)

関連動画:

- Architecting sustainably and reducing your AWS carbon footprint (持続可能なアーキテクチャ設計と AWS カーボンフットプリントの削減)

需要に合わせた調整

ユーザーとアプリケーションがワークロードやその他のリソースを使用する方法によって、持続可能性の目標を達成するための改善点を特定できます。継続的に需要に合うようにインフラストラクチャをスケールし、ユーザーをサポートするために必要な最小リソースのみを使用していることを検証します。サービスレベルをお客様のニーズと整合させます。ユーザーとアプリケーションがリソースを消費するために必要なネットワークを制限できるようにリソースを配置します。未使用のアセットを削除します。チームメンバーには、ニーズをサポートし持続可能性への影響を最小限にするデバイスを提供します。

ベストプラクティス

- SUS02-BP01 ワークロードインフラストラクチャを動的にスケールする
- SUS02-BP02 SLA を持続可能性の目標に合わせる
- SUS02-BP03 未使用アセットの創出と維持の停止
- SUS02-BP04 ネットワーク要件に基づいてワークロードの地理的配置を最適化する
- SUS02-BP05 実行されるアクティビティに応じてチームメンバーのリソースを最適化する
- SUS02-BP06 需要曲線を平坦化するためにバッファリングまたはスロットリングを実装する

SUS02-BP01 ワークロードインフラストラクチャを動的にスケールする

クラウドの伸縮性を利用してインフラストラクチャを動的にスケールすることにより、需要に合わせてクラウドリソースを供給し、ワークロード容量の過剰なプロビジョニングを回避します。

一般的なアンチパターン:

- ユーザーの負荷に合わせてインフラストラクチャをスケールしない。
- 常に手動でインフラストラクチャをスケールする。
- スケーリングイベントの後、スケールダウンして元に戻すのではなく、キャパシティーを増加させたままにする。

このベストプラクティスを確立するメリット: ワークロードの伸縮性を設定およびテストすることで、クラウドリソースの供給を需要に効率的に一致させ、容量の過剰なプロビジョニングを回避できます。クラウドの伸縮性を利用して、需要の急増時や急増後に容量を自動的にスケールすることにより、ビジネス要件を満たすために必要となる適切な数のリソースのみを運用できます。

このベストプラクティスを確立しない場合のリスクレベル: 中

実装のガイダンス

クラウドは、需要の変化に対応するためのさまざまなメカニズムを通じて、リソースを動的に拡張または縮小する柔軟性を提供します。最適な形で需要と供給を一致させることで、ワークロードに対する環境の影響を最小限に抑えることができます。

需要が一定の場合も変動する場合もあり、管理面の負担を回避するには、メトリクスと自動化が必要になります。アプリケーションのスケールは、インスタンスのサイズを変更して垂直方向 (スケールアップ/スケールダウン)、インスタンス数を変更して水平方向 (スケールイン/スケールアウト)、あるいはこれらの組み合わせで調整できます。

リソースの需要と供給は、さまざまなアプローチで一致させることができます。

- ターゲット追跡アプローチ: スケーリングメトリクスを監視し、必要に応じて容量を自動的に増減します。
- 予測スケーリング: 日単位および週単位の傾向を見越してスケールします。
- スケジュールベースのアプローチ: 予測できる負荷の変化に従って、独自のスケーリングスケジュールを設定します。
- サービススケーリング: 設計によってネイティブにスケールされるか、機能として自動スケーリングを提供するサービス (サーバーレスなど) を選択します。

使用率が低い、または使用されていない期間を特定し、リソースをスケールして余分な容量を排除し効率性を改善します。

実装手順

- 伸縮性は、持っているリソースの供給を、それらのリソースに対する需要と一致させます。インスタンス、コンテナ、機能には、伸縮性のためのメカニズムがあり、自動スケーリングと組み合わせて、またはサービスの機能として提供されます。AWS では、ユーザー負荷が低い期間には迅速かつ簡単にワークロードをスケールダウンできるように、幅広い自動スケーリングメカニズムを提供しています。自動スケーリングメカニズムの例を次に示します。

Auto scaling mechanism	Where to use
Amazon EC2 Auto Scaling	アプリケーションのユーザー負荷を処理するために、Amazon EC2 インスタンスを適切な数に調整します。
Application Auto Scaling	Amazon EC2 だけでなく、個々の AWS サービス (Lambda 関数や Amazon Elastic Container Service (Amazon ECS) サービスなど) のリソースを自動的にスケールします。
Kubernetes Cluster Autoscaler	AWS の Kubernetes クラスターを自動的にスケールします。

- スケーリングに関する議論は、Amazon EC2 インスタンスや AWS Lambda 関数などのコンピューティングサービスに関連していることが一般的です。需要に合わせて、Amazon DynamoDB の読み取り/書き込みキャパシティユニットや Amazon Kinesis Data Streams シャードなどの非コンピューティングサービスに関する設定も検討してみてください。
- スケールアップまたはスケールダウンのメトリクスが、デプロイされているワークロードのタイプに対して検証されていることを確認します。動画トランスコーディングアプリケーションをデプロイしようとする場合、100% の CPU 使用率が想定されるため、プライマリメトリクスにするべきではありません。必要に応じて、カスタマイズされたメトリクス (メモリ使用率など) をスケーリングポリシーに使用することもできます。適切なメトリクスを選ぶには、Amazon EC2 の以下のガイダンスを考慮してください。
  - メトリクスは有効な利用率メトリクスでなければならず、インスタンスのどの程度ビジーかを記述する必要があります。
  - メトリクス値は、Auto Scaling グループ内のインスタンス数に比例して増減する必要があります。

- Auto Scaling グループの手動スケーリングではなく、動的スケーリングを使用します。また、動的スケーリングでターゲット追跡スケーリングポリシーを使用することをお勧めします。
- スケールアウトとスケールインの両方のイベントに対処できるようにワークロードをデプロイします。スケールインイベントのテストシナリオを作成して、ワークロードが期待どおりに動作し、ユーザーエクスペリエンスに影響しない (スティッキーセッションが失われない) ことを確認します。アクティビティ履歴を使用すると、Auto Scaling グループのスケーリングアクティビティを検証できます。
- ワークロードを評価して予測可能なパターンを見つけ、あらかじめわかっていた、および計画的な需要の変化を予測してプロアクティブにスケールします。予測スケーリングを使用すると、容量を過剰にプロビジョニングする必要がなくなります。詳細については、Amazon EC2 Auto Scaling を使用した予測スケーリングに関するブログを参照してください。

リソース

関連するドキュメント:

- Amazon EC2 Auto Scaling の使用を開始する
- 機械学習を利用した EC2 の予測スケーリング
- Amazon OpenSearch Service、Amazon Data Firehose、および Kibana を使用してユーザーの行動を分析する
- Amazon CloudWatch とは
- Amazon RDS での Performance Insights を使用した DB 負荷のモニタリング
- Amazon EC2 Auto Scaling での予測スケーリングのネイティブサポートの概要
- Karpenter の概要 - オープンソースの高性能 Kubermetes Cluster Autoscaler
- Amazon ECS クラスター Auto Scaling の詳細

関連動画:

- Build a cost-, energy-, and resource-efficient compute environment (コスト面、エネルギー面、リソース面で効率的なコンピューティング環境の構築)
- Better, faster, cheaper compute: Cost-optimizing Amazon EC2 (CMP202-R1) (より良く、より速く、より安価なコンピューティング: Amazon EC2 でのコストの最適化 (CMP202-R1))

関連する例:

- ラボ: Amazon EC2 Auto Scaling グループの例
- ラボ: Karpenter による自動スケーリングの実装

SUS02-BP02 SLA を持続可能性の目標に合わせる

持続可能性の目標に基づいてワークロードのサービスレベルアグリーメント (SLA) をレビューし最適化して、ビジネスニーズを満たしながらワークロードをサポートするために必要なリソースを最小化します。

一般的なアンチパターン:

- ワークロード SLA がわからない、またはあいまいである。
- SLA を可用性とパフォーマンスのためにのみ定義している。
- すべてのワークロードに同じ設計パターン (マルチ AZ アーキテクチャなど) を使用している。

このベストプラクティスを活用するメリット: SLA を持続可能性の目標に合わせることで、ビジネスニーズを満たしながら最適なリソース使用量を実現できます。

このベストプラクティスが確立されていない場合のリスクレベル: 低

実装のガイダンス

SLA は、クラウドワークロードで期待できるサービスのレベルを定義します。応答時間、可用性、データ保持などです。アーキテクチャ、リソース使用量、クラウドワークロードの環境への影響に関わります。定期的に SLA をレビューして、リソースの使用量を大幅に削減できる事項と、サービスレベルの許容できる範囲での低下をトレードオフします。

実装手順

- ビジネス要件を超えるのではなく満たしながら、持続可能性の目標をサポートする SLA を定義または再設計します。
- 持続可能性への影響を大幅に削減できる事項と、サービスレベルの許容できる範囲での低下をトレードオフします。
  - 持続可能性と信頼性: 可用性が高いワークロードはより多くのリソースを消費する傾向があります。
  - 持続可能性とパフォーマンス: より多くのリソースを使用してパフォーマンスを強化すると、環境への影響が大きくなる場合があります。

- 持続可能性とセキュリティ: 過剰に保護されたワークロードは環境への影響が大きくなる場合があります。
- ビジネスクリティカルな機能を優先し、クリティカルでない機能にはサービスレベル (応答時間や回復時間目標など) を引き下げる設計パターン (AWS のマイクロサービスなど) を使用します。

リソース

関連するドキュメント:

- AWS サービスレベルアグリーメント (SLA)
- Importance of Service Level Agreement for SaaS Providers (SaaS プロバイダーにとってのサービスレベルアグリーメントの重要性)

関連動画:

- Delivering sustainable, high-performing architectures (持続可能でパフォーマンスが高いアーキテクチャを実現する)
- Build a cost-, energy-, and resource-efficient compute environment (コスト面、エネルギー面、リソース面で効率的なコンピューティング環境の構築)

SUS02-BP03 未使用アセットの創出と維持の停止

ワークロードの未使用アセットを廃止して、需要をサポートするために必要なクラウドリソース数を削減し、無駄を最小限に抑えます。

一般的なアンチパターン:

- アプリケーションを分析して冗長または不要になったアセットを見つけていない。
- 冗長または不要になったアセットを削除していない。

このベストプラクティスを活用するメリット: 不要なアセットを削除すると、リソースが解放され、ワークロード全体の効率が向上します。

このベストプラクティスが確立されていない場合のリスクレベル: 低

実装のガイダンス

未使用のアセットは、ストレージ容量やコンピューティングパワーなどのクラウドリソースを消費します。このようなアセットを特定して排除することで、これらのリソースを解放できるため、クラウドアーキテクチャの効率性が向上します。事前コンパイル済みのレポート、データセット、静的イメージなどのアプリケーションアセットと、アセットのアクセスパターンを定期的に分析し、冗長性、低使用率、および廃止できそうなターゲットを特定します。このような冗長アセットを削除して、ワークロード内のリソースの無駄を削減します。

実装手順

- モニタリングツールを使用して、不要になった静的アセットを特定します。
- アセットを削除する前に、削除によるアーキテクチャに対する影響を評価します。
- 計画を立て、不要になったアセットは削除します。
- 重複して生成されるアセットは統合し、冗長プロセスを排除します。
- 不要なアセットをこれ以上生成および保存しないようにアプリケーションを更新します。
- 不要になったアセットをお客様に代わって管理しているサードパーティに、アセットの生成と保存を止めるように指示します。
- サードパーティに、お客様の代わりに生成されている冗長アセットを統合するように指示します。
- ワークロードを定期的に見直して、未使用のアセットを特定し削除します。

リソース

関連するドキュメント:

- Optimizing your AWS Infrastructure for Sustainability, Part II: Storage (サステナビリティのための AWS インフラストラクチャの最適化、パート II : ストレージ)
- AWS アカウントで不要になったアクティブなリソースを終了するにはどうすればよいですか。

関連動画:

- How do I check for and then remove active resources that I no longer need on my AWS アカウント? (AWS アカウントで不要になったアクティブなリソースを確認し削除する方法を教えてください)

SUS02-BP04 ネットワーク要件に基づいてワークロードの地理的配置を最適化する

ワークロード向けにネットワークトラフィックが経由しなければならない距離を削減できるクラウドのロケーションとサービスを選択し、ワークロードをサポートするために必要なネットワークリソースの総量を減らします。

一般的なアンチパターン:

- 自分の場所に基づいてワークロードのリージョンを選択する。
- すべてのワークロードリソースを 1 つの地理的場所に統合する。
- すべてのトラフィックが既存のデータセンターを通過する。

このベストプラクティスを活用するメリット: ワークロードをユーザーの近くに配置することで、ネットワーク上のデータ移動を減らし、環境負荷を低減しながら、最小限のレイテンシーを実現します。

このベストプラクティスを活用しない場合のリスクレベル: 中

実装のガイダンス

AWS クラウドインフラストラクチャは、リージョン、アベイラビリティーゾーン、プレイスメントグループ、および AWS Outposts および AWS ローカルゾーンといったエッジロケーションなどのロケーションオプションを中心に構築されています。これらのロケーションオプションは、アプリケーションコンポーネント、クラウドサービス、エッジネットワーク、オンプレミスのデータセンター間の接続を維持する役割を担っています。

ワークロードのネットワークアクセスパターンを分析して、このようなクラウドロケーションオプションの使用方法や、ネットワークトラフィックが経由する距離を減らす方法を特定します。

実装手順

- ワークロードのネットワークアクセスパターンを分析して、ユーザーがアプリケーションをどのように使用しているかを特定します。
  - ネットワーク活動に関するデータを収集するため、 Amazon CloudWatch および AWS CloudTrailのようなツールを使用します。
  - データを分析して、ネットワークアクセスパターンを特定します。

- 以下の主な要素に基づいて、ワークロードのデプロイに適切なリージョンを選択します。
  - 持続可能性目標: 以下で説明されています リージョンの選択.
  - データの場所: 大量のデータを使用するアプリケーション (ビッグデータや機械学習など) では、アプリケーションコードをできるだけデータの近くで実行してください。
  - ユーザーの場所: ユーザー向けアプリケーションの場合は、ワークロードの顧客に近いリージョン (または複数のリージョン) を選択します。
  - その他の制約: 以下で説明されているように、コストやコンプライアンスなどの制約を考慮します。 What to Consider when Selecting a Region for your Workloads (ワークロードに応じたリージョンを選択する際の注意点)。
- ローカルキャッシュまたは AWS キャッシュソリューション を、頻繁に使用するアセットに使用すると、パフォーマンスを向上させ、データ移動を削減し、環境への影響を低減できます。

サービス	使用する状況
Amazon CloudFront	画像、スクリプト、動画などの静的コンテンツだけでなく、API 応答やウェブアプリケーションなどの動的コンテンツのキャッシュに使用します。
Amazon ElastiCache	ウェブアプリケーションのコンテンツをキャッシュします。
DynamoDB Accelerator	DynamoDB テーブルにインメモリアクセラレーションを追加します。

- ワークロードのユーザーの近くでコードを実行できるサービスを使用します。

サービス	使用する状況
Lambda@Edge	オブジェクトがキャッシュにないときに開始される、コンピューティング負荷の高いオペレーションに使用します。
Amazon CloudFront 関数	HTTP リクエストまたはレスポンス操作など、短時間実行の関数で実行できるシンプルなユースケースに使用します。

サービス	使用する状況
AWS IoT Greengrass	接続されたデバイスのローカルコンピューティング、メッセージング、データキャッシュを実行します。

- 接続プーリングを使用して、接続の再利用を可能にし、必要なリソースを削減します。
- 永続的な接続や同期更新に依存しない分散されたデータストアを使用して、リージョンのユーザーに一貫性のあるサービスを提供します。
- 事前にプロビジョンされた静的ネットワーク容量を、共有の動的容量に置き換え、持続可能性に対するネットワーク容量の影響を他のサブスクライバーと共有します。

リソース

関連するドキュメント:

- Optimizing your AWS Infrastructure for Sustainability, Part III: Networking
- Amazon ElastiCache のドキュメント
- 「Amazon CloudFront とは」
- Amazon CloudFront の主な特徴

関連動画:

- Demystifying data transfer on AWS
- 次世代 Amazon EC2 インスタンスでのネットワークパフォーマンスのスケーリング

関連サンプル:

- AWS Networking Workshops
- 持続可能性を考慮したアーキテクチャ - ネットワーク間のデータ移動を最小限に抑える

SUS02-BP05 実行されるアクティビティに応じてチームメンバーのリソースを最適化する

チームメンバーに提供されるリソースを最適化することで、ニーズをサポートしながら環境の持続可能性への影響を最小限に抑えます。

一般的なアンチパターン:

- クラウドアプリケーションの全体的な効率性に関して、チームメンバーが使用するデバイスの影響を無視する。
- チームメンバーが使用するリソースを手動で管理および更新している。

このベストプラクティスを活用するメリット: チームメンバーのリソースを最適化すると、クラウド対応アプリケーションの全体的な効率性が向上します。

このベストプラクティスが確立されていない場合のリスクレベル: 低

実装のガイダンス

サービスを利用するためにチームメンバーが使用しているリソース、その予想ライフサイクル、および金融および持続可能性に対する影響を理解します。これらのリソースを最適化する戦略を策定します。例えば、レンダリングやコンパイルなどの複雑なオペレーションを、使用率が低く高性能な単一ユーザーのシステムで行うのではなく、使用率の高いスケーラブルなインフラストラクチャで行います。

実装手順

- 使用方法に合わせてワークステーションや他のデバイスをプロビジョンします。
- 仮想デスクトップとアプリケーションストリーミングを使用して、アップグレードとデバイス要件を制限します。
- プロセッサーやメモリの負荷が大きいタスクをクラウドに移動して、その伸縮性を活用します。
- デバイスのライフサイクルにおけるプロセスやシステムの影響を評価し、ビジネス要件を満たしながらデバイスを交換する必要性を最小限にするソリューションを選択します。
- デバイスのリモート管理を実装して出張を少なくします。
  - AWS Systems Manager Fleet Manager は、AWS やオンプレミスで実行されているノードをリモートで管理できる、統合ユーザーインターフェイス (UI) エクスペリエンスです。

リソース

関連するドキュメント:

- Amazon WorkSpaces とは
- Cost Optimizer for Amazon WorkSpaces (Amazon WorkSpaces の Cost Optimizer)
- Amazon AppStream 2.0 のドキュメント
- NICE DCV

関連動画:

- Managing cost for Amazon WorkSpaces on AWS (AWS での Amazon WorkSpaces のコストを管理する)

SUS02-BP06 需要曲線を平坦化するためにバッファリングまたはスロットリングを実装する

バッファリングやスロットリングは、需要曲線を平坦化し、ワークロードに必要なプロビジョンドキャパシティを削減します。

一般的なアンチパターン:

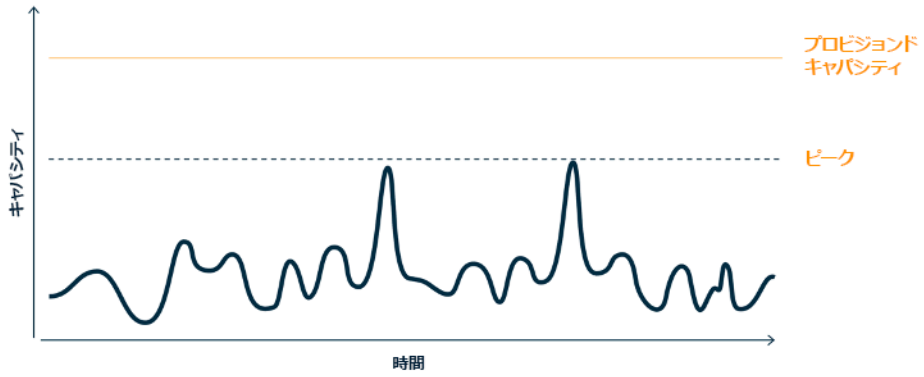
- 即時対応が不要なクライアントのリクエストを即時処理している。
- クライアントのリクエストの要件を分析していない。

このベストプラクティスを活用するメリット: 需要曲線を平坦化することで、ワークロードに必要なプロビジョンドキャパシティを削減できます。プロビジョンドキャパシティが削減されると、エネルギーの消費量が少なくなり、環境への影響が小さくなります。

このベストプラクティスが確立されていない場合のリスクレベル: 低

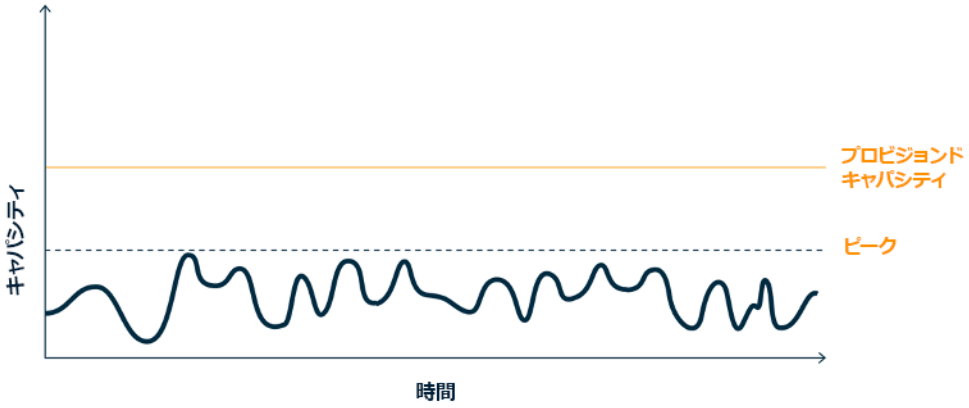
ワークロードの需要曲線を平坦化することで、ワークロードに必要なプロビジョンドキャパシティを削減し、環境への影響を減らすことができます。以下の図に示す需要曲線を持つワークロードがあるとします。このワークロードには 2 つのピークがあり、これらのピークを処理するために、オレンジの線で示されるリソース容量がプロビジョニングされます。このワークロードで使用されるリソースとエネルギーは需要曲線の下の領域ではなく、プロビジョンドキャパシティのラインの下の領域で

示されます。これら 2 つのピークを処理するには、プロビジョンドキャパシティが必要であるためです。



大きな容量をプロビジョニングする必要がある 2 つの大きなピークがある需要曲線

バッファリングやスロットリングを使用して需要曲線を変化させ、ピークをならすことができます。つまり、プロビジョンドキャパシティや消費されるエネルギーを減らすことができます。クライアントが再試行を実行できるときはスロットリングを実装します。バッファリングは、リクエストを保存し、後日まで処理を延期するために実装します。



需要曲線とプロビジョンドキャパシティに対するスロットリングの効果

実装手順

- クライアントのリクエストを分析して、それらに応答する方法を決定します。考慮すべき問題は以下のとおりです。
  - このリクエストは非同期で処理できるか?

  - クライアントは再試行できるか?
- クライアントが再試行できる場合、スロットリングを実装できます。これにより、現在リクエストを処理できない場合は、後で再試行する必要があることが送信元に通知されます。
  - Amazon API Gateway を使用してスロットリングを実装できます。
- 再試行できないクライアントの場合は、バッファを実装して需要曲線を平坦化する必要があります。バッファはリクエスト処理を延期し、アプリケーションが異なる動作速度で実行されていても効果的に通信できるようにします。バッファベースのアプローチでは、キューまたはストリーミングを使用して、プロデューサーからメッセージを受信します。メッセージはコンシューマーによって読み取られ、処理されるため、コンシューマーのビジネス要件を満たせる動作速度でメッセージを実行できます。
  - Amazon Simple Queue Service (Amazon SQS) は、1 人のコンシューマーが個別のメッセージを読むことができるキューを提供するマネージドサービスです。
  - Amazon Kinesis は、多数のコンシューマーが同じメッセージを読み取ることができるストリームを提供します。
- 全体的な需要、変化率、および要求される応答時間を分析して、必要なスロットルまたはバッファのサイズを適正化します。

リソース

関連するドキュメント:

- Getting started with Amazon SQS (Amazon SQS の開始方法)
- Application integration Using Queues and Messages (キューとメッセージを使用したアプリケーション統合)

関連動画:

- Choosing the Right Messaging Service for Your Distributed App (分散アプリケーションに適したメッセージングサービスを選択する)

ソフトウェアとアーキテクチャ

負荷平滑化を実行しデプロイされたリソースが一貫して高使用率で維持されるパターンを実装し、リソースの消費を最小化します。時間の経過とともにユーザーの行動が変化したため、コンポーネントが使用されずアイドル状態になることがあります。パターンとアーキテクチャを改定して、使用率の

低いコンポーネントを統合し、全体の使用率を上げます。不要になったコンポーネントは使用停止にします。ワークロードコンポーネントのパフォーマンスを理解し、リソースの消費が最も大きいコンポーネントを最適化します。顧客がお客様のサービスにアクセスするために使用するデバイスを把握し、デバイスをアップグレードする必要性を最小化するパターンを実装します。

ベストプラクティス

- SUS03-BP01 非同期のジョブおよびスケジュールされたジョブ向けにソフトウェアとアーキテクチャを最適化する
- SUS03-BP02 使用率が低い、またはまったく使用しないワークロードのコンポーネントを削除またはリファクタリングする
- SUS03-BP03 時間やリソースを最も多く消費するコード領域を最適化する
- SUS03-BP04 デバイスや機器への影響を最適化する
- SUS03-BP05 データアクセスとストレージパターンのサポートが最も優れたソフトウェアパターンとアーキテクチャを使用する

SUS03-BP01 非同期のジョブおよびスケジュールされたジョブ向けにソフトウェアとアーキテクチャを最適化する

キュー駆動型などの効率的なソフトウェアおよびアーキテクチャパターンを使用して、デプロイされたリソースの使用率を一貫して高く維持します。

一般的なアンチパターン:

- 予期せぬ需要の急増に対応するために、クラウドワークロードのリソースを過剰にプロビジョニングしています。
- お使いのアーキテクチャでは、メッセージングコンポーネントによって非同期メッセージの送信者と受信者が切り離されていません。

このベストプラクティスを活用するメリット:

- 効率的なソフトウェアとアーキテクチャのパターンは、ワークロード内の未使用リソースを最小限に抑え、全体的な効率を向上させます。
- 非同期メッセージの受信とは無関係に処理を拡張できます。
- メッセージングコンポーネントを使用することで、可用性要件が緩和され、より少ないリソースで対応できるようになります。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

イベント駆動型アーキテクチャなどの効率的なアーキテクチャパターンを使用することで、コンポーネントを均等に使用し、ワークロードの過剰プロビジョニングを最小限に抑えることができます。効率的なアーキテクチャパターンを使用することで、時間の経過に伴う需要の変化により、使用されずにアイドル状態になるリソースを最小限に抑えることができます。

ワークロードコンポーネントの要件を理解し、リソース全体の利用率を高めるアーキテクチャパターンを採用します。不要になったコンポーネントは廃止します。

実装手順

- ワークロードの需要を分析し、それらに対応する方法を決定します。
- 同期応答を必要としないリクエストやジョブには、キュー駆動型アーキテクチャとオートスケーリングワーカーを使用して使用率を最大化します。キュー駆動型アーキテクチャを検討する場合の例を次に示します。

Queuing mechanism	Description
AWS Batch ジョブキュー	AWS Batch ジョブはジョブキューに送信され、コンピューティング環境で実行されるようにスケジューリングされるまで、そこに留まります。
Amazon Simple Queue Service および Amazon EC2 スポットインスタンス	Amazon SQS とスポットインスタンスのペアリングにより、耐障害性が高く効率的なアーキテクチャを構築します。

- いつでも処理できるリクエストやジョブについては、スケジューリングメカニズムを利用してジョブをバッチ処理することで効率化を図ります。AWS でのスケジューリングメカニズムの例を次に示します。

Scheduling mechanism	Description
Amazon EventBridge Scheduler	スケジュールされたタスクを大規模に作成、実行、管理できるようにするための Amazon EventBridge の機能です。
AWS Glue の時間ベースのスケジュール	AWS Glue で、クローラーやジョブに対して時間ベースのスケジュールを定義します。
Amazon Elastic Container Service (Amazon ECS) のスケジュールされたタスク	Amazon ECS では、スケジュールされたタスクの作成をサポートします。スケジュールされたタスクは、Amazon EventBridge ルールを使用して、スケジュールに従って、または EventBridge イベントへの応答としてタスクを実行します。
Instance Scheduler	Amazon EC2 および Amazon Relational Database Service インスタンスの開始および停止スケジュールを設定します。

- ご使用のアーキテクチャでポーリングやウェブフックのメカニズムを使用している場合、それらをイベントに置き換えます。イベント駆動型アーキテクチャを使用して、効率性の高いワークロードを構築します。
- AWS のサーバーレスを活用し、過剰にプロビジョニングされたインフラストラクチャを排除します。
- アーキテクチャの個別のコンポーネントの適切なサイズを設定し、リソースが入力を待ってアイドル状態になるのを防ぎます。

リソース

関連するドキュメント:

- Amazon Simple Queue Service とは
- Amazon MQ とは
- Amazon SQS に基づいたスケーリング
- AWS Step Functions とは

- AWS Lambda とは
- AWS Lambda を Amazon SQS に使用する
- Amazon EventBridge とは

関連動画:

- Moving to event-driven architectures (イベント駆動型アーキテクチャへの移行)

SUS03-BP02 使用率が低い、またはまったく使用しないワークロードのコンポーネントを削除またはリファクタリングする

未使用のコンポーネントや不要になったコンポーネントを削除し、使用率の低いコンポーネントはリファクタリングして、ワークロードの無駄を最小化します。

一般的なアンチパターン:

- ワークロードの個別のコンポーネントの使用率レベルを定期的に確認していない。
- AWS Compute Optimizer のような AWS サイズ最適化ツールからのレコメンデーションを確認し分析しない。

このベストプラクティスを活用するメリット: 未使用のコンポーネントを削除すると、無駄が最小化され、クラウドワークロード全体の効率が向上します。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

ワークロードを見直して、アイドルや未使用のコンポーネントを特定します。これは、需要の変化や新しいクラウドサービスのリリースに伴う、反復的な改善プロセスです。例えば、AWS Lambda 関数の実行時間が大幅に低下した場合は、メモリーサイズを小さくする指標になり得ます。また、AWS で新しいサービスや機能がリリースされると、ワークロードに最適なサービスやアーキテクチャが変化する可能性があります。

ワークロードのアクティビティを継続的にモニターして、個別のコンポーネントの使用率レベルを改善する機会を見逃さないようにします。アイドルのコンポーネントを削除しアクティビティのサイズ最適化を行って、最小限のクラウドリソースでビジネス要件を満たすようにします。

実装手順

- ワークロードの重要なコンポーネントの使用率メトリクス (Amazon CloudWatch メトリクスの CPU 使用率、メモリ使用率、ネットワークスループットなど) をモニターし、キャプチャします。
- 安定したワークロードの場合は、AWS Compute Optimizer などの AWS サイズ最適化ツールを定期的に確認して、アイドル、未使用、または使用率の低いコンポーネントを特定します。
- 一次的なワークロードについては、使用率メトリクスを評価して、アイドル、未使用、または使用率の低いコンポーネントを特定します。
- 不要になったコンポーネントや関連アセット (Amazon ECR イメージなど) を廃止します。
- 使用率の低いコンポーネントをリファクタリングまたは他のリソースと統合して、使用効率を改善します。例えば、使用率の低い個別のインスタンスでデータベースを実行するのではなく、単体の Amazon RDS データベースインスタンスに複数の小さなデータベースをプロビジョニングできます。
- 作業の単位を完了するためにワークロードによってプロビジョニングされたリソースを理解します。

リソース

関連するドキュメント:

- AWS Trusted Advisor
- Amazon CloudWatch とは
- Automated Cleanup of Unused Images in Amazon ECR (Amazon ECR における未使用画像の自動クリーンアップ)

関連する例:

- Well-Architected Lab - Rightsizing with AWS Compute Optimizer(Well-Architected ラボ - Compute Optimizer を使用したサイズ最適化)
- Well-Architected Lab - Optimize Hardware Patterns and Observe Sustainability KPIs (Well-Architected ラボ - ハードウェアパターンの最適化と持続可能性 KPI の観察)

SUS03-BP03 時間やリソースを最も多く消費するコード領域を最適化する

アーキテクチャの異なるコンポーネント内で実行されているコードを最適化して、パフォーマンスを最大化しながらリソースの使用量を最小化します。

一般的なアンチパターン:

- リソースの使用量のためにコードを最適化しない。
- 通常、パフォーマンスの問題にはリソースを増やすことで対処している。
- コードの見直しおよび開発プロセスで、パフォーマンスの変化を追跡していない。

このベストプラクティスを活用するメリット: 効率的なコードは、リソースの使用量を最小化し、パフォーマンスを向上させます。

このベストプラクティスを活用しない場合のリスクレベル: 中

実装のガイダンス

クラウドに構築されたアプリケーションのコードを含むあらゆる機能領域を精査して、そのリソース使用量とパフォーマンスを最適化することが重要です。ビルド環境および本稼働環境でワークロードのパフォーマンスを継続的にモニターし、リソースの使用量が特に高いコードスニペットを改善する機会を特定します。定期的な見直しプロセスを導入して、コードの中でリソースを効率的に使用していないバグまたはアンチパターンを特定します。自分のユースケースに合わせて、同じ結果になるシンプルで効率的なアルゴリズムを活用します。

実装手順

- ワークロードを開発する際に、自動化されたコードレビュープロセスを導入して、品質を向上させ、バグやアンチパターンを特定します。
  - Automate code reviews with Amazon CodeGuru Reviewer (Amazon CodeGuru Reviewer を使用したコードレビューの自動化)
  - Detecting concurrency bugs with Amazon CodeGuru (Amazon CodeGuru を使用した並列バグの検出)
  - Raising code quality for Python applications using Amazon CodeGuru (Amazon CodeGuru を使用して Python アプリケーションのコード品質を向上させる)
- ワークロードを実行しながら、リソースをモニターして、作業単位でリソースを多く必要とするコンポーネントを特定し、コードレビューの対象とします。
- コードレビューでは、コードプロファイラーを使用して、時間またはリソースを最も多く使用するコードの領域を特定し、最適化の対象とします。
  - Reducing your organization's carbon footprint with Amazon CodeGuru Profiler (Amazon CodeGuru Profiler を使用して組織のカーボンフットプリントを削減する)

- Understanding memory usage in your Java application with Amazon CodeGuru Profiler (Amazon CodeGuru Profiler を使用して Java アプリケーションのメモリ使用量を理解する)
- Improving customer experience and reducing cost with Amazon CodeGuru Profiler (Amazon CodeGuru Profiler を使用してカスタマーエクスペリエンスを改善しコストを削減する)

- 最も効率的なオペレーティングシステムとプログラム言語をワークロードに使用します。エネルギー効率に優れたプログラム言語 (Rust など) の詳細については、 Sustainability with Rust を参照してください。
- コンピューティング負荷が高いアルゴリズムを、結果が同じであり、よりシンプルでより効率的なバージョンに置き換えます。
- ソートや書式設定などの不要なコードを削除します。

リソース

関連するドキュメント:

- Amazon CodeGuru Profiler とは?
- FPGA インスタンス
- The AWS SDKs on Tools to Build on AWS (AWS の構築ツールの AWS SDK)

関連動画:

- Improve Code Efficiency Using Amazon CodeGuru Profiler
- Automate Code Reviews and Application Performance Recommendations with Amazon CodeGuru

SUS03-BP04 デバイスや機器への影響を最適化する

アーキテクチャで使用されているデバイスや機器を理解し、それらの使用量を削減する戦略を使用します。これにより、環境に対するクラウドワークロードの全体的な影響を最小化できます。

一般的なアンチパターン:

- 顧客が使用するデバイスの環境に対する影響を無視する。
- 顧客が使用するリソースを手動で管理および更新している。

このベストプラクティスを活用するメリット: 顧客のデバイスに合わせて最適化されたソフトウェアパターンや機能を実装することで、クラウドワークロード全体の環境に対する影響を削減できます。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

顧客のデバイスに合わせて最適化されたソフトウェアパターンや機能を実装することで、複数の方法で環境に対する影響を削減できます。

- 後方互換性がある新機能を実装することで、ハードウェアの置換数を削減できます。
- アプリケーションを最適化してデバイスで効率的に実行できるようにすることで、エネルギー消費を削減し、バッテリー寿命を延ばすことができます (バッテリー駆動の場合)。
- また、アプリケーションをデバイスに合わせて最適化すると、ネットワーク経由のデータ転送も削減できます。

アーキテクチャで使用されているデバイスや機器、それらの予想ライフサイクル、およびそれらコンポーネントを置換した場合の影響を理解します。デバイスのエネルギー消費、顧客がデバイスを置換する必要性、およびデバイスを手動でアップグレードする必要性を最小限にできるソフトウェアパターンや機能を実装します。

実装手順

- アーキテクチャで使用されているデバイスをリストアップします。デバイスには、モバイル、タブレット、IoT デバイス、スマートライト、さらに工場のスマートデバイスも含まれます。
- デバイスで実行されているアプリケーションを最適化します。
  - バックグラウンドでのタスク実行などの戦略を使用して、エネルギーの消費量を削減します。
  - ペイロードを構築する際にネットワーク帯域幅とレイテンシーを考慮し、低帯域幅、高レイテンシーのリンクでもアプリケーションが問題なく動作できる能力を実装します。
  - ペイロードやファイルを、デバイスが必要とする最適な形式に変換します。例えば、Amazon Elastic Transcoder や AWS Elemental MediaConvert を使用して、サイズが大きい高品質デジタルメディアファイルを、ユーザーがモバイルデバイス、タブレット、ウェブブラウザ、コネクテッドテレビで再生できる形式に変換します。
  - コンピューティングの負荷が高いアクティビティはサーバー側 (画像のレンダリングなど) で実行するか、アプリケーションストリーミングを使用して、古い型のデバイスでのユーザーエクスペリエンスを改善します。

  - 特にインタラクティブセッションの場合は、出力を分割してページ番号を付け、ペイロードを管理しローカルストレージの要件を制限します。
- 自動化された無線通信 (OTA) の仕組みを使用して、1 つ以上のデバイスに更新をデプロイします。
  - モバイルアプリケーションの更新には、CI/CD パイプラインを使用できます。
  - AWS IoT Device Management を使用して、コネクテッドデバイスを大規模にリモートで管理できます。
- 新機能や更新をテストするには、ハードウェアの代表的なセットを備えたマネージド型 Device Farm を使用し、サポート対象のデバイスを最大化する開発を繰り返します。詳細については、SUS06-BP04 マネージド型 Device Farm を使用してテストする を参照してください。

リソース

関連するドキュメント:

- AWS Device Farm とは何ですか?
- Amazon AppStream 2.0 のドキュメント
- NICE DCV
- FreeRTOS を実行するデバイスでファームウェアを更新する OTA チュートリアル

関連動画:

- Introduction to AWS Device Farm(AWS Device Farm の紹介)

SUS03-BP05 データアクセスとストレージパターンのサポートが最も優れたソフトウェアパターンとアーキテクチャを使用する

データがどのようにワークロード内で使用されているか、ユーザーに消費されているか、転送されているか、保存されているかを理解します。データへのアクセスと保存を最適にサポートするソフトウェアパターンとアーキテクチャを使用して、ワークロードのサポートに必要なコンピューティング、ネットワーク、ストレージのリソースを最小化します。

一般的なアンチパターン:

- すべてのワークロードのデータの保存とアクセスのパターンが類似していると考えている。

- ストレージ階層を 1 つだけ使用し、すべてのワークロードがその階層に適していると考えている。
- 時間が経過してもデータアクセスパターンが変わらないと考えている。
- アーキテクチャはデータアクセスの高バーストの可能性をサポートしているが、その結果リソースがほとんどの時間でアイドルのままになる。

このベストプラクティスを活用するメリット: データのアクセスパターンおよびストレージパターンにもとづいてアーキテクチャを選択し最適化すると、開発の複雑性が下がり、全体の使用率が増加します。グローバルテーブル、データのパーティショニング、キャッシュをいつ使用するべきかを理解することで、運用上の諸経費を減らし、ワークロードのニーズに応じてスケーリングできるようになります。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

データの特性やアクセスパターンに最も合うソフトウェアやアーキテクチャのパターンを使用します。例えば、お客様独自の分析ユースケースに合わせて最適化された目的別サービスを使用できる、AWS のモダンデータアーキテクチャを使用します。このようなアーキテクチャパターンを使用すると、データ処理が効率的になり、リソースの使用量を削減できます。

実装手順

- データの特性やアクセスパターンを分析して、クラウドリソースに最適な構成を特定します。考慮する主な特徴には次のものがあります。
  - データタイプ: 構造、半構造、非構造
  - データ成長: 制限あり、制限なし
  - データ保存期間: 永続、一時的、一過性
  - アクセスパターン: 読み取りまたは書き取り、更新頻度、急増、または安定
- データアクセスとストレージパターンのサポートが最も優れたアーキテクチャパターンを使用します。
  - Let's Architect! Modern data architectures (構築してみよう! モダンデータアーキテクチャ)
  - Databases on AWS: The Right Tool for the Right Job (AWS のデータベース: 適材適所で使い分けるツール)
- 圧縮データをネイティブに操作するテクノロジーを使用します。
- アーキテクチャ内のデータ処理に、目的別の分析サービスを使用します。

- 主要なクエリパターンに対して最も優れたサポートをするデータベースエンジンを使用します。データベースインデックスを管理して、効率的なクエリ実行を確実にします。詳細については、AWS のデータベースを参照してください。
- アーキテクチャで消費されるネットワーク容量が削減できるネットワークプロトコルを選択します。

リソース

関連するドキュメント:

- Athena でサポートされる圧縮ファイル形式
- Amazon Redshift を使用した列指向のデータ形式からの COPY
- Converting Your Input Record Format in Firehose (Kinesis Data Firehose の入力レコード形式を変換する)
- AWS Glue の ETL 入出力の形式オプション
- 列指向形式に変換して Amazon Athena でのクエリパフォーマンスを改善する
- Amazon Redshift を使用して圧縮されたデータファイルを Amazon S3 からロードする
- Amazon Aurora での Performance Insights を使用した DB 負荷のモニタリング
- Amazon RDS での Performance Insights を使用した DB 負荷のモニタリング
- Amazon S3 Intelligent-Tiering storage class (Amazon S3 Intelligent-Tiering ストレージクラス)

関連動画:

- Building modern data architectures on AWS(AWS にモダンデータアーキテクチャを構築する)

データ管理

データ管理プラクティスを実装して、ワークロードのサポートに必要なプロビジョンされたストレージと、それを使用するために必要なリソースを削減します。データを理解し、データのビジネス価値とデータの使用方法を最もよくサポートするストレージテクノロジーと設定を使用します。必要性が小さくなった場合はより効率的で性能を落としたストレージにデータをライフサイクルし、データが不要になった場合は削除します。

ベストプラクティス

- SUS04-BP01 データ分類ポリシーを実装する

- SUS04-BP02 データのアクセスパターンとストレージパターンをサポートするテクノロジーを使用する
- SUS04-BP03 ポリシーを使用してデータセットのライフサイクルを管理する
- SUS04-BP04 伸縮性とオートメーションを使用してブロックストレージまたはファイルシステムを拡張する
- SUS04-BP05 不要なデータや重複するデータを削除する
- SUS04-BP06 共有ファイルシステムまたはストレージを使用して共通データにアクセスする
- SUS04-BP07 ネットワーク間でのデータ移動を最小限に抑える
- SUS04-BP08 データは再作成が難しい場合にのみバックアップする

SUS04-BP01 データ分類ポリシーを実装する

データを分類してビジネス成果に対する重要度を理解し、データの保存にエネルギー効率の高い適切なストレージ層を選択します。

一般的なアンチパターン:

- 処理または保存されているデータアセットの中で、類似の特徴 (機密度、ビジネス上の重要度、規制要件など) を持つものを特定していない。
- データアセットのインベントリにデータカタログを実装していない。

このベストプラクティスを活用するメリット: データ分類ポリシーを実装すると、データに対して最もエネルギー効率の高いストレージ層を検討できます。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

データ分類には、組織が所有または運用する情報システムで処理中または保存中のデータのタイプの特定を含めます。また、データの重要度と、データの侵害、損失、誤使用によって考えられる影響についても検討します。

データ分類ポリシーは、データを使用する流れから逆算して実装し、あるデータセットの組織の運営における重要度のレベルを考慮に入れて、カテゴリ分けのスキームを作成します。

実装手順

- ワークロードに存在するさまざまなデータタイプのインベントリを実施します。

  - データ分類カテゴリの詳細については、Data Classification whitepaper (データ分類ホワイトペーパー) をご覧ください。
- 組織に対するリスクにもとづいて、データの重要度、機密度、整合性、可用性を判断します。このような要件を使用して、導入するデータ分類層のいずれかにデータをグループ分けします。
  - 例として、Four simple steps to classify your data and secure your startup (データを分類しスタートアップを保護する 4 つのシンプルなステップ) を参照してください。
- 環境を定期的に監査してタグ付けおよび分類されていないデータを探し、そのデータを適切に分類してタグ付けします。
  - 例として、AWS Glue のデータカタログとクローラーを参照してください。
- 監査およびガバナンス機能があるデータカタログを作成します。
- データクラスごとに処理手順を決定して文書化します。
- 自動化を使用し、環境を継続的に監査してタグ付けおよび分類されていないデータを探し、そのデータを適切に分類してタグ付けします。

リソース

関連するドキュメント:

- Leveraging AWS クラウド to Support Data Classification (AWS クラウドを活用したデータ分類のサポート)
- AWS Organizations のタグポリシー

関連動画:

- Enabling agility with data governance on AWS(AWS 上でのデータガバナンスで俊敏性を実現する)

SUS04-BP02 データのアクセスパターンとストレージパターンをサポートするテクノロジーを使用する

データへのアクセス方法や保存方法を最もよくサポートするストレージ技術を使用し、ワークロードをサポートしながらプロビジョニングされるリソースを最小化します。

一般的なアンチパターン:

- すべてのワークロードのデータの保存とアクセスのパターンが類似していると考えている。

- ストレージ階層を 1 つだけ使用し、すべてのワークロードがその階層に適していると考えている。
- 時間が経過してもデータアクセスパターンが変わらないと考えている。

このベストプラクティスを活用するメリット: データのアクセスとストレージのパターンに基づいてストレージ技術を選択し最適化すると、ビジネスニーズを満たすために必要なクラウドリソースが削減し、クラウドワークロードの全体的な効率が向上します。

このベストプラクティスを活用しない場合のリスクレベル: 低

実装のガイダンス

アクセスパターンに最適なストレージソリューションを選択するか、パフォーマンス効率を最大にするためにストレージソリューションに合わせてアクセスパターンを変更することを検討してください。

- データの特徴とアクセスパターンを評価し、ストレージのニーズにおける主な特徴を収集します。考慮する主な特徴には次のものがあります。
  - データタイプ: 構造、半構造、非構造
  - データの増加: 制限あり、制限なし
  - データの耐久性: 永続、一過性、一時的
  - アクセスパターン: 読み取りまたは書き込み、頻度、急増、または安定
- データの特徴とアクセスパターンをサポートする適切なストレージ技術にデータを移行します。AWS ストレージ技術とその主な特徴を例としていくつか挙げます。

タイプ	テクノロジー	主な特徴
オブジェクトストレージ	Amazon S3	無制限のスケーラビリティ、高可用性、およびアクセシビリティに関して複数のオプションがあるオブジェクトストレージサービスです。Amazon S3 内外のオブジェクトの転送やアクセスには、Transfer Acceleration または Access Points などのサー

タイプ	テクノロジー	主な特徴
		ビスを使用して、ロケーション、セキュリティニーズ、アクセスパターンをサポートします。
アーカイブストレージ	Amazon S3 Glacier	データアーカイブのために構築された Amazon S3 のストレージクラスです。
共有ファイルシステム	Amazon Elastic File System (Amazon EFS)	さまざまなタイプのコンピューティングソリューションからアクセスできる、マウント可能なファイルシステムです。Amazon EFS は、ストレージを自動的に拡大/縮小し、パフォーマンスに最適化されているため、一貫した低レイテンシーを実現します。
共有ファイルシステム	Amazon FSx	最新の AWS コンピューティングソリューションをベースに構築されており、一般的に使用されている 4 つのファイルシステム (NetApp ONTAP、OpenZFS、Windows File Server、Lustre) をサポートしています。Amazon FSx レイテンシー、スループット、および IOPS は ファイルシステムごとに異なるため、ワークロードのニーズに合わせて適切なファイルシステムを選択する際に考慮する必要があります。

タイプ	テクノロジー	主な特徴
ブロックストレージ	Amazon Elastic Block Store (Amazon EBS)	Amazon Elastic Compute Cloud (Amazon EC2) 向けに設計された、スケーラブルで高パフォーマンスのブロックストレージサービスです。Amazon EBS には、処理が多く IOPS が高いワークロード向けの SSD 搭載ストレージと、スループットが高いワークロード向けの HDD 搭載ストレージが含まれます。
リレーショナルデータベース	Amazon Aurora、Amazon RDS、Amazon Redshift	ACID (atomicity、consistency、isolation、durability) トランザクションをサポートし、参照整合性と強固なデータ整合性を維持するように設計されています。従来のアプリケーション、エンタープライズリソースプランニング (ERP)、顧客関係管理 (CRM)、e コマースシステムの多くは、リレーショナルデータベースを使用してデータを保存します。

タイプ	テクノロジー	主な特徴
key-value データベース	Amazon DynamoDB	一般的に大量のデータを保存および取得するために、一般的なアクセスパターン用に最適化されています。高トラフィックのウェブアプリケーション、e コマースシステム、ゲーミングアプリケーションは、key-value データベースの典型的なユースケースです。

- Amazon EBS や Amazon FSx など固定サイズのストレージシステムの場合、利用可能なストレージ容量をモニタリングして、しきい値に達した場合のストレージ割り当てを自動化します。Amazon CloudWatch を活用して、 Amazon EBS および Amazon FSx のさまざまなメトリクスを収集および分析できます。
- Amazon S3 ストレージクラスは、オブジェクトレベルで設定でき、単一のバケットのすべてのストレージクラスのオブジェクトを含めることができます。
- また、Amazon S3 ライフサイクルポリシーを使用して、アプリケーションを変更せずにストレージクラス間でオブジェクトを自動的に移動したり、データを削除したりすることができます。一般的に、このようなストレージメカニズムを考える場合、リソース効率、アクセスのレイテンシー、信頼性の間でトレードオフを行う必要があります。

リソース

関連するドキュメント:

- Amazon EBS ボリュームタイプ
- Amazon EC2 インスタンスストア
- Amazon S3 Intelligent-Tiering
- Amazon EBS I/O の特性
- Amazon S3 のストレージクラスを使用する
- Amazon S3 Glacier とは?

関連動画:

- Architectural Patterns for Data Lakes on AWS
- Deep dive on Amazon EBS (STG303-R1)
- Optimize your storage performance with Amazon S3 (STG343)
- Building modern data architectures on AWS

関連サンプル:

- Amazon EFS CSI Driver
- Amazon EBS CSI Driver
- Amazon EFS Utilities
- Amazon EBS Autoscale
- Amazon S3 のサンプル

SUS04-BP03 ポリシーを使用してデータセットのライフサイクルを管理する

すべてのデータのライフサイクルを管理し、自動的に削除を実行することで、ワークロードに必要なストレージの総量を最小限に抑えます。

一般的なアンチパターン:

- データを手動で削除する。
- ワークロードデータは削除しない。
- データ保持やアクセス要件に基づいて、より エネルギー効率の高いストレージ階層にデータを移動することがない。

このベストプラクティスを活用するメリット: データライフサイクルポリシーを使用することで、ワークロードのデータアクセスと保持を効率的に行うことができます。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

データセットには通常、そのライフサイクルにおいて異なる保持要件とアクセス要件があります。例えば、限られた期間のみ頻繁にデータセットにアクセスする必要があるアプリケーションもあります。その後、それらのデータセットにアクセスすることはほとんどありません。

データセットをライフサイクル全体で効率的に管理するには、データセットの処理方法を定義するルールであるライフサイクルポリシーを設定します。

ライフサイクル設定ルールを使用すると、特定のストレージサービスに対して、データセットをよりエネルギー効率の高いストレージ層に移行する、アーカイブする、または削除するように指示できます。

実装手順

- ワークロード内のデータセットを分類します。
- データクラスごとに処理手順を定義します。
- ライフサイクルルールを適用するための自動ライフサイクルポリシーを設定します。さまざまな AWS ストレージサービスの自動ライフサイクルポリシーを設定する方法の例を次に示します。

Storage service	How to set automated lifecycle policies
Amazon S3	Amazon S3 ライフサイクルを使用して、オブジェクトのライフサイクル全体を管理できます。アクセスパターンが不明、変化している、または予測できない場合、アクセスパターンを監視し、アクセスされていないオブジェクトをより低コストのアクセス層に自動的に移動する Amazon S3 Intelligent-Tiering を使用できます。Amazon S3 ストレージレンズのメトリクスを活用し、ライフサイクル管理における最適化の機会やギャップを特定できます。
Amazon Elastic Block Store	Amazon Data Lifecycle Manager を使用すると、Amazon EBS スナップショットや Amazon EBS でサポートされた AMI の作成、保持、削除を自動化できます。

Storage service	How to set automated lifecycle policies
Amazon Elastic File System	Amazon EFS ライフサイクル管理は、ファイルシステムのファイルストレージを自動的に管理します。
Amazon Elastic Container Registry	Amazon ECR ライフサイクルポリシーは、経過時間やカウントに基づいてイメージを期限切れにし、コンテナイメージのクリーンアップを自動化するものです。
AWS Elemental MediaStore	オブジェクトライフサイクルポリシーを使用して、MediaStore コンテナ内にオブジェクトを保存する期間を管理できます。

- 未使用のボリューム、スナップショット、保存期間を過ぎたデータを削除します。削除には、Amazon DynamoDB の有効期限や Amazon CloudWatch ログ保持などのネイティブサービス機能を活用します。
- ライフサイクルルールに基づいて、該当する場合はデータを集約および圧縮します。

リソース

関連するドキュメント:

- Optimize your Amazon S3 Lifecycle rules with Amazon S3 Storage Class Analysis (Amazon S3 ストレージクラス分析によって Amazon S3 ライフサイクルルールを最適化する)
- AWS Config ルール を使用してリソースを評価する

関連動画:

- Simplify Your Data Lifecycle and Optimize Storage Costs With Amazon S3 Lifecycle (Amazon S3 ライフサイクルによってデータライフサイクルを簡素化し、ストレージコストを最適化する)
- Reduce Your Storage Costs Using Amazon S3 Storage Lens (Amazon S3 ストレージレンズを使用してストレージコストを削減する)

SUS04-BP04 伸縮性とオートメーションを使用してブロックストレージまたはファイルシステムを拡張する

伸縮性とオートメーションを使用して、データの増加につれてブロックストレージまたはファイルシステムを拡張し、プロビジョニングされるストレージの合計を最小化します。

一般的なアンチパターン:

- 将来必要になるかもしれない大きなブロックストレージやファイルシステムを調達している。
- ファイルシステムの IOPS (input and output operations per second、入出力操作毎秒) を過剰プロビジョニングしている。
- データボリュームの使用率をモニターしていない。

このベストプラクティスを活用するメリット: ストレージシステムの過剰プロビジョニングを最小化すると、アイドル状態のリソースが削減され、ワークロード全体の効率が向上します。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

ワークロードに適したサイズ割り当て、スループット、レイテンシーで、ブロックストレージやファイルシステムを作成します。伸縮性とオートメーションを使用して、データの増加につれてブロックストレージまたはファイルシステムを拡張し、これらのストレージサービスを過剰プロビジョニングしないようにします。

実装手順

- Amazon EBS などの固定サイズのストレージシステムについては、使用済みのストレージの量を全体的なストレージサイズに照らしてモニタリングし、可能であれば、しきい値に到達したときにストレージサイズを増加させるオートメーションを作成していることを検証します。
- 伸縮自在なボリュームとマネージド型のブロックデータサービスを使用して、永続的データの増加に応じて追加のストレージの割り当てを自動化します。例えば、Amazon EBS Elastic Volumes を使用して、Amazon EBS ボリュームのボリュームサイズやボリュームタイプを変更したり、パフォーマンスを調整したりできます。
- ファイルシステムに適したストレージクラス、パフォーマンスモード、スループットモードを選択して、ビジネスニーズを超えることなく対処できるようにします。
  - Amazon EFS パフォーマンス
  - Linux インスタンスでの Amazon EBS ボリュームのパフォーマンス

- データボリュームの使用率の目標レベルを設定し、予想される範囲外のボリュームはサイズ変更します。
- データに合わせて読み取り専用ボリュームのサイズを最適化します。
- データをオブジェクトストアに移行して、ブロックストレージの固定ボリュームサイズを超える容量をプロビジョンするのを回避します。
- 伸縮自在なボリュームやファイルシステムを定期的に見直して、アイドルなボリュームを停止し、現在のデータサイズに合わせて過剰プロビジョンされたリソースを縮小します。

リソース

関連するドキュメント:

- Amazon FSx ドキュメント
- What is Amazon Elastic File System? (Amazon Elastic File System とは)

関連動画:

- Deep Dive on Amazon EBS Elastic Volumes (Amazon EBS Elastic Volumes の詳細)
- Amazon EBS and Snapshot Optimization Strategies for Better Performance and Cost Savings (パフォーマンスとコスト節約の向上を目指した Amazon EBS とスナップショットの最適化戦略)
- Optimizing Amazon EFS for cost and performance, using best practices (ベストプラクティスを使用した Amazon EFS のコストとパフォーマンスの最適化)

SUS04-BP05 不要なデータや重複するデータを削除する

不要なデータや重複するデータを削除し、データセットの保存に必要なストレージリソースを最小限に抑えます。

一般的なアンチパターン:

- 簡単に取得または再作成できるデータを複製している。
- データの重要性を考慮せず、すべてのデータをバックアップしている。
- データの削除は、不定期、運用イベント時のみ、または全く行わない。
- ストレージサービスの耐久性に関係なく、データを冗長に保存している。
- ビジネス上の正当な理由なく Amazon S3 バージョニングを実行している。

このベストプラクティスを確立するメリット: 不要なデータを削除することで、ワークロードに必要なストレージサイズを縮小し、ワークロードの環境に対する影響も軽減します。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

不要なデータを保存しない。不要なデータの削除を自動化する。ファイルおよびブロックレベルでデータの重複を排除するテクノロジーを使用する。サービスのネイティブデータレプリケーションと冗長性機能を活用する。

実装手順

- AWS Data Exchange およびOpen Data on AWSで公開されている既存のデータセットを利用することで、データの保存を回避できないかを評価します。
- ブロックレベルとオブジェクトレベルでデータを重複排除できる仕組みを使用します。AWS でデータの重複をなくす方法の例を次に示します。

Storage service	Deduplication mechanism
Amazon S3	新しい FindMatches ML Transform を使用して、データセット全体 (識別子のないレコードを含む) で一致するレコードを検索するには、AWS Lake Formation FindMatches を使用します。
Amazon FSx	Windows 向けの Amazon FSx でデータ重複排除を有効にします。
Amazon Elastic Block Store スナップショット	スナップショットは増分バックアップです。つまり、直近のスナップショットの後に変更されたデバイス上のブロックのみが保存されます。

- データアクセスを分析し、不要なデータを特定します。ライフサイクルポリシーを自動化します。削除のための Amazon DynamoDB 有効期限、Amazon S3 ライフサイクル、Amazon CloudWatch ログ保持などのネイティブサービス機能を活用します。
- AWS のデータ仮想化機能を使用してデータをソースに保持し、データの重複を回避します。
  - AWS でのクラウドネイティブデータ仮想化

  - ラボ: Amazon Redshift データ共有を使用したデータパターンの最適化
- 増分バックアップが可能なバックアップテクノロジーを使用します。
- セルフマネージドテクノロジー (RAID (Redundant Array of Independent Disks) など) の代わりに、Amazon S3 の耐久性と Amazon EBS のレプリケーションを活用して、耐久性の目標を達成します。
- ログおよび追跡データを一元化し、同一のログエントリの重複を排除して、必要に応じて冗長性を調整するメカニズムを確立します。
- キャッシュの事前入力は、正当な場合にのみ行います。
- キャッシュのモニタリングとオートメーションを確立し、それに従ってキャッシュをサイズ変更します。
- ワークロードの新しいバージョンをプッシュする際に、オブジェクトストアとエッジキャッシュから古いデプロイとアセットを削除します。

リソース

関連するドキュメント:

- CloudWatch Logs のログデータ保持期間を変更する
- Amazon FSx for Windows File Server でのデータの重複排除
- データの重複排除を含む Amazon FSx for ONTAP の機能
- Amazon CloudFront でのファイルの無効化
- AWS Backup を使用してバックアップを行い、Amazon EFS ファイルシステムを復元する
- Amazon CloudWatch Logs とは
- Amazon RDS でのバックアップの操作

関連動画:

- Fuzzy Matching and Deduplicating Data with ML Transforms for AWS Lake Formation (AWS Lake Formation の機械学習トランスフォームによるファジーマッチングとデータの重複排除)

関連する例:

- Amazon Athena を使用して Amazon S3 サーバーのアクセスログを分析するにはどうすればよいですか?

SUS04-BP06 共有ファイルシステムまたはストレージを使用して共通データにアクセスする

共有ファイルシステムまたはストレージを導入して、データの重複を避け、ワークロードのインフラストラクチャの効率を向上させます。

一般的なアンチパターン:

- クライアントそれぞれにストレージをプロビジョンしている。
- 非アクティブなクライアントからデータボリュームをデタッチしていない。
- プラットフォームやシステムを横断してストレージに対するアクセスを提供していない。

このベストプラクティスを活用するメリット: 共有のファイルシステムやストレージを使用すると、データをコピーすることなく、1 人以上のコンシューマーがデータを共有できます。これにより、ワークロードに必要なストレージリソースを削減できます。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

同じデータセットにアクセスするユーザーやアプリケーションが複数の場合、共有ストレージ技術を使用することが、ワークロードの効率的なインフラストラクチャを実現するために重要です。共有ストレージ技術を利用すると、データセットを 1 か所で保存および管理し、データの重複を避けることができます。また、異なるシステム間でデータの一貫性を維持できます。さらに、共有ストレージ技術を利用すると、複数のコンピューティングリソースが並列して同時にデータにアクセスして処理できるため、コンピューティング性能をより効率的に使用できます。

必要なときにのみ、このような共有ストレージサービスからデータを取得し、未使用のボリュームはデタッチしてリソースを解放します。

実装手順

- データに複数のコンシューマーが存在する場合は、データを共有ストレージに移行します。AWS の共有ストレージ技術の例をいくつか示します。

Storage option	When to use
Amazon EBS Multi-Attach	Amazon EBS Multi-Attach を利用すると、単一のプロビジョンド IOPS SSD (io1 または

Storage option	When to use
	io2) ボリュームを、同一アベイラビリティーゾーン内の複数のインスタンスに添付できます。
Amazon EFS	When to Choose Amazon EFS (Amazon EFS を選択するケース) を参照してください。
Amazon FSx	Amazon FSx ファイルシステムの選択を参照してください。
Amazon S3	ファイルシステム構造を必要とせず、オブジェクトストレージを使用して動作するように設計されたアプリケーションは、非常にスケーラブルで耐久性が高い低コストのオブジェクトストレージソリューションである Amazon S3 を使用できます。

- 必要なときにのみ、共有ファイルシステムにデータをコピーしたり、共有ファイルシステムからデータを取得したりします。例えば、Amazon S3 を搭載した Amazon FSx for Lustre ファイルシステムを作成して、ジョブの処理に必要なデータのサブセットのみを Amazon FSx にロードできます。
- SUS04-BP03 ポリシーを使用してデータセットのライフサイクルを管理するに概説されているように、使用パターンに応じてデータを削除します。
- クライアントがアクティブに使用していないボリュームをクライアントからデタッチします。

リソース

関連するドキュメント:

- Amazon S3 バケットにファイルシステムをリンクする
- Using Amazon EFS for AWS Lambda in your serverless applications (サーバーレスアプリケーションの AWS Lambda に Amazon EFS を使用する)
- Amazon EFS Intelligent-Tiering Optimizes Costs for Workloads with Changing Access Patterns (Amazon EFS Intelligent-Tiering はアクセスパターンを変更しワークロードのコストを最適化する)
- オンプレミスデータリポジトリで Amazon FSx を使用する

関連動画:

- Storage cost optimization with Amazon EFS (Amazon EFS を使用したストレージコストの最適化)

SUS04-BP07 ネットワーク間でのデータ移動を最小限に抑える

共通データへのアクセスに共有ファイルシステムまたはオブジェクトストレージを使用して、ワークロードにおけるデータ移動をサポートするために必要なネットワークリソースの総量を最小化します。

一般的なアンチパターン:

- データユーザーの所在地とは別の、同じ AWS リージョンにすべてのデータを保存している。
- データをネットワーク経由で移動する前に、データサイズや形式を最適化していない。

このベストプラクティスを活用するメリット: ネットワーク経由のデータの移動を最適化すると、ワークロードに必要なネットワークリソースの総量を削減でき、環境への影響を抑えることができます。

このベストプラクティスを活用しない場合のリスクレベル: 中

実装のガイダンス

組織のあちこちにデータを移動するには、コンピューティング、ネットワーキング、ストレージのリソースが必要です。データ移動を最小限にするテクニックを使用して、ワークロード全体の効率を向上させます。

実装手順

- ワークロードのリージョンを選択する際は、データまたはユーザーの近接性を 意思決定の要素として考慮します。
- リージョン固有のデータが消費されるリージョン内に保存されるよう、リージョン内で消費されるサービスをパーティションします。
- 効率的なファイル形式 (Parquet や ORC など) を使用してデータを圧縮してから、ネットワーク経由で移動します。
- 未使用のデータは移動しないようにします。未使用のデータ移動を防止するために参考となる事例をいくつかご紹介します。

  - API リソースを関連データのみに削減します。
  - データは詳細 (レコードレベルの情報が不要) を集約します。
  - 詳細は、 Well-Architected Lab - Optimize Data Pattern Using Amazon Redshift Data Sharing を参照してください。
  - 検討 AWS Lake Formation のクロスアカウントデータ共有を考慮します。
- ワークロードのユーザーの近くでコードを実行できるサービスを使用します。

サービス	使用する状況
Lambda@Edge	オブジェクトがキャッシュにないときに実行される、コンピューティング負荷の高いオペレーションに使用します。
CloudFront 関数	HTTP(s) リクエストまたはレスポンス操作など、短時間実行の関数で実行できるシンプルなユースケースに使用します。
AWS IoT Greengrass	コネクテッドデバイスのローカルコンピューティング、メッセージング、データキャッシュを実行します。

リソース

関連するドキュメント:

- Optimizing your AWS Infrastructure for Sustainability, Part III: Networking
- AWS グローバルインフラストラクチャ
- Amazon CloudFront の主な特徴 (CloudFront グローバルエッジネットワークなど)
- Amazon OpenSearch Service での HTTP リクエストの圧縮
- Amazon EMR を使用して中間データを圧縮する
- 圧縮されたデータファイルを Amazon S3 から Amazon Redshift にロードする
- Amazon CloudFront を使用して圧縮ファイルを提供する

関連動画:

- Demystifying data transfer on AWS

関連サンプル:

- 持続可能性を考慮したアーキテクチャ - ネットワーク間のデータ移動を最小限に抑える

SUS04-BP08 データは再作成が難しい場合にのみバックアップする

ビジネス価値のないデータのバックアップを避け、ワークロードに必要なストレージリソースを最小化します。

一般的なアンチパターン:

- データのバックアップ戦略がない。
- 簡単に再作成できるデータをバックアップしている。

このベストプラクティスを活用するメリット: 重要ではないデータのバックアップを避けると、ワークロードに必要なストレージリソースを削減し、環境への影響を減らすことができます。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

必要ではないデータのバックアップを避けると、コストを下げ、ワークロードが使用するストレージリソースを削減できます。ビジネス価値のあるデータまたはコンプライアンス要件を満たすために必要なデータのみをバックアップします。バックアップポリシーを精査し、リカバリーシナリオでは価値のないエフェメラルストレージを除外します。

実装手順

- SUS04-BP01 データ分類ポリシーを実装するで概説されているように、データ分類ポリシーを実装します。
- データ分類の重要度を使用し、目標復旧時間 (RTO) および目標復旧時点 (RPO) に基づいてバックアップ戦略を策定します。重要ではないデータのバックアップを避けます。
  - 簡単に再作成できるデータを除外します。
  - バックアップから一時データを除外します。
  - 共通の場所からデータを復元するために必要な時間がサービスレベルアグリーメント (SLA) を超える場合を除き、データのローカルコピーを除外します。

- 自動化されたソリューションまたはマネージドサービスを使用してビジネスクリティカルなデータをバックアップします。
  - AWS Backup は、フルマネージドサービスで、AWS サービス、クラウド、オンプレミス全体にわたるデータ保護の一元化と自動化を容易にします。AWS Backup を使用した自動バックアップの作成方法に関するハンズオントレーニングについては、Well-Architected Labs - Testing Backup and Restore of Data (Well-Architected ラボ - データのバックアップと復元のテスト) を参照してください。
  - Automate backups and optimize backup costs for Amazon EFS using AWS Backup (AWS Backup を使用して Amazon EFS のバックアップを自動化しコストを最適化する)。

リソース

関連するベストプラクティス:

- REL09-BP01 バックアップが必要なすべてのデータを特定し、バックアップする、またはソースからデータを再現する
- REL09-BP03 データバックアップを自動的に実行する
- REL13-BP02 復旧目標を満たすため、定義された復旧戦略を使用する

関連するドキュメント:

- AWS Backup を使用してバックアップを行い、Amazon EFS ファイルシステムを復元する
- Amazon EBS スナップショット
- Amazon Relational Database Service でのバックアップの操作
- APN パートナー: バックアップを支援できるパートナー
- AWS Marketplace: バックアップに活用できる製品
- Backing Up Amazon EFS (Amazon EFS ファイルシステムのバックアップ)
- Amazon FSx for Windows File Server のバックアップ
- Amazon ElastiCache for Redis のバックアップと復元

関連動画:

- AWS re:Invent 2021 - Backup, disaster recovery, and ransomware protection with AWS(AWS re:Invent 2021 - AWS によるバックアップ、ディザスタリカバリ、ランサムウェア保護)

- AWS Backup Demo: Cross-Account and Cross-Region Backup (AWS Backup デモ: クロスアカウントおよびクロスリージョンバックアップ)
- AWS re:Invent 2019: Deep dive on AWS Backup, ft.Rackspace (STG341) (AWS re:Invent 2019: AWS Backup の詳細、Rackspace 特集 (STG341))

関連する例:

- Well-Architected Lab - Testing Backup and Restore of Data (Well-Architected ラボ - データのバックアップと復元のテスト)
- Well-Architected Lab - Backup and Restore with Failback for Analytics Workload (Well-Architected ラボ - 分析ワークロードのフェイルバックによるバックアップと復元)
- Well-Architected Lab - Disaster Recovery - Backup and Restore (Well-Architected ラボ - ディザスタリカバリ - バックアップと復元)

ハードウェアとサービス

ハードウェア管理のプラクティスを変更することで、ワークロードの持続可能性に対する影響を軽減する機会を探します。プロビジョンおよびデプロイする必要があるハードウェア数を最小化し、個別のワークロードにおいて最も効率のいいハードウェアとサービスを選択します。

ベストプラクティス

- SUS05-BP01 ニーズに合わせて最小限のハードウェアを使用する
- SUS05-BP02 影響が最も少ないインスタンスタイプを使用する
- SUS05-BP03 マネージドサービスを使用する
- SUS05-BP04 ハードウェアベースのコンピューティングアクセラレーターの使用を最適化する

SUS05-BP01 ニーズに合わせて最小限のハードウェアを使用する

ワークロードには最小限のハードウェアを使用し、ビジネスニーズを効率的に満たします。

一般的なアンチパターン:

- リソースの使用率をモニターしていない。
- アーキテクチャに使用率が低いリソースがある。
- 静的ハードウェアの使用率を見直してサイズを変更するかどうかを判断していない。

- ビジネス KPI に基づいたコンピューティングインフラストラクチャのハードウェア使用率目標を設定していない。

このベストプラクティスを活用するメリット: クラウドリソースのサイズを最適化すると、環境に対するワークロードの影響を削減し、費用を節約して、パフォーマンスベンチマークを維持できます。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

ワークロードに必要なハードウェアの総数を適切に選択して、全体の効率を改善します。AWS クラウドは、AWS Auto Scaling などのさまざまなメカニズムを通じて、リソース数を動的に拡張または縮小する柔軟性を提供し、需要の変化に対応します。AWS には、最低限の労力でリソースを変更できる API や SDK も用意されています。これらの機能を使用して、ワークロードの実装を頻繁に変更できます。さらに AWS ツールが提供するサイズ最適化のガイドラインを使用して、クラウドリソースを効率的に運用し、ビジネスニーズを満たすことができます。

実装手順

- ニーズに最適なインスタンスタイプを選択します。
  - ワークロードに適切な Amazon EC2 インスタンスタイプを選択する方法を教えてください。
  - Amazon EC2 フリートの属性ベースのインスタンスタイプの選択
  - 属性ベースのインスタンスタイプの選択を使用して Auto Scaling グループを作成します。
- ワークロードの変動に合わせて少しずつスケールします。
- 複数のコンピューティング購入オプションを使用して、インスタンスの柔軟性、スケーラビリティ、コスト節約のバランスを取ります。
  - オンデマンドインスタンスは、インスタンスタイプ、ロケーション、処理時間に柔軟性がないワークロードで、新規かつステートフルな、スパイクが発生するワークロードに最適です。
  - スポットインスタンスは、耐障害性が高く、柔軟性があるアプリケーションにおいて、他のオプションを補完する優れた方法です。
  - 自分のニーズ (AZ、リージョン、インスタンスファミリー、インスタンスタイプ) が変化した場合に柔軟に対応できる安定した状態のワークロードには、Compute Savings Plans を活用します。
- さまざまなインスタンスや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最大化し、可能なときは余剰キャパシティを活用します。
- AWS ツールの適切なサイズのレコメンデーションを使用して、ワークロードを調整します。

- AWS Compute Optimizer
- AWS Trusted Advisor
- 容量の一時的な削減ができるようにサービスレベルアグリーメント (SLA) を交渉すると同時に、オートメーションを使用して代替リソースをデプロイします。

リソース

関連するドキュメント:

- Optimizing your AWS Infrastructure for Sustainability, Part I: Compute (サステナビリティのための AWS インフラストラクチャの最適化、パート I : コンピューティング)
- Attribute based Instance Type Selection for Auto Scaling for Amazon EC2 Fleet (EC2 フリート向け Auto Scaling 用の属性ベースのインスタンスタイプの選択)
- AWS Compute Optimizer ドキュメント
- Operating Lambda: Performance optimization (Lambda の操作: パフォーマンス最適化 – パート 2)
- Auto Scaling のドキュメント

関連動画:

- Build a cost-, energy-, and resource-efficient compute environment (コスト面、エネルギー面、リソース面で効率的なコンピューティング環境の構築)

関連する例:

- Well-Architected Lab - Rightsizing with AWS Compute Optimizer and Memory Utilization Enabled (Level 200) (Well-Architected ラボ - AWS Compute Optimizer およびメモリ使用率の有効化によるサイズ最適化 (レベル 200))

SUS05-BP02 影響が最も少ないインスタンスタイプを使用する

新しいインスタンスタイプを継続的にモニターして使用し、エネルギー効率の改善を活用します。

一般的なアンチパターン:

- インスタンスの 1 つのファミリーのみを使用する。

- x86 インスタンスのみを使用する。
- Amazon EC2 Auto Scaling 設定で 1 つのインスタンスタイプを指定する。
- AWS インスタンスが設計されていない方法で使用されている (例えば、メモリ集中型のワークロードに計算用に最適化されたインスタンスを使用した場合)。
- 新しいインスタンスタイプを定期的に評価しない。
- 次のような AWS サイズ最適化ツールからのレコメンデーションを確認しない : AWS Compute Optimizer。

このベストプラクティスを活用するメリット: エネルギー効率が高く、適切なサイズのインスタンスを使用することで、環境への影響とワークロードのコストを大幅に削減できます。

このベストプラクティスを活用しない場合のリスクレベル: 中

実装のガイダンス

クラウドワークロードに効率的なインスタンスを使用することは、リソースの使用量を下げ、コスト効率を高めるために重要です。新しいインスタンスタイプのリリースを継続的にモニターし、エネルギー効率の改善を活用します。例えば、機械学習のトレーニングや推論、ビデオのトランスコーディングなど、特定のワークロードをサポートするように設計されたインスタンスタイプなどです。

実装手順

- ワークロード環境への影響を削減できるインスタンスタイプを学習し、試します。
  - 例えば、 AWS の最新情報 を購読して、最新の AWS テクノロジーとインスタンスの情報を入手します。
  - さまざまな AWS インスタンスタイプについて学びます。
  - Amazon EC2 のエネルギー使用量 1 ワットあたりで最高のパフォーマンスを発揮する AWS Graviton ベースのインスタンスについて、以下のビデオをご覧ください。 re:Invent 2020 - Deep dive on AWS Graviton2 processor-powered Amazon EC2 instances (AWS Graviton2 プロセッサ搭載 Amazon EC2 インスタンスの詳細) と Deep dive into AWS Graviton3 and Amazon EC2 C7g instances.
- ワークロードを計画し、最も影響の少ないインスタンスタイプに移行することができます。
  - ワークロードの新機能やインスタンスを評価するプロセスを定義します。クラウドの俊敏性を利用して、新しいインスタンスタイプがワークロード環境の持続可能性をどのように改善するかをすばやくテストします。プロキシメトリクスを使用して、1 つの作業単位を完了するのに必要なリソース数を測定します。

- 可能な場合は、異なる数の vCPU と異なる量のメモリで動作するようにワークロードを変更して、インスタンスタイプの選択肢を最大化します。
- ワークロードのパフォーマンス効率を向上させるために、Graviton ベースのインスタンスへの移行を検討します。
  - AWS Graviton Fast Start
  - Considerations when transitioning workloads to AWS Graviton-based Amazon Elastic Compute Cloud instances
  - AWS Graviton2 for ISVs を参照
- 以下の利用において、AWS Graviton オプションの選択を検討します : AWS マネージドサービス。
- 持続可能性に対する影響が最も少なく、かつビジネス要件を満たすインスタンスを提供するリージョンにワークロードを移行します。
- 機械学習のワークロードには、 AWS Trainium、 AWS Inferentia、および Amazon EC2 DL1 など、ワークロードに特化した専用ハードウェアを利用します。 Inf2 インスタンスなどの AWS Inferentia インスタンスは、同等の Amazon EC2 インスタンスと比較してワットあたりのパフォーマンスが最大 50% 向上します。
- Amazon SageMaker Inference Recommender を使用して、機械学習推論エンドポイントのサイズを適切に設定します。
- ワークロードの急増 (追加のキャパシティが必要になることがあまりないワークロード) の場合は、 バーストパフォーマンスインスタンスを使用します。
- ステートレスで耐障害性の高いワークロードについては、 Amazon EC2 スポットインスタンスを使用して、クラウドの全体使用率を高め、未使用のリソースの持続可能性に対する影響を軽減します。

- ワークロードインスタンスを操作して、最適化します。
  - 一次的なワークロードについては、 インスタンス Amazon CloudWatch メトリクス (`CPUUtilization` など) を評価して、インスタンスがアイドルか使用率が低いかを識別します。
  - 安定したワークロードの場合は、AWS サイズ適正化ツール (AWS Compute Optimizer など) を定期的にチェックし、インスタンスの最適化とサイズ適正化の機会を識別します。
    - Well-Architected Lab - Rightsizing Recommendations (Well-Architected ラボ - サイズ最適化のレコメンデーション)
    - Well-Architected Lab - Rightsizing with Compute Optimizer (Well-Architected ラボ - Compute Optimizer を使用したサイズ最適化)

- Well-Architected Lab - Optimize Hardware Patterns and Observe Sustainability KPIs (Well-Architected ラボ - ハードウェアパターンの最適化と持続可能性 KPI の観察)

リソース

関連するドキュメント:

- 持続可能性のために AWS インフラストラクチャを最適化する、パート I: コンピューティング
- AWS Graviton
- Amazon EC2 DL1 など、ワークロードに特化した専用ハードウェアを利用します
- Amazon EC2 キャパシティ予約フリート
- Amazon EC2 スポットフリート
- 関数: Lambda 関数の設定
- Amazon EC2 フリートの属性ベースのインスタンスタイプの選択
- AWS での持続可能で効率的、コストが最適化されたアプリケーションの構築
- Contino サステナビリティダッシュボードがお客様のカーボンフットプリントの最適化に役立つ仕組み

関連動画:

- Deep dive on AWS Graviton2 processor-powered Amazon EC2 instances (AWS Graviton2 プロセッサ搭載 Amazon EC2 インスタンスの詳細)
- Deep dive into AWS Graviton3 and Amazon EC2 C7g instances
- Build a cost-, energy-, and resource-efficient compute environment (コスト面、エネルギー面、リソース面で効率的なコンピューティング環境の構築)

関連サンプル:

- Solution: Guidance for Optimizing Deep Learning Workloads for Sustainability on AWS (ソリューション: AWS における持続可能性に向けた深層学習ワークロードの最適化ガイダンス)
- Well-Architected Lab - Rightsizing Recommendations (Well-Architected ラボ - サイズ最適化のレコメンデーション)
- Well-Architected Lab - Rightsizing with Compute Optimizer (Well-Architected ラボ - Compute Optimizer を使用したサイズ最適化)

- Well-Architected Lab - Optimize Hardware Patterns and Observe Sustainability KPIs (Well-Architected ラボ - ハードウェアパターンの最適化と持続可能性 KPI の観察)
- Well-Architected Lab - Migrating Services to Graviton (Well-Architected ラボ - サービスの Graviton への移行)

SUS05-BP03 マネージドサービスを使用する

マネージドサービスを使用して、クラウドでより効率的に運用します。

一般的なアンチパターン:

- アプリケーションの実行に使用率が低い Amazon EC2 インスタンスを使用している。
- 社内チームはワークロードの管理のみを行っており、イノベーションや簡易化に焦点を当てる時間がない。
- マネージドサービスではより効率的に実行できるタスク向けの技術をデプロイして維持している。

このベストプラクティスを活用するメリット:

- マネージドサービスを使用すると、AWS に責任を移行できます。AWS は、数百万のお客様から得られたインサイトで、新規イノベーションと効率性を促進しています。
- マネージドサービスは、マルチテナントコントロールプレーンのおかげで、サービスの環境に対する影響を、多くのお客様に分散します。

このベストプラクティスが確立されていない場合のリスクレベル: ミディアム

実装のガイダンス

マネージドサービスは、使用率を高く保つ責任と、デプロイされたハードウェアの持続可能性に対する最適化の責任を AWS に移します。また、マネージドサービスによって、サービス維持に伴う運用上および管理上の負担が軽減されるため、チームに時間の余裕ができイノベーションに集中できます。

ワークロードを見直して、AWS マネージドサービスに置き換えることができるコンポーネントを特定します。例えば、Amazon RDS、Amazon Redshift、Amazon ElastiCache は、マネージドデータベースサービスを提供します。Amazon Athena、Amazon EMR、Amazon OpenSearch Service はマネージド分析サービスを提供します。

実装手順

1. サービスとコンポーネントのワークロードをリストアップします。
2. コンポーネントを評価して、マネージドサービスに置き換えることができるものを特定します。マネージドサービスの使用を検討する場合の例を次に示します。

Task	What to use on AWS
データベースのホスティング	Amazon Elastic Compute Cloud (Amazon EC2) で独自の Amazon RDS インスタンスを維持するのではなく、マネージド Amazon Relational Database Service (Amazon RDS) インスタンスを使用します。
コンテナワークロードのホスティング	独自のコンテナインフラストラクチャを実装するのではなく、AWS Fargate を使用します。
ウェブアプリケーションのホスティング	フルマネージド CI/CD、および静的ウェブサイトとサーバー側のレンダリング済みウェブアプリケーションのホスティングサービスとして、AWS Amplify ホスティングを使用します。

3. 依存関係を特定して移行計画を作成します。同様にランブックやプレイブックも更新します。
   - AWS Application Discovery Service は、アプリケーションの依存関係と使用状況に関する詳細な情報を自動的に収集して提示するサービスです。これにより、充分な情報に基づいて移行計画の意思決定を行うことができます。
4. サービスをテストしてから、マネージドサービスに移行します。
5. 移行計画を使用して、自己ホスト型サービスをマネージドサービスに置き換えます。
6. 移行完了後は、サービスを継続的にモニターして、必要に応じて調整しサービスを最適化します。

リソース

関連するドキュメント:

- AWS クラウド 製品
- AWS 総保有コスト (TCO) 計算ツール

- Amazon DocumentDB
- Amazon Elastic Kubernetes Service (EKS)
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)

関連動画:

- Cloud operations at scale with AWS Managed Services(AWS マネージドサービスを使用した大規模なクラウド運用)

SUS05-BP04 ハードウェアベースのコンピューティングアクセラレーターの使用を最適化する

高速コンピューティングインスタンスの使用を最適化することで、ワークロードの物理インフラストラクチャの需要を低減します。

一般的なアンチパターン:

- GPU の使用状況を監視していない。
- 専用インスタンスがより高い性能、低コスト、ワットあたりの性能を実現できるのに対し、ワークロードに汎用インスタンスを使用している。
- CPU ベースのコンピューティングアクセラレーターを使用した方が効率的なタスクに、ハードウェアベースのコンピューティングアクセラレーターを使用している。

このベストプラクティスを活用するメリット: ハードウェアベースのアクセラレーターの使用を最適化することで、ワークロードの物理インフラストラクチャの需要を低減できます。

このベストプラクティスを活用しない場合のリスクレベル: 中

実装のガイダンス

高い処理能力が必要な場合、高速コンピューティングインスタンスを使用すると、グラフィック処理ユニット (GPU) やフィールドプログラマブルゲートアレイ (FPGA) などのハードウェアベースのコンピューティングアクセラレーターを利用できるというメリットが得られます。これらのハードウェアアクセラレーターは、グラフィック処理やデータパターンマッチングなどの特定の機能を、CPU ベースの代替手段よりも効率的に実行します。レンダリング、トランスコーディング、機械学習など、多くの高速ワークロードは、リソースの使用量に大きなばらつきがあります。このハードウェア

は必要な時間だけ実行し、不要になったら自動で廃止することで、消費されるリソースを最小化します。

実装手順

- どの 高速コンピューティングインスタンスが お客様の要件に対応できるかを特定します。
- 機械学習のワークロードには、 AWS Trainium、 AWS Inferentia、 Amazon EC2 DL1 など、ワークロードに特化した専用ハードウェアを利用します。Inf2 インスタンスなどの AWS Inferentia インスタンスは、 Amazon EC2 インスタンスと比較して、ワットあたりのパフォーマンスが最大 50% 向上します。
- 高速コンピューティングインスタンスの使用状況メトリクスを収集します。例えば、CloudWatch エージェントを使用して、GPU の `utilization_gpu` および `utilization_memory` などのメトリクスを収集できます (「Amazon CloudWatch で NVIDIA GPU メトリクスを収集する」を参照)。
- ハードウェアアクセラレーターのコード、ネットワーク操作、設定を最適化し、基盤となるハードウェアが十分に活用されるようにします。
  - GPU 設定を最適化する
  - Deep Learning AMI での GPU のモニタリングと最適化
  - Amazon SageMaker における深層学習トレーニングの GPU パフォーマンスチューニングのための I/O の最適化
- 最新の高性能ライブラリと GPU ドライバーを使用します。
- 使用しないときは、自動化を使用して GPU インスタンスを解放します。

リソース

関連するドキュメント:

- 高速コンピューティング
- Let's Architect! Architecting with custom chips and accelerators
- ワークロードに適切な Amazon EC2 インスタンスタイプを選択する方法を教えてください。
- Amazon EC2 VT1 インスタンス
- Choose the best AI accelerator and model compilation for computer vision inference with Amazon SageMaker

関連動画:

- 深層学習用の Amazon EC2 GPU インスタンスの選択方法
- Deploying Cost-Effective Deep Learning Inference (費用対効果の高い深層学習推論の導入)

プロセスと文化

開発、テスト、デプロイのプラクティスを変更することで、持続可能性に対する影響を減らす機会を探します。

ベストプラクティス

- SUS06-BP01 持続可能性の改善を迅速に導入できる方法を採用する
- SUS06-BP02 ワークロードを最新に保つ
- SUS06-BP03 ビルド環境の利用率を高める
- SUS06-BP04 マネージド型 Device Farm を使用してテストする

SUS06-BP01 持続可能性の改善を迅速に導入できる方法を採用する

改善の可能性の検証、テストコストの最小化、小規模な改善の提供を行う手段やプロセスを導入します。

一般的なアンチパターン:

- 持続可能性についてアプリケーションをレビューするのは、プロジェクトの開始時に 1 回だけである。
- リリースプロセスが複雑すぎてリソース効率化のための小規模な変更を導入しづらいため、ワークロードが古くなった。
- 持続可能性のためにワークロードを改善する仕組みがない。

このベストプラクティスを活用するメリット: 持続可能性に関する改善を導入および追跡するプロセスを確立することで、継続的に新しい機能や能力を導入し、問題を排除して、ワークロードの効率を向上させることができます。

このベストプラクティスが確立されていない場合のリスクレベル: 中

実装のガイダンス

本稼働環境にデプロイする前に、持続可能性を改善できるかをテストして検証します。改善に際して将来的に起こりうる利点を計算する際のテストにかかるコストを考慮します。低コストのテスト方法を開発し、小規模な改善を実施します。

実装手順

- 持続可能性の改善に関する要件を開発バックログに追加します。
- 反復的な改善プロセスを使用して、これらの改善を特定、評価、優先順位付け、テスト、デプロイします。
- 開発プロセスを継続的に改善および合理化します。例えば、継続的な統合および配信 (CI/CD) パイプラインを使用してソフトウェア配信プロセスを自動化して、工数レベルを削減し手動プロセスで発生するエラーを減らす可能性のある改善をテストしデプロイします。
- 最小限に実行可能である代表的なコンポーネントを使用して、潜在的な改善を開発およびテストし、テストのコストを削減します。
- 改善の影響を継続的に評価し、必要に応じて調整します。

リソース

関連するドキュメント:

- AWS がサステナビリティソリューションを実現
- Scalable agile development practices based on AWS CodeCommit(AWS CodeCommit に基づいたスケーラブルなアジャイル開発の実践)

関連動画:

- Delivering sustainable, high-performing architectures (持続可能でパフォーマンスが高いアーキテクチャを実現する)

関連する例:

- Well-Architected Lab - Turning cost & usage reports into efficiency reports (Well-Architected ラボ - コストと使用状況レポートを効率性レポートに変える)

SUS06-BP02 ワークロードを最新に保つ

ワークロードを最新の状態に保ち、効率的な機能を導入し、問題を排除し、ワークロード全体の効率性を向上させます。

一般的なアンチパターン:

- 現在のアーキテクチャが今後は静的なものとなり、しばらく更新されないと考えている。
- 更新されたソフトウェアおよびパッケージがワークロードと互換性があるかどうかを評価するためのシステムまたは定期的な予定がない。

このベストプラクティスを活用するメリット: ワークロードを最新に保つプロセスを確立することで、新しい機能と能力を採用し、問題を解決し、ワークロードの効率性を高めることができます。

このベストプラクティスが確立されていない場合のリスクレベル: 低

実装のガイダンス

最新のオペレーティングシステム、ランタイム、ミドルウェア、ライブラリ、アプリケーションを使用すると、ワークロードの効率が上がり、さらに効率的なテクノロジーを簡単に導入できます。最新のソフトウェアにはまた、ワークロードの持続可能性に対する影響をより正確に測定する機能が含まれている場合があります。これは、ベンダーが独自の持続可能性の目標を満たすための機能でもあります。定期的に最新の機能やリリースを導入し、ワークロードを最新に保ちます。

実装手順

- ワークロードに応じた新しい機能やインスタンスを評価するプロセスとスケジュールを定義します。クラウドの俊敏性を利用して、新しい機能がワークロードをどのように改善するかをすばやくテストします。
  - 持続可能性への影響を削減する。
  - パフォーマンスの効率を高める。
  - 計画した改善にとっての障壁を取り除く。
  - 持続可能性に対する影響の測定能力と管理能力を高める。
- ワークロードソフトウェアおよびアーキテクチャをインベントリに登録して、更新する必要があるコンポーネントを特定する。
  - AWS Systems Manager インベントリを使用して、Amazon EC2 インスタンスからオペレーティングシステム (OS)、アプリケーション、インスタンスのメタデータを収集し、どのインス

タンスがソフトウェアポリシーで要求されるソフトウェアと設定を実行しているか、どのインスタンスが更新する必要があるかを迅速に把握できます。

- ワークロードのコンポーネントを更新する方法を理解します。

Workload component	How to update
マシンイメージ	EC2 Image Builder を使用して、Linux または Windows サーバーイメージの Amazon マシンイメージ (AMI) の更新を管理します。
コンテナイメージ	既存のパイプラインに Amazon Elastic Container Registry (Amazon ECR) を使用して、Amazon Elastic Container Service (Amazon ECS) イメージを管理します。
AWS Lambda	AWS Lambda には、バージョン管理機能が含まれています。

- 更新プロセスにオートメーションを使用して、新しい機能をデプロイする労力のレベルを軽減し、手動プロセスに起因するエラーを抑制します。
  - CI/CD を使用して、AMI、コンテナイメージ、その他クラウドアプリケーションに関連するアーティファクトを自動的に更新できます。
  - AWS Systems Manager Patch Manager などのツールを使用して、システム更新のプロセスを自動化し、AWS Systems Manager Maintenance Windows を使用してアクティビティをスケジュールできます。

リソース

関連するドキュメント:

- AWS アーキテクチャセンター
- AWS の最新情報
- AWS のデベロッパー用ツール

関連する例:

- Well-Architected Labs: Inventory and Patch Management (Well-Architected ラボ: インベントリおよびパッチ管理)
- Lab: AWS Systems Manager (ラボ: AWS Systems Manager)

SUS06-BP03 ビルド環境の利用率を高める

リソースの使用率を上げて、ワークロードを開発、テスト、構築します。

一般的なアンチパターン:

- ビルド環境を手動でプロビジョニングおよび停止している。
- テスト、ビルド、リリースアクティビティとは無関係にビルド環境を実行し続けている (例えば、開発チームメンバーの就業時間外に環境を実行している)。
- ビルド環境にリソースを過剰プロビジョニングしている。

このベストプラクティスを活用するメリット: ビルド環境の使用率を上げることで、構築者が効率的に開発、テスト、構築できるようにリソースを配分しながら、クラウドワークロード全体の効率を上げることができます。

このベストプラクティスが確立されていない場合のリスクレベル: 低

実装のガイダンス

オートメーションと Infrastructure as Code を使用して、必要に応じてビルド環境を起動し、使用しないときは停止します。一般的なパターンとしては、開発チームのメンバーの勤務時間と重なるように可用性期間のスケジュールを設定することがあります。テスト環境は、本稼働構成とよく似たものにする必要があります。ただし、バーストキャパシティ、Amazon EC2 スポットインスタンス、自動スケールするデータベースサービス、コンテナ、サーバーレステクノロジーを備えたインスタンスタイプを使用して、開発およびテストのキャパシティを使用に合わせて調整できる機会を探ります。データ量を、テスト要件を満たす量だけに制限します。本稼働データをテストに使用する場合は、データを移動するのではなく、本稼働環境とデータを共有できる可能性を探ります。

実装手順

- Infrastructure-as-Code を使用してビルド環境をプロビジョニングします。
- オートメーションを使用して開発環境とテスト環境のライフサイクルを管理し、ビルド用リソースの効率を最大化します。

- 開発環境とテスト環境を最大限に活用する戦略を使用します。
  - 最小限に実行可能である代表的な環境を使用して、潜在的な改善を開発およびテストします。
  - 可能な限り、サーバーレス技術を利用します。
  - オンデマンドインスタンスを使用して開発者のデバイスを補完します。
  - バーストキャパシティ、スポットインスタンス、その他のテクノロジーを備えたインスタンスタイプを使用して、ビルドキャパシティを使用状況に合わせて調整します。
  - 踏み台ホストのフリートをデプロイするのではなく、ネイティブなクラウドサービスを採用して、インスタンスシェルのアクセスを保護します。
  - ビルドジョブに合わせてビルドリソースを自動的にスケールします。

リソース

関連するドキュメント:

- AWS Systems Manager Session Manager
- Amazon EC2 バーストパフォーマンスインスタンス
- AWS CloudFormation の概要
- AWS CodeBuild とは
- AWS での Instance Scheduler

関連動画:

- Continuous Integration Best Practices (継続的インテグレーションのベストプラクティス)

SUS06-BP04 マネージド型 Device Farm を使用してテストする

マネージド型 Device Farm を使用して、ハードウェアの代表的なセットで新機能を効率的にテストします。

一般的なアンチパターン:

- 個別の物理デバイス上で、アプリケーションを手動でテストおよびデプロイしている。
- アプリケーションテストサービスを使用せずに、実際の物理デバイス上でアプリケーションをテストおよび操作している (Android、iOS、ウェブアプリケーションなど)。

このベストプラクティスを活用するメリット: マネージド型 Device Farm を使用してクラウド対応アプリケーションをテストすると、多くのメリットがあります。

- 幅広い種類のデバイスでアプリケーションをテストする、より効率的な機能などです。
- これにより、テスト用の社内インフラストラクチャが必要なくなります。
- あまり使われない古いハードウェアを含む、さまざまデバイスタイプが提供されているため、不要なデバイスをアップグレードする必要がなくなります。

このベストプラクティスが確立されていない場合のリスクレベル: 低

実装のガイダンス

マネージド型 Device Farm を使用すると、代表的な一連のハードウェアで新機能をテストするプロセスを合理化できます。マネージド型 Device Farm は、あまり使われない古いハードウェアを含むさまざまなデバイスタイプを提供するため、不要なデバイスのアップグレードによるお客様の持続可能性に対する影響を回避できます。

実装手順

- テストの要件と計画 (テストの種類、オペレーティングシステム、テストのスケジュールなど) を定義します。
  - Amazon CloudWatch RUM を使用して、クライアント側のデータを収集および分析し、テスト計画を策定できます。
- テスト要件をサポートできるマネージド型 Device Farm を選択します。例えば、AWS Device Farm を使用して、代表的な一連のハードウェア上での変更の影響をテストし把握できます。
- 継続的統合/継続的デプロイ (CI/CD) を使用して、テストをスケジュールし実行します。
  - Integrating AWS Device Farm with your CI/CD pipeline to run cross-browser Selenium tests (AWS Device Farm を CI/CD パイプラインに統合してブラウザ間で Selenium のテストを実行する)
  - Building and testing iOS and iPadOS apps with AWS DevOps and mobile services (AWS DevOps およびモバイルサービスを使用して iOS および iPadOS アプリケーションを構築およびテストする)
- テスト結果を継続的に見直し、必要な改善を行います。

リソース

関連するドキュメント:

- AWS Device Farm デバイスリスト
- CloudWatch RUM ダッシュボードの表示

関連する例:

- AWS Device Farm Sample App for Android
- AWS Device Farm Sample App for iOS
- Appium Web tests for AWS Device Farm

関連動画:

- Optimize applications through end user insights with Amazon CloudWatch RUM (Amazon CloudWatch RUM を使用したエンドユーザーインサイトを通してアプリケーションを最適化する)

まとめ

政府規制、競争優位性、顧客、従業員、投資家の要求の変化に対応するために、持続可能性の目標を設定する組織が増加しています。CTO、アーキテクト、デベロッパー、オペレーションチームのメンバーは、組織の持続可能性目標に直接貢献できる方法を模索しています。AWS のサービスに支えられたこれらの設計原則とベストプラクティスを使用することにより、セキュリティ、コスト、パフォーマンス、信頼性、および運用上の卓越性と AWS クラウド ワークロードの持続可能性成果とバランスを取りながら、情報に基づいた意思決定を下すことができます。リソースの使用量を削減し、ワークロードの効率を高めるために行うすべての行動が、環境への影響を軽減し、組織の幅広い持続可能性目標への貢献につながります。

寄稿者

本書の寄稿者は次のとおりです。

- Amazon Web Services、Senior Efficiency Lead Solutions Architect、Sam Mokhtari
- Brendan Sisson、プリンシパルサステナビリティソリューションアーキテクト、Amazon Web Services
- Margaret O'Toole、サステナビリティテックリーダー、Amazon Web Services
- Steffen Grunwald、プリンシパルサステナビリティソリューションアーキテクト、Amazon Web Services
- Ryan Eccles、プリンシパルエンジニア、Sustainability、Amazon
- Rodney Lester、プリンシパルアーキテクト、Amazon Web Services
- Adrian Cockcroft、サステナビリティアーキテクチャ担当バイスプレジデント、Amazon Web Services
- Ian Meyers、テクノロジー担当ディレクター、Solutions Architecture、Amazon Web Services

その他の資料

詳細については、次を参照してください。

- AWS Well-Architected
- AWS アーキテクチャセンター
- クラウド上での持続可能性
- AWS がサステナビリティソリューションを実現
- The Climate Pledge
- 国連の持続可能な開発目標
- 温室効果ガス (GHG) プロトコル

改訂履歴

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

変更	説明	日付
リスクレベルの更新	ベストプラクティスのリスクレベルのマイナーな更新。	October 3, 2023
ベストプラクティスガイダンスの更新	ベストプラクティスが更新され、 需要に合わせた調整、ソフトウェアとアーキテクチャ、データ、 ハードウェアとサービスを確認してください 。	July 13, 2023
新しいフレームワーク用に更新	規範ガイダンスを使用してベストプラクティスを更新、および新しいベストプラクティスを追加。	April 10, 2023
ホワイトペーパーの更新	新しい実装ガイダンスを使用してベストプラクティスを更新。	December 15, 2022
ホワイトペーパーの更新	ベストプラクティスに加筆し、改善計画を追加。	October 20, 2022
初版発行	持続可能性の柱 - AWS Well-Architected フレームワークが公開されました。	December 2, 2021

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、(b) AWS の現行製品と慣行について説明しており、これらは予告なしに変更されることがあり、(c) AWS およびその関連会社、サプライヤーまたはライセンサーからの契約上の義務や保証をもたらすものではありません。AWS の製品やサービスは、明示または暗示を問わず、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いかなる契約の一部でもなく、その内容を修正するものでもありません。

© 2021 Amazon Web Services, Inc. or its affiliates.All rights reserved.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the AWS の用語集 Reference.