AWS Well-Architected フレームワーク

持続可能性の柱

持続可能性の柱: AWS Well-Architected フレームワーク

Table of Contents

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

発行日: 2024 年 11 月 6 日 (ドキュメントの改訂)

このホワイトペーパーでは、Amazon Web Services (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満たす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。

序章

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。フレームワークは、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。

- オペレーショナルエクセレンス
- セキュリティ
- 信頼性
- パフォーマンス効率
- コスト最適化
- 持続可能性

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。

クラウドの持続可能性

持続可能性を守ることで、ビジネスが環境、経済、社会に与える長期的な影響に対処します。国際連合の「環境と開発に関する世界委員会」では、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。企業または組織は、環境によくない影響を与える可能性があります。これは、直接的または間接的な二酸化炭素の排出、再利用できない廃棄物、上水などの共有資源への損害などです。

クラウドワークロードの構築における持続可能性の実践とは、使用しているサービスの影響を理解すること、ワークロードのライフサイクル全体における影響を数値化すること、設計原則とベストプラクティスを適用してそれらの影響を軽減することなどです。このドキュメントでは、アーキテクトがリソース使用量削減のための直接的行動を理解する上で重要な、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。

環境への影響に焦点を当てる際は、これらの影響が通常どのように計上されるか、および組織独自の排出量の計上に対する後続の影響について理解する必要があります。温室効果ガスプロトコルは、二酸化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: 組織の活動に由来する、または組織の管理下にあるすべての直接的な排出量。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の燃焼などです。
- スコープ 2: データセンターなどが購入して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な排出量。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、間接的な排出量、およびスコープ 3 の排出量の一部とみなされます。デプロイされた各ワークロードは、前の各スコープからの AWS の総排出量の一部を形成しています。ワークロードあたりの実際の量は、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、AWS による再生可能エネルギーの調達など、いくつかの要素によって異なります。

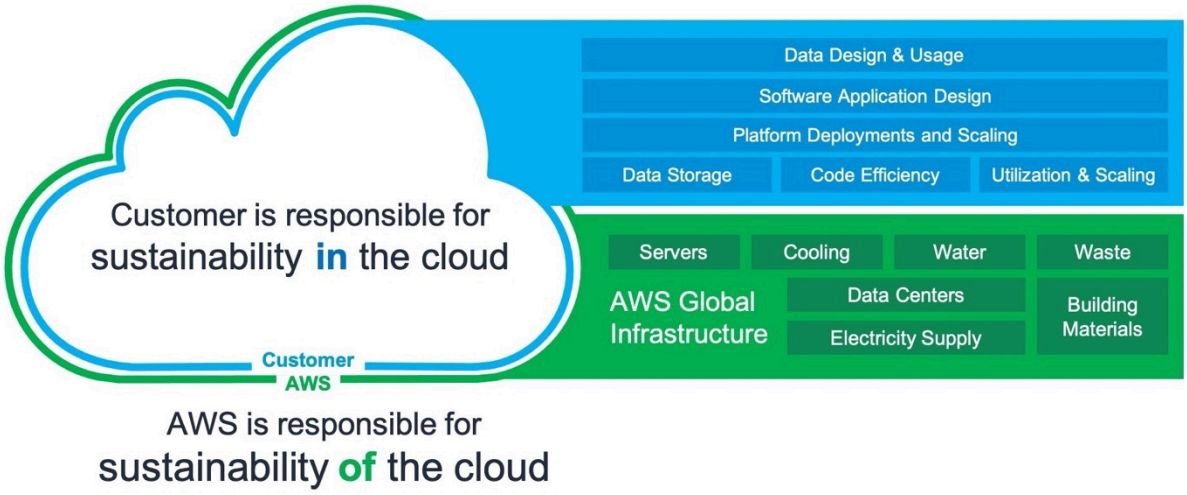
このドキュメントでは、最初に環境持続可能性の責任共有モデルについて説明し、次にアーキテクチャ上のベストプラクティスについて説明します。これにより、AWS データセンターでのワーク

ロードの稼働に必要な総リソース量を削減し、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、効率的な共有インフラストラクチャの提供、水資源の管理、再生可能エネルギーの調達などを行う必要があります。
- お客様は、クラウドの持続可能性に責任があり、ワークロードとリソースの利用率を最適化し、ワークロードにデプロイする必要があるすべてのリソースを最小化する必要があります。



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、ネットワーキング、電力、冷却、物理設備などの共有リソースを利用することで、影響を軽減します。より効率的な技術が利用可能になり次第、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

- The Carbon Reduction Opportunity of Moving to Amazon Web Services

- 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

クラウド上での持続可能性は、プロビジョニングされたリソースを最大限に活用し、必要なすべての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を削減し、効率化することを主眼とした継続的な取り組みです。この取り組みは、初期の効率的なプログラミング言語の選択から、最新のアルゴリズムの導入、効率的なデータストレージ技術の使用、適切にサイジング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 クラウドを使用することで、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。

クラウドによる持続可能性は、AWS 技術を使用して、より幅広い持続可能性の課題を解決することで実現します。例えば、機械学習サービスを使用して、産業機械の異常な動作を検出できます。この検出データを使用し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を保証できます。

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は、これらの設計原則を適用して持続可能性を最大化し、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収益率をモデル化し、持続可能性目標に必

要な投資のリソースを所有者に提供します。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、プロセス、ストレージを排除するか最小化して、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し評価します。最新の効率的な技術を迅速に導入できるように、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、電力やネットワーキングなど、データセンターに共通するコンポーネントの影響をお客様間で共有できます。マネージドサービスは AWS が大規模に運用し、効率的な運用を保証しています。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、Simple Storage Service (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、軽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、そしてこの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: 本番環境に変更をデプロイします。
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。

シナリオの例

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細については、このドキュメントの後半で説明します。

ワークロードと使用するリソースをレビューします。大規模なデプロイや頻繁に使用する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 シナリオの例 に適用することで、改善目標として考えられる次のベストプラクティスを特定しました。

- 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

- 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
- 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
- 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 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る、作業の単位を完了するために必要なリソースを理解します。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 サンプルコードは、Amazon Athena の助けを借りてレポートを確認および分析するのに役立ちます。

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ます。評価する改善点のタイプ、および改善の対象となるリソースに最も当てはまるプロキシメトリクスを選択します。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、例えば、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。

重要業績評価指標

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。

$$Resources\ provisioned\ per\ unit\ of\ work = \frac{Proxy\ metric\ for\ provisioned\ resource}{Business\ metric\ for\ outcome}$$

KPI 式

作業単位あたりのリソースを 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	トランザクションあたりの 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	トランザクションあたりの GB	プロビジョニング合計を減らす
ネットワーク	トランザクションあたりの転送 GB またはトランザクションあ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、作業単位あ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	トランザクションあたりの 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	トランザクションあたりの GB の削減 %	プロビジョニング合計を減らす
ネットワーク	トランザクションあたりの転送 GB またはトランザクションあ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。これらのトレードオフ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 シナリオの例 に適用することで、目標改善点を次のように評価しました。

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	中程度	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	高い	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オペレーションが含まれます。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 シナリオの例 に適用して、目標の改善点を次のように優先しました。

優先度	改善点	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	高い	低	低
2	予測スケーリングを実装する	中程度	低	低

ファイル圧縮の更新は、可能性が高く低コスト、そしてリスクを考慮すると、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します。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あたりの使用リソースを測定します。

作業単位あ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。

作業単位あ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]
期待どおり	期待どおり	改善を続行
期待に満たなかった	期待どおり	改善を続行
期待どおり	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 シナリオの例 に適用し、テストを実行して成功を検証しました。

改善した圧縮アルゴリズムでテストを行った結果、作業単位あ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% と期待どおりであり、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 シナリオの例 に適用し、次のアクションを実行しました。

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:

- 作業単位あ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 シナリオの例 に適用して、次の結果を測定しました。

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、貴社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。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。例えば、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。それぞれの領域には、使用率を最大化し、無駄を最小化し、ワークロードをサポートするためにデプロイ、供給されるリソースの総量を最小化することで、クラウドワークロードの持続可能性に対する影響を軽減するためのベストプラクティスがあります。

トピック

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

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活用するメリット: ワークロードを Amazon 再生可能エネルギープロジェクトまたは公開されている炭素強度の低いリージョンの近くに配置すると、クラウドワークロードのカーボンフットプリントを減らすのに役立ちます。

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

- 候補地域の絞り込み: 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 (データ主権など) 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がリージョンに備えられているかどうかを確認します。
  - AWS 料金見積りツール 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。
- 候補の選択: 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 (市場ベースおよびロケーションベースの方法) に基づいて、毎年のカーボンフットプリントを追跡して比較します。
  - 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

関連ドキュメント:

- 炭素排出量の推定の理解
- 世界中の Amazon
- 再生可能エネルギーの方法論
- 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:

- AWS re:Invent 2023 - Sustainability innovation in AWS Global Infrastructure
- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future
- AWS re:Invent 2022 - Delivering sustainable, high-performing architectures
- AWS re:Invent 2022 - Architecting sustainably and reducing your AWS carbon footprint
- AWS re:Invent 2022 - Sustainability in AWS global infrastructure

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さずに、容量を増加させたままにする。

このベストプラクティスを活用するメリット: ワークロードの伸縮性を設定およびテストすることで、需要とクラウドリソースの供給を効率的に一致させ、容量の過剰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で、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備えます。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。

需要は、一定の場合も変動する場合もあり、管理面の負担を軽減するには、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垂直方向に変更し (スケールアップ/スケールダウン)、インスタンス数を水平方向に変更して (スケールイン/スケールアウト)、またはこれらの組み合わせで調整します。

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モニタリングし、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: 日次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: ネイティブにスケーリングするサービス (サーバーレスなど) を設計によって選択するか、機能として自動スケーリングを提供します。

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することで、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以下に示します。

自動スケーリングメカニズム	使用する場所
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使用できる Amazon EC2 インスタンスの数が正しいことを検証するために使用します。
Application Auto Scaling	Lambda 関数や Amazon Elastic Container Service (Amazon ECS) サービスなど、Amazon EC2 を超えた個々の AWS のサービスのリソースを自動的にスケールするために使用します。
Kubernetes Cluster Autoscaler	AWS の Kubernetes クラスターを自動的にスケールするために使用します。

- スケーリングは、一般的に、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議論されます。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の設定を検討してください。
- 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種類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する場合は、100% の CPU 使用率が予想されるため、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、スケーリングポリシーにカスタマイズされたメトリクス (メモリ使用率など) を使用できます。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ある必要があり、インスタンスが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の代わりに、動的スケーリングを使用します。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して、Auto Scaling グループのスケーリングアクティビティを確認できます。
- 予測可能なパターンについてワークロードを評価し、予測および計画された需要の変化を想定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 の予測スケーリング」を参照してください。

リソース

関連ドキュメント:

- 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
- Introducing Native Support for Predictive Scaling with Amazon EC2 Auto Scaling
- Karpenter の概要 - オープンソースの高性能 Kubernetes Cluster Autoscaler
- Amazon ECS クラスターの Auto Scaling を深く探る

関連動画:

- AWS re:Invent 2023 - Scaling on AWS for the first 10 million users
- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future
- AWS re:Invent 2022 - Build a cost-, energy-, and resource-efficient compute environment

- AWS re:Invent 2022 - Scaling containers from one user to millions
- AWS re:Invent 2023 - Scaling FM inference to hundreds of models with Amazon SageMaker AI
- AWS re:Invent 2023 - Harness the power of Karpenter to scale, optimize & upgrade Kubernetes

関連する例:

- Autoscaling

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、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と整合すると、ビジネスニーズを満たすと同時にリソース使用を最適化できます。

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: 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かかわります。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

- 持続可能性の目標を理解する: 炭素削減やリソース使用率の向上など、組織内の持続可能性の目標を特定します。
- SLA の確認: SLA を評価して、ビジネス要件をサポートしているかどうかを評価します。SLA を超えている場合は、さらに見直しを行います。

- トレードオフを理解する: ワークロードの複雑さ (大量の同時接続ユーザーなど)、パフォーマンス (レイテンシーなど)、持続可能性への影響 (必要なリソースなど) のトレードオフを理解します。通常、2 つの要素に優先順位を付けると、3 つ目の要素が犠牲になります。
- SLA を調整する: 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: 可用性の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向上させると、環境への影響が大きくなる可能性があります。
  - 持続可能性とセキュリティ: ワークロードが過度に安全であれば、環境への影響が大きくなる可能性があります。
- 可能であれば持続可能性 SLA を定義する: ワークロードに持続可能性 SLA を含めます。例えば、コンピューティングインスタンスの持続可能性に関する SLA として最小の使用率レベルを定義します。
- 効率的な設計パターンを使用する: 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 AWS 上のマイクロサービスなどの設計パターンを使用します。
- 説明責任を伝達して確立する: 開発チームや顧客を含む関連するすべての関係者と SLA を共有します。レポートを使用して SLA を追跡およびモニタリングします。SLA の持続可能性目標を達成するための説明責任を割り当てます。
- インセンティブと報酬を使用する: インセンティブと報酬を使用して、持続可能性の目標に沿った SLA を達成または超過します。
- 見直しと反復: SLA を定期的に見直して調整し、進化する持続可能性とパフォーマンスの目標に合致していることを確認します。

リソース

関連ドキュメント:

- 回復性のパターンとトレードオフを理解して、効率的なクラウド設計を実現
- SaaS プロバイダ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:

- AWS re:Invent 2023 - Capacity, availability, cost efficiency: Pick three

- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future
- AWS re:Invent 2023 - Advanced integration patterns & trade-offs for loosely coupled systems
- AWS re:Invent 2022 - Delivering sustainable, high-performing architectures
- AWS re:Invent 2022 - Build a cost-, energy-, and resource-efficient compute environment

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: 未使用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

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

- 棚卸しを実施する: 包括的な棚卸しを実施して、ワークロード内のすべてのアセットを特定します。
- 使用率を分析する: 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。
- 未使用のアセットを削除する: 不要になったアセットを削除する計画を立てます。
  - 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。

  - 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。
- サードパーティーに伝達する: 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ーに、アセットの生成と保存を止めるように指示します。冗長アセットを統合するように依頼します。
- ライフサイクルポリシーを使用する: ライフサイクルポリシーを使用して、未使用のアセットを自動的に削除します。
  - Amazon S3 ライフサイクルを使用すると、オブジェクトのライフサイクル全体を管理できます。
  - Amazon Data Lifecycle Manager を使用して、EBS スナップショットと Amazon EBS-backed AMI の作成、保持、削除を自動化できます。
- 確認と最適化をする: 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

関連ドキュメント:

- 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:

- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future
- AWS re:Invent 2022 - Preserving and maximizing the value of digital media assets using Amazon S3
- AWS re:Invent 2023 - Optimize costs in your multi-account environments

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などのロケーションオプション、そして AWS Outposts、AWS Local Zones などのエッジロケーションから構成されます。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。
  - Amazon CloudWatch や AWS CloudTrail などのモニタリングツールを使用して、ネットワークアクティビティに関するデータを収集します。
  - 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。
  - サステナビリティの目標: 「リージョンの選択」を参照してください。
  - データがある場所: 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。
  - ユーザーがいる場所: ユーザー向けアプリケーションの場合は、ワークロードのユーザーに近いリージョン (1 つまたは複数) を選択します。
  - その他の制約:「ワークロードのリージョンを選択する際の考慮事項」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。

- 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データに使用すると、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	どのようなときに使うか
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。

- 以下のように、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	どのようなときに使うか
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 Functions	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。
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

関連ドキュメント:

- 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 III : ネットワーキング
- Amazon ElastiCache のドキュメント
- Amazon CloudFront とは何ですか?
- Amazon CloudFront の主な特徴
- AWS グローバルインフラストラクチャ
- AWS Local Zones および AWS Outposts などエッジワークロードに適したテクノロジーの選択
- プレイスメントグループ
- AWS Local Zones
- AWS Outposts

関連動画:

- Demystifying data transfer on AWS
- Scaling network performance on next-gen Amazon EC2 instances
- AWS Local Zones 解説動画
- AWS Outposts: 概要と機能紹介
- AWS re:Invent 2023 - エッジワークロードとオンプレミスワークロードの移行戦略
- AWS re:Invent 2021 - AWS Outposts: Bringing the AWS experience on premises
- AWS re:Invent 2020 - AWS Wavelength: Run apps with ultra-low latency at 5G edge
- AWS re:Invent 2022 - AWS Local Zones: Building applications for a distributed edge
- AWS re:Invent 2021 - Amazon CloudFront で低レイテンシーウェブサイトを構築
- AWS re:Invent 2022 - AWS Global Accelerator でパフォーマンスと可用性を向上
- AWS re:Invent 2022 - AWS でグローバルなワイドエリアネットワークを構築
- AWS re:Invent 2020 - Amazon Route 53 でグローバルなトラフィック管理を実現

関連する例:

- AWS ネットワーキングワークショップ
- 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

サービスを利用するためにチームメンバーが使用するリソース、その予想ライフサイクル、および経営と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

- 省エネ型ワークステーションを使用する: チームメンバーに省エネ型ワークステーションと周辺機器を支給します。これらのデバイスで効率的な電源管理機能 (低電力モードなど) を使用して、エネルギー使用量を削減します。
- 仮想化を利用する: 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、アップグレードの必要性とデバイス要件を軽減します。
- リモートコラボレーションを奨励する: Amazon Chime および AWS Wickr などのリモートコラボレーションツールを使用するようチームメンバーに奨励して、出張の必要性や出張に関連する炭素排出量を削減します。
- エネルギー効率が優れたソフトウェアを使用する: 不要な機能やプロセスを削除または無効にして、チームメンバーにエネルギー効率が優れたソフトウェアを支給します。
- ライフサイクルを管理する: 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。ワークステーションまたはソフトウェアの定期的なメンテナンスと更新を行って、効率性を維持、改善します。

- リモートデバイス管理を実施する: 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

関連ドキュメント:

- Amazon WorkSpaces とは
- Amazon WorkSpaces のコストオプティマイザー
- Amazon AppStream 2.0 のドキュメント
- NICE DCV

関連動画:

- Managing cost for Amazon WorkSpaces on AWS

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:

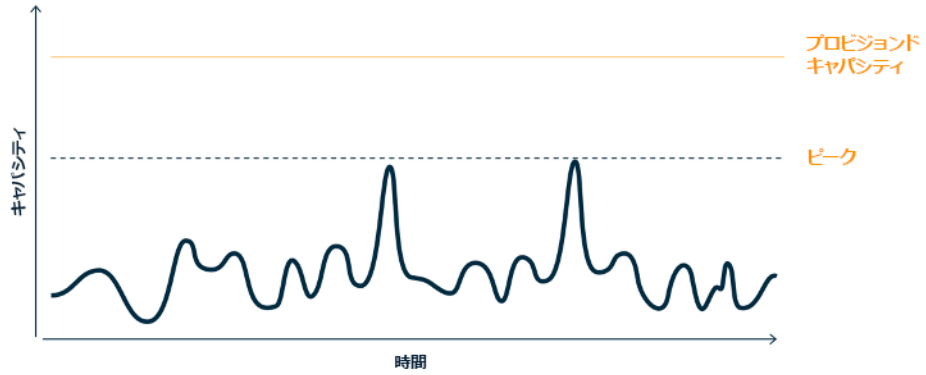
- 即時対応が不要なクライアントのリクエストを即時処理している。
- クライアントのリクエストの要件を分析していない。

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を活用しない場合のリスクレベル: 低

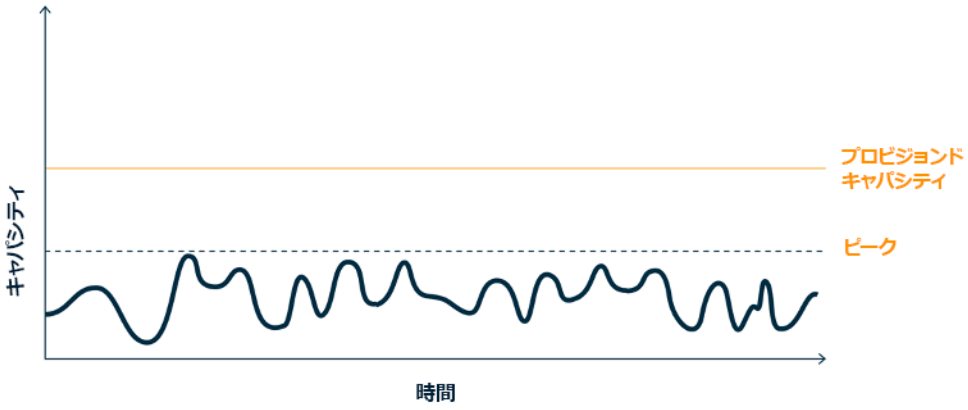
実装のガイダンス

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。



高いプロビジョンドキャパシティを必要とする 2 つの異なるピークの需要曲線

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。バッファリングを実装して、リクエストを保存し、処理を延期できます。



需要曲線とプロビジョニングされた容量に対するスロットリングの影響。

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。考慮すべき課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 を使用すると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、単独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

関連ドキュメント:

- Amazon SQS の開始方法
- キューとメッセージを使用したアプリケーション統合
- ワークロードでの API スロットリングの管理と監視
- 階層化されたマルチテナント REST API を API Gateway を使用して大規模にスロットリング
- 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:

- AWS re:Invent 2022 - Application integration patterns for microservices

- AWS re:Invent 2023 – Smart savings: Amazon EC2 cost-optimization strategies
- AWS re:Invent 2023 - Advanced integration patterns & trade-offs for loosely coupled systems

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廃止します。ワークロードコンポーネントのパフォーマンスを把握し、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スケールできます。
- 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を活用しない場合のリスクレベル: 中

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と、コンポーネントの使用率が均等になり、ワークロードのオーバープロビジョニングを抑えます。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自動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。

キューイングメカニズム	説明
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 と 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を組み合わせると、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。

スケジューリングメカニズム	説明
Amazon EventBridge スケジューラ	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 Amazon EventBridge の機能です。
AWS Glue 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 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、スケジュールに基づいて、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、停止スケジュールを設定します。

- 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、高効率のワークロードを構築します。
- AWS でサーバーレスを活用して、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。
  - または AWS Cost Explorer または AWS Compute Optimizer で適切なサイズ設定に関する推奨事項を使用して、適切なサイズ設定の機会を特定できます。
  - 詳細については、「適切なサイジング: ワークロードに適したインスタンスのプロビジョニング」を参照してください。

リソース

関連ドキュメント:

- Amazon Simple Queue Service とは
- Amazon MQ とは
- Amazon SQS に基づくスケーリング
- AWS Step Functions とは
- AWS Lambda とは
- Amazon SQS での AWS Lambda の使用
- Amazon EventBridge とは
- REST API を使用した非同期ワークフローの管理

関連動画:

- AWS re:Invent 2023 - Navigating the journey to serverless event-driven architecture
- AWS re:Invent 2023 - Using serverless for event-driven architecture & domain-driven design
- AWS re:Invent 2023 - Advanced event-driven patterns with Amazon EventBridge
- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future
- Asynchronous Message Patterns | AWS Events

関連する例:

- AWS Graviton プロセッサと Amazon EC2 スポットインスタンスを使用したイベント駆動型アーキテクチャ

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 など AWS の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。

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、無駄が最小限に抑えられ、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: 中

実装のガイダンス

クラウドワークロードで未使用または十分に活用されていないコンポーネントは、不要なコンピューティング、ストレージ、またはネットワークリソースを消費します。これらのコンポーネントを削除またはリファクタリングして、無駄を直接削減し、クラウドワークロードの全体的な効率を向上させます。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 関数のランタイムが大幅に低下すると、メモリサイズを小さくする必要があることを示す指標になります。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

- AWS リソースのインベントリ作成: AWSリソースのインベントリを作成します。AWS では、AWS Resource Explorer を有効にして AWS リソースを探索および整理できます。詳細については、「AWS re:Invent 2022 - How to manage resources and applications at scale on AWS」を参照してください。
- 使用率のモニタリング: ワークロードの重要なコンポーネント (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) の使用率メトリクスをモニタリング、キャプチャします。
- 未使用コンポーネントの特定: アーキテクチャ内の未使用のコンポーネントや使用率の低いコンポーネントを特定します。
  - 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チェックして、アイドル状態、未使用、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。

- 不要コンポーネントの削除: 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。
  - Amazon ECR における未使用イメージの自動クリーンアップ
  - AWS Config および AWS Systems Manager を使用して未使用の Amazon Elastic Block Store (Amazon EBS) ボリュームを削除
- 低使用率コンポーネントのリファクタリング: 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代わりに、1 つの Amazon RDS データベースインスタンスで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。
- 改善の評価: ワークロードによってプロビジョニングされる、作業の単位を完了するために必要なリソースを理解します。この情報を使用して、コンポーネントを削除またはリファクタリングすることによって達成された改善を評価します。
  - 持続可能性プロキシメトリクスを使用してクラウド効率を測定して追跡する、パート I: プロキシメトリクスとは
  - 持続可能性プロキシメトリクスを使用してクラウド効率を測定して追跡する、パート II: メトリクスパイプラインの確立

リソース

関連ドキュメント:

- AWS Trusted Advisor
- Amazon CloudWatch とは
- 適切なサイジング: ワークロードに適したインスタンスのプロビジョニング
- 適切なサイズ設定の推奨事項によるコストの最適化

関連動画:

- AWS re:Invent 2023 - Capacity, availability, cost efficiency: Pick three

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:

- リソースの使用量に対して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: 効率的なコードを使用すると、リソースの使用量が最小限に抑えられ、パフォーマンスが向上します。

このベストプラクティスを活用しない場合のリスクレベル: 中

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

- 効率的なプログラミング言語を使用する: ワークロードに効率的なオペレーティングシステムとプログラミング言語を使用します。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、「Rust での持続可能性」を参照してください。
- AI コーディングコンパニオンを使用する: Amazon Q Developer などの AI コーディングコンパニオンを使用して、コードの効率的な記述を検討します。
- コードレビューを自動化する: 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。
  - Amazon CodeGuru Reviewer でのコードレビューの自動化
  - Amazon CodeGuru での同時実行バグの検出
  - 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
- コードプロファイラーを使用する: 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。
  - 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
  - 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

    - 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
- モニタリングと最適化をする: 継続的なモニタリングリソースを使用して、リソース要件が高い、または最適ではない構成のコンポーネントを特定します。
    - 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。
- コードのリファクタリングまたは変換を使用する: アプリケーションのメンテナンスとアップグレードに Amazon Q コード変換を検討します。
    - Amazon Q コード変換による言語バージョンのアップグレード
    - AWS re:Invent 2023 - Amazon Q コード変換を使用してアプリケーションのアップグレードとメンテナンスを自動化する

リソース

関連ドキュメント:

- Amazon CodeGuru Profiler とは
- FPGA インスタンス
- 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:

- 顧客によって使用されるデバイスの環境に対する影響を無視する。
- 顧客によって使用され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: 顧客のデバイス用に最適化されたソフトウェアパターンと機能を実装することで、クラウドワークロードの全体的な環境への影響を軽減できます。

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

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。

- 下位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

- 棚卸しを実施する: 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。
- エネルギー効率が優れたデバイスを使用する: エネルギー効率が優れたデバイスをアーキテクチャで使用することを検討してください。デバイスの電源管理設定を使用して、使用していないときは低電力モードに切り替えます。
- 効率的なアプリケーションを実行する: 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 または AWS Elemental MediaConvert を使用して、サイズが大きい高品質の

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、およびネット接続したテレビで再生できる形式に変換できます。

  - 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、または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。
- サプライヤーを関与させる: 持続可能な資材を使用し、サプライチェーンと環境認定に透明性を持ったデバイスサプライヤーと連携します。
- 無線通信 (OTA) アップデートを使用する: 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。
  - CI/CD パイプラインを使用してモバイルアプリケーションを更新できます。
  - AWS IoT Device Management を使用して、接続されたデバイスを大規模にリモートで管理できます。
- マネージド型 Device Farm を使用する: 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 を使用して、サポート対象のデバイスを拡大する開発を繰り返します。詳細については、以下を参照してください。SUS06-BP05 マネージド型 Device Farm を使用してテストする
- モニタリングと改善を続ける: デバイスのエネルギー使用量を追跡して、改善が必要な分野を特定します。新しいテクノロジーやベストプラクティスを活用して、これらのデバイスの環境に配慮した取り組みを強化します。

リソース

関連ドキュメント:

- AWS Device Farm とは
- WorkSpaces アプリケーションドキュメント
- NICE DCV
- FreeRTOS 実行デバイスでファームウェアを更新するための OTA チュートリアル
- 環境持続可能性のための IoT デバイスの最適化

関連動画:

- AWS re:Invent 2023 - Improve your mobile and web app quality using AWS Device Farm

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: データアクセスとストレージパターンに基づいてアーキテクチャを選択、最適化すると、開発の複雑さが軽減され、全体的な使用率が向上します。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ルできるようになります。

このベストプラクティスを活用しない場合のリスクレベル: 中

実装のガイダンス

ワークロードの長期的な持続可能性を向上させるには、ワークロードのデータアクセスとストレージ特性をサポートするアーキテクチャパターンを使用します。これらのパターンは、データを効率的に取得して処理するのに役立ちます。例えば、独自の分析ユースケースに最適化された専用サービスを使用して、AWS で最新の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

- データ特性の理解: 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。

  - データ型: 構造、半構造、非構造
  - データの増加: 制限あり、無制限
  - 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、安定
- 最適なアーキテクチャパターンの使用: 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。
  - データ永続化を有効にするパターン
  - Let's Architect! モダンデータアーキテクチャ
  - AWS のデータベース: 特定のジョブに最適なデータベースを
- 専用サービスの使用: 目的に合ったテクノロジーを使用します。
  - 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。
    - Athena でサポートされる圧縮ファイル形式
    - での ETL 入力および出力の形式オプションAWS Glue
    - Amazon Redshift を使用して Simple Storage Service (Amazon S3) から圧縮されたデータファイルをロードする
  - アーキテクチャでのデータ処理に専用の分析サービスを使用します。AWS 専用分析サービスの詳細については、「AWS re:Invent 2022 - Building modern data architectures on AWS」を参照してください。
  - 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、効率的なクエリを実行します。詳細については、「AWS データベース」および「AWS re:Invent 2022 - Modernize apps with purpose-built databases」を参照してください。
- データ移動を最小限にする: 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

関連ドキュメント:

- 列データ形式の COPY
- Firehose での入力レコード形式の変換
- 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 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 ストレージクラス
- Amazon DynamoDB を使用して CQRS イベントストアを構築する

関連動画:

- AWS re:Invent 2022 - Building data mesh architectures on AWS
- AWS re:Invent 2023 - Deep dive into Amazon Aurora and its innovations
- AWS re:Invent 2023 - Improve Amazon EBS efficiency and be more cost-efficient
- AWS re:Invent 2023 - Optimizing storage price and performance with Amazon S3
- AWS re:Invent 2023 - Building and optimizing a data lake on Amazon S3
- AWS re:Invent 2023 - Advanced event-driven patterns with Amazon EventBridge

関連する例:

- AWS 目的別データベースワークショップ
- AWS モダンデータアーキテクチャ Immersion Day
- でのデータメッシュの構築AWS

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技術と設定を使用します。必要性が減少した場合は、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、不要になったデータは削除します。

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、データの保存に工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、データの最も省工ネ的なストレージ階層を決定できます。

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

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

- データインベントリを実施する: 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。
- データをグループ分けする: 組織に対するリスクに基づいて、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、「データを分類してスタートアップ企業を保護するための 4 つの簡単なステップ」を参照してください。

- データ分類レベルとポリシーを定義する: データグループごとに、データ分類レベル (パブリックポリシーや機密ポリシーなど) と処理ポリシーを定義します。分類にそってデータにタグを付けます。データ分類カテゴリの詳細については、データ分類に関するホワイトペーパーを参照してください。
- 定期的にレビューする: タグ付けされていないデータや分類されていないデータがないか、環境を定期的にレビューして監査します。オートメーションを使用してこのデータを特定し、データを適切に分類してタグ付けします。例として、「AWS Glue でのデータ検出とカタログ化」を参照してください。
- データカタログを作成する: 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。
- 文書化する: 各データクラスのデータ分類ポリシーと処理手順を文書化します。

リソース

関連ドキュメント:

- データ分類に AWS クラウドを活用
- AWS Organizations Tag policies

関連動画:

- AWS re:Invent 2022 - Enabling agility with data governance on AWS
- AWS re:Invent 2023 - Data protection and resilience with AWS storage

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良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。

実装手順

- データとアクセスパターンの特徴を評価する: 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。
  - データ型: 構造、半構造、非構造
  - データの増加: 制限あり、無制限
  - 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、頻度、急増、安定
- 適切なストレージ技術を選択する: 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 との間でオブジェクトを転送し、オブジェクトにアクセスするには、Transfer Acceleration やアクセスポイントなどのサービスを使用して、ロケー

タイプ	テクノロジー	主な特徴
		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 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)	複数の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張および縮小し、一貫した低レイテンシーを実現するようにパフォーマンスが最適化されています。
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 のレイテンシー、スループット、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適したファイルシステムを選択する際には、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) のために設計された、スケーラブルな高性能ブロックストレージサービスです。Amazon EBS には、トランザクション、IOPS を多用するワークロード用の SSD ベースのストレージと、スループットを多用するワークロード用の HDD ベースのストレージが含まれています。
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。

- ストレージ割当を自動化する: 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 と 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。
- 適切なストレージクラスを選択する: データに適したストレージクラスを選択します。
  - 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ます。1 つのバケットには、すべてのストレージクラスに保存されているオブジェクトを含めることができます。
  - Amazon S3 ライフサイクルポリシーを使用して、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、アプリケーションに変更を必要としません。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

関連ドキュメント:

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

関連動画:

- AWS re:Invent 2023 - Improve Amazon EBS efficiency and be more cost-efficient
- AWS re:Invent 2023 - Optimizing storage price and performance with Amazon S3
- AWS re:Invent 2023 - Building and optimizing a data lake on Amazon S3
- AWS re:Invent 2022: Building modern data architectures on AWS
- AWS re:Invent 2022: Modernize apps with purpose-built databases
- AWS re:Invent 2022 - Building data mesh architectures on AWS
- AWS re:Invent 2023 - Deep dive into Amazon Aurora and its innovations
- AWSre:Invent 2023: Advanced data modeling with Amazon DynamoDB

関連する例:

- Amazon S3 の例
- AWS 目的別データベースワークショップ
- 開発者向けデータベース
- AWS モダンデータアーキテクチャ Immersion Day
- AWS でのデータメッシュの構築

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と、ワークロード内の効率的なデータアクセスと保持が保証されます。

このベストプラクティスを活用しない場合のリスクレベル: 中

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。データストレージと計算の経時的な効率を向上させるには、データの経時的な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実装します。

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。このプラクティスにより、アクティブなデータの保存と取得が最小限に抑えられ、エネルギー消費量が低下します。さらに、古いデータのアーカイブや削除などのプラクティスは、規制コンプライアンスとデータガバナンスをサポートします。

実装手順

- データ分類の使用: ワークロード内のデータセットを分類します。
- 処理規則の定義: データクラスごとに処理手順を定義します。
- 自動化の有効化: 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。

ストレージサービス	自動ライフサイクルポリシーを設定する方法
Amazon S3	Amazon S3 ライフサイクルを使用すると、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、変更中、または予測不可能な場合は、Amazon S3 Intelligent-Tiering を使用できます。これにより、アクセスパターンがモニタリングされ、アクセスされていないオブジェクトが低コストのアクセス階層に自動的に移動します。Amazon S3 ストレージレンズメトリクスを活用して、ライフサイクル管理の最適化の機会とギャップを特定できます。

ストレージサービス	自動ライフサイクルポリシーを設定する方法
Amazon Elastic Block Store	Amazon Data Lifecycle Manager を使用して、EBS スナップショットと Amazon EBS-backed AMI の作成、保持、削除を自動化できます。
Amazon Elastic File System	Amazon EFS のライフサイクル管理では、ファイルシステムのファイルストレージが自動的に管理されます。
Amazon Elastic Container Registry	Amazon ECR ライフサイクルポリシーでは、年数またはカウントに基づいたイメージの有効期限を使用してコンテナイメージのクリーンアップを自動化できます。
AWS Elemental MediaStore	オブジェクトの MediaStore コンテナ内保存期間を管理する、オブジェクトのライフサイクルポリシーを作成することができます。

- 未使用アセットの削除: 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 Logs 保持などのネイティブサービス機能を活用します。
- 集約と圧縮: 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

関連ドキュメント:

- Amazon S3 ストレージクラス分析を使用して Amazon S3 ライフサイクルルールを最適化する
- Evaluating Resources with AWS Config ルール

関連動画:

- AWS re:Invent 2021 - Amazon S3 Lifecycle best practices to optimize your storage spend
- AWS re:Invent 2023 - Optimizing storage price and performance with Amazon S3
- Simplify Your Data Lifecycle and Optimize Storage Costs With Amazon S3 Lifecycle

- Reduce Your Storage Costs Using Amazon S3 Storage Lens

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オーバープロビジョニングを最小限に抑えると、アイドル状態のリソースが減少し、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: 中

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するようにします。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。Amazon EBS Elastic Volumes では、EBS ボリュームのボリュームサイズの増加、ボリュームタイプの変更、パフォーマンスの調整を行うことができます。
- 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

関連ドキュメント:

- Extend the file system after resizing an EBS volume
- Modify a volume using Amazon EBS Elastic Volumes
- Amazon FSx Documentation
- Amazon Elastic File System とは

関連動画:

- Deep Dive on Amazon EBS Elastic Volumes
- Amazon EBS and Snapshot Optimization Strategies for Better Performance and Cost Savings
- Optimizing Amazon EFS for cost and performance, using best practices

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、またはまったく行わない。
- 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 バージョニングを有効にしている。

このベストプラクティスを活用するメリット: 不要なデータを削除すると、ワークロードに必要なストレージサイズとワークロードの環境への影響が軽減されます。

このベストプラクティスを活用しない場合のリスクレベル: 中

実装のガイダンス

不要な冗長データセットを削除すると、ストレージコストと環境フットプリントを削減できます。この方法により、コンピューティングリソースが不要なデータではなく重要なデータのみを処理するため、コンピューティングの効率も向上する可能性があります。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。ネイティブデータレプリケーションと冗長性のためのサービス機能を使用します。

実装手順

- パブリックデータセットの評価: AWS Data Exchange および AWS の Open Data で公開されている既存のデータセットを使用し、データの保存を回避できるかどうかを評価します。
- データの重複排除: 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。

ストレージサービス	重複排除メカニズム
Amazon S3	AWS Lake Formation FindMatches の新しい FindMatches ML Transform を使用して、データセット (識別子のないレコードを含む) 間で一致するレコードを検索します。
Amazon FSx	Amazon FSx for Windows のデータ重複排除を使用します。
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、最後にスナップショットを作成した時点から、ボリューム上で変更のあるブロックだけが保存されます。

- ライフサイクルポリシーの使用: ライフサイクルポリシーを使用して、未使用のアセットを自動的に削除します。削除には、Amazon DynamoDB の有効期限や Amazon S3 Lifecycle、Amazon CloudWatch Logs 保持などのネイティブサービス機能を使用します。
- データ仮想化の使用: 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
  - Amazon Redshift データ共有を使用したデータパターンの最適化
- 増分バックアップの使用: 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。
- ネイティブ耐久性の使用: セルフマネージドテクノロジー (独立ディスクの冗長アレイ (RAID) 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。
- 効率的なログの使用: 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。
- 効率的なキャッシュの使用: 正当化された場合にのみキャッシュを事前入力します。
- 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。
- 古いバージョンのアセットの削除: 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

関連ドキュメント:

- Change log data retention in CloudWatch Logs
- Data deduplication on Amazon FSx for Windows File Server
- Features of Amazon FSx for ONTAP including data deduplication
- Amazon CloudFront のファイルを無効化する
- AWS Backup を使用してAmazon EFS ファイルシステムをバックアップおよび復元する
- What is Amazon CloudWatch Logs?
- バックアップの概要
- AWS Lake Formation を使用してデータセットの統合および重複の削除を実施

関連動画:

- Amazon Redshift Data Sharing Use Cases

関連する例:

- Amazon Athena で Amazon S3 サーバーアクセスログを分析する方法を教えてください。

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: 共有ファイルシステム、ストレージを使用すると、データをコピーしなくても 1 つ以上のコンシューマーに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を活用しない場合のリスクレベル: 中

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

- 共有ストレージの使用: 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。

ストレージオプション	どのようなときに使うか
Amazon EBS マルチアタッチ	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 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アタッチできます。
Amazon EFS	「Amazon EFS を選択するタイミング」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。

- 必要なときにのみデータを取得: 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 にバックアップされた Amazon FSx for Lustre ファイルシステムを作成し、処理ジョブ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。
- 不要データの削除:「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」で説明されているように、使用パターンに応じてデータを削除します。
- 非アクティブなクライアントのデタッチ: 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

関連ドキュメント:

- Linking your file system to an Amazon S3 bucket
- Using Amazon EFS for AWS Lambda in your serverless applications
- 新機能 – Amazon EFS Intelligent-Tiering がアクセスパターンの変化に応じてワークロードのコストを最適化
- 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
- AWS re:Invent 2023 - What's new with AWS file storage
- AWS re:Invent 2023 - File storage for builders and data scientists on Amazon Elastic File System

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

- 近接性の使用: ワークロードのリージョンを選択するときは、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。
- パーティションサービス: 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の使用: 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。
- データ移動を最小限に抑える: 未使用のデータは移動しません。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。
  - 詳細なデータ (レコードレベルの情報は不要) を集約します。
  - 「Well-Architected Lab - Optimize Data Pattern Using Amazon Redshift Data Sharing」を参照してください。
  - AWS Lake Formation のクロスアカウントのデータ共有を考慮します。
- エッジサービスの使用: 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	どのようなときに使うか
Lambda@Edge	オブジェクトがキャッシュにないときに実行され、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 Functions	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

関連ドキュメント:

- 持続可能な AWS インフラストラクチャの最適化、第三部:ネットワーキング編
- AWS グローバルインフラストラクチャ
- Amazon CloudFront 特徴 (グローバルエッジネットワーク他)
- Compressing HTTP requests in Amazon OpenSearch Service
- Intermediate data compression with Amazon EMR
- 圧縮されたデータファイルを Amazon S3 からロードする
- 圧縮ファイルを供給する

関連動画:

- Demystifying data transfer on AWS

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: 重要度の低いデータのバックアップを回避することでワークロードに必要なストレージリソースを減らし、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を活用しない場合のリスクレベル: 中

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

- データの分類: SUS04-BP01 データ分類ポリシーを実装する で解説しているとおりにデータ分類ポリシーを実装します。
- バックアップ戦略の策定: データの重要度区分を用いて、目標復旧時間 (RTO) と目標復旧時点 (RPO) に基づき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。
- 自動化されたバックアップの使用: 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 はフルマネージド型のバックアップサービスであり、AWS のサービス、クラウド内、およびオンプレミス間で簡単に一元化およびデータ保護を自動化できます。AWS Backup を使用した自動バックアップの作成方法に関する実践的ガイダンスについては、「Well-Architected Labs - Testing Backup and Restore of Data」を参照してください。
  - AWS Backup を使用して Amazon EFS のバックアップを自動化しバックアップコストを最適化します。

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

関連ドキュメント:

- AWS Backup を使用してAmazon EFS ファイルシステムをバックアップし復元する
- Amazon EBS snapshots
- Amazon Relational Database Service でバックアップを操作する
- APN パートナー: バックアップの支援が可能なパートナー

- AWS Marketplace: バックアップに活用できる製品
- Amazon EFS のバックアップ
- Amazon FSx for Windows File Server のバックアップ
- Amazon ElastiCache (Redis OSS) のバックアップと復元

関連動画:

- AWS re:Invent 2023 - Backup and disaster recovery strategies for increased resilience
- AWS re:Invent 2023 - What's new with AWS Backup
- AWS re:Invent 2021 - Backup, disaster recovery, and ransomware protection with AWS

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の数を最小化し、個別のワークロードにおいて最も効率的な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ことで、ワークロードによる環境への影響を減らし、費用を節約して、パフォーマンス基準を維持することができます。

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

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 クラウドでは、AWS Auto Scaling などさまざまなメカニズムによって、リソースの数を必要に応じて柔軟に拡張または縮小することができ、需要の変化に対応することができます。また、リソースの変更を最小限の労力で実行できる API と 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 ツールが提供する適切なサイジング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

- インスタンスタイプを選択する: ニーズに最適なインスタンスタイプを選びます。Amazon Elastic Compute Cloud インスタンスの選び方や、属性ベースのインスタンスの選択といったメカニズムの使用方法については、以下を参照してください。
  - ワークロードに適した 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 Fleet 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 グループを作成する
- スケールする: 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する: 複数のコンピューティング購入オプションを使用することで、インスタンスの柔軟性、スケーラビリティ、コスト削減の間のバランスを取ります。
  - Amazon EC2 オンデマンドインスタンスは、インスタンスタイプやロケーション、処理時間の柔軟性が低い、ステートフルでスパイクが発生しやすい新規のワークロードに最適です。
  - Amazon EC2 スポットインスタンスは、耐障害性と柔軟性を備えたアプリケーションに関して、他の方法を補完する優れた方法です。
  - 定常状態のワークロードには、Compute Savings Plans を活用すれば、ニーズの変化 (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など) に柔軟に対応できます。
- 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する: 多様なインスタンスやアベイラビリティーゾーンを使用することで、アプリケーションの可用性を最大化し余剰のキャパシティを活用することができます。

- インスタンスを適切なサイズに設定する: AWS ツールの適切なサイジング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。詳細については、「Optimizing your cost with Rightsizing Recommendations」と「適切なサイジング: ワークロードに適したインスタンスのプロビジョニング」を参照してください。
  - 適切なサイジングの機会を特定するには、AWS Cost Explorer の適切なサイジングのレコメンデーション、または AWS Compute Optimizer を使用します。
- サービスレベルアグリーメント (SLA) を見直す: 容量を一時的に減らせるように SLA を見直すと同時に、オートメーションを使用して代替のリソースをデプロイします。

リソース

関連ドキュメント:

- 持続可能な AWS インフラストラクチャの最適化、第一部:コンピュート編
- 新機能 — EC2 Auto Scaling と 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 Compute Optimizer のドキュメント
- Operating Lambda: パフォーマンスの最適化 – Part 2
- Auto Scaling ドキュメント

関連動画:

- AWS re:Invent 2023 – What's new with Amazon EC2
- AWS re:Invent 2023 - Smart savings: Amazon Elastic Compute Cloud cost-optimization strategies
- AWS re:Invent 2022 - Optimizing Amazon Elastic Kubernetes Service for performance and cost on AWS
- AWS re:Invent 2023 - Sustainable compute: reducing costs and carbon emissions with AWS

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 の設定で 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。
- AWS Compute Optimizer など、AWS の適切なサイジングツールのレコメンデーションを確認しない。

このベストプラクティスを活用するメリット: エネルギー効率に優れた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下げることができます。

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

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

- インスタンスタイプを探して詳しく調べる: ワークロードによる環境への影響を減らすことができるインスタンスタイプを特定します。
  - AWS の最新情報を購読して、AWS のテクノロジーとインスタンスに関する最新情報を入手します。
  - 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 でのワットあたりのエネルギー使用量が最も優れた、AWS Graviton ベースのインスタンスの詳細については、「re:Invent 2020 - Deep dive on AWS Graviton2 processor-powered Amazon EC2 instances」と「Deep dive into AWS Graviton3 and Amazon EC2 C7g instances」をご覧ください。
- 環境への影響が最少のインスタンスタイプを使用する: 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します。
  - 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。ワークロードを AWS Graviton に移行する方法の詳細については、「AWS Graviton Fast Start でイノベーションを加速する」と「Considerations when transitioning workloads to AWS Graviton-based Amazon Elastic Compute Cloud instances」を参照してください。
- AWS マネージドサービスを使用するときは、AWS Graviton の利用を検討します。
- 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、Amazon EC2 DL1 など、特定のワークロード専用のハードウェアを活用します。AWSInferentia インスタンス (Inf2 インスタンスなど) を使用することで、同等の Amazon EC2 インスタンスに比べ、ワットあたりのパフォーマンスが最大で 50% 向上します。
- ML 推論エンドポイントのサイズを適正化するときは、Amazon SageMaker AI Inference Recommender を使用します。
- スパイクが発生しやすいワークロード (追加の容量が必要になる頻度が低いワークロード) には、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あるワークロードには、Amazon EC2 スポットインスタンスを使用することで、クラウドの全体的な使用率を増やし、未使用のリソースが持続可能性に与える影響を減らします。

- 運用しながら最適化する: ワークロードインスタンスを運用し、最適化します。
  - エフェメラルなワークロードの場合は、インスタンスの Amazon CloudWatch メトリクス (`CPUUtilization` など) を測定し、インスタンスがアイドル状態か、またはあまり利用されていないかを特定します。
  - 安定したワークロードの場合は、AWS の適切なサイジングツール (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適切なサイジングの機会を特定します。その他の例と推奨事項については、以下のラボを参照してください。
    - Well-Architected Lab - Rightsizing Recommendations
    - Well-Architected Lab - Rightsizing with Compute Optimizer
    - Well-Architected Lab - Optimize Hardware Patterns and Observe Sustainability KPIs

リソース

関連ドキュメント:

- 持続可能な AWS インフラストラクチャの最適化、第一部:コンピュート編
- AWS Graviton プロセッサ
- Amazon EC2 DL1 インスタンス
- キャパシティ予約フリート
- EC2 フリートとスポットフリート
- Function Configuration
- Amazon EC2 Fleet の属性ベースのインスタンスタイプの選択
- 持続可能で、効率的かつコストが最適化されたアプリケーションを AWS で構築する
- How the Contino Sustainability Dashboard Helps Customers Optimize Their Carbon Footprint

関連動画:

- AWS re:Invent 2023 - AWS Graviton: The best price performance for your AWS workloads
- AWS re:Invent 2023 - New Amazon Elastic Compute Cloud generative AI capabilities in AWS マネジメントコンソール
- AWS re:Invent 2023 = What's new with Amazon Elastic Compute Cloud
- AWS re:Invent 2023 - Smart savings: Amazon Elastic Compute Cloud cost-optimization strategies
- AWS re:Invent 2021 - Deep dive into AWS Graviton3 and Amazon EC2 C7g instances
- AWS re:Invent 2022 - Build a cost-, energy-, and resource-efficient compute environment

関連する例:

- Solution: Guidance for Optimizing Deep Learning Workloads for Sustainability on AWS

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。当社は、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を活用しない場合のリスクレベル: 中

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 にはデータベースのマネージドサービスがあります。Amazon Athena、Amazon EMR 、Amazon OpenSearch Service には分析のマネージドサービスがあります。

実装手順

1. ワークロードのリストを作成する: 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. コンポーネントの候補を特定する: 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。

タスク	AWS で使用するもの
データベースのホスティング	独自の Amazon RDS インスタンスを Amazon Elastic Compute Cloud (Amazon EC2) で維

タスク	AWS で使用するもの
	持する代わりに、マネージド型の 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代わりに、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	AWS Amplify Amplify ホスティングを、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、使用します。

3. 移行計画を作成する: 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提供するサービスです。移行の計画を立てる際に、充分な情報に基づいて意思決定を行えるようになります。
4. テストを行う: マネージドサービスに移行する前にサービスをテストします。
5. セルフホスト型のサービスを置き換える: 作成した移行計画に基づいて、セルフホスト型のサービスをマネージドサービスに置き換えます。
6. モニタリングし調整する: 移行の完了後は、サービスを継続的にモニタして、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:

- AWS re:Invent 2021 - Cloud operations at scale with AWS Managed Services
- AWS re:Invent 2023 - Best practices for operating on AWS

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

- コンピューティングアクセラレーターの調査: 要件に対応できる高速コンピューティングインスタンスを特定します。
- 専用ハードウェアの使用: 機械学習ワークロードには、AWSTrainium、AWSInferentia、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活用してください。AWSInf2 インスタンスなどの 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。
- 使用状況メトリクスのモニタリング: 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、「Amazon CloudWatch で NVIDIA GPU メトリクスを収集する」のように、CloudWatch エージェントを使用して GPU の `utilization_gpu` や `utilization_memory` などのメトリクスを収集できます。
- 適切なサイズ: 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 AI での深層学習トレーニング時における、GPU パフォーマンスチューニングのための I/O 最適化
- 最新に保つ: 最新の高性能ライブラリと GPU ドライバーを使用します。
- 不要なインスタンスの解放: 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
- ワークロードに適した 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 Instances
- Amazon SageMaker AI でコンピュータビジョン推論に最適な AI アクセラレータとモデルコンパイルを選択

関連動画:

- AWS re:Invent 2021 - How to select Amazon EC2 GPU instances for deep learning
- AWS Online Tech Talks - Deploying Cost-Effective Deep Learning Inference
- AWS re:Invent 2023 - Cutting-edge AI with AWS and NVIDIA
- AWS re:Invent 2022 - [NEW LAUNCH!] Introducing AWS Inferentia2-based Amazon EC2 Inf2 instances
- AWS re:Invent 2022 - Accelerate deep learning and innovate faster with AWS Trainium
- AWS re:Invent 2022 - Deep learning on AWS with NVIDIA: From training to deployment

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、持続可能性への影響を軽減する機会を探します。

ベストプラクティス

- SUS06-BP01 持続可能性の目標を伝え、段階的に広める
- SUS06-BP02 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-BP03 ワークロードを最新に保つ
- SUS06-BP04 ビルド環境の利用率を高める
- SUS06-BP05 マネージド型 Device Farm を使用してテストする

SUS06-BP01 持続可能性の目標を伝え、段階的に広める

テクノロジーは持続可能性の重要なイネーブラーです。IT チームは、組織の持続可能性目標に向けて有意義な変化を推進する上で重要な役割を果たします。これらのチームは、会社の持続可能性目標を明確に理解し、それらの優先事項をオペレーション全体に伝え、広めるよう努める必要があります。

一般的なアンチパターン:

- 組織の持続可能性の目標や、それらがチームにどのように適用されるかがわからない。
- クラウドワークロードの環境への影響に関する認識とトレーニングが不十分である。
- 優先すべき具体的な工リアが不明である。
- 持続可能性の取り組みに従業員や顧客を関与させていない。

このベストプラクティスを活用するメリット: インフラストラクチャとシステムの最適化から革新的なテクノロジーの使用まで、IT チームは組織の二酸化炭素排出量を削減し、リソースの消費を最小限に抑えることができます。持続可能性の目標を伝えることによって、IT チームは進化する持続可能性の課題を継続的に改善し、適応できるようになります。さらに、これらの持続可能な最適化はコスト削減につながることも多く、ビジネスケースを強化できます。

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

IT チームの主な持続可能性の目標は、システムとソリューションを最適化してリソース効率を高め、組織の二酸化炭素排出量と全体的な環境への影響を最小限に抑えることです。トレーニングプログラムや運用ダッシュボードなどの共有サービスやイニシアチブは、組織が IT オペレーションを最適化し、二酸化炭素排出量を大幅に削減できるソリューションを構築する際に役立ちます。クラウドは、物理インフラストラクチャとエネルギー調達の責任をクラウドプロバイダーの共同責任に移行するだけでなく、クラウドベースのサービスのリソース効率を継続的に最適化する機会も提供します。

チームがクラウド固有の効率性と責任共有モデルを使用すると、組織の環境への影響を大幅に削減できます。これにより、組織全体の持続可能性の目標に貢献し、より持続可能な未来に向けた取り組みにおける戦略的パートナーとしてのこれらのチームの価値を実証できます。

実装手順

- 目標と目的の定義: IT プログラムの明確に定義された目標を設定します。これには、IT、持続可能性、財務など、さまざまな部門から責任ある利害関係者からの情報収集も含まれます。これらのチームは、炭素削減やリソースの最適化などの分野を含め、組織の持続可能性目標に沿った測定可能な目標を定義する必要があります。
- ビジネスの炭素会計の境界を理解する: Greenhouse Gas (GHG) Protocol などの炭素会計方法がクラウド内のワークロードにどのように関連しているかを理解します (詳細については、「クラウドの持続可能性」を参照してください)。
- 炭素会計にクラウドソリューションを使用する: AWS の炭素会計ソリューションなどのクラウドソリューションを使用して、オペレーション、ポートフォリオ、バリューチェーン全体で GHG 排出量の範囲 1、2、3 を追跡します。これらのソリューションにより、組織は GHG 排出データの取得を合理化し、レポートを簡素化し、気候戦略に役立つインサイトを得ることができます。
- IT ポートフォリオのカーボンフットプリントをモニタリングする: IT システムの二酸化炭素排出量を追跡して報告します。AWS Customer Carbon Footprint Tool を使用して、AWS使用状況から発生する炭素排出量を追跡、測定、レビュー、予測します。

- プロキシメトリクスを使用してリソースの使用状況をチームに伝達する: プロキシメトリクスを使用してリソースの使用状況を追跡してレポートします。クラウドのオンデマンド価格設定モデルでは、リソース使用量はコストに関連しており、これは一般的に理解できるメトリクスです。少なくとも、コストをプロキシメトリクスとして使用して、各チームによるリソースの使用状況と改善点を伝えます。
  - Cost Explorer で時間単位の詳細度を有効にし、コストと使用状況レポート (CUR) を作成する: CUR は、すべての AWS サービスの日単位または時間単位の使用状況の詳細度、レート、コスト、使用状況属性を示します。クラウドインテリジェンスダッシュボード とその持続可能性プロキシメトリクスダッシュボードを、コストと使用状況に基づくデータの処理と視覚化の開始点として使用します。詳細については、以下をご参照ください。
  - 持続可能性プロキシメトリクスを使用してクラウド効率を測定して追跡する、パート I: プロキシメトリクスとは
  - 持続可能性プロキシメトリクスを使用してクラウド効率を測定して追跡する、パート II: メトリクスパイプラインの確立
- 継続的な最適化と評価: 改善プロセスを使用して、効率と持続可能性のためのクラウドワークロードを含む IT システムを継続的に最適化します。最適化戦略の実装前後にカーボンフットプリントをモニタリングします。カーボンフットプリントの削減を使用して、有効性を評価します。
- 持続可能性カルチャーの醸成: トレーニングプログラム (AWS スキルビルダーなど) を使用して、持続可能性について従業員を教育します。持続可能性イニシアチブにエンゲージさせます。成功事例を共有し、称賛します。持続可能性の目標を達成した場合は、インセンティブを使用して報奨します。

リソース

関連ドキュメント:

- 炭素排出量の推定の理解

関連動画:

- AWS re:Invent 2023 - Accelerate data-driven circular economy initiatives with AWS
- AWS re:Invent 2023 - Sustainability innovation in AWS Global Infrastructure
- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future
- AWS re:Invent 2022 - Delivering sustainable, high-performing architectures
- AWS re:Invent 2022 - Architecting sustainably and reducing your AWS carbon footprint

- AWS re:Invent 2022 - Sustainability in AWS global infrastructure

関連する例:

- Well-Architected Lab - Turning cost & usage reports into efficiency reports

関連するトレーニング:

- AWS での持続可能性の変革
- SimuLearn - 持続可能性レポート
- AWS による脱炭素化

SUS06-BP02 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を活用し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

- 組織の持続可能性目標の理解と周知: 二酸化炭素削減やウォータースチュワードシップなど、組織の持続可能性目標を理解します。これらの目標をクラウドワークロードの持続可能性要件に変換します。これらの要件を主なステークホルダーに伝えます。
- 持続可能性要件のバックログへの追加: 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。
- 反復と改善: 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。
- 実用最小限の製品 (MVP) を使用したテスト: 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、テストのコストと環境への影響を削減します。
- プロセスの合理化: 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。
- トレーニングと啓発: チームメンバーを対象にトレーニングプログラムを実施して、持続可能性、および活動が組織の持続可能性目標にどのように影響するかについて教育します。
- 評価と調整: 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

関連動画:

- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future
- AWS re:Invent 2022 - Delivering sustainable, high-performing architectures
- AWS re:Invent 2022 - Architecting sustainably and reducing your AWS carbon footprint
- AWS re:Invent 2022 - Sustainability in AWS global infrastructure
- AWS re:Invent 2023 - What's new with AWS observability and operations

SUS06-BP03 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を活用し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

- プロセスの定義: 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使用します。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。
- インベントリの作成: 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。
  - AWSSystems Manager Inventory を使用すれば、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アップデートする必要があるかを迅速に把握することが可能です。

- 更新手順の学習: 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。

ワークロードコンポーネント	更新方法
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あります。

- 自動化の使用: 更新を自動化して、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 を使用すると、AMI、コンテナイメージなど、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するとシステム更新のプロセスを自動化でき、AWS Systems Manager Maintenance Windows を使用するとアクティビティをスケジュールできます。

リソース

関連ドキュメント:

- AWS アーキテクチャセンター
- AWS の最新情報
- AWS 開発者用ツール

関連動画:

- AWS re:Invent 2022 - Optimize your AWS workloads with best-practice guidance
- All Things Patch: AWS Systems Manager

SUS06-BP04 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を活用し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 (IaC) 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。本番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 (IaC) を使用する: 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する: 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。

- 使用率を最大化する: 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 とは
- AWS CodeBuild とは
- AWS での Instance Scheduler

関連動画:

- AWS re:Invent 2023 - Continuous integration and delivery for AWS

SUS06-BP05 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を活用し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

- テストの要件を定義する: 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。
- マネージド型 Device Farm を選択する: テスト要件に対応できるマネージド型 Device Farm を選択します。例えば、AWS Device Farm を使用すると、代表的なハードウェア一式における変更をテストし、その影響を理解できます。
- オートメーションを使用する: 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。
  - AWS Device Farm を CI/CD パイプラインと統合してクロスブラウザの Selenium テストを実行する
  - Building and testing iOS and iPadOS apps with AWS DevOps and mobile services
- 見直して調整する: 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

関連動画:

- AWS re:Invent 2023 - Improve your mobile and web app quality using AWS Device Farm
- AWS re:Invent 2021 - Optimize applications through end user insights with Amazon CloudWatch RUM

関連する例:

- AWS Device Farm Android 向けサンプルアプリ
- AWS Device Farm iOS 向けサンプルアプリ
- AWS Device Farm 向け Appium ウェブテスト

結論

政府の規制、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、開発者、オペレーションチームのメンバーは、組織の持続可能性の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、および運用上の優秀性と AWS クラウドワークロードの持続可能性の成果との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。リソース使用量を削減し、ワークロード全体の効率性を向上させるために行うすべてのアクションは、環境への影響の低減と、組織の広範な持続可能性の目標に貢献します。

寄稿者

本ドキュメントの寄稿者は次のとおりです。

- Amazon Web Services、Senior Efficiency Lead Solutions Architect、Sam Mokhtari
- Amazon Web Services、Principal Sustainability Solutions Architect、Brendan Sisson
- Amazon Web Services、Sustainability Tech Leader、Margaret O'Toole
- Amazon Web Services、Principal Sustainability Solutions Architect、Steffen Grunwald
- Amazon、Principal Engineer、Sustainability、Ryan Eccles
- Amazon Web Services、Principal Architect、Rodney Lester
- Amazon Web Services、VP Sustainability Architecture、Adrian Cockcroft
- Amazon Web Services、Director of Technology、Solutions Architecture、Ian Meyers

詳細情報

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

ドキュメントの改訂

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

変更	説明	日付
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は、SUS 1、SUS 3、SUS 4、SUS 5、SUS 6 の分野で新しいガイダンスで更新されました。これらのベストプラクティス分野全体でガイダンスが改善されました。SUS 6 には、「SUS06-BP01 Communicate and cascade your sustainability goals」が追加されました。SUS 6 の既存のベストプラクティスの番号が変更されました。	2024 年 11 月 6 日
ベストプラクティスガイダンスの更新	柱全体の小さな変更。	2024 年 6 月 27 日
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。	2023 年 10 月 3 日
ベストプラクティスガイダンスの更新	次の分野: 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のベストプラクティスが、新しいガイダンスで更新されました。	2023 年 7 月 13 日
新しいフレームワーク用の更新	規範ガイダンスを使用してベストプラクティスを更新、お	2023 年 4 月 10 日

	よび新しいベストプラクティスを追加。	
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。	2022 年 12 月 15 日
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。	2022 年 10 月 20 日
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。	2021 年 12 月 2 日

注意

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

AWS 用語集

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