AWS Well-Architected Framework

ゲーム業界レンズ

ゲーム業界レンズ: AWS Well-Architected Framework

Table of Contents

はじめに

発行日: 2021 年 9 月 19 日 (ドキュメント履歴)

クラウドアーキテクトは、 AWS Well-Architected Framework を活用して、クラウドアーキテクトはアプリケーションやワークロード向けの安全性、高性能、回復力、効率性を備えたインフラストラクチャを構築できます。Well-Architected Framework は、5 つの柱 (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) に基づく一貫したアプローチを通じて、お客様や AWS パートナーがアーキテクチャを評価し、リスクを修正して、ビジネス価値をもたらす設計を実装できるようにします。

このレンズでは、AWS クラウド でゲームワークロードの設計、アーキテクチャの構築、デプロイを行う方法に焦点を当てます。コンポーネントを定義し、一般的なワークロードシナリオを検討して、Well-Architected Framework の適用に役立つ設計原則について説明します。アーキテクチャの設計を開始するに当たっては、 AWS Well-Architected Framework のホワイトペーパーに記載しているベストラクティスや質問を考慮してください。このドキュメントでは、ゲーム業界のお客様向けに追加のベストプラクティスを紹介します。

このレンズでは、AWS が世界中のゲーム業界のデベロッパーやパブリッシャーとの協働を通じて培った経験に基づいて、クラウド内でのゲームの構築と運用に固有の特性に対処するためのベストプラクティスを示します。コストを最適化し、世界中のプレイヤーからの需要の変動にスケーラブルに対応できるように環境を設計および運用するためのガイダンスを提供します。また、このレンズでは、ゲームインフラストラクチャのセキュリティを確保し、パフォーマンスをチューニングして、プレイヤーに満足をもたらすためのガイダンスも提供します。

このドキュメントは、最高技術責任者 (CTO)、ゲームスタジオのテクニカルディレクター、アーキテクト、デベロッパー、オペレーションチームメンバーなど、技術担当者の方々を対象にしています。このドキュメントに目を通すことで、ゲームのアーキテクチャ設計に役立つ AWS のベストプラクティスと戦略を理解できます。

定義

AWS Well-Architected Framework は、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、コスト最適化という 5 つの柱に基づいています。AWS には、ゲームワークロード用の最新アーキテクチャの設計に役立つ複数のコアコンポーネントが用意されています。このセクションでは、主要な定義について概説します。

このホワイトペーパーでは、ゲームアーキテクチャの一部として、ゲームの構築と運用に必要なバックエンドの技術インフラストラクチャも含めています。ゲームの中には、ソーシャル機能、マルチプレイヤー機能、その他のオンライン機能を備えていないものがあり、本書で説明しているバックエンドの技術インフラストラクチャの一部は該当しない場合があります。ゲームアーキテクチャをサポートするためにデプロイすることが多いさまざまなワークロードタイプの詳細については、「シナリオ」を参照してください。

AWS クラウド は、 AWS リージョン と アベイラビリティーゾーンを中心に構築されています。リージョンは世界中の物理的場所であり、複数のアベイラビリティーゾーンが配置されています。アベイラビリティーゾーンは 1 つ以上の独立したデータセンターで構成されます。各データセンターは、冗長性のある電源、ネットワーク、接続を備えており、別々の設備に収容されています。ゲームの特性によっては、ゲームアーキテクチャの特定のコンポーネントを複数のリージョンにデプロイしてプレイヤーのパフォーマンスを向上させたり、プレイヤーの所在地に応じてカスタマイズしたエクスペリエンスを提供したりできます。

ゲームにはさまざまな種類があり、ゲームをサポートするために必要なバックエンドの技術インフラストラクチャは、開発しているゲームの種類によって異なります。人気のあるゲームの種類としては、ファーストパーソンシューティング (FPS)、ロールプレイング (RPG)、マルチプレイヤーオンライン (MMO)、バトルロワイヤル (BR)、スポーツゲーム、パズルゲームなどがあります。また、ゲームのアーキテクチャに影響を与えるさまざまなインタラクションモード (ターン制や同時プレイなど) があり、インタラクションモードごとにパフォーマンス特性は大きく異なります。

ゲームは、デスクトップ、ウェブ、モバイル、コンソールなど、1 つ以上のゲームプラットフォームでプレイできるように開発されています。また、拡張現実 (AR)、バーチャルリアリティ (VR) などの最新のインタラクションモードや、ゲームストリーミングプラットフォームを備えている場合もあります。クロスプラットフォームのゲームプレイをサポートすることが一般的な傾向になりつつあります。つまり、プレイヤーはゲームプレイを途中で保存し、他のプラットフォームで再開できます。また、他のプラットフォームのプレイヤーとゲームプレイセッションを開始することもできます。ゲームパブリッシャーは、ビデオゲームから収益を生み出すために、広告、デジタルおよび小売ベースのゲーム購入、ダウンロード可能コンテンツ (DLC) のゲーム内購入 (マイクロトランザクションと

呼ばれます)、有料サブスクリプションを通じてのゲームプレイなど、さまざまな戦略を使用できます。ゲーム業界で最も一般的な重要業績評価指標 (KPI) としては、1 日あたりのアクティブユーザー数 (DAU)、1 か月あたりのアクティブユーザー数 (MAU)、同時実行ユーザー数 (CCU)、セッション期間、インストール単価 (CPI)、プレイヤーのライフタイムバリュー (LTV)、ユーザーあたりの平均収益 (ARPU) の変動などがあります。

トピック

ゲームプラットフォーム

ビデオゲームは、プレイヤーエクスペリエンスを提供するゲームプラットフォーム上でプレイするように開発されています。ゲームプラットフォームは一般的に、クライアント入力コントロール、グラフィックス、クライアントソフトウェア (ゲームクライアントと呼ばれます)、ハードウェアで構成されており、さらにゲームプレイをサポートするプラットフォーム専用の機能を持つ場合もあります。

ゲームプラットフォームは、通常、以下のカテゴリに分かれます。

- コンソール – ゲームプレイ専用に設計されたエンターテインメントシステムです。よく知られているものとして、Sony PlayStation、Microsoft Xbox、Nintendo Switch などが該当します。コンソールでは、ゲームプラットフォームプロバイダーが製造したコンソールハードウェアに、物理配信またはデジタル配信されたゲームコンテンツをインストールすることで、ゲームをプレイできます。この定義に従うと、コンソールには Nintendo Switch などのハンドヘルド型や、Xbox、PlayStation などの据え置き型の家庭用娯楽機器が含まれます。
- パーソナルコンピュータ (PC) ゲーム – クライアントマシンにインストールしたコンピュータソフトウェアを使用してプレイするゲームであり、プレイヤーがカスタマイズできます。このため、PC ゲームは柔軟性とコントロール性が高く、プレイヤーの間で人気があります。
- ウェブゲーム – ウェブブラウザを使用してプレイするように設計されたゲームです。ウェブブラウザはさまざまな種類のデバイスにインストールできるため、通常、プレイヤーはデフォルトで複数のプラットフォーム (クロスプラットフォーム) からゲームにアクセスできるというメリットがあります。

- モバイルゲーム – 携帯電話でプレイするように開発されたゲームです。通常、iOS や Android など、スマートフォンのオペレーティングシステムを使用します。モバイルゲームは一般的に、デジタルアプリケーションストアからダウンロードし、携帯電話にインストールします。

上記のプラットフォームに加えて新進のプラットフォームもあります。これらは、比較的新しく、成長途上であり、優勢なプラットフォームに比べると市場シェアは非常に小さいプラットフォームです。このカテゴリのゲームプラットフォームの例としては、拡張現実 (AR)、バーチャルリアリティ (VR)、ゲームストリーミング (クラウドゲームとも呼ばれます) があります。ゲームストリーミング では、クラウド内でゲームプレイをレンダリングし、シンクライアント (通常はウェブブラウザ) にストリーミングします。プレイヤーは、ゲームストリーミングサービスプロバイダーが完全にリモートで (通常はクラウド内で) ホストしているゲームをプレイできます。ゲームストリーミングの場合、プレイヤーはクラウドベースのゲームに接続する方法として、ウェブブラウザを使用するか、クラウドゲームサービスプロバイダーが提供するシンクライアント (ゲームプラットフォーム) を使用します。

ゲームサーバー

ゲームサーバーは、ゲームのコンピューティングインフラストラクチャの最も重要な要素の 1 つです。ゲームサーバーは、専用ゲームサーバーとも呼ばれ、マルチプレイヤーゲームを開発する場合や、ゲームプレイのイベント処理にサーバー権限を要する場合に使用します。ゲームサーバーは、ゲームアーキテクチャの中心に位置し、コアロジックの実行場所となります。コアロジックには、プレイヤーおよびゲームの状態の管理や、接続したゲームクライアントとゲームサーバーとの間のやり取りの管理が含まれます。ゲームサーバーは、プレイヤーのゲームクライアントからの入力を処理し、その結果を他の接続先のプレイヤーにリアルタイムで適切に配信する役割を果たすため、通常、ゲームアーキテクチャにおいて最もパフォーマンスが要求される要素の 1 つです。ゲームサーバーのパフォーマンスが低いと、ゲーム体験全体のパフォーマンスに影響を与える可能性があります。したがって、特にゲームのリリース時やゲームプレイのピーク時には、ゲームサーバーのパフォーマンスを最適化し、十分な容量を確保することが重要です。

このドキュメントにおいて、ゲームサーバー (またはゲームサーバーインスタンス) とは、1 つ以上のゲームサーバープロセスをホストする、仮想マシン (VM) などのコンピューティングリソースを指します。ゲームサーバープロセスは、ゲームセッションをホストするゲームサーバービルドの 1 つのインスタンスを表します。ゲームセッションとは、プレイヤーがプレイヤーセッションを介して接続できる実行中のゲームのインスタンスです。したがって、このドキュメントでは、 ゲームサーバープロセス と ゲームセッション を同じ意味で使用します。これは、ゲームセッションとそれをホストしているゲームサーバープロセスとの間に暗黙の 1 対 1 の関係があるためです。AWS には、コンピューティングリソースでゲームサーバーをホストするためのオプションが複数あります。いずれ

のオプションでも、リソースの伸縮自在なプロビジョニングを通じて、クラウドベースのスケーラブルな容量が提供されます。

Amazon EC2 は、クラウドベースの仮想サーバー (インスタンスと呼ばれます) を提供し、Linux と Windows の複数のバージョンをサポートしています。インスタンスを作成し、他のサーバーや仮想マシン (VM) と同じようにインスタンスを直接管理できます。通常、複数のゲームサーバープロセスを 1 つのインスタンスにデプロイして、効率向上とコスト削減を図ります。コンピューティングインフラストラクチャを最大限に制御したい場合、Amazon EC2 がゲームサーバーとして最適です。

Amazon Amazon GameLift は、クラウド内で専用ゲームサーバーをホストするフルマネージドソリューションであり、Amazon GameLift FlexMatch を通じてマッチメイキングなどの追加機能も提供します。Amazon GameLift は、ゲームサーバーの管理を容易にするために Amazon EC2 上に抽象化レイヤーを提供します。ほとんどの AWS リージョン では、これを利用してプレイヤーの近くでゲームサーバーをホストし、レイテンシーの短縮、高可用性の実現、スポットインスタンスを使用したコストの大幅削減を行うことができます。Amazon GameLift は、既存のゲームバックエンドに統合できます。特に、ゲームデベロッパーが、独自のゲームサーバー管理やマッチメイキングソリューションを開発せずに、AWS のマネージドソリューションを使用してゲームの成長に合わせてスケールできるようにする場合に役立ちます。

Amazon Elastic Container Service (Amazon ECS) は、Docker ベースのコンテナを実行できるフルマネージドのコンテナオーケストレーションサービスです。 Amazon Elastic Kubernetes Service (Amazon EKS) を使用すると、Kubernetes で構築した Docker ベースのコンテナを実行できます。Amazon ECS や Amazon EKS が提供しているようなコンテナテクノロジーを使用すると、多数のゲームサーバープロセスや他のゲームアプリケーションインスタンスを EC2 インスタンス内に効率的にパッケージできるため、コンピューティングインフラストラクチャの使用率を高めることができます。また、コンテナを使用すると、デベロッパーの生産性も向上します。デベロッパーは、開発時にローカルマシンで使用するのと同じ Docker イメージ操作ランタイムを使用してアプリケーションをホストできます。運用のオーバーヘッドをさらに削減するには、 AWS Fargate を使用できます。これは、コンテナを実行するためのサーバーレスコンピューティングプラットフォームであり、Amazon EKS および Amazon ECS の両方と互換性があります。Fargate は、コンテナを実行する基盤のインスタンスを操作せずに、コンテナ内でゲームサーバーを実行するようなユースケースに最適です。

AWS Outposts では、あらゆるデータセンターまたはオンプレミス施設で AWS のサービスを実行できるため、同じサービスを使用してオンプレミス環境と AWS でゲームを実行し、ハイブリッドクラウド導入戦略をサポートできます。 AWS ローカルゾーン は AWS リージョンの拡張として、ゲームサーバーやレイテンシーの影響を受けやすいその他のワークロードをプレイヤーや開発チームの近くで実行できるようにします。さらに、ゲームサーバーのグローバルネットワークレイテンシーを短縮

するために、 AWS Global Accelerator を使用してゲームサーバーに対するプレイヤートラフィックのパフォーマンスを向上させることができます。

AWS Lambda は、サーバーのプロビジョニングや管理をしなくてもコードを実行できるサーバーレスコンピューティングサービスであり、ターン制ゲームなどの非同期ゲームサーバーのユースケースに役立ちます。Lambda は、コンピューティング要件が軽く、コードベースが小さく、ステートレスなマイクロサービスアーキテクチャを使用してゲームプレイ機能を設計できるゲームにも適しています。Lambda 関数は、長時間実行するゲームサーバープロセスの一部としてではなく、イベント駆動型のリクエストごとに実行されます。Lambda は、コードをホストする基盤のアプリケーションをデベロッパーが選択してすぐに利用できるため、このホワイトペーパーで説明しているオプションのうち、ランタイム抽象化のレベルが最も高いオプションです。

ゲームサーバーホスティングのアプローチを選択する場合は、運用のオーバーヘッド、レガシーコードベース、パフォーマンス要件、スケールなど、さまざまな要件を考慮する必要があります。Amazon EC2 インスタンスとコンテナは、クラウド移行に伴う変更が最小限であるため、レガシーコードベースに適しています。また、EC2 インスタンスを使用すると、コンピューティングインスタンスのすべてのリソースを専用として利用できます。コンテナを使用すると、管理が容易になり、使用率も高まる傾向にあります。サーバーレス関数は、最高レベルの抽象化を提供します。また、イベントにのみ応答して実行するコードを定義できるため、コストを削減できます。

ゲームクライアント

この ゲームクライアント は、プレイヤーがゲームプレイに使用するソフトウェア、ハードウェアデバイス、またはその両方を指します。ゲームクライアントは、プレイヤーの入力をメッセージに変換してサーバーに送信し、処理できるようにするソフトウェアを提供します。また、サーバーから受信した応答を処理し、プレイヤーへの出力 (グラフィックスなど) をレンダリングする役割も果たします。リアルタイムのネットワークマルチプレイヤーゲームの場合、ゲームクライアントは、通常、ゲームプレイセッション中にゲームサーバーへの継続的なネットワーク接続を維持し、ネットワークレイテンシーを短縮して処理時間を最小限に抑えます。ただし、ゲームクライアントは、ゲームサーバーやバックエンドサービスと REST (Representational State Transfer) を介してやり取りすることもあります。

メッセージング

ゲーム内のメッセージは、通常、以下の 3 種類に大別されます。

- 特定のユーザーやユーザー集団を対象としたプレイヤーエンゲージメントメッセージング (ゲームへの招待やプッシュ通知など)

- プレイヤー間のグループメッセージング (ゲーム内チャットなど)
- サービス間メッセージング (複数のアプリケーションを統合するために使用する JSON メッセージなど)

これらの種類のメッセージを送受信するための一般的な戦略は、パブリッシャー/サブスクライバー (pub/sub) や非同期処理のアーキテクチャパターンを使用することです。AWS は、ゲームにメッセージングを実装するのに役立つサービスをいくつか用意しています。

Amazon Simple Notification Service (Amazon SNS) は、pub/sub アーキテクチャパターンを使用してパブリッシャーとサブスクライバーとの間でメッセージを配信するマネージドサービスを提供します。パブリッシャーは、API を介して Amazon SNS にメッセージを送信します。Amazon SNS は、サブスクライブしているアプリケーションにメッセージを非同期で配信し、モバイルクライアントやデスクトップにプッシュ通知を直接配信できます。この場合、最も広く使用されているプッシュ通知サービスの一部を追加設定なしで使用できます。Amazon SNS は、クライアントへのプッシュ通知だけでなく、サービス間メッセージングのユースケースにも使用できます。

Amazon Simple Queue Service (Amazon SQS) は、フルマネージドのキューサービスです。ゲームサーバーやゲームで使用しているプログラミング言語を問わず、両方を簡単に統合できます。リーダーボードの更新やデータベース内のプレイタイム値の更新など、多くのゲームタスクを切り離してバックグラウンドで処理できます。このアプローチは、ゲームのさまざまな部分を切り離し、プレイヤー向けの機能をバックエンド処理から独立してスケールするのに非常に効果的です。

Amazon Managed Streaming for Apache Kafka (Amazon MSK) は、人気のオープンソースプラットフォームである Apache Kafka を使用して、データストリーミングアプリケーションやプロデューサー/コンシューマーアプリケーションを簡単に構築できるフルマネージドサービスです。Kafka は、通常、リアルタイムのストリーミングデータの取り込みと処理に使用します。サービス間メッセージングにも使用できます。

Amazon ElastiCache (Redis OSS) は、フルマネージドのインメモリデータストアを提供します。これに含まれている Redis の一般的な pub/sub 機能は、通常、チャットルームアプリケーションやハイパフォーマンスのサービス間メッセージングの開発に役立ちます。Redis は、リストやセットなどの豊富なデータ型もサポートしているため、デベロッパーは Redis を使用してハイパフォーマンスのキューを実現できます。

Amazon Pinpoint は、E メール、SMS、音声、プッシュ通知を介したユーザーエンゲージメントメッセージングを提供します。例えば、Amazon Pinpoint は、ユーザーエンゲージメントメッセージングを配信してプレイヤーをゲームに呼び戻すために使用したり、多要素認証トークンのサポート、注文

確認やパスワードリセットの E メールなどのトランザクションのユースケースに使用したりできます。

ライブゲームオペレーション (Live Ops)

ライブオペレーション (Live Ops) とは、ゲームの管理や運用のスタイルであり、ゲームをライブサービスとして扱い、発売したゲームに新機能、更新プログラム、プロモーション、ゲーム内イベント、機能強化などを継続的に提供してプレイヤーコミュニティのエクスペリエンスを改善します。

従来、ゲームはサービスとしてではなく、製品として提供されるのが一般的であり、新しいコンテンツや機能は、発売した製品内ではなく、その後のリリースや続編に組み込まれていました。Live Ops のゲーム管理アプローチにより、ゲームオペレーションチームは、発売したゲームに実験、プロモーション、ゲーム内イベント、新しいイノベーションを追加することで、プレイヤーを常に楽しませ、熱心なプレイヤーコミュニティを維持できます。このアプローチには、新しいプレイヤーエンゲージメント戦略の開拓と継続的な収益源の確保というメリットがあります。ただし、運用に関するより深い専門知識が必要になります。例えば、Live Ops 戦略を成功させるには、デベロッパーが必要に応じてクラウドサービスとの統合や独自のバックエンド技術インフラストラクチャの運用を行う場合があります。また、ゲーム内やプレイヤーコミュニティ内で発生した問題を効果的な方法で特定および対処し、プレイヤーエクスペリエンスに悪影響を及ぼさないようにする必要があります。

一般的な設計原則

AWS Well-Architected Framework は、クラウド内でのゲームワークロードの適切な設計を促進するために、以下の一般的な設計原則を定めています。

プレイヤーの行動と使用パターンを理解することで、ゲームを進化させ、プレイヤーを保護する: ゲームを継続的に改善し、プレイヤーエクスペリエンスを効果的に管理するには、プレイヤーがゲーム自体およびエコシステム内の他のプレイヤーとどのようにやり取りしているかを可視化することが重要です。そうすることで、ゲームを改善する方法、コストを管理する方法、プレイヤーエクスペリエンスにリスクをもたらす不正な使用行為をモニタリングして対処する方法を理解できます。

ゲームの操作を簡素化し、開発速度を高めるテクノロジーを使用する: 新機能や機能強化をプレイヤーに提供する期間を短縮し、運用のオーバーヘッドを削減するために役立つテクノロジーを優先的に採用します。ゲームはヒット志向で、プレイヤーには多くの選択肢があるため、ゲームを成功させるには、すばやく行動して変化に適応することが重要です。独自のソフトウェアを問題なく運用できるかどうか、あるいは、AWS、AWS パートナー、またはその両方から同等なマネージドサービスを導入するかどうかを検討します。

アーキテクチャを最適化し、実際のプレイヤーエクスペリエンスを反映するようにメトリクスを改善する: アーキテクチャを時間の経過に伴って適応および進化させながら、これらの改善と変更がプレイヤーエクスペリエンスにどのような影響を与えるかを考えます。ゲームワークロードでは、障害に耐えてその影響を最小限に抑え、ゲームプレイに広範囲にわたる混乱が及ぶのを防止する必要があります。ゲーム機能とシステムとの相互依存関係が重要ではない場合、両者を切り離して障害の影響範囲を限定し、プレイヤーに影響する問題を切り分けます。

ピーク時のプレイヤーの同時実行数に対応し、必要に応じて動的にスケールするようにインフラストラクチャを設計する: インフラストラクチャは、プレイヤーの需要に合わせてスケールするように設計する必要があります。プレイヤーセッションの同時実行数やログイン数などのメトリクスを使用すると、システムが過負荷になる前に未然にスケールできます。CPU やメモリの消費率など、システム使用率に関するリアクティブなメトリクスは、システムが過負荷になった後でスケールする場合に使用できます。インフラストラクチャを動的にスケールすることで、ゲームの運用コストを削減できます。

ランブックを実装してゲームの操作を改善する: オペレーションランブックは、ゲームの定期的な運用タスクを一貫して管理するために役立ちます。ランブックは、ゲームの一般的な運用ワークフローのために必要です。例えば、プレイヤーからの報告を調査して対応するために使用します。また、インフラストラクチャの規模の拡大や縮小を事前に管理して、新シーズンの開始やゲームコンテンツの

リリースなどの大規模なイベントに備えたり、ゲームの一般的なメンテナンス作業に対処したりするために使用します。

シナリオ

このセクションでは、ゲームアーキテクチャの一般的なシナリオをいくつか紹介します。各シナリオには、設計を推進する一般的な特性と、サンプルのリファレンスアーキテクチャ図を含めています。

シナリオ

- リアルタイム同期ゲームのためのゲームホスティング
- ゲームバックエンド
- クラウド内でのゲーム制作 (GPIC)
- ゲーム分析パイプライン

リアルタイム同期ゲームのためのゲームホスティング

リアルタイム同期ゲームプレイでは、複数のプレイヤーが同時にゲームに参加してやり取りできます。接続されたプレイヤー間でゲームプレイの状態を共有し、リアルタイムエクスペリエンスを創出します。同期ゲームの例としては、ファーストパーソンシューティングゲーム、MMOG (大規模多人数同時参加型オンラインゲーム)、スポーツやアクションゲームなど、プレイ体験をリアルタイムで共有するために複数のプレイヤーを接続する必要があるオンラインゲームがあります。

リアルタイム同期ゲームのプレイアーキテクチャには、以下のような特徴があります。

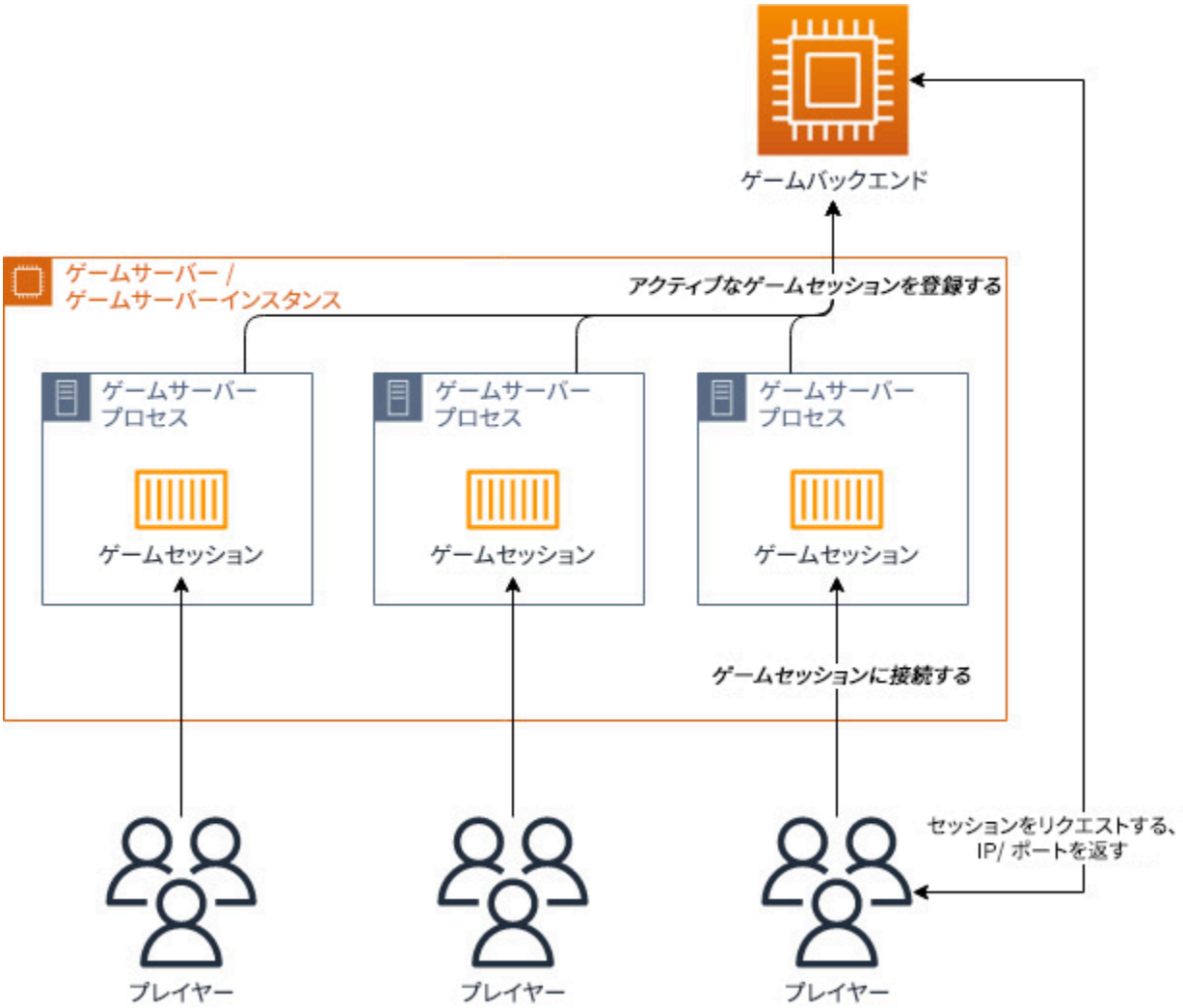
- ゲームは、ゲームサーバーで実行するゲームサーバープロセスを通じて、ゲームセッションとしてホストされます。ゲームサーバーは、複数のデータセンターと AWS リージョンにわたってグローバルにホストされます。
- ゲームクライアントがゲームセッションに参加するには、ゲームバックエンドプラットフォームでホストしている集中型マッチメイキングサービスにマッチをリクエストするか、利用可能なゲームサーバーの事前定義済みリストからマッチを選択します。ゲームクライアントには、接続先の IP アドレスとポートが返されます。
- ファーストパーソンシューティングゲームや大規模多人数同時参加型オンラインゲームなど、多くの同期ゲームにはレイテンシーが大きく影響します。これらのゲームでは、通常、高レイテンシーの状況で発生する可能性があるプレイヤーのラグを減らすために、レイテンシー許容値を事前に定義し、慎重に測定して最適化を図ります。このレイテンシー情報を判断するには、ゲームクライアントから利用可能なゲームサーバーの所在地に ping を送信し、レイテンシー、ネットワークジッター、その他ゲームプレイ体験にとって重要なメトリクスを取得するようにゲームクライアントを設定します。これらのメトリクスは、ゲームバックエンドプラットフォームの中央メトリクス収

集サービスに送られるため、ライブオペレーションチームはゲームのヘルスをモニタリングできます。ゲームクライアントは、マッチメイキングプロセスでマッチをリクエストするときに、現在のレイテンシーデータをリクエストパラメータの 1 つとして提供できます。マッチメイキングサービスは、このレイテンシーデータを変数の 1 つとして使用し、プレイヤーをホストするゲームサーバーを選択します。

- 通常、ゲームプレイは、複数のプロトコルを組み合わせて実行します。例えば、ゲームサーバーではより高速な UDP ベースのメッセージングを使用し、マッチメイキング、認証、その他のクライアントサーバーのトラフィックでは HTTP を使用します。
- ゲームサーバーは、悪意のあるアクティビティの標的となることが多く、AWS Shield Advanced などの DDoS 対策ソリューションで保護する必要があります。

ゲームサーバープロセス

次の図は、ゲームサーバーの一般的なアーキテクチャを示しています。この図は、ゲームサーバーインスタンスと、ゲームセッションをホストするゲームサーバープロセスとの間の論理的な関係を表しています。



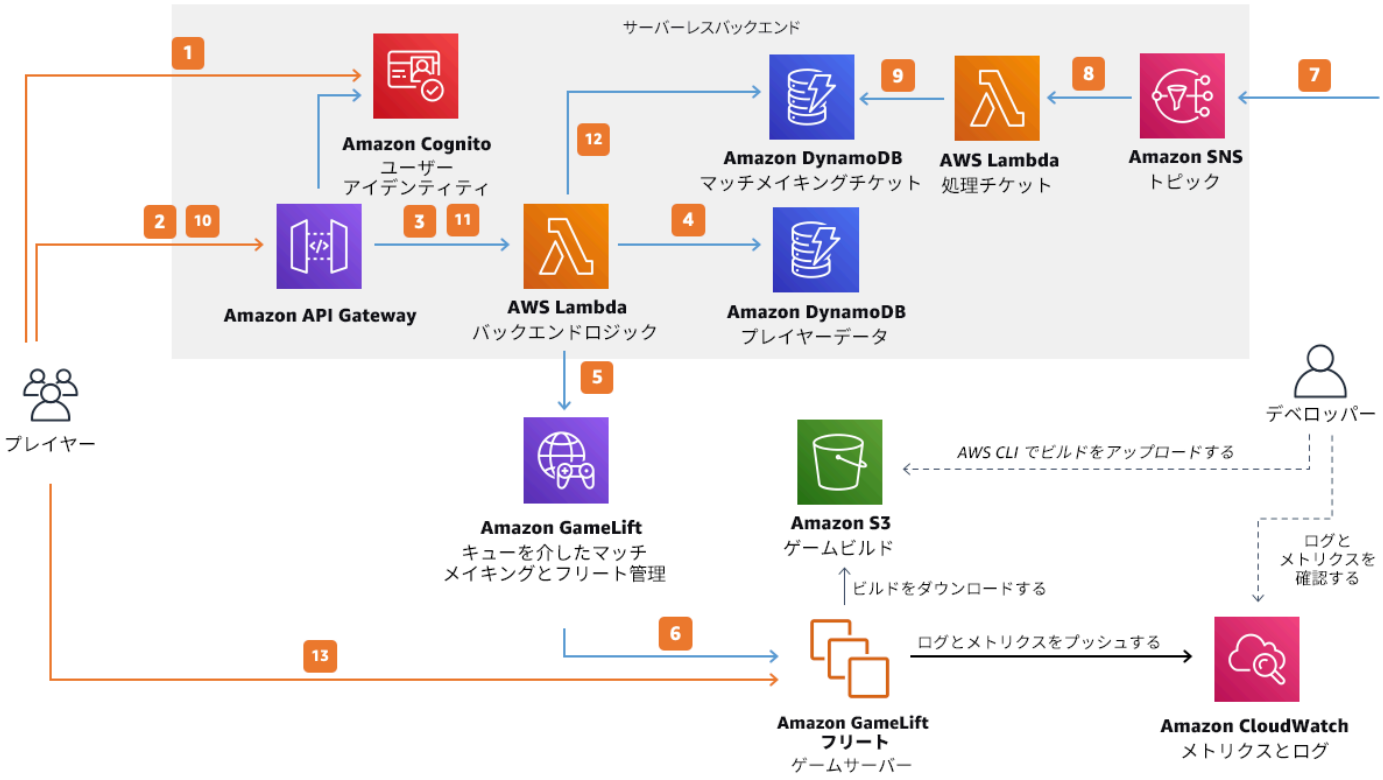
論理的なゲームサーバーアーキテクチャ

- ゲームサーバー (ゲームサーバーインスタンスとも呼ばれます) として Amazon EC2 インスタンスを使用します。ゲームサーバーは、1 つ以上のゲームサーバープロセスをホストし、各ゲームサーバープロセスがゲームサーバービルドのコピーを実行します。通常、複数のゲームサーバープロセスを 1 つのゲームサーバーインスタンスで実行することで、コンピューティングリソースを効率的に使用し、コストを削減します。ゲームセッションがアクティブになってプレイヤーセッションをホストする準備が整うと、ゲームセッションのステータスがゲームバックエンド (通常はマッチメイキングサービス) で更新され、ゲームセッションでプレイヤーをホストできるようになります。
- ゲームバックエンドは、プレイヤーのゲームクライアントに、ゲームセッションをホストしているサーバーの IP アドレスとポートを返し、プレイヤーが接続してプレイできるようにします。

サーバーレスバックエンドによるセッションベースのゲームサーバーホスティング

ゲームのアーキテクチャを開発する場合は、必要な機能や性能と、運用管理のオーバーヘッドの許容レベルを検討する必要があります。運用の容易さと柔軟性との最適なバランスを取るために、クラウドプロバイダーのマネージドサービスを使用してゲームを構築できます。これにより、独自のカスタムゲーム機能を開発したりカスタマイズを行ったりするための制御権を行使できるとともに、インフラストラクチャのデプロイと管理に伴う負担を軽減できます。

セッションベースのマルチプレイヤーゲームをホストするには、ゲームサーバープロセスをホストするためのサーバーインフラストラクチャと、マッチメイキングおよびセッション管理用のスケーラブルなバックエンドが必要です。次のリファレンスアーキテクチャは、Amazon GameLift マネージドホスティングとサーバーレスバックエンドを使用してセッションベースのゲームを管理する方法を示しています。



セッションベースのゲーム用の Amazon GameLift マネージドホスティング

この図は、Amazon GameLift マネージドゲームホスティングで実行しているゲームにプレイヤーを参加させるプロセスを示しています。このプロセスには以下のステップが含まれます。

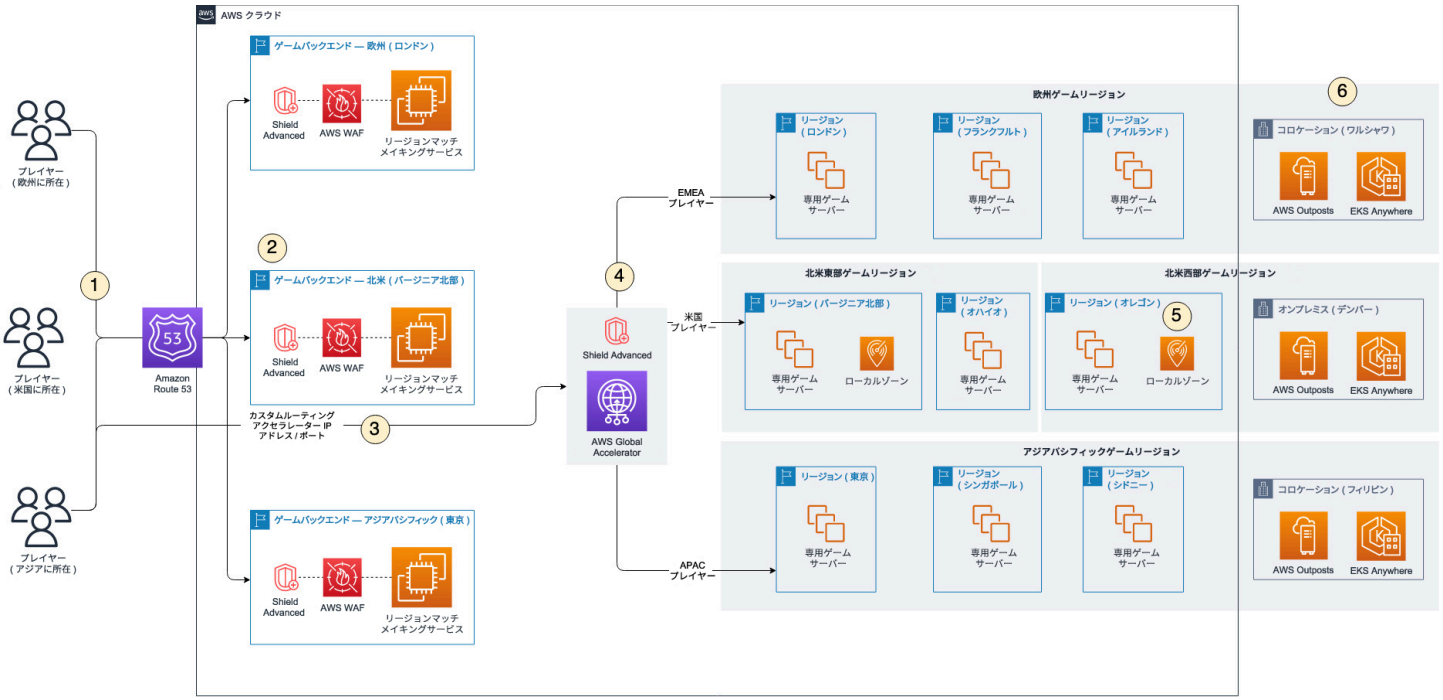
1. ゲームクライアントが、 Amazon Cognito アイデンティティを Amazon Cognito に対してリクエストします。オプションとして、外部のアイデンティティプロバイダーに接続できます。

2. ゲームクライアントは、一時的なアクセス認証情報を受け取り、 API Gateway でホストしている API を通じてゲームセッションをリクエストします。そのために Amazon Cognito の認証情報を使用します。
3. API Gateway が AWS Lambda 関数を呼び出します。
4. Lambda 関数が DynamoDB テーブルのプレイヤーデータをリクエストします。リクエストに対して認証済みのアイデンティティが返されるため、Amazon Cognito アイデンティティを使用して正しいプレイヤーデータを安全にリクエストできます。
5. Lambda 関数は、追加情報 (プレイヤーのスキルレベルなど) に関する正しいプレイヤーデータを使用して、 Amazon GameLift FlexMatch マッチメイキングを通じてマッチをリクエストします。FlexMatch マッチメイキングの設定は、JSON ベースの設定ドキュメントを使用して定義できます。ゲームクライアントは、さまざまなリージョンのサーバーエンドポイントに ping を送信することで、レイテンシーメトリクスを生成できます。また、そのレイテンシーデータを使用してレイテンシーベースのマッチメイキングをサポートできます。
6. FlexMatch は、プレイヤーのグループとリージョンへの適切なレイテンシーとをマッチさせた後で、Amazon GameLift キューを通じてゲームセッションの配置をリクエストします。キューには、1 つ以上の登録済みのリージョン所在地にあるフリートが含まれています。
7. セッションをフリートのいずれかの所在地に配置すると、イベント通知が Amazon SNS トピックに送信されます。
8. Lambda 関数が Amazon SNS イベントを受け取って処理します。
9. Amazon SNS メッセージが `MatchmakingSucceeded` イベントである場合、Lambda 関数は、結果をサーバーのポートおよび IP アドレスと共に DynamoDB に書き込みます。マッチメイキングチケットが不要になると、有効期限 (TTL) の値を使用してチケットが DynamoDB から確実に削除されます。
10. ゲームクライアントから API Gateway に対して署名付きのリクエストを行い、マッチメイキングチケットのステータスを一定の間隔で確認します。
11. API Gateway が、マッチメイキングチケットのステータスをチェックする Lambda 関数を呼び出します。
12. Lambda 関数が DynamoDB をチェックし、チケットが成功したかどうかを判断します。成功した場合、Lambda 関数は IP アドレス、ポート、プレイヤーセッション ID をクライアントに返します。チケットがまだ成功していない場合、Lambda 関数はマッチの準備が整っていないことを示す応答を送信します。
13. ゲームクライアントが、バックエンドから返されたポートと IP アドレスを使用してゲームサーバーに接続します。ゲームクライアントは、プレイヤーセッション ID をゲームサーバーに送信し、ゲームサーバーは Amazon GameLift Server SDK を使用してそれを検証します。

上記とは別に、Amazon GameLift で API Gateway WebSockets を使用するように、上記のアーキテクチャを変更することもできます。この方法では、ゲームクライアントとゲームバックエンドサービスとの間の通信が、 WebSockets ベースの実装を使用して行われます。この実装を使用すると、ゲームバックエンドの Lambda 関数はポーリングモデルを実装せずに、WebSocket 経由でゲームクライアントへのサーバー側メッセージを開始できます。

低レイテンシーゲーム向けのマルチリージョンおよびハイブリッドアーキテクチャ

このセクションでは、低レイテンシーゲーム向けのマルチリージョンおよびハイブリッドアーキテクチャについて説明します。



ネットワークアクセラレーションとゲームサーバーのグローバルなデプロイによるレイテンシーの短縮

1. グルーバルに利用可能なゲームの場合、プレイヤーはどこからでもプレイを開始できます。プレイヤーがゲームセッションまたはマッチをリクエストすると、ゲームクライアントが Amazon Route 53 に登録されているゲームバックエンドサービスにリクエストを送信します。
2. ゲームバックエンドは、プレイヤー集団に最も近い複数の AWS リージョンにデプロイします。各ゲームバックエンドでは、ゲームのすべてのリージョンからゲームセッションを見つけるリージョンマッチメイキングサービスを提供します。プレイヤーのマッチメイキングリクエストは、近くのリージョンマッチメイキングサービスによって処理されますが、マッチメイキングサービスは、必要に応じてプレイヤーを任意のゲームリージョンのゲームセッションにルーティング

できます。このアクションにより、回復力とパフォーマンスが向上します。さらに、各ゲームのバックエンドサービスは AWS WAF と Shield Advanced を使用して、レイヤー 7 のウェブフィルタリングとボットコントロール、および DDoS (分散型サービス拒否) に対する保護を提供します。ゲームのバックエンドサービスを構築する方法には、サーバーレス、コンテナ、EC2 インスタンス、独自のデータセンターでのゲームバックエンドサービスのホスティングなど、さまざまなオプションがあります。

3. ネットワークのレイテンシーやジッターを減らしてプレイヤーのエクスペリエンスを向上させるために、 AWS Global Accelerator を使用してカスタムルーティングアクセラレーターをデプロイします。これにより、ゲームクライアントからゲームサーバーへのトラフィックのルーティングが自動的に最適化されます。Global Accelerator のリスナーポートをゲームサーバーの EC2 インスタンスポートにマッピングするように、 カスタムルーティングアクセラレーター を設定します。ゲームクライアントは Global Accelerator の IP およびポートに接続することで、ゲームセッションをホストしている正しいゲームサーバーの IP およびポートにプレイヤーを決定的な方法でルーティングします。
4. ゲームには、プレイヤーが利用しやすい論理的なゲームリージョンが含まれ、これらのリージョンは地理的に相互に近いゲームサーバーのホスト場所の集まりを表します (北米や アジアパシフィックなど)。また、より詳細なリージョンとして、 北米東部 や 北米西部などを作成することもできます。レイテンシーを短縮し、地理的範囲を拡大するために、さまざまなゲームサーバーホスティングソリューションを組み合わせて使用することで、プレイヤーエクスペリエンスを向上させることができます。できる限り AWS リージョンを優先的に使用してください。これらの場所は、十分な機能を備えており、容量のフットプリントが最も大きいためです。
5. 十分なサービスを受けていないプレイヤーの地理的な場所では、 ローカルゾーン を使用します。既存のホスティング施設が存在しない場合や、AWS リージョンが利用できない場合に対応できます。
6. 既存のオンプレミスデータセンターやコロケーションプロバイダーに Outposts をデプロイし、フルマネージド型ラックとラックマウント型サーバーを使用して、各デプロイ場所にまたがるシームレスなコントロールプレーンと管理エクスペリエンスを構築します。Outposts は、オンプレミス環境に既存のサーバー容量がない場合にも役立ちます。ただし、独自の既存のサーバーインフラストラクチャでソフトウェアを実行するハイブリッド実装が必要な場合は、 EKS Anywhere を使用できます。これを使用すると、Amazon EKS への接続を備えた独自のインフラストラクチャでコンテナのクラスターを作成して実行できます。Amazon EKS は、AWS およびオンプレミスで Kubernetes クラスターの一貫したコンソールビューを提供します。

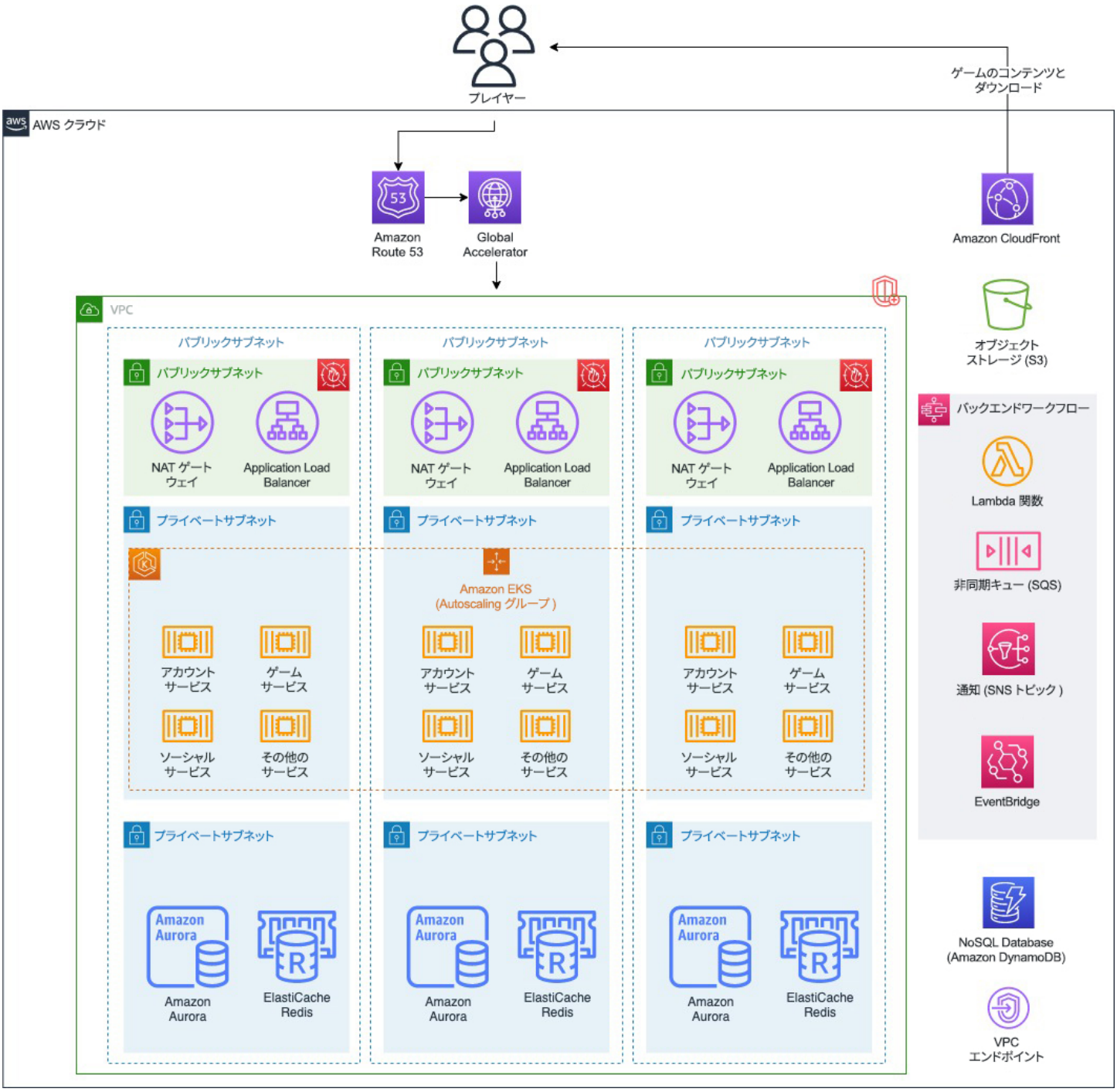
ゲームバックエンド

ゲームバックエンドは、ゲームやプレイヤーの状態を管理するために使用します。また、ソーシャルレベルやプラットフォームレベルの機能をゲームに統合して、ゲーム体験をサポートするためにも使用します。ゲームバックエンドでホストするサービスとしては、プレイヤープロファイルの管理、アイテムやインベントリの保存、統計情報、リーダーボードなどがあります。ゲームバックエンドは、通常、HTTPS を使用してクライアントからアクセスする REST API として構築します。ただし、他のアプローチも一般的に使用されています。例えば、WebSockets では、ゲーム内チャットやプレゼンスのクライアント通知などのユースケース向けに双方向チャネルを提供します。ゲームバックエンドは、インスタンス、コンテナ、サーバーレスアーキテクチャを使用するなど、さまざまなデプロイアーキテクチャを使用してデプロイできます。

トピック

コンテナベースのゲームバックエンドアーキテクチャ

このセクションでは、コンテナベースのゲームバックエンドアーキテクチャについて概説します。



コンテナを使用したゲームバックエンドのホスティング

1. プレイヤーは、ゲームクライアントソフトウェアを使用してゲームにアクセスします。ゲームクライアントソフトウェアは、ゲームプラットフォーム、デジタルストアフロント、または Amazon CloudFront などのコンテンツ配信ネットワーク (CDN) からの直接ダウンロードを通じて、プレイヤーに配信できます。CDN は、エッジロケーションでキャッシュを提供し、コンテンツをダウンロードするユーザーのパフォーマンスを向上させます。例えば、CloudFront を使用し

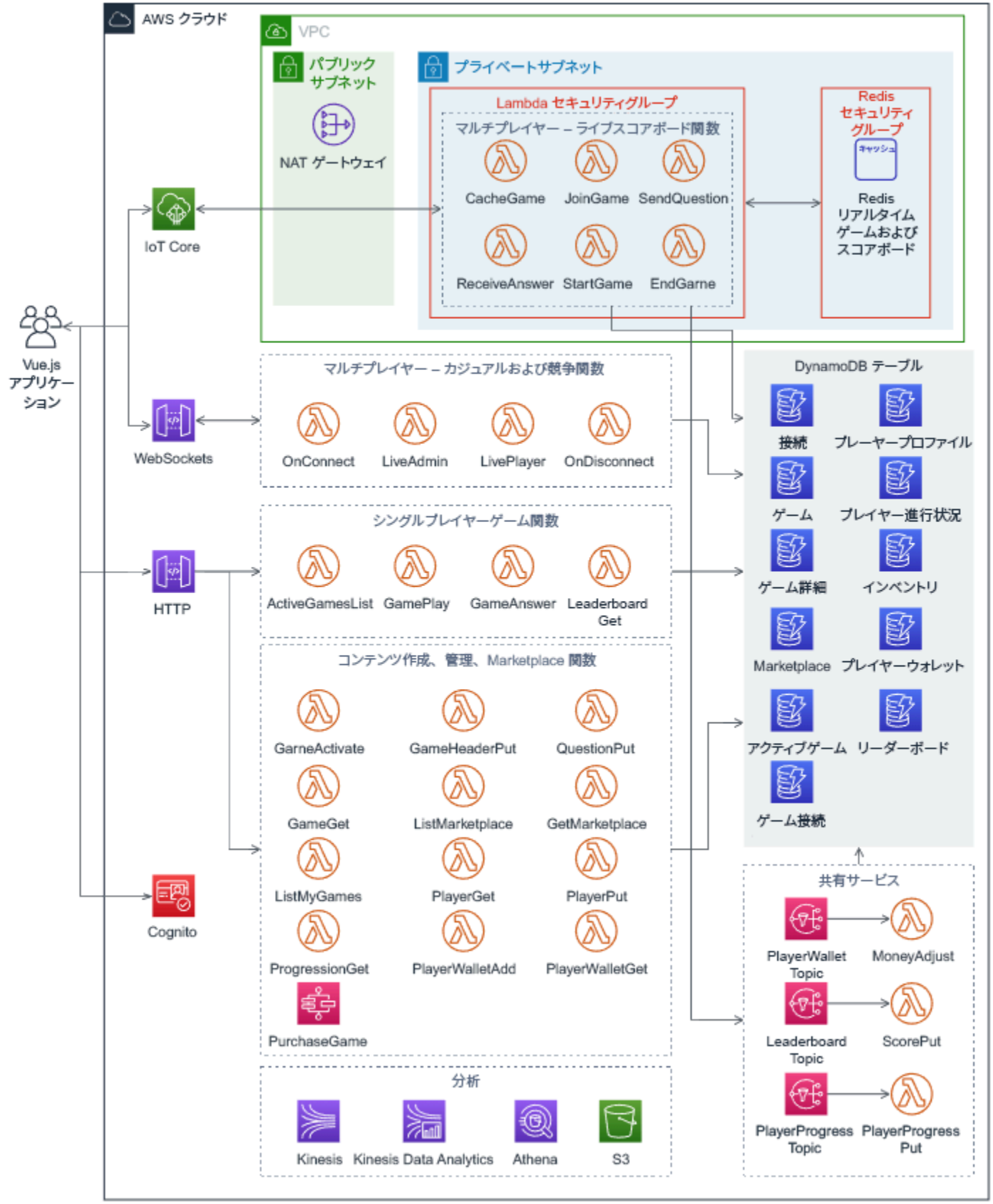
て、ゲームクライアントソフトウェアだけでなく、ゲームアセットやその他のコンテンツもプレイヤーに配信できます。

2. Global Accelerator は、トラフィックの高速化とカスタマイズ可能なコントロールを提供することで、プレイヤーゲームクライアントからロードバランサーへのトラフィックのルーティング、マルチリージョンやフェイルオーバーを目的としたリージョン間でのトラフィックのルーティングに役立ちます。ゲームバックエンド REST API のカスタムドメイン名は、トラフィックを Global Accelerator のエンドポイントにルーティングするように Route 53 で設定します。 Shield Advanced は、アクセラレーターとゲームバックエンドに DDoS 緩和機能を提供します。
3. NAT ゲートウェイと Application Load Balancer は、ゲームバックエンドが使用する各アベイラビリティーゾーンのパブリックサブネットにデプロイすることで、リージョンでの高可用性を確保します。 ウェブアプリケーションファイアウォール は、Application Load Balancer にデプロイすることで、レイヤー 7 のウェブトラフィックフィルタリングを提供します。
4. ゲームバックエンドは、回復力を高めるために複数のアベイラビリティーゾーンに分散したプライベートサブネットで、 Amazon EKS クラスター内にデプロイした個々のコンテナベースのマイクロサービスのコレクションとしてホストします。オートスケーリングにより、リソース使用率 (通常はプレイヤーの需要と関連する) に基づいて、サービスとクラスターノードの容量が動的に調整されます。この場合、 Cluster Autoscaler はクラスター内のノード数を自動的に調整するのに対し、 Horizontal Pod Autoscaler はクラスター内にデプロイしたポッドを自動的にスケールします。
5. ゲームとプレイヤーのデータは、複数のアベイラビリティーゾーンにまたがるプライベートサブネットにデプロイしたバックエンドデータベースとキャッシュに保存され、プライマリノードとレプリカノードとの間でレプリケートされます。Aurora は、プレイヤープロファイル、エンタイトルメント、ゲーム内購入などのユースケースによく使用されます。これらのユースケースでは、クエリ要件が複雑になる可能性があり、MySQL や PostgreSQL のリレーショナルデータモデリング機能が役立つ場合があります。ElastiCache for Redis は、ハイパフォーマンスのリーダーボードの構築や pub/sub メッセージングに役立ちます。また、アクセス頻度の高いデータをキャッシュしてレイテンシーを短縮したり、データベースへの負荷を軽減したりするためにも役立ちます。DynamoDB は、予測不能なアクセスパターンに最適なフルマネージドの NoSQL データストアです。プレイヤーやゲームの状態データ、セッションデータ、インベントリ、アイテムストアなどのユースケースや、オーバーヘッドを最小限に抑えたグローバルデータベースを必要とするユースケースにおいて、実質的に無制限のスループットまでスケールできます。
6. バックグラウンドで処理できる作業 (リーダーボードの更新やフレンド登録の依頼など) を実行するには、非同期処理のワークフローを使用する必要があります。この種の作業を Amazon SQS キューにプッシュするようにゲームバックエンドを設定して、ゲームの成長に合わせてスケールできるようにするか、あるいは Amazon SNS トピックを使用して作業を多数のコンシューマー

アプリケーションキューに分散して並列処理を行うことを検討します。また、 Lambda 関数 を使用してイベント駆動方式で処理を実行すると、コンピューティングインフラストラクチャのコストと管理オーバーヘッドを低減できます。存続期間が長いワークフローや、複数のステップを使用してタスクを調整する必要があるワークフローの場合は、 Step Functions を使用してワークフロー全体をオーケストレートすることを検討します。Amazon EventBridge を使用すると、AWS のサービスやカスタムアプリケーションイベントに応答する関数を開始できます。

サーバーレスベースのゲームバックエンドアーキテクチャ

多くのゲームデベロッパーは、インフラストラクチャの管理を敬遠しがちであり、ソフトウェアに専念できるようなテクノロジーを使用してゲームを構築することを望んでいます。このシナリオには、サーバーレスアーキテクチャをお勧めします。サーバーレスアーキテクチャを使用すると、機能を構築およびリリースする時間を短縮し、運用のオーバーヘッドも削減できます。サーバーレスアーキテクチャを設計するには、需要に応じて動的にスケールできるクラウドサービスを使用します。サーバーのセットアップ、管理、スケーリングは不要です。次のリファレンスアーキテクチャは、サーバーレスアーキテクチャを使用してゲームを構築する方法を示しています。


AWS クラウド
VPC
パブリック サブネット
NAT ゲートウェイ
プライベートサブネット
Lambda セキュリティグループ
マルチプレイヤー – ライブスコアボード関数
CacheGame
JoinGame
SendQuestion
ReceiveAnswer
StartGame
EndGarne
Redis セキュリティ グループ
Redis リアルタイム ゲームおよび スコアボード
IoT Core
Vue.js アプリケーション
WebSockets
マルチプレイヤー – カジュアルおよび競争関数
OnConnect
LiveAdmin
LivePlayer
OnDisconnect
HTTP
シングルプレイヤーゲーム関数
ActiveGamesList
GamePlay
GameAnswer
Leaderboard Get
コンテンツ作成、管理、Marketplace 関数
GarneActivate
GameHeaderPut
QuestionPut
GameGet
ListMarketplace
GetMarketplace
ListMyGames
PlayerGet
PlayerPut
ProgressionGet
PlayerWalletAdd
PlayerWalletGet
PurchaseGame
Cognito
分析
Kinesis
Kinesis Data Analytics
Athena
S3
DynamoDB テーブル
接続
プレーヤープロファイル
ゲーム
プレイヤー進行状況
ゲーム詳細
インベントリ
Marketplace
プレイヤーウォレット
アクティブゲーム
リーダーボード
ゲーム接続
共有サービス
PlayerWallet Topic
MoneyAdjust
Leaderboard Topic
ScorePut
PlayerProgress Topic
PlayerProgress Put

サーバーレスベースのゲームバックエンドのリファレンスアーキテクチャ

このリファレンスアーキテクチャは、シングルプレイヤーとマルチプレイヤーの機能を提供するウェブベースのトリビアゲームを示しています。

- プレイヤー認証: プレイヤーは Amazon Cognito を使用して認証します。Amazon Cognito は、プレイヤーのアイデンティティ管理用のユーザーディレクトリを備えた安全な認証を提供します。
- サーバーレス関数としてのゲームロジック: ゲームの機能とバックエンドビジネスロジックのすべては、イベントに応答して動作する Lambda 関数として実行します。関数の実行時にのみ料金が発生するため、コストを抑えることができます。Lambda では、任意のプログラミング言語を使用して、ゲームの各機能を独立したマイクロサービスとして柔軟に記述できます。例えば、C# を使用して Unity ゲームを構築した経験があれば、.NET Lambda 関数を開発できます。ウェブベースのゲームのフロントエンドとバックエンドの両方を JavaScript でプログラミングする場合は、Node.js Lambda 関数を開発できます。
- ゲームおよびプレイヤーのデータ用の NoSQL データストア: DynamoDB は、マイクロサービスの大量のデータを格納する目的で構築されているため、プレイヤーとゲームのデータを保存するために使用できます。このアーキテクチャに示しているように、各ゲーム機能のデータストレージニーズに個別のデータストアを使用することをお勧めします。これにより、機能を独立してモニタリングおよび管理しやすくなります。また、チーム内で機能やサービスの所有権が変わった場合に、分離の境界を作成するのにも役立ちます。このリファレンスアーキテクチャでは、DynamoDB テーブルを使用して、接続状態、ゲームの詳細、プレイヤーの進行状況、リーダーボード情報などのデータを保存しています。
- シングルプレイヤーゲームプレイ: シングルプレイヤー機能では、プレイヤーがゲームを選択してプレイしたり、リーダーボードを表示したりするなどのアクションを実行できます。これらの機能は、RESTful バックエンドサービスとして実装し、Amazon API Gateway HTTP API を使用してホストします。この API は、適切な Lambda 関数を呼び出して DynamoDB テーブルのデータを取得および設定します。ゲームプレイが完了すると、バックエンドは SNS トピックにも通知を送信し、Lambda 関数を非同期に開始してプレイヤーの進行状況や統計情報を保存します。
- マルチプレイヤーゲームプレイ: マルチプレイヤーゲーム機能では、プレイヤーがポイントツーポイント通信でゲームとやり取りできること、および接続している他のプレイヤーに更新をブロードキャストしたり、逆に更新を受信したりできることが必要です。WebSockets の実装は、トリビアなどの軽量なゲームでのポイントツーポイント通信に適しています。プレイヤーは、Amazon API Gateway WebSockets への WebSockets 接続を確立できます。接続先の WebSockets では、接続を管理し、プレイヤーに対して送受信するメッセージがある場合にのみ、Lambda 関数を呼び出します。プレイヤー間で 1 対多の通信が必要なユースケースの場合、AWS IoT Core は MQTT 経由の WebSockets を使用したメッセージングをサポートし、クライアントがトピックをサブスク

ライブして、受信したメッセージに対してアクションを実行できるようにします。このアーキテクチャでは、WebSockets over MQTT を使用して、ゲーム内ライブ更新のブロードキャストや、接続しているすべてのプレイヤーへの質問などのユースケースをサポートしています。AWS IoT Core の代わりに、メッセージ配信に Redis Pub/Sub を選択したり、メッセージを保持する必要がある場合に Redis Streams を選択したりできます。

- VPC 対応の Lambda 関数を使用してプライベートサブネット内のリソースにアクセスする: VPC 対応の Lambda 関数を設定して、VPC のプライベートサブネット内のリソースにアクセスできるようにします。例えば、ElastiCache for Redis にアクセスできるようにして、低レイテンシーのデータセット (ライブリーダーボードなど) のクエリ時間を短縮します。

詳細については、 シンプルなトリビアサービスのコードサンプルを参照してください。

クラウド内でのゲーム制作 (GPIC)

クラウド内でのゲーム制作 (GPIC) とは、ゲーム開発ライフサイクルに必要なインフラストラクチャとツールを使用してゲームを構築、テスト、開発することを指します。ゲーム開発は、ユーザー間での共同作業であり、インフラストラクチャの要件は開発の各段階を通じて頻繁に変わります。多くのゲームデベロッパーは、世界中に分散したリモート開発チームを採用しているため、この種の開発をサポートするテクノロジーが必要です。ゲームデベロッパーは、これらの環境のすべてまたは一部を AWS でホストし、AWS リージョンのグローバルな可用性を利用してリソースをユーザーの近くに配置できます。また、必要に応じてコンピューティングとストレージをスケールすることで、開発環境をよりコスト効率よく管理できます。

環境はゲームデベロッパーのニーズによって異なりますが、通常、アーティスト、デザイナー、エンジニア、QA テスター、請負業者、その他の担当者が作業を行うためのデベロッパーワークステーションが含まれます。これらの環境には、通常、ビルドファームも含まれます。ビルドファームは、ユーザーが変更をチェックインするためのソースコードリポジトリと、開発したアーティファクトをビルド、パーケージ化、テストするための CI/CD インフラストラクチャで構成されます。

これらのゲーム制作アーキテクチャには、以下の特徴があります。

- ユーザーは、ウェブブラウザまたはローカルデスクトップクライアントを介して仮想ワークステーションにアクセスできる必要があります。例えば、 NICE DCV にアクセスできる場合、低レイテンシーのストリーミングセッションを通じて、社内や開発スタジオのマシンで作業する場合と同じソフトウェアやツールを利用できます。これらの仮想ワークステーション (通常はクラウドベースのサーバー) により、LAN または WAN を介して、ユーザーはプロジェクトの共同作業をすべてクラウド環境で実行できます。ユーザーがマシンを頻繁には使用しない場合は、耐久性のあるク

ラウドストレージに作業をバックアップします。例えば、 Amazon Elastic File System (EFS) や Amazon FSx などのソース管理リポジトリやファイルシステムにバックアップし、マシンをシャットダウンしてコストを削減します。

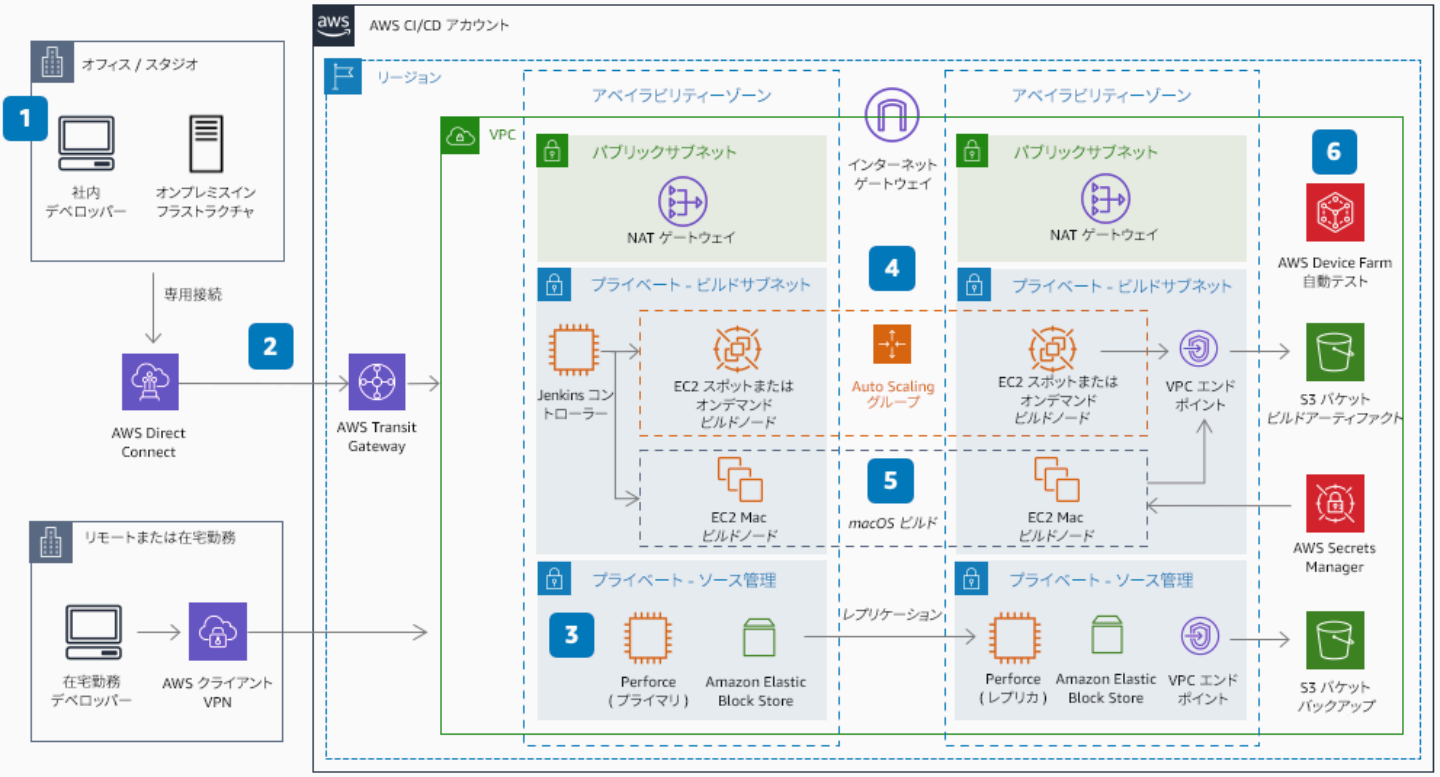
- ソース管理リポジトリ (Perforce など) は、アベイラビリティーゾーン間またはオンプレミス間でのレプリケーションを通じて高可用性を備えた設計にし、バックアップを Amazon S3 などのクラウドストレージに保存する必要があります。例えば、クラウドベースの Perforce サーバーは、1 つのアベイラビリティーゾーンでプライマリコミットサーバーをホストし、同じリージョンの別のアベイラビリティーゾーンでスタンバイサーバーへのレプリケーションをホストする必要があります。
- ゲーム開発ビルドファームリソースは、オートスケーリングを使用してコンピューティングリソースが必要に応じてプロビジョニングされるように設計する必要があります。また、 EC2 スポットインスタンス を使用して、ビルドに必要なサーバー数のスケールアウトに伴って発生するコストを削減する必要があります。

トピック

- クラウド内でのゲーム制作 – CI/CD
- クラウド内でのゲーム制作 – ワークステーション

クラウド内でのゲーム制作 – CI/CD

ゲームの開発には CI/CD インフラストラクチャが必要です。ゲーム開発の CI/CD パイプラインは、通常、高可用性のソース管理サーバーおよびストレージ、ビルドを実行するコンピューティングリソース、自動テストを実行するソフトウェア、開発マシンからの適切なネットワーク接続で構成されます。次のリファレンスアーキテクチャは、ゲームビルドをリモートまたはオンプレミスのゲーム開発環境から AWS クラウドにオフロードし、デベロッパーを支援してビルドファームを移行したり、新しいビルドファームを構築したりする方法を示しています。



クラウドへのゲームビルドのオフロード

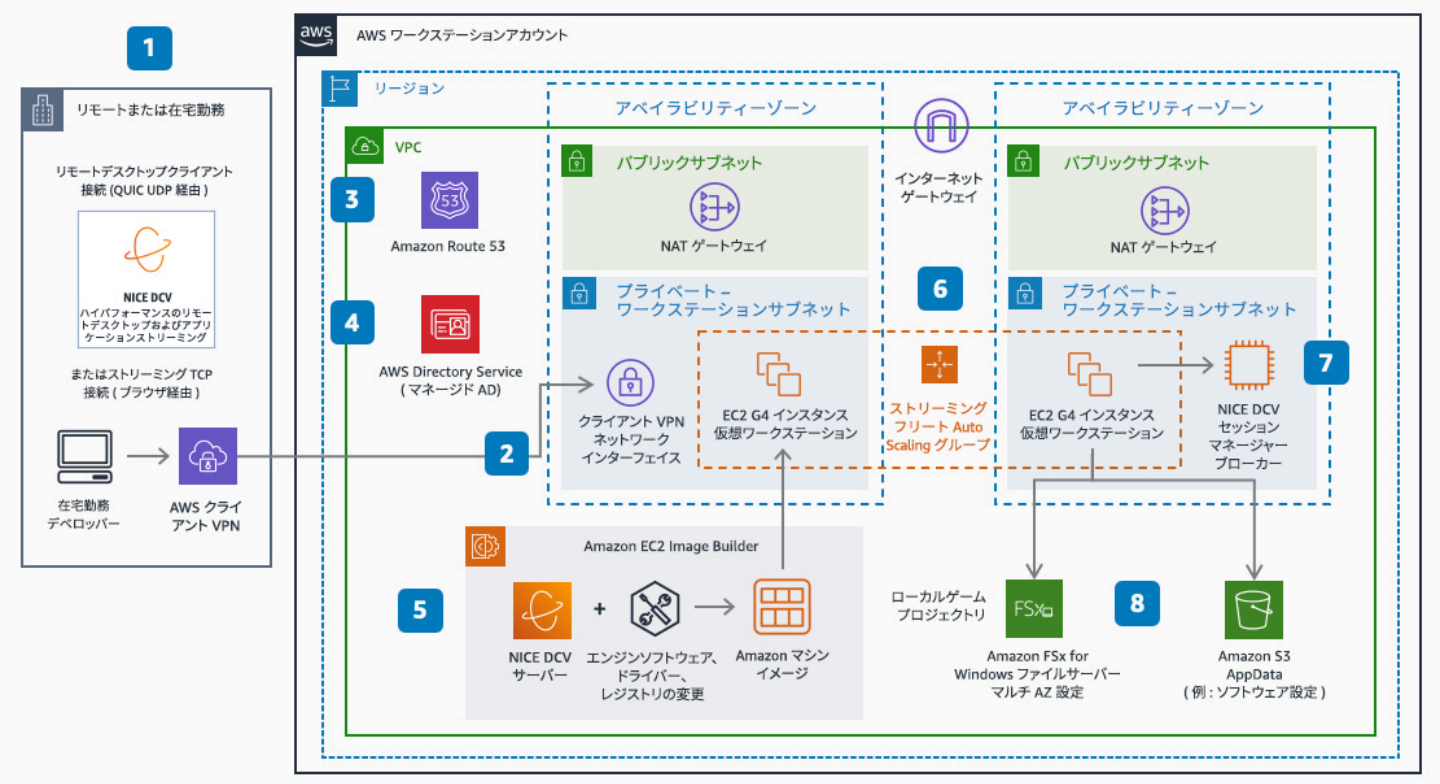
1. AWS Direct Connect は、社内のデベロッパーに対して AWS への低レイテンシーのプライベート専用接続を提供します。リモートデベロッパーは、 AWS クライアント VPN を使用します。
2. AWS Transit Gateway は、VPC 間の接続およびオンプレミスからの接続に対するネットワーク管理を簡素化します。
3. Perforce は、 Amazon EBS ストレージによってバッキングされたソースおよびバージョンコントロール (CI) を管理し、迅速にアクセスできる永続的なデータを提供します。Perforce Helix Core (P4D) は、 AWS Marketplace から入手できます。
4. デベロッパーがブランチに関連付けた Perforce に変更をプッシュすると、コミットによって Jenkins でビルド (CD) が開始されます。Perforce は、Jenkins に対する JSON ペイロードの POST をトリガーします。Jenkins コントローラーは、エンジンの「headless」 CLI コマンドを呼び出し、エフェメラル、Docker ノード (Amazon EC2 スポットインスタンスなど)、または Amazon EC2 オンデマンドインスタンスとの間でビルドプロセスを実行し、並列化します。デベロッパーは、ロードバランサーの背後にある 2 つの Jenkins コントローラー (各アベイラビリティーゾーンに 1 つずつ) を使用することで、可用性を高めることができます。一部のエンジンでは、デベロッパーは追加のライセンスインフラストラクチャを追加のサブネットに設定し、同時ビルドを実行するたびにビルドコンテキストのライセンスを販売することが必要になる場合があります。

5. iOS ビルド の Xcode 部分を Amazon EC2 Mac インスタンス にオフロードして、 `.IPA` ファイルの署名、構築、エクスポートを行い、プロセスを分割してビルド時間を短縮します。AWS Secrets Manager は、プロビジョニングプロファイル、プライベートキー、証明書を保持します。
6. ビルドアーティファクトを Amazon S3 に配信します。Amazon S3 は成功または失敗の通知を送信します。AWS Device Farm によって、自動テストを実行できます。

クラウド内でのゲーム制作 – ワークステーション

ゲーム開発プロセスは、さまざまな場所からゲームのさまざまな面に取り組んでいるチームに高度に分散される場合があり、生産性を高めるには、チームがワークステーションおよび共有ストレージにアクセスできることが重要です。単純なシナリオでは、特定のオフィス所在地にある 1 つのゲーム開発スタジオでゲームを開発できます。ただし、実際には、地理的に異なるさまざまな場所からデベロッパーが 1 つのゲームプロジェクトに協力するのがより一般的です。例えば、単一のゲームスタジオが、さまざまな都市や国にグローバルに分散した複数のチームを抱えている場合があります。または、雇われて仕事をするスタジオ、請負業者、共同開発スタジオパートナーが、アート開発やゲームテスト、QA、ローカライゼーションなどの特定の専門知識を持ち込むことで 1 つのプロジェクトをサポートする場合があります。また、ゲーム開発では、リモートワークをサポートして、デベロッパーが自宅や他の場所で生産性を発揮できるようにする必要があります。ゲーム業界では、新興のスタジオがオフィス環境での作業を完全に排除してリモートワークだけにする傾向が顕著になっています。

次のリファレンスアーキテクチャは、AWS で NICE DCV プロトコルを使用して、リモートゲーム開発ワークステーションをホストする方法を示しています。



NICE DCV を使用したあらゆる場所からのゲーム開発のストリーミング

1. NICE DCV は、4K、60-FPS ストリーミングをサポートするストリーミングプロトコルです。ブラウザを使用するデベロッパーは TCP 経由で接続しますが、デスクトップクライアントはポート 8443 経由で QUIC UDP を使用してパフォーマンスを高めることができます。
2. デベロッパーは、 AWS クライアント VPN を使用し、ソースネットワークアドレス変換 (SNAT) を通じてワークステーションサブネット内のネットワークインターフェイスに安全に接続できます。
3. Amazon Route 53 は、インバウンドおよびアウトバウンド DNS 転送だけでなく、VPC 内のリソースにプライベート DNS を提供します。
4. AWS Directory Service は、マネージド型の Microsoft Active Directory を提供し、ローカルゲームプロジェクトのストレージを個々のユーザーにマッピングできるようにします。
5. ワークステーションを作成するには、 Amazon マシンイメージ (AMI) を Image Builder で構築して使用します。イメージには、NICE DCV Server、デベロッパーソフトウェア、レジストリの変更、ドライバー (GPU ゲームドライバーや周辺機器ドライバーなど) が含まれます。AWS Marketplace には、ワークステーションで使用する一般的な AMI が用意されています。
6. ワークステーションのフリートでは、GPU を提供するグラフィックス Amazon EC2 インスタンスタイプを使用し、EC2 Auto Scaling グループを使用してスケールします。
7. セッションマネージャーブローカーを使用すると、NICE DCV セッションを管理できます。

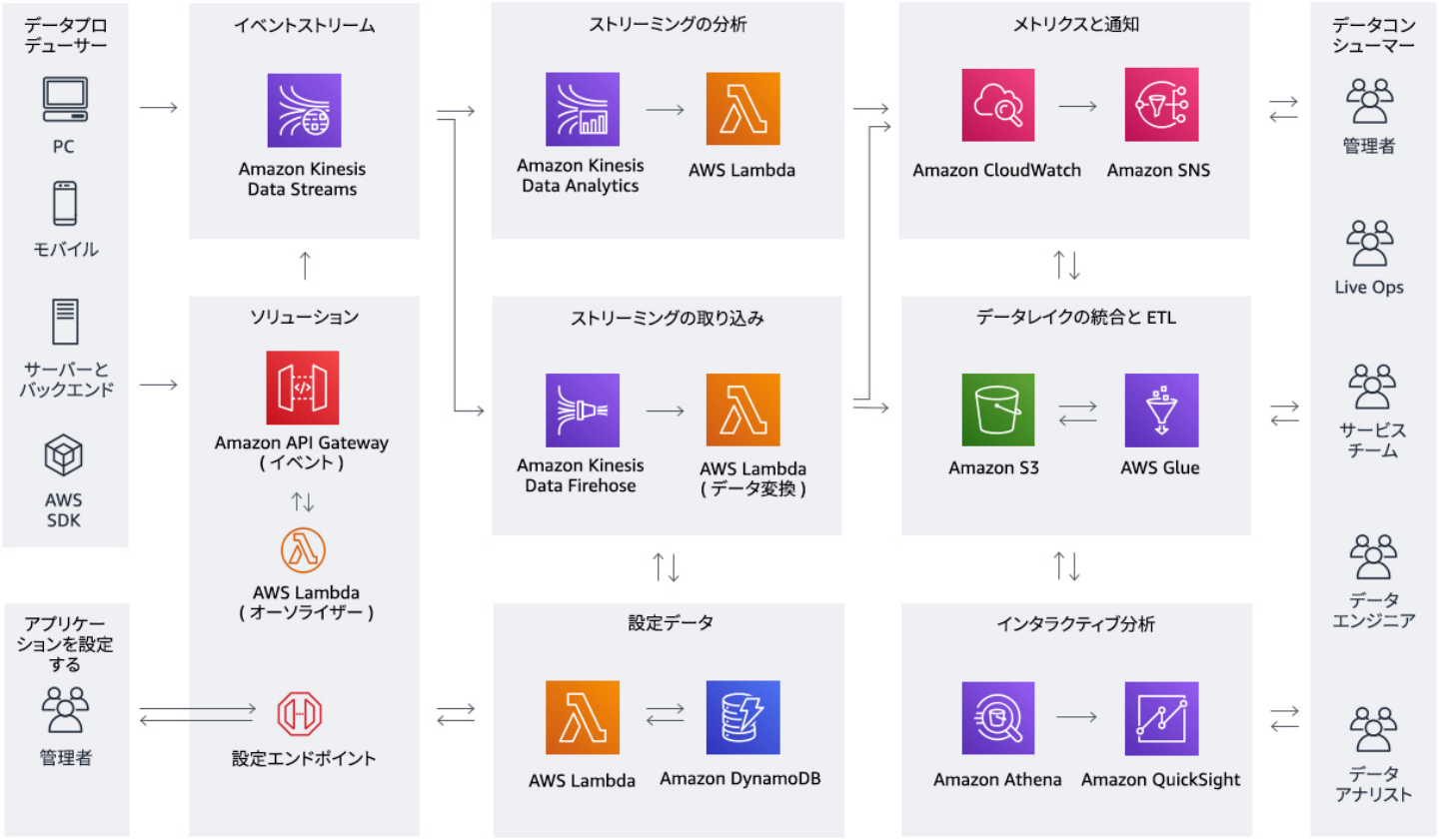
8. プロジェクトのローカルファイルストレージは、 Amazon FSx for Windows ファイルサーバーでホストします。デベロッパーは、ローカルストレージからソース管理にプッシュすることで、別の CI/CD パイプラインにコミットします。

ゲーム分析パイプライン

ゲームデベロッパーは、ゲームプレイ体験を向上させてプレイヤーベースを保持および成長させるために、プレイヤーの行動をよりよく理解する方法を熱心に模索しています。ゲーム分析とは、ゲームおよび関連サービスから生成されるすべてのデータを理解および分析するために必要な技術的なインフラストラクチャとプロセスを指します。これには、通常、このエンドツーエンドのプロセスをサポートできる分析パイプラインアーキテクチャを使用する必要があります。例えば、 ゲーム分析パイプライン ソリューションの実装が考えられます。

ゲーム分析アーキテクチャには、以下の特徴があります。

- データソースは、JSON などの一般的な形式でデータを送信します。データソースには、通常、ゲームサーバーやゲームバックエンドサービスに加えて、PC、モバイルデバイス、ゲームコンソールなどのゲームクライアントも含まれます。
- ゲーム分析パイプラインは、raw データを取り込んで保存し、それを使用可能な出力形式として処理するワークフロー全体を自動化することで、エンドユーザーや分析アプリケーションなどのデータコンシューマーが効率的で費用対効果の高い方法で分析できるようにします。
- ゲーム分析パイプラインは、ゲームの成長に合わせてスケールするために、大量のリアルタイムデータの取り込みと処理をサポートします。
- リアルタイムレポートとバッチレポートの両方のユースケースをサポートします。例えば、リアルタイムのダッシュボードとアラートは、通常、Live Ops チームがゲームインフラストラクチャとプレイヤーの行動をモニタリングして問題を検出するために使用します。データアナリストは、通常、アドホックレポートとバッチレポートを使用して経時的な傾向を把握します。



ゲームプレイテレメトリ用のサーバーレスゲーム分析パイプライン

ゲームデータは、ゲームクライアント、ゲームサーバー、その他のアプリケーションから取り込まれます。ストリーミングデータは Amazon S3 に取り込まれ、データレイクの統合とインタラクティブな分析に使用されます。ストリーミング分析は、リアルタイムのイベントを処理し、メトリクスを生成します。データコンシューマーは、Amazon CloudWatch のメトリクスデータと Amazon S3 の raw イベントを分析します。

1. ソリューション API と設定データ: Amazon API Gateway を使用して REST API を提供することで、ゲーム分析パイプラインを管理したり、Lambda 関数を通じて Amazon DynamoDB に設定データを保存したりします。この API や管理用のカスタムコマンドラインインターフェイス上に内部ポータルを構築できます。REST API はサーバー認証も提供します。この認証を使用して、データソースからゲームプレイデータを取り込み、テレメトリデータを Amazon Kinesis Data Streams に転送してリアルタイム処理とストレージへの取り込みを行います。
2. イベントストリーム: Kinesis Data Streams は、ゲームからストリーミングデータをキャプチャし、データを処理して保存します。
3. ストリーミング分析: Kinesis Data Analytics は、Kinesis Data Streams からのストリーミングイベントデータを分析し、Lambda 関数を使用して CloudWatch に発行するカスタムメトリクスとアラートを生成できます。

4. メトリクスと通知: Amazon CloudWatch を使用して、ソリューションのメトリクス、ログ、アラームをモニタリングします。Amazon SNS を使用して、オンコールのエンジニアや他のデータコンシューマーに通知を送信します。
5. ストリーミングの取り込み: Kinesis Data Firehose を使用すると、Kinesis Data Streams からのストリーミングデータを簡単に受け取って、そのデータを Amazon S3 のデータレイクに配信し、長期保存、変換、他のデータとの統合を行うことができます。
6. データレイクの統合と ETL: ETL 処理ワークフローに Glue を使用し、Glue データカタログでメタデータを整理します。このデータカタログは、柔軟な分析ツールと統合するためのデータレイクの基盤となります。
7. インタラクティブ分析: エンドユーザーは、Amazon S3 に保存したデータセットに対して、Amazon Athena を使用して アドホックインタラクティブクエリを実行できます。また、QuickSight を使用してダッシュボードを構築できます。

CloudFormation を使用してアカウントにデプロイできる分析パイプラインの自動リファレンス実装については、 ゲーム分析パイプライン を参照してください。

Well-Architected の柱

このセクションでは、Well-Architected Framework の 5 本の柱という面から、ゲーム業界レンズについて説明します。柱ごとに、設計原則、定義、ベストプラクティス、評価質問、検討事項、主要サービス、役立つリンクを紹介します。

柱

- 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

運用上の優秀性

運用上の優秀性の柱では、クラウドベースのゲームをあらゆる規模でデプロイおよび運用するためのベストプラクティスに注目します。優れたプレイヤーエクスペリエンスを維持し、エクスペリエンスに影響を与える問題に備えて予防策を講じて回復を図るためには、運用上の優秀性に重点を置くことが重要です。

設計原則

運用上の優秀性を達成するには、Well-Architected Framework のホワイトペーパーに記載している設計原則に加えて、以下の設計原則が役立ちます。

ゲームオペレーションチームのために測定可能かつ達成可能な目標を定義し、必要に応じて適応させる: ゲームはヒット志向であるため、ゲームのリリース時に何人のプレイヤーが参加するか、進行中のゲームオペレーションに対してプレイヤーがどのような期待を持つかを事前に判断することは困難です。野心的でありながらも達成可能な目標をステークホルダーと設定し、ゲームが予想を上回った場合はスケールアップし、ゲーム開発チームがプレイヤーエクスペリエンスを最適化するまではスケールダウンできるアプローチを設計することが重要です。これらの要件を満たすために十分な準備とテストを事前に実施し、ビジネスおよび技術上のステークホルダーが目標に即してゲームを運用していることを確認します。目標を定義しておくと、ゲームチームは、コストとパフォーマンスの適切なバランスを取りながら、ゲームバックエンドインフラストラクチャの計画、設計、プロビジョニング、デプロイ、運用を行うことができます。

オペレーションランブックを使用してゲームのリリースや特別なイベントを事前に計画およびスケールする: ゲームオペレーションチームは、ビジネスステークホルダーと連携して、イベントのピーク時のプレイヤー同時実行数の推定をモデル化し、インフラストラクチャの容量を事前にスケールするための先を見越した計画を策定します。イベント中のプレイヤートラフィックは変動しやすいため、事前の計画と事前の規模の拡大や縮小によって既存の自動スケーリングシステムを強化して、イベント中の成功率を高め、ポジティブなプレイヤーエクスペリエンスを提供するための十分なリソースがあることを確認します。オペレーションランブックを作成して、プロセスの一貫性を確保します。

プレイヤーのサポートリクエストを受け取り、調査し、対応するための運用モデルを確立する: リリース後は、ゲームに関する苦情や問題の報告をモニタリングすることが重要です。コミュニティフォーラム、ソーシャルメディア、E メール、チケットシステム、コールセンター、自動化されたチャットボットソリューションなど、プレイヤーの問題を適切に解決するために、プレイヤーと安全かつ効果的な方法でやり取りするための適切なシステムを実装します。

定義

クラウドベースのゲーム用に Well-Architected Framework の運用上の優秀性の柱を拡張します。このレンズを通して、クラウドベースのゲームの運用上の優秀性に、以下の要因がどのように貢献するかを確認します。

クラウド内でのゲームワークロードに関する運用上の優秀性には、以下の 3 つのベストプラクティス分野があります。

- 準備
- 運用
- 進化

ベストプラクティス

以下のトピックは、クラウドアーキテクチャのベストプラクティスに関するものです。

トピック

準備

GAMEOPS01 - ゲームのライブオペレーション (LiveOps) 戦略をどのようにして定義しますか?

GAMEOPS_BP01: ゲームの目標とビジネスのパフォーマンスメトリクスを使用して、ライブオペレーション戦略を策定します。

ゲームの目的とパフォーマンスメトリクスを決定するには、ゲームプロデューサーやパブリッシングパートナーなどのビジネスステークホルダーに相談する必要があります。これにより、メンテナンスウィンドウ、ソフトウェアとインフラストラクチャの更新スケジュール、システムの信頼性と回復性の目標の定義など、ゲームの管理方法に関する計画を立てることができます。

例えば、プレイヤーの同時実行数 (CCU)、日次および月次のアクティブユーザーのターゲット数 (DAU/MAU)、インフラストラクチャ予算、財務目標のターゲットを定義したり、その他のパフォーマンスメトリクスとして、新しいコンテンツや機能のリリース頻度、ゲーム内イベントやプレイヤーエンゲージメントを高めるためのプロモーションの頻度などのターゲットを定義したりできます。これらの目標とメトリクスに基づいて、ゲーム設計、リリース管理、可観測性、効率的な運営に必要なサポートに関する意思決定を下すことができます。

ゲームに、少なくとも毎月 1 回は新しいコンテンツ更新をリリースし、リリース中にダウンタイムは発生させないという目標があるとします。この情報に基づいて、リリースのデプロイ戦略を定義し、ダウンタイムを必ず伴うメンテナンスを該当する月の他の時間帯にスケジュール調整して、可用性の SLA を達成できます。

これらのメトリクスは、ゲームのヘルスのモニタリング、直接的なゲームフィードバックの収集、合理化および自動化したリリースプロセスの構築を行うために、ゲームのライフサイクルのどの段階にライブオペレーションチーム (Live Ops) を関与させるかを判断するためにも役立ちます。例えば、新しいゲームの場合は、アクティブなプレイヤー数、収益、または別の一連のメトリクスに基づいて、ゲームが特定の規模に達したことを確認した後で、専任のライブオペレーションチームを編成する場合があります。体制が確立されているゲーム開発スタジオでは、以前のゲームでライブオペレーションを経験済みであるため、新しいゲームをオンボードするだけで済む場合があります。

GAMEOPS_BP02: 既存のゲームソフトウェアをゲームで再利用する前に検証およびテストします。

組織は、開発時間とコストを節約するために既存の (以前のゲームの) コンポーネントやソースコードを再利用する傾向があります。これらのレガシーコンポーネントやコードは、綿密なレビューや詳細な統合テストを受けていない可能性があり、代わりに過去のパフォーマンスに依存する場合があり

ます。再利用は生産性の向上に役立ちますが、以前のパフォーマンスの問題や安定性の問題が新しいプロジェクトに引き継がれるリスクもあります。したがって、以前のゲームの既存のコンポーネントやソースコードを再利用する場合は、厳密なテストを実施する必要があります。

例えば、ゲーム A 向けに設計、作成、テストしたソースコードとコンポーネントをゲーム B で再利用すると、ゲーム B に要求される条件のすべては処理できない可能性があります。本番稼働環境でインシデントが発生した場合、デベロッパーにはそのコードやコンポーネントをデバッグして修正する十分な知識や作成し直す時間がないことがあり、オペレーションの問題を軽減できない場合があります。コードの原作者がいない場合、適切な修正を実装するのに時間がかかる可能性があります。以前に使用したコードやコンポーネントに問題があった場合は、再利用する前に優先事項として置換や修正を行います。オペレーションで再び問題が発生してから事後処理することは避けます。

GAMEOPS02 - ゲーム環境をホストするためのアカウントはどのように構成しますか?

GAMEOPS_BP03: マルチアカウント戦略を採用し、ゲームやアプリケーションごとに独自のアカウントに分離します。

AWS でデプロイするゲームアーキテクチャでは、複数のアカウントを論理的に整理して適切に分離できるようにします。これにより、問題の影響範囲を限定し、ゲームインフラストラクチャがスケールした際のオペレーションを簡素化できます。ゲームインフラストラクチャをホストする AWS アカウントは、通常、以下の論理環境にグループ化されます。

- ゲーム開発環境 (Dev): デベロッパーがゲームのソフトウェアやシステムを開発するために使用します。
- テストまたは QA 環境: 統合テスト、手動の品質保証 (QA)、他の必須の自動テストを実行するために使用します。
- ステージングまたは本番稼働前環境: 最終ビルドのソフトウェアをホストするために使用し、本番稼働環境に移行する前に負荷テストとスモークテストを実施できるようにします。
- ライブまたは本番稼働環境: ライブソフトウェアおよびインフラストラクチャをホストし、プレイヤーからの本番トラフィックを処理するために使用します。
- 共有サービスまたはツール環境: さまざまなチームが使用する共通のプラットフォーム、ソフトウェア、ツールにアクセスできます。例えば、中央のセルフホスト型ソース管理リポジトリやゲームビルドファームは、共有サービスアカウントでホストしている場合があります。
- セキュリティ環境: クラウドセキュリティを重視するチームが使用する、一元化されたログとセキュリティのテクノロジーを統合するために使用します。

AWS のゲームインフラストラクチャでは、ゲーム環境ごと (開発、テスト、ステージング、本番稼働) に個別のアカウントを作成するとともに、セキュリティ、ログ記録、中央の共有サービス用のアカウントを作成することをお勧めします。

通常、限られた数のインフラストラクチャリソース (通常は数百台以下のサーバー) を管理する小規模なゲーム開発スタジオでは、1 つの本番稼働用アカウント、1 つの開発アカウント、1 つのステージングアカウントなど、環境ごとに 1 つの AWS アカウントを作成できます。ただし、ゲームのインフラストラクチャやチームの規模が時間の経過とともに大きくなると、この単純なモデルは適切にスケールできなくなる可能性があります。これらの環境を設定する場合は、特定のリージョン内のアカウント全体において、AWS の多くのサービスがリソースや API レベルの Service Quotas を共有することを考慮することが重要です。この点を考慮して、アカウントを論理的に整理する方法を決定する必要があります。AWS アカウントでは、アカウント内にデプロイしたサービスを使用した分だけに料金が発生します。したがって、これにより、リソースの競合と Service Quotas を効果的に削減できます。特にゲームが成長して、より多くのデベロッパーがリソースを構築および管理するためのアクセスを要求するようになった場合に効果的です。

AWS は、大規模なゲーム開発スタジオ (一般的に、数千のサーバーを運用し、数百人のデベロッパーがリソースにアクセスする) と協働した経験に基づいて、よりきめ細かいアカウント構造を設計し、ゲームをサポートするアプリケーションごとに独自の開発、テスト、ステージング、および本番稼働用アカウントを設定することをお勧めします。大規模で成功したゲームをサポートしてきた経験から、ゲームをリリースした後で AWS マルチアカウント戦略を再設計することは困難で時間がかかることがわかっています。ライブシステムの計画と移行は複雑であるためです。将来のスケーリングニーズを考慮した上で、適切なマルチアカウント構造を決定してください。

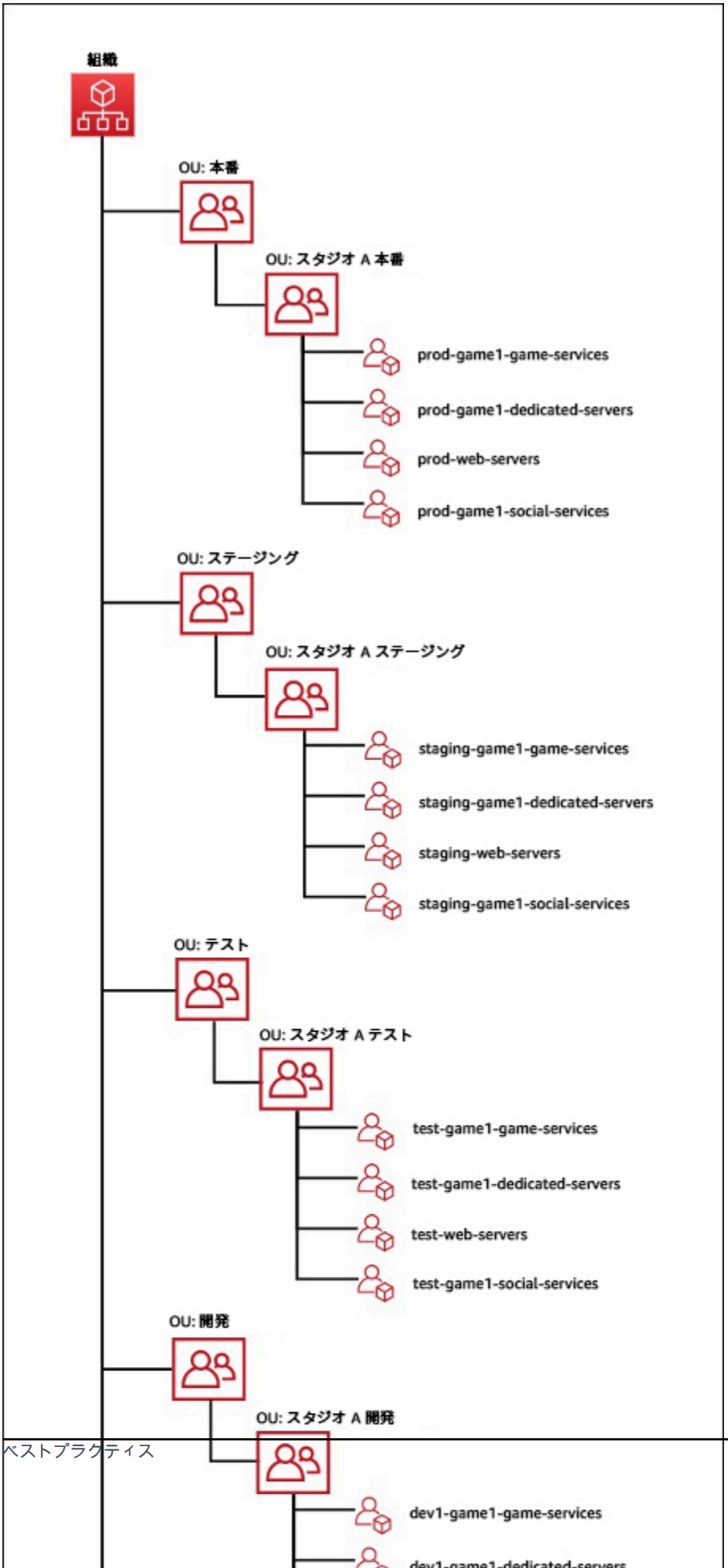
AWS Organizations を使用すると、AWS アカウントの階層とグループを設定できます。また、組織単位 (OU) を定義して共通の OU レベルのポリシーを サービスコントロールポリシー (SCP) を通じて適用できます。AWS Organizations は、リソースを増やしてスケールする際に、環境を一元的に管理および統制する場合に役立ちます。新しいアカウントをプログラムで作成してリソースを割り当てる、アカウントをグループ化してワークフローを整理する、アカウントやグループにポリシーを適用してガバナンスを確保する、すべてのアカウントの支払い方法を一本化して請求を簡素化するなどの処理を行うことができます。さらに、Organizations は他のサービスと統合しているため、中央での設定、セキュリティメカニズム、監査要件、組織内のアカウント間でのリソース共有を定義できます。

AWS Control Tower は、ランディングゾーンと呼ばれる安全なマルチアカウント環境をセットアップおよび管理するための最も簡単な方法を提供します。Control Tower は、AWS Organizations を使用してランディングゾーンを作成し、継続的なアカウント管理とガバナンスに加えて、クラウドに移行する数千のお客様と連携してきた AWS の経験に基づいて実装のベストプラクティスを提供しま

す。 AWS Config、AWS Trusted Advisor、および Security Hub は、アカウントの衛生状態の集約ビューまたは一元化されたビューを提供するサービスです。

このような分離により、ゲーム環境ごとにカスタムまたは個別の許可とガードレールを設定できます。本番稼働用アカウントにはすべての必要なガードレール、アクセス制限、モニタリングとアラート、セキュリティツールが備わっている必要があり、本番稼働用アカウント以外のアカウントには同じレベルのガードレールや許可は必要ない場合があります。本番稼働以外の環境は、営業時間後にリソースをシャットダウンしてコストを削減するように自動化できます。このレベルのきめ細かさでアカウントを分離すると、ゲームをサポートする環境ごとにインフラストラクチャコストをモニタリングしやすくなります。

次の例は、AWS Organizations と組織単位 (OU) を使用して、AWS アカウントを個別の環境とスタジオに論理的にグループ化するゲーム会社のマルチアカウント構造を示しています。この例では、OU を使用して環境ごとにアカウントをグループ化し、次に環境を運用するスタジオごとにアカウントをグループ化しています。これは、ネスト階層を作成して、アプリケーションやゲームごとに独自のアカウントにデプロイできるようにする方法を示しています。この方法は、複数のゲームを開発して運用する場合に便利です。マルチアカウント戦略を整理するために検討できる追加の戦略については、この柱の「リソース」セクションに記載しているドキュメントとホワイトペーパーを参照してください。


組織
OU: 本番
OU: スタジオ A 本番
prod-game1-game-services
prod-game1-dedicated-servers
prod-web-servers
prod-game1-social-services
OU: ステージング
OU: スタジオ A ステージング
staging-game1-game-services
staging-game1-dedicated-servers
staging-web-servers
staging-game1-social-services
OU: テスト
OU: スタジオ A テスト
test-game1-game-services
test-game1-dedicated-servers
test-web-servers
test-game1-social-services
OU: 開発
OU: スタジオ A 開発
dev1-game1-game-services

ゲーム環境のアカウント構造の例

GAMEOPS_BP04: リソースのタグ付けを使用してインフラストラクチャリソースを整理します。

効果的に インフラストラクチャリソース を AWS で管理および追跡するには、 リソースの適切なタグ付け と グループ化 を利用して各リソースの所有者、プロジェクト、アプリケーション、コストセンター、その他のデータを特定します。タグ付けしたリソースは、リソースグループを使用してグループ化すると、オペレーションのサポートに役立ちます。

ベストプラクティスとして、 タグ付けポリシーを定義する必要があります。一般的な戦略には、チーム名や個人名、ゲーム/アプリケーション/プロジェクトの名前、スタジオ名、環境 (開発、テスト、ステージング、本番稼働、共有など)、リソースのロール (データベースサーバー、ウェブサーバー、専用ゲームサーバー、アプリケーションサーバー、キャッシュサーバーなど) など、リソース所有者を特定するためのリソースタグがあります。任意のタグを追加して、ビジネスや IT のニーズに対応できます。AWS Config は、リソースの作成時と更新時に タグ付けポリシー を適用する場合にも役立ちます。タグとリソースグループは、AWS Management Console、AWS CLI、API オペレーションから使用できます。

GAMEOPS03 - ゲームのデプロイはどのようにして管理しますか?

GAMEOPS_BP05: プレイヤーへの影響を最小限に抑えるデプロイ戦略を採用します。

プレイヤーがゲームをプレイできないダウンタイムを最小限に抑えるような、ゲームのソフトウェアとインフラストラクチャのデプロイ戦略を組み込む必要があります。更新の種類によってはゲームクライアントに新しい更新をインストールする必要がありますが、ゲームバックエンドは、デプロイ中のダウンタイムを極力または完全になくすように設計する必要があります。

ゲームのデプロイ戦略を策定する際に考慮すべき最も重要なステップの 1 つは、ゲームインフラストラクチャの管理方法を決定することです。ゲームのインフラストラクチャを管理するベストプラクティスは、Infrastructure as Code (IaC) ツールを使用することです。例えば、 AWS CloudFormation または Terraform by Hashicorp を使用して、環境の準備中の人的エラーを減らします。インフラストラクチャテンプレートは、自動パイプラインでデプロイおよびテストできるため、さまざまなゲーム環境を一貫した方法で設定できます。

ゲームでは、以下のいくつかのデプロイ戦略を使用できます。

ローリング置換: デプロイのローリング置換の主な目的は、ゲームをシャットダウンしたり、プレイヤーに影響を与えたりすることなく、リソースを実行することです。実行するアップグレードや変更

には下位互換性があること、およびシステムの以前のバージョンに隣接して動作することが重要です。

このデプロイでは、その名前が示すように、サーバーインスタンスが、更新バージョンを実行するインスタンスに段階的に入れ替わり (置換またはロールアウト) ます。このローリング置換は、いくつかの方法で実行できます。例えば、専用ゲームサーバーのフリートにローリング更新を実装するには、一般的な方法として、新しいゲームサーバービルドバージョンをデプロイしている EC2 インスタンスの新しい Auto Scaling グループを作成し、この新しいサーバーフリートでホストしているゲームセッションにプレイヤーを徐々にルーティングします。新しいゲームサーバービルドを使用する前提条件として、関連するゲームクライアント更新が必要である場合は、検証チェックを含めます。それにより、この新しいゲームクライアント更新をインストールしたプレイヤーのみを、これらのゲームセッションにルーティングできます。

サーバーフリート (EC2 Auto Scaling グループなど) に古いゲームサーバービルドバージョンが含まれている場合は、すべてのアクティブなプレイヤーセッションを適切な方法で終了した後でのみ、これらのフリートの稼働を停止できます。通常は、そのためにサーバーごとのメトリクスを設定し、ゲームオペレーションチームがこのプロセスを自動化できるようにします。また、ローリングデプロイを実行するためのインフラストラクチャと時間を削減するには、1 つの方法として、既存の本番インスタンスの稼働を停止し、新しいゲームサーバービルドで更新してから、本番フリートに戻します。この方法により、必要なインフラストラクチャの量は減りますが、サーバーを入れ替える際にプレイヤーが利用できるライブゲームサーバーの数が少なくなるため、リスクも増大します。

このモデルは、ゲームプレイをホストしないデータベース、キャッシュ、アプリケーションサーバーなどのバックエンドサービスに対してローリングデプロイを実行する際にも使用できます。クラスター化した複数のインスタンスを使用して可用性の高い方法でこれらのサービスをデプロイしている限り、これらのサービスへのデプロイは専用ゲームサーバーへのデプロイよりも複雑さが少ないはずです。

ブルー/グリーンデプロイ: ゲームでのブルー/グリーンデプロイの主な目的は、ダウンタイムを最小限に抑えながら、問題を特定した場合に、以前のデプロイに安全にロールバックできるようにすることです。このデプロイは、ゲームバックエンドの 2 つのバージョン間に互換性があり、同時にプレイヤーにサービスを提供できるようにする場合に適しています。ブルー/グリーンデプロイ戦略では、2 つの同じ環境 (ブルーとグリーン) を設定し、既存のゲームバージョンにはブルーというラベルを付け、デプロイ先である新しいゲームバージョンにはグリーンというラベルを付けます。グリーン環境で移行準備が整ったら、トラフィック先をグリーン環境に切り替えるようにルーティングレイヤーを設定できます。古い環境 (ブルー) は、フェイルバックが必要な場合に備えて、一定期間、使用可能な状態に保持します。このシナリオでは、ルーティングの更新に伴い、必要に応じてマッチメイキングサービスを更新してゲームセッションを新しいフリートに送信するように設定する

か、ゲームバックエンドサービスの場合は、サービスの Route 53 の DNS レコードを更新するか、Application Load Balancer の重み付けをシフト して新しいターゲットグループにトラフィックを送信します。

ブルー/グリーンデプロイ戦略の欠点の 1 つは、デプロイの実行時にインフラストラクチャを追加する必要があるため、スタンバイ環境に必然的にコストが伴うことです。この追加のインフラストラクチャコストを軽減するには、ブルー/グリーンデプロイのバリエーションとして、本番稼働環境にデプロイ済みの同じサーバーに新しいゲームソフトウェアをデプロイすることを検討します。このシナリオでは、既存のブルーサーバープロセスと並行して新しいグリーンサーバープロセスで新しいソフトウェアを開始できます。カットオーバーは、完全に分離した物理インフラストラクチャ間ではなく、サーバープロセス間で発生します。また、このアプローチでは、クラウドで新しいサーバーが起動するまで待つ必要がないため、大量のインフラストラクチャ全体にわたってゲームのデプロイを高速化できます。このデプロイアプローチのベストプラクティスの詳細については、 AWS でのブルー/グリーンデプロイ のホワイトペーパーを参照してください。

canary デプロイメント: canary デプロイメントは、ゲームの初期のアルファ/ベータビルドや、新しいゲームモード/マップ/チャレンジなどのゲーム機能を、本稼働環境内の特定または少数のプレイヤーにリリースするために適用できるという点で、ゲームデベロッパーに特に役立つ戦略です。このようなデプロイを canary と呼びます。リリースに追跡やレポートの機能を追加すると、実際のプレイヤーが該当するゲームや機能をプレイしたときに、ゲームプレイのテレメトリを収集し、異常や問題がないか分析できます。新機能の場合は、これを該当するプレイヤーに常に通知するわけではなく、プレイヤーに問題が発生しているかどうか、リリースをロールバックする必要があるかどうかを判断するには、ゲームテレメトリが主な情報源となります。同時に、重大な問題が特定されなければ、この機能をさらに多くのプレイヤーにロールアウトして追加のデータを収集できます。プレイヤーに通知する場合は、プレイヤーエクスペリエンスに関する定期的なフィードバックを提供するよう依頼できます。このようなテストアクティビティは、ライブオペレーションチームが調整するのが最適です。

戦略として、canary デプロイメントを標準リリースにも使用して、新しい機能を徐々にプレイヤーに利用してもらうこともできます。標準のブルー/グリーン環境に優る潜在的な利点は、フルスケールの第 2 環境を設定する必要がないことです。新しいスケールダウンした環境の容量によって、新機能にオンボードするプレイヤーの数が決まります。より多くのプレイヤーを追加する前に、容量を適切にスケールする必要があります。このカスタマイズしたブルー/グリーン手法が標準のブルー/グリーンよりもコスト安であるとしても、canary デプロイメントのローリング置換手法よりは多くのコストが発生すると推定されます。

本番稼働環境で canary を 1 つだけ実行し、その目的をデータとフィードバックに絞ることをお勧めします。複数の canary をデプロイすると、本番稼働環境でのトラブルシューティングと問題の切り分けが複雑になり、収集するデータセットとフィードバックの品質が損なわれます。

canary のバリエーションとして、1 つ以上の実験 (通常は UI テスト) をターゲットデプロイを介して実行します。その際、1 式のゲームバックエンドサーバーで 1 つのバージョンの機能を提供し、同様のサイズの別の 1 式で同じ機能の別のバージョンを提供します。この場合、追加のインフラストラクチャや特殊なインフラストラクチャのスピンアップはなく、選択したバックエンドサーバーだけがこれらの更新を受け取ります。実験の結果として、同じ機能の各バージョンに対するプレイヤーの反応と、全体的に好評か不評かという統一した見解が見られるかを観察したり、使いやすさや機能に問題がないかどうかなど、その他の想定していた結果を確認したりします。このような戦略的実験は A/B テストとも呼ばれます。このプロセス全体は A/B テスティングと呼ばれます。これらの実験が完了したら、必要なテストデータを収集した後で、テストに使用したサーバー上のゲームバックエンドシステムを現行バージョンに戻します。

従来のレガシーデプロイ: 従来のスタイルのデプロイでは、スケジュールしたメンテナンスウィンドウ中にゲームをシャットダウンし、接続しているすべてのプレイヤーをドロップまたはドレインした後で、ゲームバックエンド内のすべてのサーバーインスタンスを最新のコードビルドで更新します。このデプロイは、実行するたびにすべてのプレイヤーに影響するため、スケジュールに先立ってプレイヤーに通知する必要があります。結果として、このモデルはプレイヤーに最も大きな影響を与えるため、できる限り避けてください。ゲームの更新をデプロイした後で、ゲームの再開を待っているプレイヤーにゲームを開放する前に、ゲームのスモークテストを行うことができます。この場合、すべてのプレイヤーが短時間内にログインしてプレイしようとすると、トラフィックが急増する可能性があります。したがって、このようなトラフィックの急増を処理するようにゲームを設計していない場合は、プレイヤーをバッチに分けて徐々にゲームに戻すことができます。別の方法としては、開始時のトラフィックの急増に耐えられるようにインフラストラクチャをオーバープロビジョニングします。最終的にゲームのトラフィックが落ち着いたら、リソースをスケールダウンできます。このタイプのデプロイは、プレイヤー数が最も少ないオフピーク時に実施することをお勧めします。メンテナンスを頻繁にスケジュールしたり、時間を長くしたりすると、プレイヤー数や収益の減少につながるリスクがあります。また、プレイヤーは新しいリリース後に変更を期待するため、一定期間のダウンタイム後に戻ったときに変更がなければ、ゲームへの信頼を失うおそれがあります。

GAMEOPS_BP06: ピーク時の要件をサポートするために必要なインフラストラクチャを事前にスケールします。

大規模なゲームイベントに先立ってインフラストラクチャをスケールし、プレイヤーの需要の急増に対応できるようにする必要があります。

ライブゲームでは、新しいゲームのリリースに加えて、プレイヤーのエンゲージメントを維持し向上させる方法として、一般的にゲーム内イベント、プロモーション、新規コンテンツ、シーズンリリースなどを行います。このようなアクティビティでは、イベントやプロモーションの期間中に、プレイヤーのトラフィックが大量に発生します。ビジネスでは、イベントの意図した目標を達成するか上回ることを期待しており、ゲームインフラストラクチャは、イベント期間中の大量のトラフィックに耐えてサポートする必要があります。大規模なイベント中に予想されるプレイヤーロードをサポートできるように、事前にインフラストラクチャを準備することが重要です。準備として、ゲームオペレーションチームはセールスおよびマーケティングのステークホルダーと連携し、過去のプレイヤーの同時実行数、エンゲージメントメトリクス、セールスデータを調べて、予定のイベントで発生する予測需要を見積もる必要があります。新しいゲームのリリースを目的としたイベントの場合、ゲームオペレーションチームはこれらのステークホルダーと協力し、予測需要の規模を現実的に特定する必要があります。ゲームがどの程度の成功を達成するか予測するのは難しい場合がありますが、成功の期待値を全員が理解し、この期待値の目標をサポートできるようにインフラストラクチャをスケールおよびテストすることが重要です。

多くのゲームは、いくつかのステージを経てリリースします。ゲームを少数のプレイヤーに公開するソフトローンチから始まり、各ステージでプレイヤー数を有機的にスケールしながら、最終的に完全に一般公開します。ソフトローンチ期間中に問題をモニタリング、特定、追跡、解決するとともに、一般公開に向けて予測を調整することをお勧めします。

インフラストラクチャ要件を適切に見積もるには、ゲームをリリースする前に、本番稼働環境またはこれと似たステージング環境で実行しているゲームバックエンドに対してロードテストやパフォーマンステストを実行し、データを収集する必要があります。これらのテストは複数回実施し、ゲームのさまざまな状況をシミュレートして、バックエンドがあらゆる状況下でロードに耐えられることを検証します。デベロッパーは、これを実現するためにゲームプレイボットを記述して、ゲーム内のさまざまなワークフローを走査し、各種の状況をエミュレートできます。これらのテストでは、ゲームバックエンドのさまざまなシステムレイヤーをすべて検査し、各レイヤーやコンポーネントをテストして詳細を記録することが不可欠です。これらのテストから収集したデータは、ゲームのリリース時のプロビジョン計画に使用します。

アプリケーションの可用性と耐障害性を高めることで、単一障害点 (SPOF) をできる限り排除する必要があります。ロードテストを使用して、アップストリームとダウンストリームのさまざまなレイヤーで障害をエミュレートし、ゲームやその他のコンポーネントの動作を検証することで、SPOF を検出します。

ゲームのリリース、ゲーム内イベント、プロモーションの準備に必要と推定されたインフラストラクチャをプロビジョニングすることに加えて、オンデマンドで自動的にスケールするようにシステムを設定します。スケーリングイベントのしきい値を定義、設定、モニタリングし、ゲームバックエンド

をスケールして大量のプレイヤートラフィックに耐えられるようにする必要があります。トラフィックが変動する場合は、スケールアウトする時間が十分にない場合があるため、事前プロビジョニングが最適です。ゲームの初回リリース時には、予想を超える需要が発生して、自動システムによるリソースのスケーリングでは間に合わない場合があり、手動スケーリングが必要になることがあります。

AWS では、組織はゲームバックエンドで使用するサービスに対して、より高い Service Quotas をリクエストする必要があります。Service Quotas は、すべてのアカウントに対して設定し、意図した以上のインフラストラクチャをお客様が不注意で起動したり、スケールしたりするのを防ぎます。クォータを引き上げても、追加のリソースを消費するまではコストが発生しません。アカウントで実行しているゲームが、そのリージョンに設定されているサービスクォータの上限に達すると、プロビジョニングしたクォータを超えるすべてのリクエストおよびバーストしたプロビジョンは、サービスによってスロットリングされます。スロットリングは、意図しないエラーや予期しないエラーを引き起こし、プレイヤーエクスペリエンスを損なう可能性があります。スロットリングを回避するには、本番稼働中のゲームで使用するすべてのサービスについてサービスクォータのしきい値をモニタリングおよび追跡し、定期的に確認することが重要です。使用量が許容可能なサービスクォータのしきい値を超えた場合は、コンソールサポートセンターから サポートケース を作成することで、クォータの引き上げをリクエストできます。最初に該当するアカウントにログインするか、 Support API を使用する必要があります。

ゲームを Amazon GameLift でホストして立ち上げる場合は、 リリース前のチェックリスト を確認して準備してください。

GAMEOPS_BP07: プレイヤーの行動をシミュレートして、リリース前にパフォーマンスエンジニアリングおよびロードテストを実施します。

リリースに向けて準備するには、インフラストラクチャに対して大規模にテストできるゲームプレイシミュレーションを開発し、ピーク時の使用量要件を満たすようにスケールできることを確認します。

パフォーマンスエンジニアリングとは、アプリケーションの複数の重要な運用メトリクスをモニタリングして、アプリケーションのパフォーマンスをさらに改善できる最適化の機会を発見するプロセスです。これは、コード、その依存関係、関連プロセス、ホストオペレーティングシステム、基盤となるインフラストラクチャを最初にテストし、次にこれらを最適化する反復プロセスです。

アプリケーションのパフォーマンスをより詳細に分析するには、アプリケーションパフォーマンスモニタリング (APM) またはデバッグツールをアプリケーションコードに統合することをお勧めします。これにより、アプリケーションのすべてのフローにわたってアプリケーションの動作に関して

異常を追跡し、問題を切り分け、トラブルシューティング時間を短縮できます。APM ツールでパフォーマンスの遅いメソッドや外部操作を特定することもできます。

AWS X-Ray は、パフォーマンスのボトルネックを特定したり、本番稼働環境のエラーを分析およびデバッグしたりするなど、デベロッパーのパフォーマンスエンジニアリング活動を支援します。X-Ray を使用すると、アプリケーションおよび基盤となるサービスがどのように動作しているかを理解して、パフォーマンスの問題やエラーの根本原因を特定し、トラブルシューティングを行うことができます。ロードテストでは、アプリケーションとそのインフラストラクチャに合成プレイヤートラフィックを段階的にロードします。このロードテストを何回も繰り返すことで、他の品質保証 (QA) テストでは見過ごされた可能性のある、さまざまなシステムのボトルネック、アプリケーションエラー、例外、OS 問題などの問題を特定できます。

人為的なプレイヤートラフィックをシミュレートするには、ボットが必要です。このボットを使用して、ゲームクライアントのフローをエミュレートし、ゲームバックエンドとやり取りして実際のプレイヤーの行動をシミュレートします。通常、このデータは、ゲームプレイのログと人間が実施した QA テストで生成したデータから取得します。または、現実世界の小規模なアルファテストやベータテストに実際のプレイヤーを招待してゲームをプレイしてもらうことで生成したデータから取得します。

また、ロードテストを実行し、これら各種のロードテスト中にさまざまな種類の障害をゲームバックエンドに挿入して、各障害に対してシステムがどのように動作するかを確認することをお勧めします。将来発生する可能性がある障害のトラブルシューティングに役立てるために、システムの動作をオペレーションランブックに記録することが重要です。ロードテスト中に人間のテスターにゲームをテストしてもらう環境は、ロードテストを行っているのと同じ環境である必要があります。人間は、ロードテスト中にボットや他のメトリクスでは検出できないものを発見できます。

ゲームのリリース、主要なゲーム内イベントやプロモーションなど、重要なイベントには、AWS インフラストラクチャイベント管理 (IEM) を利用できます。IEM は、ゲームのリリース、主要なゲーム内イベント、プロモーション、移行など、計画したイベントの準備とデプロイの際に、アーキテクチャと運用のガイダンス、および運用サポートを提供します。IEM プロセスを通じて、システム運用の準備状況の評価、リスクの特定と軽減、イベントのデプロイを、他のすべてのエキスパートと共に自信を持って進めることができます。

AWS Fault Injection Simulator は、フォールト挿入実験を実行するためのフルマネージドサービスです。これを使用すると、アプリケーションのパフォーマンス、可観測性、回復力を容易に改善できます。フォールト挿入実験は、カオスエンジニアリングで使用します。これは、テスト環境や本番稼働環境でアプリケーションにストレスを与える方法であり、CPU やメモリの消費量の急増などの破壊的なイベントを作成し、システムの反応を観察して、改善を実装するために使用します。フォールト挿入実験により、チームは、分散システムでは検出しにくい隠れたバグ、モニタリングのブラインド

スポット、パフォーマンスのボトルネックを発見するために必要な現実的な状況を作り出すことができます。

運用

GAMEOPS04 - ゲームのヘルスをどのようにモニタリングしますか?

GAMEOPS_BP08: プレイヤーに影響する問題を検出およびモニタリングする機能をゲームに備えます。

ソーシャルメディアやプレイヤーからの問題の報告に対応するほかに、プレイヤーに影響する問題に伴う調査が必要なときに、これを検出できるモニタリングソリューションをゲームに備える必要があります。

テストをどれだけ増やしても、ゲーム内のすべての問題を特定できるわけではありません。通常、ゲームのリリース時には既知の問題があり、これらはゲームの次回のリリースで段階的に修正される予定になっています。既知の問題や再現可能な問題は、対処と修正が容易です。プレイヤーが経験する可能性があるすべての問題やバグを未然に防ぐことは困難です。このような問題の特定を支援するために、ゲームクライアントでは、さまざまな戦略的な場所でアプリケーションにログ記録とレポートを実装することをお勧めします。これにより、バックエンドチームはクライアント側の問題を特定できます。このような問題を早期に発見できれば、ゲームデベロッパーは問題が広まる前にトラブルシューティングと修正を行うことができます。追跡コードから報告されるデータとログには、プレイヤーを特定できる情報 (PII) を絶対に含めないでください。デバッグに役立つゲーム固有のメタデータのみを含める必要があります。

ゲームのクラッシュやバグなどの問題を検出して対応するための可観測性ソリューションを実装します。次に Amazon CloudWatch Synthetics を使用することで、プレイヤー向けのバックエンドゲームサービスのヘルスをモニタリングできる canary を作成できます。バックエンドサービスに X-Ray を装備し、分散サービス全体でリクエストをトレースし、カスタムログおよびメトリクスを Amazon CloudWatch に送信できます。サードパーティーソリューションの Backtrace.io や Sentry などは、ゲームのエラー報告用の一般的なソリューションです。パートナーの New Relic、Splunk、Datadog、 Honeycomb.io などが提供するアプリケーションパフォーマンスモニタリング (APM) ソリューションもよく使用されています。

ゲームのライブオペレーションチームとコミュニティマネージャーは、公式サポートチャネルに加えて、さまざまなソーシャルネットワークやチャネルもモニタリングして、プレイヤーによるフィードバック、苦情、バグレポートがないかを調べる必要があります。ゲーム固有の苦情を受け取ったら、

その都度確認して再現を試みるか、QA チームに送って審査を受ける必要があります。再現可能な場合は、ゲームデベロッパーに問題をエスカレーションして、プレイヤーベース全体に影響が及ぶ前にトラブルシューティングと修正を行います。

進化

GAMEOPS05 - ゲームを長期的に最適化するにはどうすればよいでしょうか?

GAMEOPS_BP09: プレイヤーの傾向とパターンの特定に役立つ主要なゲームメトリクスをモニタリングし、その情報を使用してゲームのリバランス、ゲーム設計の最適化、インフラストラクチャの改善を行います。

ゲームクライアントシステムやアプリケーションの使用状況、例外、クラッシュデータに加えて、ゲームバックエンドシステムに送信されるゲームテレメトリデータをキャプチャすることを強くお勧めします。このデータは、プレイヤーのアクティビティを示し、プレイヤーがゲームのさまざまな機能とどのようにやり取りしているかを理解できるものである必要があります。

実装によっては、ゲームクライアントは定義済みのゲーム機能またはゲーム環境内の場所でテレメトリデータを収集できます。データは、処理のためにバックエンド取り込みサービスに送信されます。何らかの理由でバックエンドサービスにアクセスできない場合、クライアントはバックエンドサービスが再び使用可能になるまで、ローカルデバイスにデータをローカルに保存できます。ゲームデザイナーは、このテレメトリデータを使用して、プレイヤーがゲームをどのようにプレイしているか、ゲームに異常がないかを確認します。例えば、テレメトリデータからプレイヤーの動作やマップ内のアイテムとのやり取りを抽出し、設定した時間枠におけるゲーム内でのすべてのプレイヤーによるアクティビティのヒートマップとしてプロットできます。このようなデータは、ゲームデザイナーが、武器の威力、ゲーム内のキャラクターの力、マップの複雑さなど、ゲーム内のさまざまな要素をどのようにバランスさせる必要があるかを確認するために役立ちます。通常、テレメトリの raw データは保存して処理し、分析結果を抽出してアナリストが視覚化できるようにします。

ゲームデベロッパーは、 ゲーム分析パイプライン ソリューションの実装を使用して、ゲームやサービスから生成したテレメトリデータの取り込み、保存、分析を行うためのスケーラブルなサーバーレスデータパイプランを立ち上げることができます。このソリューションはデータのストリーミング取り込みをサポートしているため、ユーザーは数分以内にゲームやその他のアプリケーションからインサイトを取得できます。

カスタムゲームテレメトリデータの取り込み、保存、処理、分析のために、AWS は ビッグデータの処理と分析に特化したサービス も多く提供しています。

リソース

運用上の優秀性に関する AWS のベストプラクティスの詳細については、以下のリソースを参照してください。

ドキュメントとブログ

- Architecture Best Practices for Game Tech (Game Tech のアーキテクチャに関するベストプラクティス)
- ベストプラクティスの AWS 環境を確立する
- AWS コントロールタワーのlanding zone に対する AWS マルチアカウント戦略
- ゲーム分析パイプライン
- AWS で CI/CD パイプラインをセットアップする
- その他の関連する CI/CD ブログ
- ブログ記事「Game DevOps made easy with AWS Game-Server CD Pipeline (ゲームサーバー CD パイプラインを使用してゲームの DevOps を簡単に)」
- Amazon GameLift Prepare for launch (リリース準備)
- Best Practices for Organizational Units with AWS Organizations (組織単位に関するベストプラクティス)
- AWS X-Ray
- AWS インフラストラクチャイベント管理 (IEM)

パートナーのソリューション

- New Relic
- Splunk APM
- Backtrace.io
- Sentry
- Datadog APM
- Honeycomb.io

ホワイトペーパー

- Organizing Your AWS Environment Using Multiple Accounts (複数のアカウントを使用して環境を整理する)
- Introduction to Scalable Game Development Patterns on AWS (スケーラブルなゲーム開発パターン)

動画コンテンツ

- YouTube シリーズ: Building Games on AWS (AWS でのゲーム構築)
- Re:Invent 2017: How Amazon Scales Its Infrastructure to Handle Billions of Trans (Amazon がインフラストラクチャをスケールして数十億件のトランザクションを処理している方法)
- Re:Invent 2019: Scaling up to your first 10 million users (最初の 1,000 万人のユーザーへのスケールアップ)

トレーニング資料

- カリキュラム - Game Tech Starter Pack (Game Tech スターターパック)

セキュリティ

セキュリティの柱とは、リスクの評価と軽減を通じてビジネス価値を提供しながら、情報、システム、アセットを保護する能力のことを指します。ゲームは、グローバルな認知度とプレイヤー数の多さから、エクスプロイターやハッカーなど、システムを悪用または不正使用する方法を探している者にとって格好のターゲットとなります。その結果、プレイヤーエクスペリエンスの質が低下し、ゲームデベロッパーではコスト増が生じる可能性があります。強力なセキュリティ体制を維持する準備を整えるには、 責任共有モデルで説明しているように、セキュリティに関する AWS の責任範囲とお客様の責任範囲を理解することが重要です。この柱では、クラウド内でゲームを開発および運用する際に考慮すべきクラウドセキュリティのベストプラクティスに関するガイダンスを提供します。

設計原則

Well-Architected Framework のホワイトペーパーに記載しているセキュリティの柱に関する設計原則に加えて、以下の設計原則がクラウド内でのゲームワークロードのセキュリティ強化に役立ちます。

プレイヤーの利用行動をモニタリングおよびモデレートする: 使用状況データをキャプチャして分析し、プレイヤーがゲームやソーシャル機能とどのようにやり取りしているかを理解するのに役立てます。このデータを分析することで、プレイヤーエクスペリエンスを損なうような攻撃的および不適切な行動を検出して対処できます。

定義

クラウド内でのセキュリティには、5 つのベストプラクティス分野があります。

- アイデンティティとアクセスの管理
- 発見的統制
- インフラストラクチャの保護
- データ保護
- インシデント対応

システムを設計する前に、セキュリティ対策とアクセスコントロールを確立する必要があります。また、セキュリティインシデントを特定し、システムやサービスを保護して、データ保護によってデータの機密性と完全性を維持できるようにする必要があります。明確に定義され、実践に裏打ちされたプロセスを利用して、セキュリティインシデントに対応してください。これらのツールやテクニックは、金銭的な損失の予防や規制遵守という目的を達成するためにも重要です。

ベストプラクティス

クラウド内でのセキュリティには 5 つのベストプラクティス分野があります。

トピック

アイデンティティとアクセスの管理

GAMESEC01: プレイヤーのアイデンティティとアクセスの管理はどのようにして行いますか?

ゲームの開発に当たっては、ゲームや関連サービスへのアクセス権をプレイヤーに付与する方法を決定する必要があります。この決定は、プレイヤー獲得と収益化の戦略、プレイヤーエクスペリエンス、その他の要因 (ゲームパブリッシングパートナーから提供されている既存の機能など) の影響を受けます。例えば、ゲームでは、購入を要求し、ユーザープロファイルを作成して支払い方法を各自のアカウントに関連付けることをプレイヤーに義務付ける場合があります。

または、プレイヤーをゲームに導入しやすくするために、ユーザーアカウントを作成しなくてもゲームプレイを開始できるようにして、プレイヤーが初めてゲームを試す可能性を高めようとする場合があります。通常、ゲームには、プレイヤーのアイデンティティとアクセスの管理に関して、さまざまなアプローチから 1 つまたは複数を組み合わせて実装します。

未認証または匿名のアクセス: このアクセスレベルは、プレイヤーが新しいユーザーアカウントを作成したり、ソーシャルネットワークやゲームプラットフォームで各自の ID を関連付けたりする必要がないゲームに利用できます。これは、プレイヤーがゲームを始める最も簡単で迅速な方法であり、特にゲームデベロッパーがモバイルゲームへのプレイヤーの初回の導入を容易にしたい場合に役立ちます。このアクセスシナリオでは、インストールしたゲームの使用状況を確認する場合、ゲームクライアントのプログラミングを通じて、デバイスでのすべてのゲームセッションをまたいでプレイヤーを識別するための一意の識別子を生成し、これをプレイヤーのデバイスに保存します。これにより、経時的な使用状況に関する分析レポートを作成できるようになります。後日、プレイヤーがアカウントを作成することを選択した場合は、その新しいユーザーアカウントを、以前に生成した一意の識別子と関連付けて、統計やゲーム実績などのプレイヤーの使用履歴を新しいプレイヤー ID とリンクさせることができます。プレイヤーが最終的にアカウントの作成やリンクを行わなかった場合、プレイヤーがゲームとやり取りするために使用するデバイスは一意に識別できますが、プレイヤーに関する回復可能な情報は収集および保存されません。したがって、プレイヤーのデバイスが破損すると、そのデバイスに関連付けられていた以前の保存データも失われ、回復できない可能性があります。

ユーザーネームとパスワードによる認証: ゲームでは、ゲームのバックエンド内に保存しているユーザーネームとパスワードを使用して、プレイヤーが各自のユーザーアカウントを作成できる場合があります。この一般的な理由としては、ゲームデベロッパーが協働しているゲームパブリッシャーが既存のプレイヤーアカウントシステムを既に持っていて、これをデベロッパーが統合している場合があります。または、デベロッパーが独自のゲームを発行してプレイヤーエクスペリエンスを簡素化する

ために、発行するすべてのゲームにアクセスできる単一のユーザーアカウントを作成することをプレイヤーに許可する場合があります。

サードパーティーのソーシャルネットワークやゲームプラットフォームとリンクした認証とアカウント: 通常、オンラインゲームやソーシャル機能を備えたゲームでは、プレイヤーエクスペリエンスを簡素化する方法として、サードパーティーのアイデンティティプロバイダーフェデレーションを提供します。認証のためにユーザーネームとパスワードの組み合わせを作成するようにプレイヤーに要求する代わりに、ID フェデレーションを使用して、ソーシャルネットワークやゲームプラットフォームでサードパーティーアカウントを介して認証することをプレイヤーに許可できます。このログインプロセスにより、サインインと登録が簡単になり、アカウント作成が強制されない便利な方法を通じて、ゲームへのスムーズなアクセスをプレイヤーに提供できます。ゲームデベロッパーは、フェデレーションログインによってプレイヤー検証ワークフローを合理化できます。また、プレイヤーデータをより信頼性の高い方法で管理してパーソナライゼーションを実現できます。プレイヤーがサードパーティーのアイデンティティプロバイダーに既に提供している可能性があるデータを再度プレイヤーに要求する必要はありません。さらに、これらのプラットフォームは、プレイヤーをその友人とリンクするなど、追加のソーシャル機能との統合も提供します。

ゲームにセキュアなアクセスコントロールを組み込むには、以下のベストプラクティスが役立ちます。

GAMESEC_BP01: ゲームバックエンドサービスへのリクエストを認証します。

ゲームバックエンドサービスに送信したリクエストを認証することで、ゲームバックエンドへの望ましくないリクエストを排除する必要があります。

プレイヤーのログイン用に認証サービスを提供し、プレイヤーを正常に認証したら、JSON Web Token (JWT) などの存続期間が短い安全なトークンをゲームクライアントに返す必要があります。これらのトークンには、プレイヤー属性やその他の関連メタデータを含むクレームアサーションを含めることができます。これらのアサーションは、ゲームクライアントからゲームバックエンドに送信される後続のリクエストで使用し、リクエストを認証したり、認証済みのプレイヤーのコンテキストでリクエストを承認したりできます。独自のプレイヤー認証システムを設計して構築し、必要な改善とメンテナンスを継続的に行うか、 Amazon Cognito が提供するスケーラブルで安全なユーザーサインアップ、サインイン、アクセスコントロール機能を利用するかを選択できます。Amazon Cognito ユーザープールを使用すると、ユーザーディレクトリを作成し、API をゲーム内に統合してサインアップ、サインイン、パスワードリセットの各ワークフローに使用できます。このユーザーディレクトリは、サードパーティーのアイデンティティプロバイダーと統合できます。Application Load Balancer と Amazon API Gateway は、どちらも Cognito との統合を提供し、これらのサービスでホストしているカスタムゲームバックエンドに送信されるリクエストのユーザー認証を簡単に統合できます。

ゲームが匿名アクセスをサポートしていて、プレイヤーを認証できない場合は、ゲームバックエンドと統合する際に、クライアント認証アプローチを使用して安全なエクスペリエンスを提供します。ゲームクライアントが AWS サービスを使用している場合、これらのサービスへのリクエストには認証情報を使用して署名する必要があります。未認証のユーザーの認証情報をゲームクライアントに提供するには、AWS SDK を使用して Amazon Cognito ID プール から存続期間が短い認証情報を取得し、これを使用して AWS のサービスへのリクエストに署名できます。これらの認証情報はゲームクライアントから更新できます。ゲームクライアントから AWS SDK と直接統合するだけでなく、カスタム認可をサポートする Amazon API Gateway などのサービスを使用して、独自のゲームバックエンドを構築することもできます。独自のゲームバックエンドサービスを設計することで、カスタムのサーバー側ロジックを使用して、すべてのリクエストを権限を持って制御できます。Amazon GameLift を使用してホストしているゲームのバックエンドサービスを構築する方法の詳細については、「 Design your backend service (バックエンドサービスを設計する)」を参照してください。

GAMESEC_BP02: マルチプレイヤーゲームに参加するためのプレイヤーリクエストは、ゲームバックエンドサービスで検証する必要があります。

通常、マルチプレイヤーゲームでは、プレイヤーが利用可能なセッションのリストから直接オプションを選択してゲームセッションに参加します。または、リクエストを送信してマッチを見つけてもらいます。この場合、該当するゲームセッションを特定して接続情報 (通常は IP アドレスとポート) をプレイヤーのゲームクライアントに提供することは、ゲームデベロッパーの責任となります。実装は、開発するゲームのジャンルによって異なる場合がありますが、いずれにしても、プレイヤーのゲーム参加リクエストをサーバー側で検証するのがセキュリティのベストプラクティスです。

例えば、セッションベースのマルチプレイヤーゲームでは、プレイヤーからのゲームセッションへの参加リクエストは、サーバーへの接続を承認する前に、ゲームバックエンドマッチメイキングサービスを使用してゲームサーバーソフトウェアで検証する必要があります。プレイヤーがゲームセッションへの参加をリクエストすると、ゲームサーバーは、プレイヤーセッション ID やゲームバックエンドのマッチメイキングサービスからゲームクライアントに以前に提供されたサーバー生成チケットなど、リクエストの一意識別子を確認する必要があります。ゲームサーバーへの接続を開始すると、サーバー側のソフトウェアは、この情報を使用してプレイヤーの接続リクエストが有効であることをマッチメイキングサービスと検証し、プレイヤーが参加するスポットが別のプレイヤーのゲームセッションで予約済みでないことを確認します。Amazon GameLift を使用してホストするゲームについては、 Amazon GameLift のドキュメント を参照し、このタイプのサーバー側検証をどのように実装できるかの例をご覧ください。

GAMESEC_BP03: パスワードを必要とするプレイヤーユーザーアカウントでは、強力なセキュリティポリシーを適用する必要があります。

プレイヤーがユーザーアカウントと一緒にパスワードを作成できるゲームの場合は、パスワードが強力なポリシーに準拠することを義務付ける必要があります。例えば、Amazon Cognito ユーザープールでは、ユーザーアカウントの パスワード要件の定義 をサポートしています。

GAMESEC_BP04: プレイヤーが各自のアカウントに多要素認証 (MFA) を設定するためのオプションを提供します。

プレイヤーアカウントは、特にゲーム内の通貨や購入をサポートするゲームで、悪意のある攻撃者に格好のアセットとなります。ゲームデベロッパーのプレイヤーサポートチームに対するプレイヤーアカウントのハッキングやソーシャルエンジニアリングが増えているため、プレイヤーのユーザーアカウントに多要素認証 (MFA) を設定するオプションを提供して、プレイヤーのアカウントのセキュリティを強化することが重要です。

プレイヤーが MFA を使用して各自のアカウントにアクセスすると、E メール、電話番号、または専用の多要素認証モバイルアプリケーションに一時コードが送信されます。正常に認証するには、この一時コードを限られた時間内に入力してログインする必要があります。MFA は、新しいジオロケーションから認証しようとしているアカウント、悪質な行為の可能性があるとしてプレイヤーサポートがフラグを立てたアカウント、さらにはゲームに長期間ログインしていないアカウントに対する保護にも使用できます。例えば、Amazon Cognito ユーザープールでは、ユーザーディレクトリでの 多要素認証の設定 をサポートしています。

GAMESEC02: ゲームコンテンツへの不正アクセスはどのようにして防ぎますか?

現代のゲームには、ダウンロード可能コンテンツ (DLC) など、大量のコンテンツが含まれています。DLC は、プレイヤーのエンゲージメントとゲームの収益化の重要な一面です。プレイヤーは、新しいキャラクター、レベル、チャレンジが次々と提供されることを期待しており、ゲームデベロッパーはプレイヤーを保持するために、新鮮なコンテンツに対する途切れることのない需要に常に対応する必要があります。コンテンツの多様性とサイズは、ゲームの種類や、ゲームをプレイするデバイスが PC、コンソール、モバイルのいずれであるかによっても大きく異なります。ゲームのコンテンツを不正アクセスから確実に保護することが重要です。

GAMESEC_BP05: ダウンロード可能コンテンツへのアクセスは、承認済みのクライアントとユーザーに限定します。

ゲームコンテンツへのアクセスは、承認済みのアプリケーションとクライアントに限定して許可します。

ダウンロード可能なゲームコンテンツを保存するための費用対効果の高いスケーラブルなオリジンとして Amazon S3 を使用し、パフォーマンスの高いグローバルなコンテンツ配信をプレイヤーに提供するために Amazon CloudFront を使用することを検討します。どちらのサービスにも、保存しているデータへのアクセスを制限するためのメカニズムが組み込まれています。

S3 のコンテンツへのアクセスを許可する必要がある場合、考慮すべきベストプラクティスがいくつかあります。デフォルトでは、S3 バケットを作成した AWS アカウントだけが、バケット内に保存されているオブジェクトにアクセスできます。内部アプリケーションへのアクセスを許可し、Amazon S3 バケットに保存されているコンテンツを管理するには、 Identity and Access Management (IAM) を使用してポリシーを作成し、適切なアクセスを提供します。IAM ロール は、フェデレーティッドユーザー、システム、アプリケーション (Amazon EC2、Lambda などのサービスでホスト)、コンテナベースのアプリケーション (Amazon EKS や Amazon ECS でホスト) に関連付けることができます。例えば、AWS SDK や AWS CLI を使用して、S3 バケット内のゲームコンテンツアセットを公開および管理できます。このユースケースをサポートするために、S3 バケットに対してゲームコンテンツを読み書きするための適切なアクセス権を持つ IAM ロールを作成し、このロールをソフトウェアとスクリプトをホストする EC2 インスタンスに関連付けることができます。

リソースベースのポリシーは、バケットや特定のオブジェクトに対しても定義できます。 S3 バケットポリシー は、S3 バケットに関連付けて、バケットおよびバケット内のオブジェクトへのアクセスを制限したり、他のアカウントからの S3 リソースへのアクセスを許可したりするために使用できます。例えば、複数のチームや別々の開発スタジオが同じゲームコンテンツに取り組んでいて、S3 で一元的にホストしているコンテンツに対する同じアクセスが必要な場合、S3 バケットポリシーを使用して S3 リソースへのクロスアカウントアクセスの許可を定義できます。共有データへのデータアクセスの管理を簡素化するには、 S3 Access Points を使用することを検討してください。各アプリケーションや一連のアプリケーションに固有の名前と許可を持つアクセスポイントを作成することで、簡単に管理できます。Amazon S3 のドキュメントは、 Amazon S3 でのアクセスコントロールに関する追加のベストプラクティスを記載しています。

コンテンツへの短期アクセスを許可する一時的な URL を生成することをお勧めします。Amazon S3 は、 事前署名付き URL の生成をサポートしています。これにより、オブジェクト所有者は他のユーザーとオブジェクトを共有するために、独自のセキュリティ認証情報を使用して事前署名付き URL をバックエンド内に生成し、オブジェクトをダウンロードするための期限付きの許可を付与できます。これにより、アクセスを許可されたエンドユーザーまたはアプリケーションは、アカウントや IAM の許可を持つ必要がなくなり、代わりに事前署名付き URL を使用してコンテンツにアクセスできます。このベストプラクティスは、ダウンロード可能なコンテンツへのアクセスを該当する承認済みプレイヤーに許可したり、期間限定のゲームコンテンツへの一時的なアクセスを付与したりするなど、さまざまなゲームユースケースで一般的に使用できます。事前署名付き URL は、S3 バケット

にコンテンツをアップロードするための一時的な許可を付与するためにも使用できます。例えば、クライアントログをアップロードするためのアクセス権をプレイヤーに付与して、プレイヤーサポートチームがプレイヤーサポートケースのトラブルシューティングをできるようにすることができます。事前署名付き URL の機能を制限するためのベストプラクティスを参照してください。

S3 Block Public Access は一連のセキュリティコントロールであり、S3 のバケットおよびオブジェクトにパブリックアクセス権がないことを確認するために、特定のユーザーが定義した設定をオーバーライドして S3 アクセスポイント、バケット、AWS アカウントのすべてをまたいで一元管理を有効にします。

アプリケーション、ゲームデベロッパー、アーティスト、その他の担当者が、開発や管理の目的で S3 バケット内のコンテンツに直接アクセスする必要がある場合は、コンテンツ配信ネットワークを使用し、インターネット上でプレイヤーや他のユーザーに公開されているコンテンツへのアクセスを提供することをお勧めします。これにより、アクセス頻度が高いコンテンツをキャッシュすることで、ダウンロードのパフォーマンスを高め、コストを削減します。Amazon CloudFront は、コンテンツをグローバルに配信できます。そのために、コンテンツをキャッシュしてプレイヤーの近くに配信するとともに、Amazon S3 などのゲームのダウンロードオリジンに対する負荷を軽減します。

パブリックコンテンツを S3 バケットから直接提供するのではなく、このコンテンツをプライベートに保ち、CloudFront を使用してパブリックに配信することをお勧めします。CloudFront は、プレイヤーがプライベートコンテンツ (有料プレイヤー専用の新しいゲームのダウンロードなど) へのアクセスに 署名付き URL または 署名付き cookie のいずれかを使用することを要求するように設定できます。次に、署名付き URL を作成して認証済みユーザーに配信するアプリケーションか、認証済みユーザーの署名付き cookie を設定する set-cookie ヘッダーを送信するためのアプリケーションを開発します。署名付き URL または署名付き cookie を作成してファイルへのアクセスを制御する場合は、URL の有効期限が切れる最終日時を指定できます。オプションとして、コンテンツへのアクセスに使用できるコンピュータの IP アドレスまたはアドレス範囲を指定することもできます。これは、アクセスを特定のゲーム開発スタジオパートナーまたは請負業者のネットワークに制限する場合に便利です。ファイル別にアクセスを制限する場合や、ユーザーが使用しているクライアントが cookie をサポートしていない場合は、署名付き URL を使用します。複数の制限付きファイルへのアクセスを許可する場合や、現在の URL を変更したくない場合は、署名付き cookie を使用します。署名付き URL は、署名付き cookie よりも優先されます。

GAMESEC_BP06: オリジンへのアクセスは、承認済みのコンテンツ配信ネットワーク (CDN) に制限します。

ユーザーがコンテンツ配信ネットワークを迂回して、Amazon S3 バケットなどのオリジンからコンテンツに直接アクセスするのを防ぐ必要があります。オリジンへのアクセスを承認済みの CDN にの

み制限することが重要です。これにより、オリジンからの不要なコンテンツ提供に伴うデータ転送コストを削減できます。また、オリジンコンテンツへのすべてのパブリックアクセスを同じエントリポイントを通過させることで、セキュリティ体制も向上します。このエントリポイントで、AWS WAF レイヤー 7 フィルタリング、セキュリティ関連の HTTP リクエストパラメータの取り込みと検査、分散型サービス拒否 (DDoS) からの保護など、エッジセキュリティコントロールをデプロイできます。これらのコントロールを S3 オリジンに実装するには、 Amazon CloudFront オリジンアクセスアイデンティティ (OAI) を使用して、S3 オブジェクトに対するすべてのリクエストを CloudFront ディストリビューションから発信させることができます。レイヤー 7 フィルタリングを提供するために、AWS WAF を CloudFront ディストリビューションに関連付けることをお勧めします。ただし、追加の CDN からコンテンツを配信する場合は、オリジンリクエストに 1 つ以上のカスタム HTTP ヘッダーを挿入するように CDN を設定できます。このヘッダーは、AWS WAF によって検査されるため、承認済みの CDN プロバイダーから受信トラフィックが発信されたことを確認できます。このアプローチは、オリジンが Application Load Balancer (ALB) の背後でホストされている場合に、ユーザーが CDN プロバイダーを迂回するのを防ぐためにも役立ちます。ALB は、レイヤー 7 を保護するために AWS WAF に関連付けることができます。AWS WAF は、ALB で検査できるカスタム HTTP ヘッダーを挿入するように設定できます。これにより、ロードバランサーへの着信トラフィックが AWS WAF で最初に処理および検査済みであることを確認できます。

GAMESEC_BP07: 不正アクセスを防ぐために地域制限を実装します。

プレイヤーがコンテンツをリクエストすると、CloudFront はプレイヤーの場所に関係なく、最も近いエッジロケーションからリクエストされたコンテンツを提供します。ただし、コンテンツに対する世界の特定地域のユーザーからのアクセスを制限する必要がある場合もあります。例えば、ゲームのローリングデプロイ戦略に従って国別に段階的にコンテンツをリリースする場合や、国固有のアクセスコントロールに従う必要がある場合があります。地域制限 (または地理的ブロック) を使用すると、CloudFront ディストリビューションを通じて配信しているコンテンツに対する、特定地域のユーザーからのアクセスを禁止できます。CloudFront の地域制限機能を使用すると、ディストリビューションに関連付けられているすべてのファイルへのアクセスを制限したり、国レベルでアクセスを制限したりできます。また、サードパーティーの位置情報サービスを使用して、ディストリビューションに関連付けられているファイルのサブセットへのアクセスを制限したり、国レベルよりも細かい粒度でアクセスを制限したりすることもできます。

CloudFront の地域制限を使用すると、プレイヤーの所在国が、承認された国の許可リストに含まれている場合にのみ、コンテンツへのアクセスをプレイヤーに許可し、禁止された国の拒否リストに含まれている場合はコンテンツへのアクセスを禁止できます。ブロックしている地理的な場所からリクエストを受け取ると、CloudFront は 403 Forbidden HTTP ステータスコードをプレイヤーに返します。

GAMESEC_BP08: デジタル著作権管理 (DRM) ソリューションでコンテンツへのアクセスを制限します。

アクセスコントロールベースのアプローチに加えて、暗号化ベースのアプローチを採用することもできます。この場合は、デジタル著作権管理 (DRM) ソリューションを使用してプライベートコンテンツを暗号化し、復号キーを承認済みのプレイヤーに配布します。DRM ソリューションは、ゲームコンテンツを早期にダウンロードすることをプレイヤーに許可するとしても、所定の時間までコンテンツのアクセスや再生をできないようにする場合にお勧めします。例えば、これは特に PC ゲームで一般的ですが、プレイヤーはゲームを予約注文し、暗号化されたファイルの早期ダウンロードを自動的に開始するようにゲームクライアントを設定します。ゲームが正式にリリースされたときには、ゲームがダウンロード済みで準備が完了しているため、すぐにプレイを開始できます。ゲームのリリース後に、プレイヤーのゲームクライアントは DRM バックエンドソリューションに対して復号キーを要求し、以前にダウンロード済みのファイルを復号してゲームのプレイを開始できます。DRM システムは、承認済みのプレイヤーがダウンロードおよびインストールしたゲームの不正な再配布や改ざんを防止する手段としても有効です。DRM システムは、暗号化キーを交換してプレイヤーに復号キーの取得を許可するために、オリジンと統合する必要があります。商用 DRM システムプロバイダーは、特定の機能を備えた幅広いソリューションを提供し、さまざまなデバイスをサポートしています。

発見的統制

GAMESEC03: ゲーム内でのプレイヤーの利用行動をどのようにモニタリングおよび分析しますか?

ポジティブなプレイヤーエクスペリエンスを維持するには、関連するデータをキャプチャ、保存、分析するプロセスを通じて、プレイヤーがゲームの機能や他のプレイヤーとどのように関わっているかを理解する必要があります。

GAMESEC_BP09: プレイヤーの使用状況ログを収集、保存、分析して、不適切な行動を検出します。

ゲームにログを収集する機能を設定して、プレイヤーがゲームの機能をどのように使用し、他のプレイヤーとどのようにやり取りしているかを理解することで、プレイヤーエクスペリエンスを損なう不正行為を防止できます。これを行うには、構造化したログイベントを ゲーム分析パイプライン に送信するか、ログ記録ソリューションである Amazon CloudWatch Logs、Amazon OpenSearch Service などを使用するか、あるいは AWS パートナーソリューションである Datadog、Sumo Logic、New Relic、Honeycomb、 Splunk などを使用できます。これらのプレイヤーの使用状況ログ

は、プレイヤーの特定のアクションを調査する必要がある場合に利用して検出できるように構成する必要があります。

データをキャプチャしたら、不適切な利用行動の検出に役立つツールを実装することを検討します。例えば、ゲームのソーシャル機能として、プレイヤーのメッセージングやボイスチャット、オンラインフォーラムなどがゲーム内にある場合は、これらのプレイヤーエンゲージメントのログを、モデレーションのために分析できる形式で保存することをお勧めします。ゲームのボイスチャット機能を設定して録音内容を Amazon S3 にエクスポートし、 Amazon Transcribe を使用して音声をテキスト形式に変換し、保存して処理できるようにします。または、リアルタイムのストリーミング文字起こしを行うこともできます。そのためには、ゲームバックエンドのボイスチャットサービスを Transcribe API と直接統合し、リアルタイムで ストリーミングオーディオの文字起こし を行います。モデレーションチームは、コンテンツを手動で確認できます。コンテンツを標準形式にすると、AWS AI/機械学習サービスを使用してモデレーションを自動的に実行することもできます。Amazon Comprehend を使用すると、自然言語処理 (NLP) を実行して、非構造化テキストから情報を発見できます。これは、会話を該当するトピックに分類して整理し、みだらな言葉の使用といった不適切な行為を特定するのに役立ちます。

ゲームでプレイヤーがコンテンツを生成またはアップロードできる場合は、 Amazon Rekognition を使用して画像のコンテンツを識別し、モデレートすることを検討してください。プレイヤーのライブストリーミングなどの動画ユースケースでは、動画ストリームを Amazon Kinesis Video Streams に送信できます。これを Amazon Rekognition Video または独自のカスタムアプリケーションと統合して、リアルタイムで分析およびモデレートできます。ゲームでは、プレイヤーサポートエージェントに連絡する機能として、 Amazon Connect などのコールセンターや、Amazon Lex を使用したチャットボットをプレイヤーに提供する場合があります。Amazon Connect は、 ライブおよび録音した会話のモニタリングをサポートします。Amazon Lex で構築したプレイヤーサポートチャットボットとプレイヤーとの間のやり取りを分析するには、これらのやり取りの 会話ログ を Amazon CloudWatch Logs に保存し、これを前述のように S3 にエクスポートして分析できます。

また、ゲームを Amazon Fraud Detector と統合することもできます。これはフルマネージドサービスであり、機械学習を使用して潜在的な不正行為を特定し、お客様がオンライン詐欺を迅速にキャッチできるようにします。Fraud Detector を使用すると、不正行為の可能性があるアクティビティを検出してレビュー用のフラグを付けることができます。これにより、不正なゲーム内購入をリアルタイムで防止したり、行動の変化や異常を探して侵害を受けたアカウントを検出したり、正当なアカウント登録とリスクの高い新規アカウント登録を区別したりできます。

Amazon Lookout for Metrics では、機械学習を使用して、ビジネスデータと運用データの異常を自動的に検出して診断し、ビジネスにとって最も重要なメトリクスをより迅速かつ正確にモニタリングします。また、このサービスにより、収益、ログイン数、トランザクション数、リテンションの急激な

低下などの異常の根本原因も診断しやすくなります。ゲームデベロッパーは、機械学習のセットアップ経験がなくても、Amazon S3、Amazon CloudWatch、Amazon RDS、Amazon Redshift、SaaS アプリケーションなど、一般的なデータソースに接続できます。例えば、 Amazon Lookout for Metrics をゲーム分析パイプラインなどのデータソースと統合 して動作の分析を開始し、異常を検出できます。

または、カスタムの機械学習モデルを構築、トレーニング、ホストするために Amazon SageMaker を使用し、コンテンツモデレーション、毒性検出、チート検出、不正検出などのユースケースに対処することもできます。

カスタムのゲーム使用状況ログの生成に加えて、 S3 サーバーアクセスログ、CloudFront アクセスログ、 ALB アクセスログなどの関連サービスからシステムレベルのログをキャプチャして保存することもお勧めします。これらのログは、アカウントの Amazon S3 バケットに保存でき、ゲーム内のプレイヤーの使用状況に関する情報をシステムレベルの情報 (IP アドレス、リクエストヘッダー、ゲームバックエンド内で設定した関連するリクエストの操作やフィルタリングなど、接続の詳細を含む) と関連付けるのに役立ちます。これらのログは、前述したのと同じログ記録ソリューションに送信できます。また、Amazon S3 からログを移動しなくても、 Amazon Athena で SQL クエリを使用して分析 することもできます。

Access Analyzer for S3 は、バケットアクセスポリシーをモニタリングする機能であり、ポリシーから S3 リソースに提供するアクセスを、意図したものに限定します。Access Analyzer for S3 は、バケットアクセスポリシーを評価し、意図した以外のアクセス権を持つと思われるバケットを検出して迅速に修正できるようにします。

AWS 環境内の悪意のあるアクティビティや不正行為を継続的にモニタリングするには、 Amazon GuardDuty の使用を検討してください。GuardDuty は、環境内のアカウントの動作、ネットワークアクティビティ、データアクセスパターンをモニタリングすることで、脅威を特定します。GuardDuty は、CloudTrail イベントログ、Amazon VPC フローログ、DNS ログなど、複数のデータソースにわたって数百億件のイベントを分析し、潜在的な脅威を検出します。Amazon CloudWatch Events および Lambda との統合により、GuardDuty アラートを該当するセキュリティチームに自動的に転送し、詳細な分析を行うことができます。

AWS Security Hub は、AWS でのセキュリティ状態の包括的なビューを提供し、セキュリティの業界標準やベストプラクティスに照らして環境をチェックできるようにします。Security Hub を使用すると、AWS アカウント、サービス、およびサポートしているサードパーティーのパートナー製品全体からセキュリティデータを収集し、セキュリティの傾向を分析して、最も優先度の高いセキュリティ問題を特定できます。この Security Hub と Amazon GuardDuty の統合 により、GuardDuty から Security Hub に結果を送信できます。Security Hub では、この検出結果をセキュリティ体制の分析に含めることができます。

不正行為者は、一般的にボットを利用してアカウントを乗っ取ったり、ゲームでチートしたりします。 WAF Bot Control では、過剰なリソースを消費したり、メトリクスをゆがめたり、ダウンタイムを引き起こしたりするなど、望ましくないアクティビティを実行する目立った広範なボットトラフィックを可視化して制御できます。

ランサムウェアは、システムやデータセットへのアクセスを不正に取得し、そのデータを暗号化して正当なプレイヤーによるアクセスをブロックするように設計された悪意のあるコードです。ランサムウェアがプレイヤーをシステムから締め出し、機密データを暗号化すると、サーバー犯罪者はデータのロックを解除する復号キーを提供する代わりに身代金を要求します。組織は攻撃に伴って完全にシャットダウンし、多大なコストがかかり、ビジネスの生産性が低下します。インシデントの発生前、発生中、発生後にランサムウェアへの対処能力を強化するベストプラクティスについては、「Securing your Cloud Environment from Ransomware (クラウド環境をランサムウェアから保護する)」を参照してください。

セキュリティの発見的統制分野におけるその他のベストプラクティスについては、Well-Architected Framework のホワイトペーパーを参照してください。

インフラストラクチャの保護

ゲームのワークロードに適用するセキュリティを実現するための インフラストラクチャ保護 に関するベストプラクティスについては、Well-Architected Framework のホワイトペーパーを参照してください。

データ保護

ゲームのワークロードに適用するセキュリティを実現するための データ保護 に関するベストプラクティスについては、Well-Architected Framework のホワイトペーパーを参照してください。

インシデント対応

GAMESEC04 - プレイヤーの不正行為や悪意のある行為に対応するためのポリシーはどのようにして定義および適用していますか?

GAMESEC_BP10 - インシデント対応計画を実装して不正行為者や悪意のある行為に対処します。

ゲームのワークロードに適用するセキュリティを実現するための インシデント対応 に関するベストプラクティスについては、Well-Architected Framework のホワイトペーパーを参照してください。

GAMESEC_BP11 - 不正行為者に関連付けられたアカウントを禁止します。

放置すると、ゲーム内で悪意のある行為が続いて、他のユーザーのゲーム体験に影響する可能性があるため、できる限り早く軽減する必要があります。利用規約に違反していることが確認された不正行為者に対して、禁止などの方法で制限を課すプロセスを実装する必要があります。通常、このような制限を課す状況を判断するためのルールと評価プロセスは、組織内のプレイヤーコミュニティチームや信頼と安全チームなどの担当者が決定します。不正行為者にフラグを付けたら、特定したプレイヤーに対処するために実行できる事前定義のワークフローを設定します。AWS Step Functions および Lambda 関数を使用して自動化したワークフローを実行できます。このワークフローでは、プレイヤーアカウントのバッチを入力として受け取り、 Bans（##）という名前の DynamoDB テーブルのエントリを更新します。このテーブルには、プレイヤーアカウント、禁止理由、期間の詳細を含めることができます。ゲームおよびアカウント管理システムの設計方法や不正使用の種類によっては、アカウント管理システムとは別に禁止記録システムを設定しておくと役立ちます。アカウント管理システムからプレイヤーのアカウントを排除しない場合は、単にゲームをプレイする機能をオフにすることを代わりに選択します。この方法は、プレイヤーのアカウント認証情報が、利用規約やポリシーが異なる複数のゲームにアクセスするために使用されている場合に便利です。

リソース

セキュリティに関するベストプラクティスの詳細については、以下のリソースを参照してください。

ドキュメントとブログ

- 一般的な Amazon Cognito シナリオ
- 署名付き URL の使用
- Use channel flows to remove profanity and sensitive content from messages in Amazon Chime SDK messaging (チャネルフローを使用して Amazon Chime SDK メッセージングのメッセージからみだらな言葉や機密性の高いコンテンツを削除する)
- Amazon GameLift のセキュリティ

ホワイトペーパー

- Secure Content Delivery with Amazon CloudFront (Amazon CloudFront を使用した安全なコンテンツ配信)
- AWS Security Incident Response Guide (セキュリティ対応ガイド)
- AWS Best Practices for DDoS Resiliency (DDoS に対する回復力に関するベストプラクティス)
- Securing your AWS Cloud environment from ransomware (クラウド環境をランサムウェアから保護する)

パートナーのソリューション

- Datadog
- Sumo Logic
- Splunk
- Honeycomb
- New Relic
- AWS Marketplace - DRM ソリューション

トレーニング資料

- Amazon Cognito の開始方法
- セキュリティのセルフペーストレーニング

信頼性

信頼性の柱には、インフラストラクチャまたはサービスの障害から復旧する、コンピューティングリソースを動的に取得して需要を満たす、および設定ミスや一時的なネットワーク問題などによる機能停止を緩和するシステムの能力が含まれます。

設計原則

AWS Well-Architected Framework のホワイトペーパーに記載している設計原則に加えて、クラウド内でゲームワークロードの信頼性を高めるために役立つ設計原則を以下に示します。

ビジネス予測に対応するために必要なピーク時のプレイヤー同時実行数とシステムのスケーラビリティの目標について合意する: ゲームのリリース前とゲームのライブ運用中に、ピーク時に予想されるプレイヤーの同時実行数の見積もりを作成し、これらの見積もりに応じたシステムスケーラビリティの目標を確立します。これにより、ゲームの信頼性のベースラインを作成できます。スケーリングポリシーを定義し、スケーリングシステムでアクティブなプレイヤーセッションを適切に管理するなど、可用性に影響を与えることなく、需要の変動に自動的に対応できるようにします。

信頼性と、プレイヤーエクスペリエンスへの影響を測定する: ゲームのヘルスを示す重要業績評価指標 (KPI) を定義します。インフラストラクチャとゲーム機能の変更が信頼性に及ぼす影響をモニタリングします。

定義

クラウド内の信頼性に関するベストプラクティスには、以下の 3 つの分野があります。

- 基盤
- 変更管理
- 障害管理

信頼性を達成するため、システムの基盤について十分に計画し、モニタリングを実施する必要があります。需要や要件の変更に対応するためのメカニズムも必要です。障害を検出し、自動的に修復できるシステムを設計することが必要です。

ベストプラクティス

クラウドアーキテクチャのベストプラクティスを以下に示します。

トピック

基盤

ゲームのワークロードに適用する信頼性を実現するための 基盤 に関するベストプラクティスについては、Well-Architected Framework のホワイトペーパーを参照してください。

変更管理

GAMEREL01 – ゲームインフラストラクチャは、プレイヤーの需要の変動に応じてどのようにスケールしますか?

プレイヤーの需要は時間とともに変動するため、ゲームインフラストラクチャはこれらの変動する要件を反映して適切にスケールできることが必要です。ゲームの人気を事前に予測することは困難ですが、インフラストラクチャの容量を簡単に追加または削除できるようにアーキテクチャアプローチを設計して、プレイヤー集団の変動に対応する必要があります。

GAMEREL_BP01 - アクティブなプレイヤーゲームセッションの状態を反映するスケーリング戦略を実装します。

ゲームのインフラストラクチャを自動的にスケールするソリューションを実装します。このソリューションでは、アクティブに接続されたプレイヤーセッションのステートフルな性質を反映し、ゲームプレイを中断せずに規模の拡大や縮小を正常に処理します。

クラウドでゲームを開発する利点の 1 つは、需要に応じてサーバーインフラストラクチャを自動的にスケールできる伸縮性です。ステートレスまたは非同期型のゲームやバックエンドサービスは、Amazon EC2 Auto Scaling ポリシー 、またはスケーラブルなウェブアプリケーションで一般的に採用している同様の手法を使用して動的にスケールできます。ただし、ステートフルまたは同期型のゲームをスケールする場合は、通常、よりカスタマイズしたアプローチを使用して、アクティブなプレイヤーセッションの中断を防ぐ必要があります。

ステートフルなゲームでは、ゲームバックエンドで生成したカスタムメトリクスを使用してプレイヤーセッションの状態と使用可能なゲームサーバーの容量をモニタリングし、これらをカスタムメトリクスとして Amazon CloudWatch に報告するのがベストプラクティスです。このデータを使用して、ゲームサーバーのスケーリングソフトウェアを実装できます。例えば、AWS Lambda 関数や AWS Fargate を使用してサーバーレスアプリケーションとして実装し、専用ゲームサーバーインスタンスのフリートを管理できます。この場合、AWS SDK を使用して API コールを行い、ゲームサーバービルドをホストしている Auto Scaling グループ の最小、最大、および必要な容量設定を更新します。

または、Amazon GameLift を使用してゲームサーバーをホストし、 すぐに使えるゲームサーバーオートスケーリング機能 を使用して、このスケーリングプロセスを自動的に管理することもできます。Amazon GameLift のオートスケーリング機能は、アクティブなプレイヤーセッションを認識し、ゲームサーバーインスタンスの終了またはスケールインを防止するように設定できます。また、プレイヤーをアクティブにホストしているゲームサーバーインスタンスの終了またはスケールインを防止するようにも設定できます。詳細については、「 Amazon CloudWatch で Amazon GameLift をモニタリングする」を参照してください。

GAMEREL_BP02 - ゲームでの複数の EC2 インスタンスタイプの使用をサポートします。

EC2 インスタンスを使用してゲームをホストする場合、または EC2 インスタンスでホストしているコンテナを AWS アカウントで使用する場合は、ホスティング戦略で複数のインスタンスタイプを使用する必要があります。複数のインスタンスタイプを使用することで、コンピューティングオプションの数が増えます。これらのオプションを使用してゲームをスケールし、プレイヤーの成長に合わせてサーバー数を増やすことができ、主力インスタンスタイプが使用できなくなった場合の信頼性が向上します。これは、スポットインスタンスを使用してゲームをホストする場合にもベストプラクティ

スとなります。スポットインスタンスの可用性はお客様の需要に応じて変動するためです。コストとパフォーマンスの要件を満たすには、複数のインスタンスタイプでゲームをテストし、インスタンスタイプの優先順位を決定する必要があります。Amazon EC2 Auto Scaling は、複数のインスタンスタイプとサイズの使用に加え、設定での 各インスタンスタイプへの重みの割り当て もサポートしているため、コンピューティングオプションに優先順位を付けることができます。

Amazon GameLift マネージドホスティングを使用してゲームをホストする場合、このソリューションでは Amazon EC2 インスタンスを使用してゲームサーバーをデプロイし、プレイヤーのゲームセッションをホストします。新しいフリートを設定する場合は、ゲームに必要なインスタンスタイプと、これらに対してゲームサーバープロセスを実行する方法を (ランタイム設定を使用して) 決定します。フリートのリソースを選択する場合は、ゲームのオペレーティングシステム、インスタンスタイプ (コンピューティングハードウェア)、オンデマンドインスタンスとスポットインスタンスのどちら (または両方) を使用するかなど、いくつかの要因を考慮する必要があります。Amazon GameLift でのホスティングコストは、主として、どのインスタンスタイプを使用するかによって異なります。詳細については、「 コンピューティングリソースの選択」を参照してください。

障害管理

GAMEREL02 – インフラストラクチャの障害がアクティブなプレイヤーに与える影響を最小限に抑えるには、どうすればよいですか?

ゲームサーバーの障害メトリクスとこれらの障害がプレイヤーの行動に与える影響を長期にわたってモニタリングすることで、ゲームの信頼性要件を満たすようにゲームサーバーのホスティング戦略を調整する必要があります。ゲームサーバーインフラストラクチャが劣化していることが判明したら、これがプレイヤーに影響を与えている場合はただちにサービスから削除します。または、サーバーでホストしているアクティブなプレイヤーセッションがないときを選んで事前に交換します。

ゲームを REST API としてホストする場合は、システムの信頼性を従来のウェブアプリケーションアーキテクチャと同様に管理し、トラフィックを複数のサーバーに分散してロードバランスを行うことで、サーバーの障害のリスクを軽減できます。

リアルタイム同期ゲームプレイの場合、ゲームセッションは、通常、仮想マシンまたはゲームサーバーインスタンスで実行しているゲームサーバープロセスでホストします。これは、ゲームプレイの状態を効率的に維持し、すべての接続されているゲームクライアントにレプリケートする必要があるためです。この実装では、プレイヤーのエクスペリエンスが、ゲームセッションをホストするゲームサーバープロセスのパフォーマンスや信頼性と密接に関連することになります。このタイプのアーキテクチャでは、ゲームサーバーの信頼性の管理が従来のアプローチよりも複雑になります。

ゲームサーバーの障害の影響を軽減するには、 Amazon ElastiCache for Redis や Amazon MemoryDB for Redis などの高可用性キャッシュまたはデータベースに対して、プレイヤーのゲーム状態の非同期更新を継続的に実行するようにゲームを設定できます。サーバーの障害が発生した場合、プレイヤーが最後に保存したゲーム状態を外部データストアから取得し、そのセッションを新しいゲームサーバーインスタンスで復元できます。ただし、このアプローチでは、この外部状態を管理するためにコストと複雑さが増すため、ペースの速いゲームや競争の激しいゲームには適さない場合があります。このようなゲームでは、状態の変化が頻繁かつ大規模に発生するため、パフォーマンスの高いインメモリキャッシュデータストアを導入しても、レプリケーションの遅延が大きくなり、セッションの復元に活用できないことがあります。この種のゲームでは、サーバーの喪失を受け入れ、プレイヤーをゲームロビーに送り返して別のセッションを見つけるか、自動的に別のゲームセッションにリダイレクトするのが最適なアプローチです。

後で問題を調査できるように、サーバーの中断の原因に関する有用なログデータをできる限り多く収集する必要があります。Amazon GameLift は、 フリートの問題をデバッグするためのガイダンスと、 Amazon GameLift フリートインスタンスにリモートからアクセスする 機能を提供しています。

GAMEREL_BP03 - ゲーム機能の疎結合を実装して、プレイヤーエクスペリエンスへの影響を最小限に抑えながら障害を処理します。

コンポーネントのデカップリングとは、サーバーコンポーネントができる限り独立して動作できるように設計するという概念です。プレイヤーに優れたゲーム内体験を提供するには、データをできる限り最新に保つ必要があるため、ゲームにはデカップリングすることが困難な部分もあります。ただし、多くのコンポーネントとゲームタスクはデカップリングできます。例えば、リーダーボードと統計サービスはゲームプレイ体験にとって重要ではなく、これらのサービスに対する読み取りと書き込みはゲームとは非同期に実行できます。

検討事項としては、問題が検出された場合に自動的に無効したり管理者が無効にしたりできるゲーム機能の開発方法や、その機能に依存するアップストリームサービスが障害を正常に処理できるように設定することがあります。例えば、特定のプレイヤーデータがゲームクライアント内で正しく読み込まれない場合、そのデータがゲームプレイ体験にとって重要かどうかを検討します。重要でない場合は、プレイヤーのエクスペリエンスを妨げることなく、この障害を正常に処理するようにゲームクライアントを設定し、後でプレイヤーが画面に戻ったときに、このデータの取得を再試行できるようにします。タイムアウト、再試行、バックオフなどのロジックを使用して、エラーや障害を処理します。タイムアウトは、システムが不当に長時間ハングするのを防ぎます。再試行は、一時的エラーやランダムエラーでの高可用性を提供できます。

重要なコンポーネントに疎結合できる重要でないコンポーネントを定義します。疎結合では、あるコンポーネントの障害が他のコンポーネントに及ぶことはないため、システムの耐障害性が向上します。ゲーム機能がゲームサーバーやバックエンドへのステートフル接続を必要としない場合は、ス

テートレスプロトコルを実装し、動的にスケールしたり、一時的な障害からすぐに回復したりするようにします。ステートレスプロトコルと疎結合できる重要でないコンポーネントは、HTTP/JSON API を使用して開発します。また、ゲームクライアントからのネットワーク呼び出しを非同期かつブロック不可に実装して、パフォーマンスの遅いゲーム機能やその他の依存サービスがプレイヤーに与える影響を最小限に抑えるようお勧めします。

疎結合を通じて回復力をさらに向上させるには、キューイング、ストリーミング、トピックベースのシステムなどのメッセージングサービスをコンポーネント間で使用し、非同期で処理できるようにします。このモデルは、即時応答を必要としないインタラクションや、リクエストが登録済みであることを確認するだけのインタラクションに適しています。このソリューションには、イベントを生成するコンポーネントと、イベントを消費するコンポーネントがあります。この 2 つのコンポーネントは、ポイントツーポイントの直接的なインタラクションを介して統合するのではなく、耐久性のあるストレージレイヤーやキューイングレイヤーなどの中間レイヤーを介して統合します。また、処理が失敗した場合でもメッセージを保持できるため、システムの信頼性の向上にも役立ちます。メッセージングサービスごとに、順序付けや配信のメカニズムなどの特性が異なるため、適切なメッセージングサービスを選択することが重要です。オペレーションがべき等性を持つように設計し、選択したメッセージングシステムが少なくとも 1 回はメッセージを配信するようにします。例えば、一般的なゲームユースケースでプレイヤーのプレイ時間、統計、その他の関連データを追跡する必要がある場合、プレイヤーの同時実行数がピークに達すると、このユースケースは書き込みスループットの高いユースケースになる場合があります。

信頼性の高いアーキテクチャを実装するには、ユースケースにおいてリードアフターライトの一貫性がプレイヤーが認識するレベルで必要かどうかを検討します。通常、このようなシナリオは非同期処理に適しており、書き込みキューイングパターンを実装することで達成できます。その場合、リクエストは Amazon SQS などのスケーラブルで耐久性のあるメッセージキューに取り込まれ、Lambda 関数などのコンシューマーサービスを使用してバックエンドデータベースに一括挿入できます。このアプローチは、プレイヤーのゲームクライアント、バックエンドのウェブサーバーとアプリケーションサーバー、内部データベースシステムなどの複数の分散コンポーネント間の同期通信よりも、信頼性が高くなります。また、書き込みキューからのコンシューマー処理を使用して、この取り込み速度を必要に応じて遅らせることができるため、ピーク時の書き込みスループットに合わせてバックエンドデータベースをスケールする必要がなく、コストも削減されます。

詳細については、以下のドキュメントを参照してください。

- Build highly scalable and reliable workloads using microservice architecture (マイクロサービスアーキテクチャを使用して、スケーラブルで信頼性の高いワークロードを構築する)
- AWS サーバーレスサービスを使用してマイクロサービスを統合する
- Asynchronous messaging for microservices (マイクロサービス向けの非同期メッセージング)

- Introduction to Scalable Game Development Patterns on AWS (AWS でのスケーラブルなゲーム開発パターンの紹介)

GAMEREL_BP04 - インフラストラクチャの障害を長期にわたってモニタリングし、プレイヤーの行動への影響を測定します。

ゲームサーバープロセスとゲームサーバーインスタンスのメトリクスをモニタリングして、問題の根本原因を特定します。CPU とメモリのモニタリングに加えて、EC2 インスタンスのネットワーク制限 に関連するネットワークメトリクスのモニタリングをセットアップし、帯域幅の超過、1 秒あたりのパケット数、その他のネットワークレベルの問題など、サーバーリソースのプロビジョニング不足を示す問題についてアラートを受け取ることができます。Amazon GameLift を使用してホストするゲームサーバーについては、メトリクス として `GameServerInterruptions` や `InstanceInterruptions` などをモニタリングすることを検討します。これらのメトリクスは、スポットインスタンスの可用性の制限が、スポットを使用してデプロイしたゲームサーバーにどのような影響を与えているかを理解するのに役立ちます。また、`ServerProcessAbnormalTerminations` を使用してゲームサーバープロセスの異常終了を検出できます。

ゲームサーバーの信頼性の履歴メトリクスデータを保持することをお勧めします。この履歴データをレポート目的で使用し、他のデータセットと結合して、ゲームサーバーの問題に起因する可能性がある傾向を明らかにし、プレイヤーの行動への影響を長期にわたって評価します。Amazon CloudWatch はメトリクスを無期限に保持せず、メトリクスのストレージ解像度 は時間の経過とともに増加するため、これらのメトリクスを Amazon S3 などの費用対効果の高い長期ストレージにエクスポートする必要があります。メトリクスを CloudWatch から独自の S3 バケットに自動的に配信するように CloudWatch メトリクスストリーム を設定し、S3 Intelligent-Tiering などのストレージ層に長期保存して、最終的に Amazon S3 Glacier を使用してアーカイブできます。メトリクスを Amazon S3 に配置することで、データレイク内の他のデータセットと簡単に結合でき、Amazon Athena でインタラクティブなクエリを実行できます。

詳細については、以下のドキュメントを参照してください。

- Use Amazon EC2 instance-level network performance metrics (Amazon EC2 インスタンスレベルのネットワークパフォーマンスメトリクスを使用する)
- CloudWatch メトリクスストリーム

GAMEREL_BP05: 各ゲームサーバーインスタンスでホストするゲームセッションの数を調整して、影響範囲を縮小します。

ゲームサーバーでインフラストラクチャやソフトウェアの問題が発生した場合に、影響を受けるプレイヤー数のリスク許容度を決めます。この情報を使用して、ゲームサーバーインスタンスごとに問題なくホストできるゲームセッションの最大数を判断できます。

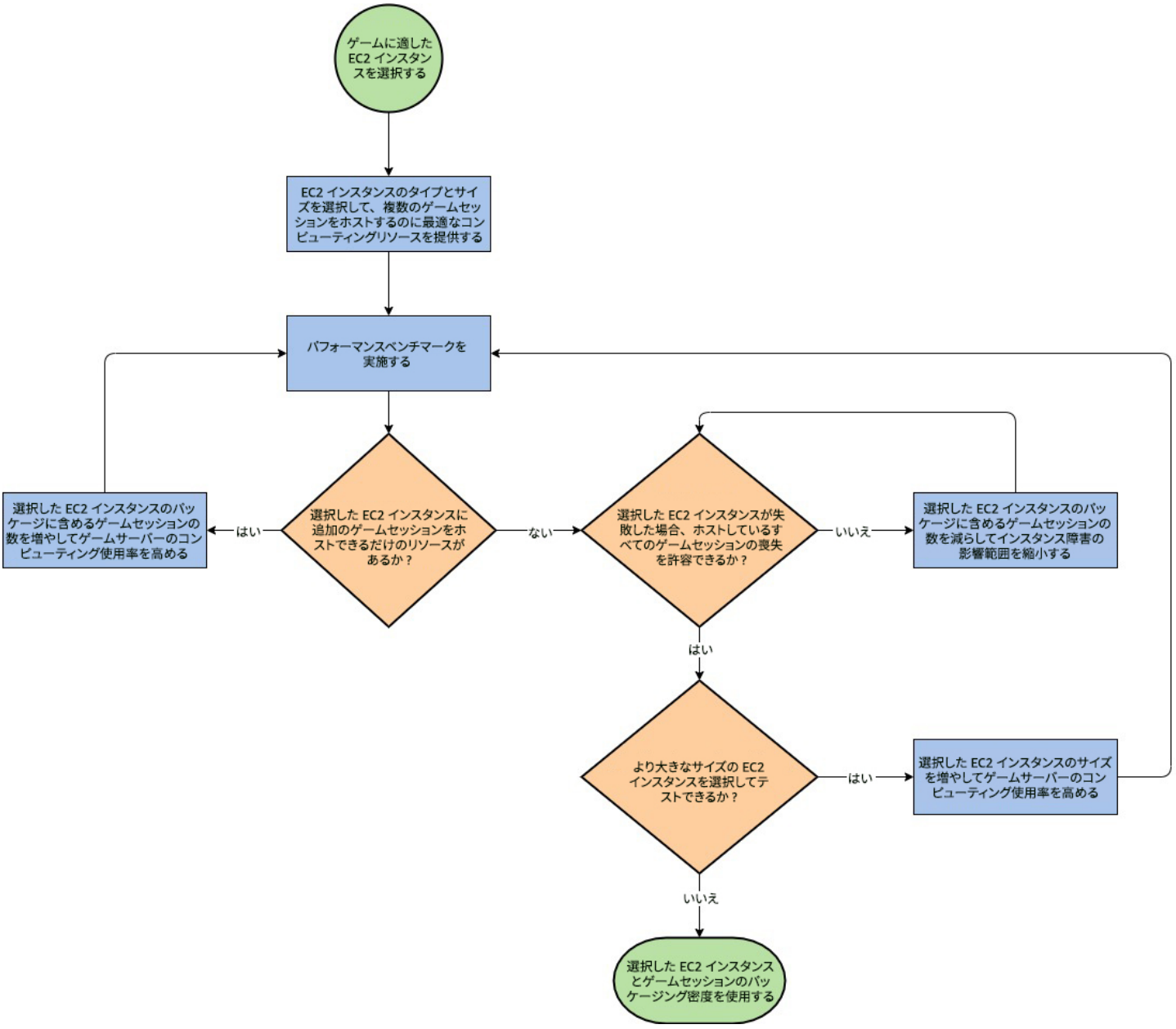
サーバーインスタンスでホストするゲームセッションの数を決定する場合、通常、ゲームデベロッパーは初めにコストに注目します。ただし、ゲームアーキテクチャを設計する際にはこの決定が信頼性に与える影響を考慮することが重要です。ゲームサーバーごとにホストするゲームセッションの密度を高めて、インフラストラクチャの利用率を高めることは重要ですが、1 つのゲームサーバーインスタンスで障害が発生した場合の影響範囲を許容できるか判断する必要があります。1 つのインスタンスで障害が発生すると、そのインスタンスでアクティブなゲームセッションをホストしているすべてのゲームサーバープロセスを失うことになるため、この障害に伴うプレイヤーの中断数は要件が許容する範囲に留める必要があります。

バトルロワイヤルやその他の MMO スタイルのゲームなど、ゲームに大きなマッチサイズが必要な場合、1 回のゲームセッションでホストできるプレイヤーの数は制限されることがあります。この制限はゲームのスタイルに伴うものであり、インフラストラクチャの決定要件ではなく、ゲーム設計やプレイヤーエクスペリエンスの要件とより密接に関連しています。EC2 インスタンスのコストは、通常、特定のインスタンスタイプ内でサイズを大きくした場合 (`2xlarge` から `4xlarge` に移行するなど) に、直線的に増大することを覚えておく必要があります。

したがって、信頼性を高め、1 つのゲームサーバーインスタンスの障害に伴う影響を軽減するには、ゲームセッションをホストするゲームサーバーインスタンスの数を増やすことを検討します。例えば、50 件のゲームセッションを `4xlarge` EC2 インスタンスでホストする代わりに、この 50 件のゲームセッションを 2 つの `2xlarge` EC2 インスタンスに均等に分割し、各インスタンスで 25 件のゲームセッションをホストすると、インスタンスで障害が発生した場合に影響を受けるプレイヤーの数を減らすことができます。

通常、この 2 つのデプロイアーキテクチャのコンピューティングコストは同じですが、コストが同じではない場合も、ゲームサーバーのホスティング戦略でこの種の変更を行うことでゲームサーバーの障害に伴うプレイヤーへの影響がどのように変わるかを考慮することが重要です。また、このアプローチで注意すべき重要な点として、ゲームセッションをホストする各ゲームサーバープロセスが使用するリソースは、この方法で均等に分割できるように一定量がゲームサーバーに事前に割り当てられていることを前提としています。

次の図は、ゲームサーバーインスタンスごとにホストするゲームセッションの数を決定する場合に役立つプロセスの例を示しています。



ゲームサーバーインスタンスごとにホストするゲームセッションの数を決定する方法の例

GAMEREL_BP06: ゲームインフラストラクチャを複数のアベイラビリティーゾーンとリージョンに分散して回復力を高める

インフラストラクチャの局所的な障害がプレイヤーに与える影響を最小限に抑えるには、インフラストラクチャのデプロイを必要に応じて複数の独立した場所に均等に分散し、予期しない障害に耐えられるようにするとともに、引き続き十分な容量を確保してプレイヤーの需要を満たす必要があります。

ゲームインフラストラクチャをデプロイする場合は、1 つのリージョン内の複数のアベイラビリティーゾーンに容量を均等に分散し、1 つまたは複数のアベイラビリティーゾーンが中断しても耐えられるようにして、円滑なプレイヤーエクスペリエンスを確保することをお勧めします。ウェブアプリケーションなどのゲームバックエンドサービスは、複数のアベイラビリティーゾーン間でロードバランスするか、AWS Lambda や Amazon API Gateway などのマネージドサービスを使用して構築し、設計でリージョンの高可用性を実現する必要があります。同様に、キャッシュ、データベース、メッセージキュー、ストレージソリューションなどの状態を維持するすべてのコンポーネントは、複数のアベイラビリティーゾーンにわたってデータの強固な永続性を提供するように設計する必要があります。これは Amazon S3、DynamoDB、Amazon SQS などのサービスでは仕様で提供されています。他のサービスでも設定できます。

回復力を考慮してゲームサーバーホスティングアーキテクチャを設計する場合は、ゲームサーバーのフリートを AWS リージョン内のすべてのアベイラビリティゾーンに均一にデプロイすることで、リージョン内の利用可能なすべてのコンピューティング性能へのアクセスを最大化するとともに、アベイラビリティーゾーンの障害の影響範囲を縮小する必要があります。例えば、すべてのアベイラビリティーゾーンを使用するように Amazon EC2 Auto Scaling を設定できます。EC2 インスタンスで異常が発生した場合、EC2 Auto Scaling はこのインスタンスを置き換えることができます。また、1 つまたは複数のアベイラビリティーゾーンが使用できなくなった場合、インスタンスを他のアベイラビリティーゾーンで起動できます。

ベストプラクティスとして、ゲームインフラストラクチャは複数のリージョンにデプロイし、高可用性を最大化します。これは、ゲームバックエンドサービスで高可用性を実現するための推奨事項ですが、特にゲームサーバーの場合にお勧めします。例えば、マルチプレイヤーゲームでは、ゲームサーバーを使用してプレイヤーとの長期のゲームセッションをホストするため、ゲームサーバーのインフラストラクチャ容量は他のサービスの容量ニーズを上回る場合があります。多くのゲームでは、北米東部、北米西部、欧州、アジアパシフィックなどの論理的なゲームリージョンにプレイヤーを分割するようになっています。プレイヤーエクスペリエンスを簡素化し、グローバルなインフラストラクチャを利用してゲームをホストしやすくするために、プレイヤー向けのゲームリージョンの名前 (「北米西部」など) を、ゲームサーバーを物理的にホストしているクラウドプロバイダーのリージョンやデータセンターの所在地から切り離すことを検討する必要があります。これには、オレゴン (us-west-2) リージョンや北カリフォルニア (us-west-1) リージョンに加えて、該当するプレイヤーゲームリージョンをサポートするゲームサーバーインスタンスをホストしている ローカルゾーン や社内データセンターなど、その他のインフラストラクチャが含まれる場合があります。

マッチメイキングサービスを設計する場合は、マルチリージョンアーキテクチャを導入し、各リージョンにソフトウェアを個別にデプロイする必要があります。マッチメイキングサービスのデプロイを、ゲームサーバーインスタンスをホストするフリートから切り離すことで、どのリージョンにデプロイしたマッチメイキングサービスがマッチメイキングリクエストを処理したかに関係なく、プレイ

ヤーを任意のリージョンのゲームサーバーにルーティングできます。レイテンシーやその他のルールに合致するゲームサーバーリージョンを優先するように、マッチメイキング実装のロジックを設計します。フリートの容量が少ない場合や、他のリージョンのインフラストラクチャが中断した場合は、プレイヤーを他のリージョンにルーティングするようにフォールバック機能を備えます。

詳細については、以下のドキュメントを参照してください。

- Amazon GameLift ゲームセッションキューのベストプラクティス
- GameLift Multi-Region fleets (Amazon GameLift マルチリージョンフリート)

リソース

信頼性に関するベストプラクティスの詳細については、以下のリソースを参照してください。

ドキュメントとブログ

- Practicing Continuous Integration and Continuous Delivery on AWS (AWS での継続的インテグレーションと継続的デリバリーの実践)
- Autoscaling Asynchronous Job Queues (非同期ジョブキューのオートスケーリング)
- お客様のワークロードサービスアーキテクチャを設計する
- ジッターを伴うタイムアウト、再試行、およびバックオフ
- 信頼性の柱 - AWS Well-Architected Framework
- Architecting for Reliable Scalability (信頼性の高いスケーラビリティを実現するアーキテクチャの設計)
- Amazon Builder's Library
- Massive Scale Real-Time Messaging for Multiplayer Games (マルチプレイヤーゲームのための大規模なリアルタイムメッセージング)

ホワイトペーパー

- Introduction to Scalable Game Development Patterns on AWS (AWS でのスケーラブルなゲーム開発パターンの紹介)
- Running Containerized Microservices on AWS (AWS でコンテナ化されたマイクロサービスを実行)
- Web Application Hosting in the AWS Cloud (クラウド内でのウェブアプリケーションのホスティング)

- Building a Scalable and Secure Multi-VPC AWS Network Infrastructure (スケーラブルで安全なマルチ VPC ネットワークインフラストラクチャの構築)

動画コンテンツ

- re:Invent 2020: Ubisoft: Building a multi-platform multiplayer game on AWS (Ubisoft - AWS でマルチプラットフォームのマルチプレイヤーゲームを構築する)
- re:Invent 2018: Supercell – Scaling Mobile Games (Supercell - モバイルゲームのスケーリング)
- re:Invent 2019: How CAPCOM builds fun games fast with containers, data, and ML (カプコンがコンテナ、データ、機械学習を使用して楽しいゲームを高速で構築している方法)
- re:Invent 2018: Globalizing Player Accounts at Riot Games While Maintaining Availability (Riot Games が可用性を維持しながらプレイヤーアカウントをグローバル化)
- re:Invent 2020: Gameloft: A zero downtime data lake migration deep dive (GameLoft - ダウンタイムゼロのデータレイク移行の詳細)

トレーニング資料

- Using Amazon GameLift FleetIQ for Game Servers (ゲームサーバーでの Amazon GameLift FleetIQ の使用)
- Game Server Hosting with Amazon EC2 (Amazon EC2 でのゲームサーバーホスティング)

パフォーマンス効率

パフォーマンス効率の柱では、コンピューティングリソースを効率的に使ってシステム要件を満たし、需要の変動と技術の進化に合わせて効率性を維持することに重点を置きます。

設計原則

AWS Well-Architected Framework のホワイトペーパーに記載している設計原則に加えて、以下の設計原則がゲームのパフォーマンス効率の達成に役立ちます。

ゲームクライアント、インターネット、ゲームインフラストラクチャなど、ゲームのパフォーマンスをエンドツーエンドで測定する: プレイヤーの視点から見たパフォーマンスを測定することが重要です。つまり、ゲームクライアント、ゲームインフラストラクチャ、インターネット接続 (プレイヤーをインフラストラクチャに接続する) のパフォーマンスを測定する必要があります。これにより、アーキテクチャのどのレベルでも、パフォーマンスを改善できる箇所を把握できます。

定義

クラウドのパフォーマンス効率を向上させるには、次の 4 つのベストプラクティス分野があります。

- 選択
- レビュー
- モニタリング
- トレードオフ

高性能なアーキテクチャの選択には、データ駆動型のアプローチを取ります。ハイレベルな設計から、リソースタイプの選択と設定に至るまで、アーキテクチャのあらゆる側面に関するデータを収集してください。

選択した内容を定期的にレビューすることで、絶えず進化し続けているプラットフォームを活用していることを確認できます。モニタリングを実施すれば、予想したパフォーマンスからのずれを把握し、その対策を講じることができます。さらに、圧縮やキャッシュを使用したり、整合性に関する要件を緩和したりするなど、アーキテクチャにおけるトレードオフを行ってパフォーマンスを向上させることができます。

ベストプラクティス

クラウドアーキテクチャのベストプラクティスを以下に示します。

トピック

選択

GAMEPERF01 – ゲームインフラストラクチャをホストする地理的リージョンは、どのようにして決定しますか?

GAMEPERF_BP01 – プレイヤーとビジネスステークホルダーからのフィードバックを確認します。

ゲームを最初に発売するときは、ビジネスステークホルダーとの話し合いに基づいて、インフラストラクチャをどこにデプロイするかを決定する必要があります。例えば、パブリッシングチームと話し合い、プレイヤーにゲームをリリースしたい場所や、リリース前のマーケティングと広告を集中的に行う場所を決定します。

また、ビジネスステークホルダーは、需要を刺激し、プレイヤーの受け入れや実現性の理解を深めるために役立つメカニズムを持っているはずです。例えば、これらのチームが用意するメカニズムには、ゲームの予約注文、マーケティングイベントやキャンペーン、関心を寄せているプレイヤーを発売前に登録するための公開メーリングリストのほか、発売時にゲームのプレイヤーが最も多いと思われる場所を判断するための関連シグナルを確立するアプローチなどがあります。ゲームでは、事前に決めたリージョンロールアウト戦略を使用する場合もあります。この戦略を使用してテストやソフトローンチを行い、リージョンのプレイヤーの需要をより詳しく判断できます。

GAMEPERF_BP02 - レイテンシーの影響を受けるゲームインフラストラクチャをプレイヤーの近くに配置してパフォーマンスを向上させるアプローチを設計します。

ゲームを最初に発売するときは、プレイヤーベースに関する十分な情報がないため、ゲームのプレイに関心が高いプレイヤーに最も近い場所としてどこにインフラストラクチャをデプロイすれよいかを適切に判断できない場合があります。これは一般的な課題であり、このような場合に備えて、ホスティング戦略を迅速に調整してサーバーのデプロイ先をプレイヤーに近づけることができるようにアーキテクチャを設計する必要があります。ゲームデベロッパーは、通常、ゲーム発売後にゲームインフラストラクチャのデプロイを定期的に評価し、反復的なアプローチを使用しての長期的な改善に向けて段階的に投資します。

ベストプラクティスは、VPC、サブセット設定、重要なゲームサービスのリリースに必要な依存関係など、インフラストラクチャの設定に AWS CloudFormation や Terraform などの Infrastructure as Code テンプレートを使用することです。これらのテンプレートを参照し、必要に応じてすばやくカスタマイズして、プレイヤーをサポートするために追加のインフラストラクチャが必要な場所にデプロイします。

また、現在のデプロイ戦略をどのように進化させれば将来の拡張につながるかについても理解しておく必要があります。例えば、ゲームサーバーをホストするためのサブネットを作成する場合は、そのサイズを考慮し、確実に成長に対応できる大きさにします。そのほか、複数の場所にデプロイしたゲームサーバーをゲームバックエンドに接続する方法も考慮する必要があります。ゲームバックエンドは、中央の場所または複数の場所でホストしている場合があり、プライベート接続をサポートするには追加の設定が必要になることがあります。これらの考慮事項は長期にわたって継続的に評価し、

ゲームの要件が時間の経過とともに進化したり、プレイヤーの要件が変化したりしたときにはゲームのホスティング戦略を変更できるようにする必要があります。

ゲームにいくつのゲームホスティング場所を使用するか決定する場合は、以下の要因を考慮する必要があります。

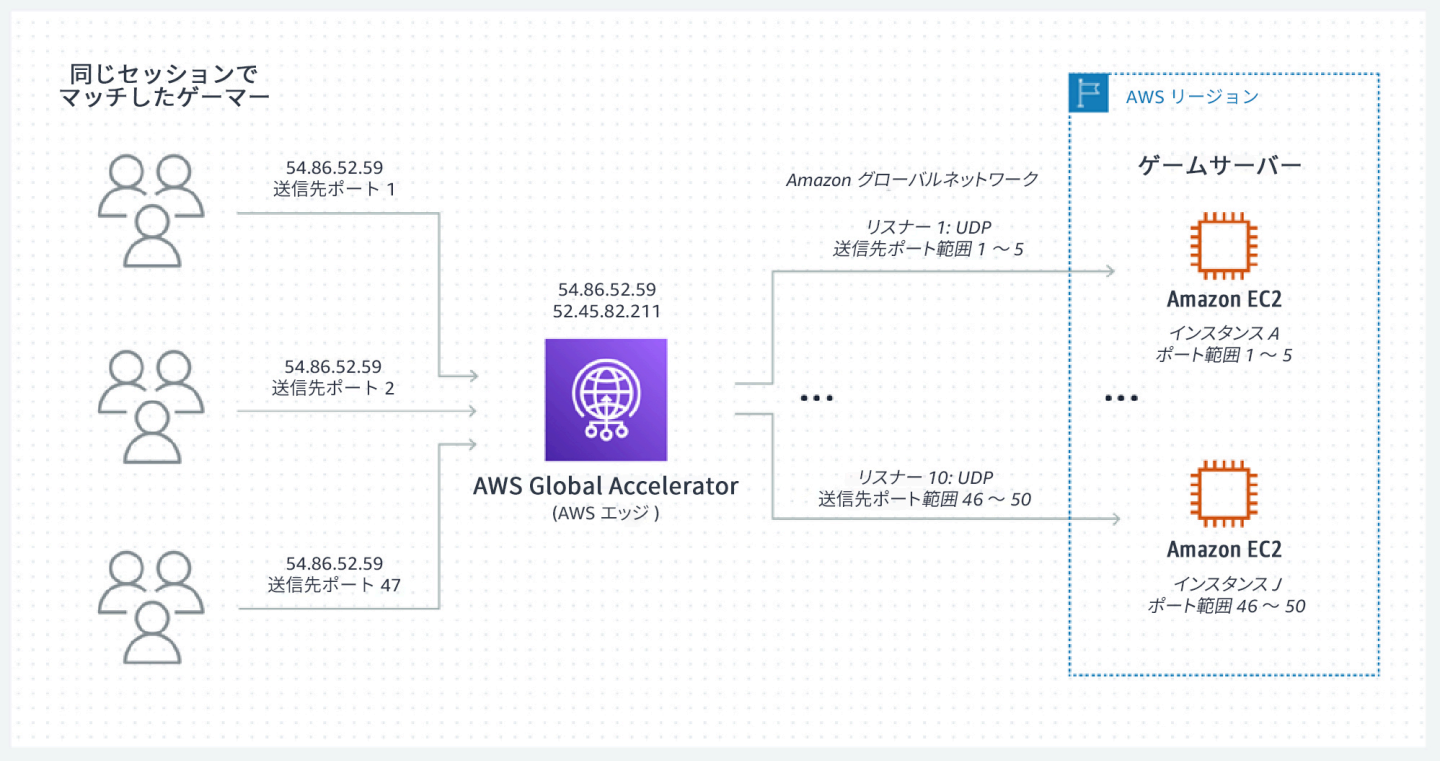
- プレイヤーエクスペリエンスの質の向上: ゲームのホスティング場所を増やすことで、プレイヤーエクスペリエンスをどの程度改善できるか。 その結果、パフォーマンスがどれくらい向上するか。 このパフォーマンスの向上をどのようにして測定するか。
- 優先させるプレイヤー集団: ゲームのホスティング場所を増やした場合、どれくらいの数のプレイヤーのエクスペリエンスを改善できるか。 どのプレイヤー集団または地理的な場所を優先するか。
- 変更によるダウンストリームへの影響: ゲームホスティング戦略を変更した場合、プレイヤーのマッチメイキングの待ち時間にどのように影響するか。 変更を導入した場合、ゲームセッションを構成するために必要なマッチサイズやプレイヤー数は、世界各地で大規模なプレイヤー集団を構築する能力に影響するか。

ゲームホスティングの場所をどこで追加または削除するかを決定する場合は、これらの各考慮事項を評価する必要があります。例えば、ゲームプレイ体験のパフォーマンスが最も低い地域のプレイヤーや、最も熱心なフィードバックを一般に公開したり、コミュニティ管理チームに伝えたりしているプレイヤーのエクスペリエンスを優先的に改善することを選択できます。また、プレイヤーの収益化を優先させることもできます。例えば、ゲームの大きな収益源である地域や、パフォーマンスを改善することで収益増が見込める地域のプレイヤーのエクスペリエンスを改善することに集中的に取り組みます。

AWS リージョンでのインフラストラクチャのホスティングに加えて、 ローカルゾーン (AWS リージョンの拡張) を使用すると、プレイヤーにより近い場所でゲームサーバーやその他のレイテンシーの影響を受けやすいアプリケーション (ボイスチャットサーバーなど) をホストできます。また、ゲーム開発インフラストラクチャをローカルゾーンで実行して、ゲーム開発チームのエクスペリエンスを改善することもできます。例えば、ローカルゾーンを使用して対応できるユースケースとして、ゲームデベロッパーにより近い場所でセルフマネージドのソース管理サーバーのレプリカをホストすることができます。または、オンプレミスでインフラストラクチャをホストせずに、開発スタジオに近い 1 つ以上のローカルゾーンにデプロイした Amazon EC2 インスタンス、EBS ボリューム、Amazon FSx ファイルシステムを使用して、ゲーム開発の仮想ワークステーションやコンテンツストレージをユーザーに提供できます。

また、既存のオンプレミスデータセンターやコロケーション施設に機能を拡張することもできます。そのためには、 Outposts を使用します。これはフルマネージドサービスであり、専用のラックとラックマウント可能なサーバーオプションを使用して、同じサービスと API へのアクセスを提供します。これにより、リージョン、ローカルゾーン、施設にデプロイした Outposts のすべてにわたって一貫したデプロイモデルを作成できます。ゲームの構築にコンテナを使用していて、自社のインフラストラクチャにデプロイできるオープンソースソフトウェアを使用したハイブリッドデプロイアーキテクチャを柔軟に採用したい場合は、 ECS Anywhere を使用できます。または、Kubernetes ベースのインフラストラクチャを運用したい場合は、 EKS Anywhere を使用できます。

GAMEPERF_BP03 - ネットワークアクセラレーションテクノロジーを使用して、インターネット全体のパフォーマンスを向上させます。



Global Accelerator を使用してゲームのために改善したネットワークパフォーマンス

プレイヤーエクスペリエンスを改善するには、レイテンシーの影響を受けやすいゲームインフラストラクチャをプレイヤーにより近い物理的な場所に配置するだけでなく、ゲームのネットワークパフォーマンスを最適化する方法もあります。プレイヤーがゲーム接続時に利用するネットワークやインターネットサービスプロバイダー (ISP) とゲームインフラストラクチャの接続を向上させるテクノロジーを使用します。ネットワークアクセラレーションでは、ネットワークパスを最適化してパフォーマンスを向上させます。最適化後のネットワークパスを通じて、プレイヤートラフィックをゲームクライアントからインターネット経由でゲームサーバーやゲームバックエンドサービスなどのゲームインフラストラクチャにルーティングします。例えば、AWS Global Accelerator は、

グローバルネットワークを使用してアプリケーションのネットワークパフォーマンスを向上させるネットワークサービスです。ゲームプレイのトラフィック、ボイスチャット、リアルタイムメッセージングトラフィック、その他のレイテンシーの影響を受けやすいアプリケーションを高速化できます。Global Accelerator カスタムルーティングアクセラレーター をマッチメイキングサービスと統合して、複数のプレイヤーを同じゲームセッションの IP アドレスとポートに決定的な方法でルーティングできます。

ゲーム開発チームは世界中に分散している場合があり、共有のコンテンツやアセットに効率的にアクセスする必要があります。Amazon S3 バケットに保存した共有コンテンツのパフォーマンスを向上させるには、 S3 クロスリージョンレプリケーション を使用してリージョン間でのデータの双方向レプリケーションをセットアップし、ユーザーが自分に近いバケットからデータにアクセスできるようにします。このアクセスパターンを簡素化するには、 S3 マルチリージョンアクセスポイント を使用します。これにより、Global Accelerator を使用してグローバルネットワーク全体で S3 へのリクエストを高速化します。詳細については、「 Improving the Player Experience by Leveraging AWS Global Accelerator and Amazon GameLift FleetIQ (Global Accelerator と Amazon GameLift FleetIQ を活用してプレイヤーエクスペリエンスを向上させる)」を参照してください。

GAMEPERF02 – ゲームセッションでの過剰なリソース使用が、同じゲームサーバーインスタンスで実行している他のプレイヤーに影響を与えないようにするには、どうすればよいですか?

GAMEPERF_BP04 - ゲームサーバーのプロセスをモニタリングして問題を検出します。

ゲームサーバーインスタンスでリソースを効率的に使用するには、インスタンスごとに複数のゲームサーバープロセスを実行できます。この場合は、ゲームセッションをホストしている各ゲームサーバープロセスが、同じゲームサーバーインスタンスでホストしている他のゲームセッションに悪影響を与えないようにアーキテクチャを設計する必要があります。

ゲームサーバーインスタンスで使用できる限られたリソースをモニタリングして、各ゲームサーバープロセスが所定のリソース割当量のしきい値を超えたときにアラートを受信できるようにします。しきい値を超えた場合は、関連するシステムやゲームサーバーのログを、中央ログソリューションなどの耐久性のあるストレージにダンプするようにゲームサーバーソフトウェアを設定し、ゲームサーバーエンジニアがこの動作の原因を調査できるようにします。また、ゲームサーバーインスタンスで実行している各ゲームサーバープロセスのメトリクスを報告するようにインスタンスを設定し、インスタンスの全体的なメトリクスに加えて各ゲームサーバープロセスをモニタリングできるようにする必要があります。例えば、Amazon GameLift は ゲームセッションをモニタリング するためのメトリクスを提供します。このメトリクスは、ゲームサーバーインスタンスに設定できる Amazon

CloudWatch Agent を使用して収集したゲーム固有のカスタムメトリクスとログで拡張できます。これらのメトリクスは、CloudWatch で表示できます。または、Single Sign-On を使用して統合した Amazon Managed Grafana などの他のツールにエクスポートして、マネジメントコンソールにアクセスできないユーザーがメトリクスに簡単にアクセスできるようにすることができます。詳細については、 Amazon GameLift を使用してログとメトリクスを管理するための以下のベストプラクティスを参照してください。Amazon GameLift では、各 ゲームセッションのログを表示することもできます。

GAMEPERF_BP05 - 実際のゲームプレイシナリオをシミュレートして、ゲームサーバーのパフォーマンスをテストします。

パフォーマンステストを実施し、さまざまなゲームプレイシナリオを評価して、ゲームサーバープロセスが EC2 インスタンスメモリ、CPU、ネットワーク帯域幅などの固定リソースの使用を適切に処理しているかどうかを判断する必要があります。

プレイヤーの一般的なゲームプレイのパスと動作をミラーリングできるボットを使用してゲームプレイをシミュレートするテストを作成し、ゲームサーバープロセスがさまざまな使用状況下でどのように処理するかを判断できるようにします。例えば、 AWS での分散負荷テスト などのソリューションを実装してカスタマイズし、ゲームクライアントのシミュレーションやゲームクライアントのビルドを実行してゲームプレイシナリオを生成できるようにします。社内でのプレイテストや、QA チームによるゲームのさまざまな機能のストレステストを行うことで、最適に動作するようにゲームが設計されていることに確信が持てます。AWS Device Farm を使用すると、iOS、Android、プラウザのゲームのモバイルおよびウェブテストを実行できます。

GAMEPERF03 - ゲームに適したコンピューティングソリューションは、どのようにして選択しますか?

GAMEPERF_BP06 - さまざまなコンピューティングタイプにわたってゲームパフォーマンスをベンチマークします。

ゲームサーバーのワークロードに関しては、ゲームサーバーをホストするための最適なコンピューティングソリューションを特定するための画一的なアプローチはありません。通常、ゲームサーバーにはコンピューティングに最適化された EC2 インスタンスを使用します。このインスタンスファミリーは、計算負荷の高いゲームサーバーなどのワークロード用に最適化されています。または、ゲームで特定の機能を実装するために大量のメモリが必要な場合は、メモリに最適化されたインスタンスが最適である場合もあります。

ワークロードが大量のネットワークリソースを使用するユースケースでは、ネットワークに最適化されたインスタンスを実装することを検討してください。ネットワークに最適化されたインスタンスは、通常、インスタンス名が「n」で始まります。ゲームは、レイテンシーや破棄されたパケットの影響を受けやすいため、EC2 拡張ネットワークを使用してゲームサーバーのネットワークパフォーマンスを向上させることをお勧めします。拡張ネットワークは、シングルルート I/O 仮想化 (SR-IOV) を使用して、 サポートしているインスタンスタイプ に対して高パフォーマンスのネットワーク機能を提供します。SR-IOV は、従来の仮想化ネットワークインターフェイスと比べて I/O パフォーマンスが高く、CPU 利用率が低いデバイス仮想化の手法です。拡張ネットワークは、高い帯域幅、1 秒あたりのパケット (PPS) の高いパフォーマンス、常に低いインスタンス間レイテンシーを実現します。Elastic Network Adapter による拡張ネットワークは、最新の EC2 インスタンスタイプで利用できます。

ゲームが複数の EC2 インスタンスタイプで同じように動作する場合は、複数のインスタンスタイプを使用してゲームサーバーをホストすることを検討します。これにより、経時的なパフォーマンスをモニタリングし、経時的なパフォーマンス傾向を特定できるだけの十分な本番ゲームセッションをホストした後で、さらに最適化を実行できます。ゲームに新しい機能を追加したことに伴って異なるリソース割り当てが必要になる場合、時間の経過とともにリソース要件が変わる場合があります。複数のインスタンスタイプを使用するように EC2 Auto Scaling グループ を設定できます。または、個別の Auto Scaling グループを使用して個別のインスタンスタイプを実行するゲームサーバーインスタンスをホストし、メトリクスの相関と集約を管理しやすくすることができます。

また、インテルベースのインスタンス、AMD ベースのインスタンス、ARM ベースの Graviton インスタンスなど、異なる種類のプロセッサごとにゲームがどのように動作するかを評価する必要があります。

さらに、コンテナと Lambda 関数を使用してゲームをホストした場合に、ゲームのパフォーマンスにどのような影響があるかをベンチマークする必要があります。非同期ゲームやゲームバックエンドサービスなど、ゲームサーバーによる存続期間が長い処理が不要なユースケースでは、Lambdaでサーバーレスアーキテクチャを使用することを検討します。これにより、ゲームオペレーションチームのために管理とオペレーションを簡素化し、ゲームをより迅速に多くの AWS リージョンにグローバルにデプロイできます。サーバーレスのベストプラクティスについては、「 サーバーレスアプリケーションレンズ - Well-Architected Framework」 を参照してください。詳細については、「 グローバルゲームサーバーに適したコンピューティング戦略を選択する」を参照してください。

GAMEPERF_BP07 - ゲーム開発向けの仮想ワークステーションでは、グラフィックスインスタンスを使用します。

ゲームデザイナー、エンジニア、アーティスト、QA、その他の担当者は、仮想ワークステーションを使用することが必要になる場合があります。これらのユースケースをサポートするには、グラ

フィックスに最適化されたインスタンスを使用します。この種のインスタンスは、インスタンス名が「g」で始まり、ゲーム開発やゲームストリーミングなどのグラフィックスのユースケースをサポートするための専用の GPU を使用して構築します。

エンドユーザーが通常必要とするのと同じツールを使用して、さまざまなグラフィックス最適化インスタンスタイプにわたってパフォーマンスを評価およびベンチマークします。例えば、AMD や NVIDIA など、メーカーの異なる GPU を搭載したさまざまなグラフィックス最適化インスタンスを使用します。これらのインスタンスをベンチマークする場合は、使用するソフトウェアが、サポートされている GPU および関連ドライバーと互換性があることを確認してください。グラフィカルアーティストのユースケースでは、 Amazon Nimble Studio を使用することを検討します。これにより、クラウドベースのスタジオを運用するために必要なクラウドベースの仮想ワークステーション、ファイルストレージやツールにアクセスできます。

EC2 を使用して独自のカスタム仮想ワークステーションを開発する場合は、これらの仮想ワークステーションにエンドユーザーがアクセスする方法を検討する必要があります。さまざまな接続オプションがありますが、 NICE DCV を使用することを検討してください。これは、ローカルにインストールしたクライアントソフトウェアやウェブブラウザからリモートデスクトップに接続するための高性能リモートディスプレイプロトコルであり、 AWS Marketplace から NICE DCV サーバーがプリインストールされた Amazon マシンイメージ (AMI) として無償で入手できます。

詳細については、以下を参照してください。

- Virtual workstations on AWS powered by NVIDIA Quadro technology (NVIDIA Quadro テクノロジーを搭載した仮想ワークステーション)
- Game Production in the Cloud - Workstations: Stream Game Development from anywhere with NICE DCV (クラウド内でのゲーム制作 - ワークステーション: NICE DCV を使用してどこからでもゲーム開発をストリーミング)
- Stream a remote environment with NICE DCV over QUIC UDP for a 4K monitor at 60 FPS (4K モニター 60 FPS で QUIC UDP 経由の NICE DCV を使用してリモート環境をストリーミング)
- Putting bitrates into perspective (大局的な視点でビットレートを考える)

GAMEPERF_BP08 - レイテンシーの影響を受けないコンピューティングタスクを非同期ワークフローにプッシュします。

ゲームのパフォーマンスを最適化する場合、クライアントとゲームバックエンドとの間のインタラクションを必ずしも同期的に実行する必要はないことを覚えておくことが重要です。プレイヤーエクスペリエンスの観点から各機能を検討し、特定のインタラクションで同期通信 (ブロッキング型で必要でリソースを集中的に消費する) が必要かどうか、各機能を非同期的に実装できるかどうかを判断す

る必要があります。ネットワーク呼び出しを実装する場合は、非同期のノンブロッキング型アプローチを使用するようにします。さらに、タスクをキューにオフロードし、できる限りクライアントへの高速応答を優先することで、作業を効率的に実行するようにゲームバックエンドを設定する必要があります。

例えば、リーダーボードの更新は、プレイヤーセッションの終了時に非同期で実行できるため、クライアントはリーダーボードの更新が完了するのを待機する必要はありません。代わりに、これをゲームクライアントで非同期に実行します。また、この種のオペレーションは Amazon SQS などのキューにプッシュするようにバックエンドサービスを設計することを検討してください。このアーキテクチャでは、リクエストを受け入れ、SQS (非同期処理のためにメッセージを永続的に保存する) のキューにリクエストを追加し、クライアントに迅速に返信するように、バックエンドを設定する必要があります。リーダーボードの更新が完了すると、バックエンドはゲームクライアントに更新を送信して、プレイヤーのリーダーボードのビューを更新します。または、プレイヤーはゲームのリーダーボード画面にアクセスするだけで最新データを取得できます。アクセスに伴って、バックエンドにウェブリクエストが発行され、キャッシュから最新データが取得されます。

詳細については、以下のドキュメントを参照してください。

- Understanding asynchronous messaging for microservices (マイクロサービスの非同期メッセージングについて)
- Using service integrations and asynchronous processing (Lambda - サービス統合と非同期処理の使用)

レビュー

ゲームのワークロードに適用する パフォーマンス効率 のベストプラクティスについては、Well-Architected Framework のホワイトペーパーを参照してください。

モニタリング

GAMEPERF04 – パフォーマンスを最適化するには、マッチメイキングサービスをどのように設計しますか?

GAMEPERF_BP09 - ゲームのネットワークレイテンシーのしきい値を定義します。

マルチプレイヤーゲームを開発する場合は、ゲームインフラストラクチャがプレイヤーに不要なレイテンシーを与えないようにします。ゲームがネットワークレイテンシーの影響を受けやすい場合は、

マッチメイキングのロジックでレイテンシーのしきい値を設定し、このしきい値以内でゲームセッションに接続すると理想的なプレイヤーエクスペリエンスが得られるようにします。この条件を満たすリージョンでホストされているゲームサーバーインスタンスにプレイヤーを優先的に配置します。

レイテンシーの影響を受けやすい多くのゲームでは、通常、ゲームの各インフラストラクチャリージョンに ping を実行する機能をゲームクライアントに実装することで、ネットワークレイテンシー、ジッター、パケット損失などのパフォーマンスデータを収集し、このデータをメトリクス収集バックエンドに報告して分析できるようにします。プレイヤーをゲームセッションとマッチさせる場合は、プレイヤーのゲームを選択する際に、マッチメイキングサービスで使用する入力の 1 つとして、ゲームクライアントがゲームサーバーインフラストラクチャに対して認識するネットワークレイテンシーを組み込むようにゲームを設定できます。

GAMEPERF_BP10 - ゲームプレイモードとゲームホスティングリージョンごとに個別のマッチメイキングサービスを実行します。

ゲームに複数のゲームプレイモードがあってプレイヤーが選択できる場合は、ゲームプレイモードごとにマッチメイキングシステムを分離することで、各ゲームプレイモードの固有の要件に基づいてパフォーマンスを個別に調整し、リソースの競合を減らすことができます。各ゲームプレイモードには、許容されるレイテンシー、マッチサイズ、その他ゲーム固有のマッチメイキングロジックに関する独自の要件がある場合があります。また、ゲームプレイモードに興味を示すプレイヤーのタイプもそれぞれ異なる場合があります。各ゲームモードのマッチメイキングサービスを個別のソフトウェアデプロイとして実行すると、ゲームモードごとのパフォーマンステストと運用がより簡単になります。例えば、これらをゲームモードごとに個別の Lambda 関数として実行したり、コンテナベースの個別のサービスデプロイとして運用したりできます。

マッチメイキングサービスは、複数のリージョン (可能であればゲームサーバーをホストしているのと同じリージョン) にデプロイします。これにより、プレイヤーを最も近いマッチメイキングサービスにルーティングできる可能性があり、低レイテンシーのゲームサーバーを見つける効率が向上します。Amazon GameLift FlexMatch は、マッチメーカーでリージョンを選択する際の追加ガイダンスを提供し、マッチメーカーと マルチリージョンのゲームセッションキュー を統合する機能を備えています。

GAMEPERF_BP11 -マッチメイキングのパフォーマンスを定期的にモニタリングします。

プレイヤーのためにゲームのパフォーマンスを最適化する最も効果的な方法の 1 つは、プレイヤーがゲームセッションに入るまでの待ち時間を短縮することです。待ち時間が長いと、プレイヤーは興味を失い、プレイヤーの減少につながります。この点を考慮してマッチメイキングのソリューションを設計することが重要です。

ゲームのマッチメイキング設定を設計する場合は、マッチの構成時に適用する条件を決定するルールを作成する必要があります。これらのルールがシステムのパフォーマンス、特にプレイヤーの待ち時間に与える影響を考慮します。新しいマッチメイキングの条件やフィルターの追加など、マッチメイキング実装の変更をデプロイする前に、変更をあらかじめ適切にテストする必要があります。または、この変更をプレイヤー集団全体に導入する前に、パフォーマンスメトリクスを収集するための canary または A/B テストとして、サンプルの少人数のプレイヤー集団に変更を段階的にリリースすることを検討してください。

各マッチメイキングリクエストに適用した条件やルールを理解するのに役立つ詳細なログを生成するようにマッチメイキングサービスを設定し、必要に応じてマッチメイキングの実装を確認および調整できるようにします。例えば、Amazon Amazon GameLift FlexMatch はフルマネージドのマッチメイキングサービスであり、独自のゲームサーバーのホスティングでスタンドアロンサービスとして使用したり、Amazon GameLift でホストしているゲームサーバーで使用したりすることができます。FlexMatch では、Amazon EventBridge (以前の CloudWatch Events) と Amazon Simple Notification Service (Amazon SNS) へのイベント通知を JSON 形式で生成できるため、このデータを自動的に処理および保存して分析し、マッチメイキングのパフォーマンスを向上させることができます。

マッチメイキングサービスがプレイヤーのために適切なゲームセッションを見つけるまでにかかる時間を追跡するためのメトリクスを設定します。マッチメイキングの所要時間のメトリクスを定期的に確認します。これらの時間をプレイヤーの行動やコミュニティのセンチメントと関連付けて、マッチメイキングのタイムアウトの適切なしきい値を策定し、このしきい値をマッチメイキングルールの設定に含めます。例えば、Amazon GameLift FlexMatch は、マッチメイキングリクエストのタイムアウトの定義をサポートするほか、マッチメイキングで 経時的に要件を緩和できるルールの作成を を可能にします。この機能を使用すると、マッチメイキングを作成して適応させることで、マッチを見つけるのが難しい場合でも、マッチを簡単に作成してプレイヤーをゲームセッションに配置できます。

トレードオフ

ゲームのワークロードに適用するパフォーマンス効率の トレードオフ に関するベストプラクティスについては、Well-Architected Framework のホワイトペーパーを参照してください。

リソース

パフォーマンス効率に関するベストプラクティスの詳細については、以下のリソースを参照してください。

ドキュメントとブログ

- Architecture Best Practices for Game Tech (アーキテクチャセンター (ゲーム業界向け))
- パフォーマンス効率の柱 - Well-Architected Framework
- Comparing your on-premises storage patterns with AWS Storage services (オンプレミスのストレージパターンとストレージサービスの比較 - ストレージブログ)
- Amazon EC2 インスタンスストア - Amazon Elastic Compute Cloud
- CloudWatch エージェントを使用して Amazon EC2 インスタンスおよびオンプレミスサーバーからメトリクスおよびログを収集する
- Allow and configure enhanced networking for EC2 instances (EC2 インスタンスの拡張ネットワークを許可および設定する)
- Global Accelerator と Amazon GameLift FleetIQ
- Riot Games のテクノロジーブログ: Scalability and Load Testing For Valorant (Valorant のスケーラビリティと負荷テスト)
- Hyper-scale online games with a hybrid AWS Solution (ハイブリッドソリューションによるハイパースケールのオンラインゲーム)

ホワイトペーパー

- Optimizing Multiplayer Game Server Performance on AWS (AWS でのマルチプレイヤーゲームサーバーのパフォーマンス最適化) ホワイトペーペーパー
- Performance at Scale with Amazon ElastiCache (Amazon ElastiCache を使用した大規模環境でのパフォーマンス)
- Database Caching Strategies Using Redis (Redis を使用したデータベースキャッシュ戦略)
- Amazon Virtual Private Cloud の接続オプション
- 設計パターンのベストプラクティス: Amazon S3 のパフォーマンスの最適化

サードパーティー製ツール

- Unreal Engine のパフォーマンスおよびプロファイリング
- Unity プロファイラー
- Open 3D Engine (O3DE) プロファイラー
- Amazon GameLift のモニタリング

動画コンテンツ

- Amazon EC2 foundations (CMP211-R2)
- Powering next-gen Amazon EC2: Deep dive into the Nitro system (次世代 Amazon EC2 の強化: Nitro system の詳細)
- Getting Started with Amazon GameLift FleetIQ (Amazon GameLift FleetIQ の開始方法)
- Riot Games: Outposts のお客様の声

トレーニング資料

- Game Server Hosting with Amazon EC2 (Amazon EC2 でのゲームサーバーホスティング)
- Using Amazon GameLift FleetIQ for Game Servers (ゲームサーバーでの Amazon GameLift FleetIQ の使用)
- Getting Started with AWS Game Tech (Game Tech の開始方法)
- Game Server Hosting on AWS (AWS でのゲームサーバーのホスティング)
- Amazon GameLift Primer

コスト最適化

コスト最適化の柱には、システムのライフサイクル全体にわたって改良、改善する継続的なプロセスが含まれます。最初の概念実証の初期設計から、本番ワークロードの継続運用まで、このホワイトペーパーのプラクティスを採用すれば、コスト対応のシステムを構築および運用し、ビジネス上の成果を達成してコストを最小化できます。その結果、ビジネスの投資利益率を最大化できます。

設計原則

Well-Architected Framework のホワイトペーパーに記載している最適化の柱に関する設計原則に加えて、以下の設計原則がクラウド内でのゲームワークロードの実行コストの最適化に役立ちます。

プレイヤー、プラットフォーム、ゲーム機能ごとにインフラストラクチャコストを測定する: ゲームの開発と運用、およびプレイヤーの獲得と保持に伴うコストは、ゲームの財務面での成功に大きな影響を与えます。したがって、ゲームプラットフォーム全体で特定のプレイヤーエクスペリエンスや機能に伴うインフラストラクチャコストを理解して追跡し、コスト最適化が必要と思われるアーキテクチャのリソースを特定できるようにすることが重要です。

コスト最適化とプレイヤーエクスペリエンスのトレードオフを評価する: プレイヤーエクスペリエンスの機能と改善に重点を置くか、コスト最適化に重点を置くかを判断します。通常、ゲームが最低限必要な規模に達し、プレイヤー集団が安定したら、ゲームの運用コストを最適化することに重点を置きます。

定義

クラウド内でのコスト最適化に関するベストプラクティスには、以下の 4 つの分野があります。

- 費用対効果の高いリソース
- 需要と供給の一致
- 費用認識
- 長期的な最適化

通常、ゲームデベロッパーは、発売後にゲームがどれくらい人気を得るか、成功するか、または長続きするかを発売前には明確に理解していません。前評判が高くてもプレイヤーを長期に保持できないゲームもあれば、プレイヤーベースを即座または徐々に拡大して持続可能で収益性の高いビジネスに発展するゲームもあります。ゲームの収益化戦略、ビジネスの優先順位、ゲームがそのライフサイクルのどこにあるかに応じて、ゲームデベロッパーはトレードオフを通じてコスト最適化の意思決定を評価する必要があります。例えば、ゲームデベロッパーが、メディアに流布され、業界で広く認知されている、待望の新ゲームを発売する準備をしているとします。

この発売前の段階では、コスト最適化よりも、市場投入までの時間、機能の開発、ゲームのパフォーマンスに重点を置くことが多いでしょう。新しいゲームを発売する場合、ゲームデベロッパーは、ピーク時のプレイヤーの需要に合わせてインフラストラクチャをスケールできるようにしたいと考えます。これに伴い、通常、ピーク時のプレイヤー数予測や最良の販売シナリオに対応するためにリソースを過剰にプロビジョニングする結果となります。または、ゲームが成功しなかったり、ゲームの開発が鈍化する時期に差し掛かったりすると、デベロッパーはできる限りコストの削減を優先させて、既存のプレイヤーのためにゲームを長く運用し続けることを考えるようになります。

ゲームデベロッパーは、複数のゲームを同時に運用する場合があり、これに伴って追加の考慮事項が必要になります。例えば、ゲームデベロッパーは、技術インフラストラクチャ、ソフトウェア、スタッフなどのリソースを複数のライブゲームで再利用し、運用コストをゲーム間で分担する場合があります。

各ゲームは、ビジネスモデル、規模、予測不能性が異なる固有のワークロードです。ゲームのコスト最適化に関する意思決定には、以下の質問が役立つ場合があります。

ベストプラクティス

クラウドアーキテクチャのベストプラクティスを以下に示します。

トピック

費用対効果の高いリソース

GAMECOST01 - ゲームサーバーに適したコンピューティングソリューションは、どのようにして選んでいますか?

他の種類のワークロードと比べて、ゲームワークロードに特有である要素の 1 つは、プレイヤーエクスペリエンスに不可欠なゲームサーバーです。プレイヤーはゲームクライアントからゲームサーバーに接続してゲームセッションをプレイするため、ゲームサーバーはマルチプレイヤーゲームの運用に伴う最大のコスト要因の 1 つでもあります。そのため、ゲームのコンピューティングインフラストラクチャの利用方法を最適化してコストを削減することが重要です。

GAMECOST_BP01: ゲームサーバーを複数のコンピューティングタイプでベンチマークします。

ゲーム開発の初期計画とテスト段階では、ベンチマークを実行して、ゲームに使用する適切なコンピューティングタイプを決定する必要があります。通常、セッションベースのマルチプレイヤーゲームやその他の低レイテンシーのゲームでは、Amazon EC2 インスタンスを使用してゲームサーバーをホストします。各 EC2 インスタンスタイプには、異なるワークロードプロファイルごとに最適化したコンピューティングリソースの組み合わせがあります。ゲームサーバーコードのベンチマークを実行して、ゲームセッションで使用する CPU、メモリ、ネットワーク帯域幅などのリソースを特定するとともに、最小限のコストでパフォーマンスについて適切なバランスが得られるオプションを選択する必要があります。一般的な市販ゲームエンジン (Unreal Engine、Unity、Lumberyard など) のほとんどには、エンジンエディタで利用できるパフォーマンスプロファイリングユーティリティが用意されているため、ゲームサーバービルドから出力されるログやメトリクスデータを活用してパフォーマンスやリソース使用率をベンチマークできます。このテレメトリは、適切な EC2 インスタンスタイプを評価および選択して使用するために役立ちます。

複数の EC2 インスタンスタイプにわたってゲームサーバーをベンチマークする一環として、ゲームを実行するために必要なオペレーティングシステムの種類やプロセッサ要件を判断する必要があります。最善のコスト最適化を行うには、ゲームコンピューティングインフラストラクチャを Linux インスタンスで実行して、Windows で発生するライセンスコストを排除することをお勧めします。また、 Graviton インスタンス は 64 ビットの Arm ベースの EC2 インスタンスであり、 Unreal Engine の専用サーバーを含むゲームサーバーの実行に使用できます。

GAMECOST_BP02 - 各ゲームサーバーインスタンスでホストするゲームセッションの数を最適化してコストを削減します。

サーバーインスタンスごとにホストするゲームセッションの数を最適化して、コンピューティングの使用率を向上させ、コンピューティングインフラストラクチャのコストを削減します。

ゲームデベロッパーは、コストを削減するために、同一の物理サーバーや仮想サーバーでホストするゲームセッションの数を最大化する必要があります。これは、ゲームサーバーのパッキング密度とも呼ばれます。これを達成するには、EC2 インスタンスで同時にホストできるゲームサーバープロセスの数を増やします。通常は、1 つのゲームサーバープロセスが EC2 インスタンスで利用可能なすべてのリソースを占有しないようにします。これは、ゲームのコンピューティングコストを削減する最も重要な方法の 1 つであり、EC2 インスタンス上の複数のサーバープロセスを別々のポートで起動および管理できるソフトウェアを使用する必要があります。例えば、Amazon GameLift には、 インスタンスあたりのゲームサーバープロセスの最大数に対するクォータがあります。このクォータを活用することで、ホスティングコストを削減できます。インスタンスあたりのゲームサーバープロセスの最大数に対する現在のクォータの詳細については、Amazon GameLift のドキュメントを参照してください。

ゲームサーバーのプロセスを EC2 インスタンスなどの仮想マシン上にデプロイする代わりに、ゲームデベロッパーがゲームサーバーをコンテナベースのアプリケーションとして実行する方法が一般的になりつつあります。そのために、Amazon Elastic Container Service (Amazon ECS) や Amazon Elastic Kubernetes Service (Amazon EKS) などのコンテナオーケストレーションプラットフォームを使用したり、 Fargate を使用してゲームサーバーをホストしたりします。コンテナプラットフォームには、ジョブスケジューリング機能があり、リソース要件やその他の指定した配置ロジックに基づいて、ゲームサーバーコンテナをホストするために使用できるコンテナインスタンスをクラスター内で自動的に検出できます。ただし、このレンズの信頼性の柱で説明しているように、スケーリングとプレイヤー配置の動作をどのように管理すれば、アクティブなプレイヤーセッションを妨げずに済むかを検討することが重要です。

GAMECOST_BP03 - 適切なコンピューティング料金を選択してコストを削減します。

ゲームサーバーソフトウェアのパフォーマンステストをさまざまなインスタンスタイプとコンピューティングオプションで実行し、どのオプションがゲームにとって最も費用対効果が高いかを判断します。

ワークロードに適した EC2 インスタンスタイプを効率的に利用することに加えて、コスト最適化の目標に対してどのコンピューティング料金オプションが最もふさわしいかを検討してください。オンデマンドインスタンス、スポットインスタンス、リザーブドインスタンス、Savings Plans など、いくつかの料金オプションが用意されています。

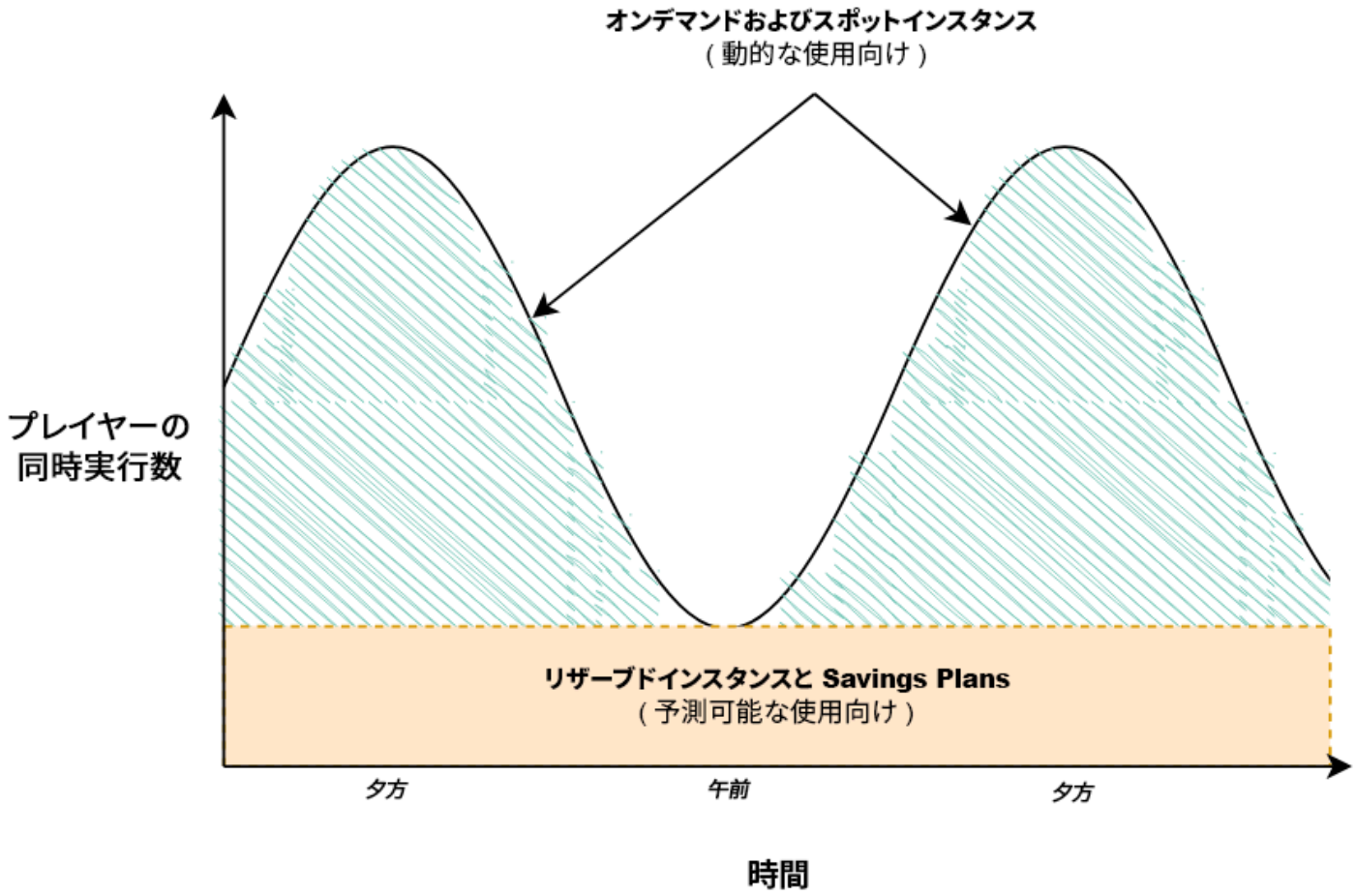
スポットインスタンスは、コンピューティングの割引率が最大で、使用量のコミットメントが不要であり、予測不能で急増する種類のワークロードにも柔軟に対応できるため、ゲームサーバーの実行に理想的です。ただし、スポットインスタンスは中断される可能性が高いため、ゲームセッション時間が短いゲームサーバーのワークロードや、中断に対する許容度が高い状況に最適です。例えば、「Running your game servers at scale for up to 90% lower compute cost (ゲームサーバーを大規模に実行してコンピューティングコストを最大 90% 削減)」というブログ記事では、EC2 スポットインスタンスによって Amazon EKS で Kubernetes を使用してゲームサーバーを実行するためのガイダンスを提供しています。スポットを使用する場合は、ゲームサーバーのワークロードを AWS リージョン内の複数の EC2 インスタンスタイプやアベイラビリティーゾーンで実行し、容量の利用を分散して中断リスクを低減することもお勧めします。また、スポットインスタンスとオンデマンドインスタンスを組み合わせて使用することで、アクティブなゲームセッションに対する中断の影響の可能性を最小限に抑え、容量最適化割り当て戦略を使用して中断のリスクをさらに低減するのもよいでしょう。その他のベストプラクティスについては、「EC2 スポットを利用するうえでのベストプラクティス」を参照してください。Amazon EC2 Auto Scaling キャパシティーの再調整を使用すると、容量を未然にモニタリングして、スポットインスタンスの中断の可能性が高くなったときに容量を追加できます。Amazon GameLift FleetIQ は、スポットインスタンスと統合して低コストのスポットインスタンスの使用を最適化するとともに、中断のリスクを低減します。Amazon GameLift を使用してゲームをホストする場合は、Amazon GameLift のドキュメントを参照してコンピューティングリソースを選択してください。

EC2 リザーブドインスタンスを使用すると、特定のリージョンやインスタンスタイプでの使用をコミットすることでコンピューティングの割引を受けることができます。また、リザーブドインスタンス (RI) の代わりに Savings Plans を使用すると、RI と同様の割引を、リージョン、インスタンスファミリー、オペレーティングシステム、テナンシー全体にわたって柔軟に適用したり、Fargate や Lambda などの他のコンピューティングサービスに適用したりできます。Savings Plans は、リージョンごとに柔軟に対応できるため、新しいゲームの発売時など、複数の地域にわたる使用量が予測できない場合に特に適しています。オンデマンド料金と比べて大幅な割引が得られ、1 年間または 3 年間の予想使用量を予測できるシナリオに最適です。

さまざまなコンピューティングサービスに割引を柔軟に適用できることは、大きなメリットであり、EC2 インスタンスで動作しているゲームサーバーや、Lambda などの他のサービスでの運用が考えられるゲームバックエンドサービスに対して、インフラストラクチャ全体にわたってコメットメントベースの使用料割引を適用できます。中断される可能性があるスポットインスタンスとは異なり、Savings Plans とリザーブドインスタンスは、課金に関する直接的なメリットが得られ、オンデマンド容量と同じ使用特性を利用できます。通常、ゲームサーバーのワークロードでは、ゲームを本番稼働環境で長期間 (少なくとも数週間から数か月) 運用し、日々の使用パターンが十分に明らかになった後で、リザーブドインスタンスを導入します。リザーブドインスタンスと Savings Plans には使用量のコミットメントが必要なため、事前に購入したリザーブドインスタンスと Savings Plans の使用率を最大化することをお勧めします。オンデマンドインスタンスやスポットインスタンスなど、他の購入オプションを追加すると、ゲームサーバーの予測できない使用量の急増に対する柔軟性を高めることができます。

例えば、プレイヤーの毎日の使用パターンによると、プレイヤーベースをサポートするために常時最低 20 台のサーバーが必要であり、さらに定期的な需要のために最大 40 台のサーバーが必要になるとします。この場合、使用需要は予測可能で安定しているため、20 個のリザーブドインスタンスまたは同数の Savings Plan コミットメントを購入することを検討します。これにより、購入した使用量コミットメントを最大限に活用できます。プレイヤーをサポートするために必要な追加容量は、スポットインスタンスとオンデマンドインスタンスを使用してホストできます。

次の図は、ゲームサーバーのワークロードに複数のコンピューティング料金オプションを使用する例を示しています。



ゲームサーバーのホストにおける複数の EC2 料金オプションの使用

この図では、プレイヤーの同時実行数が時間とともに変動しており、使用率の管理とコストの最適化が困難になっています。この変動に対処するには、さまざまなコンピューティング料金プランを組み合わせて使用することを検討します。例えば、リザーブドインスタンスと EC2 Savings Plans を使用して最小使用要件を満たす一方で、動的な使用には EC2 オンデマンドインスタンスと EC2 スポットインスタンスを使用します。

GAMECOST02 - ゲームインフラストラクチャのデータ転送コストは、どのように最適化していますか?

ゲームは、ゲームプレイ体験を提供するゲームインフラストラクチャとプレイヤーのゲームクライアントデバイスとの間だけでなく、ゲームインフラストラクチャのコンポーネント間でも、インターネットを介して大量のデータを転送する場合があります。データ転送が発生する場合の例としては、プレイヤーがゲームコンテンツの更新をゲームクライアントにダウンロードしたとき、ゲームの進行

状況をクラウドに保存したとき、友人とリアルタイムのマルチプレイヤーゲームセッションに参加したとき、また、ゲームインフラストラクチャがリージョンとアベイラビリティーゾーンの間でデータを転送したときなどがあります。アーキテクチャの選択を最適化して、このデータ転送コストを削減するために、ゲームワークロードのどこでデータ転送が発生するかを理解することが重要です。ゲームのデータ転送コストを最適化するには、以下のベストプラクティスを検討してください。

GAMECOST_BP04: インターネットを介したデータ転送のコストを最適化します。

ゲームバックエンドからプレイヤーへのデータ転送コストを削減するソリューションを実装します。

CloudFront を使用すると、コンテンツ配信のコストと、使用頻度の高い公開ウェブアプリケーションのコストを削減できます。クラウドに保存したゲームコンテンツやアセットは、通常、Amazon S3 に保存され、S3 からゲームクライアントに直接配信されるか、Amazon EC2 でホストしているウェブサーバーから配信されます。Amazon EC2 は、Amazon S3 からコンテンツを取得してクライアントに配信します。コンテンツダウンロードのデータ転送コストを削減するには、クラウドストレージの前で Amazon CloudFront を使用してユーザーにコンテンツを配信することを検討してください。CloudFront を使用すると、コンテンツをリージョンから直接配信するよりも、CloudFront のポイントオブプレゼンスから配信する方が割安であるため、データ転送のコストを削減できます。また、CloudFront では Amazon EC2 や Amazon S3 などの AWS ベースのオリジンに対するオリジン取得に対して課金は発生しません。コンテンツがキャッシュ可能であれば、CloudFront を使用してユーザーにより近い場所にコンテンツをキャッシュできるため、コストをさらに削減できます。また、キャッシングを使用しない場合でも、CloudFront では CloudWatch ネットワーク経由でトラフィックをルーティングすることでサーバーとクライアントとの間のデータ転送コストを削減できるため、公開ウェブアプリケーションやサービスの前に配置することにはメリットがあります。CloudWatch を使用して Amazon CloudFront の使用状況をモニタリングできます。複数のコンテンツ配信ネットワーク (CDN) を使用するユースケースの場合、CloudFront Origin Shield が追加のキャッシュレイヤーを提供し、さまざまなプロバイダーからのオリジンリクエストを統合して数を削減できます。コンテンツ配信のベストプラクティスについては、「Content Delivery for Games (ゲーム向けのコンテンツ配信)」ホワイトペーパーを参照してください。

VPC フローログ を使用すると、環境内のネットワークトラフィックをモニタリングし、トラフィックの送信元と送信先を特定してデータ転送コストを最適化できます。

GAMECOST_BP05: コストを最適化して、サービス、アベイラビリティーゾーン、リージョン間のデータ転送を削減します。

ゲームインフラストラクチャとインターネットとの間のデータ転送を最適化することに加え、ゲームインフラストラクチャ内のコンポーネント間のデータ転送も最適化して、同一リージョン内のアベイ

ラビリティーゾーン間、およびリージョン間のトラフィック転送量を削減する必要があります。これらのデータ転送ではいずれも、データ転送コストが発生します。

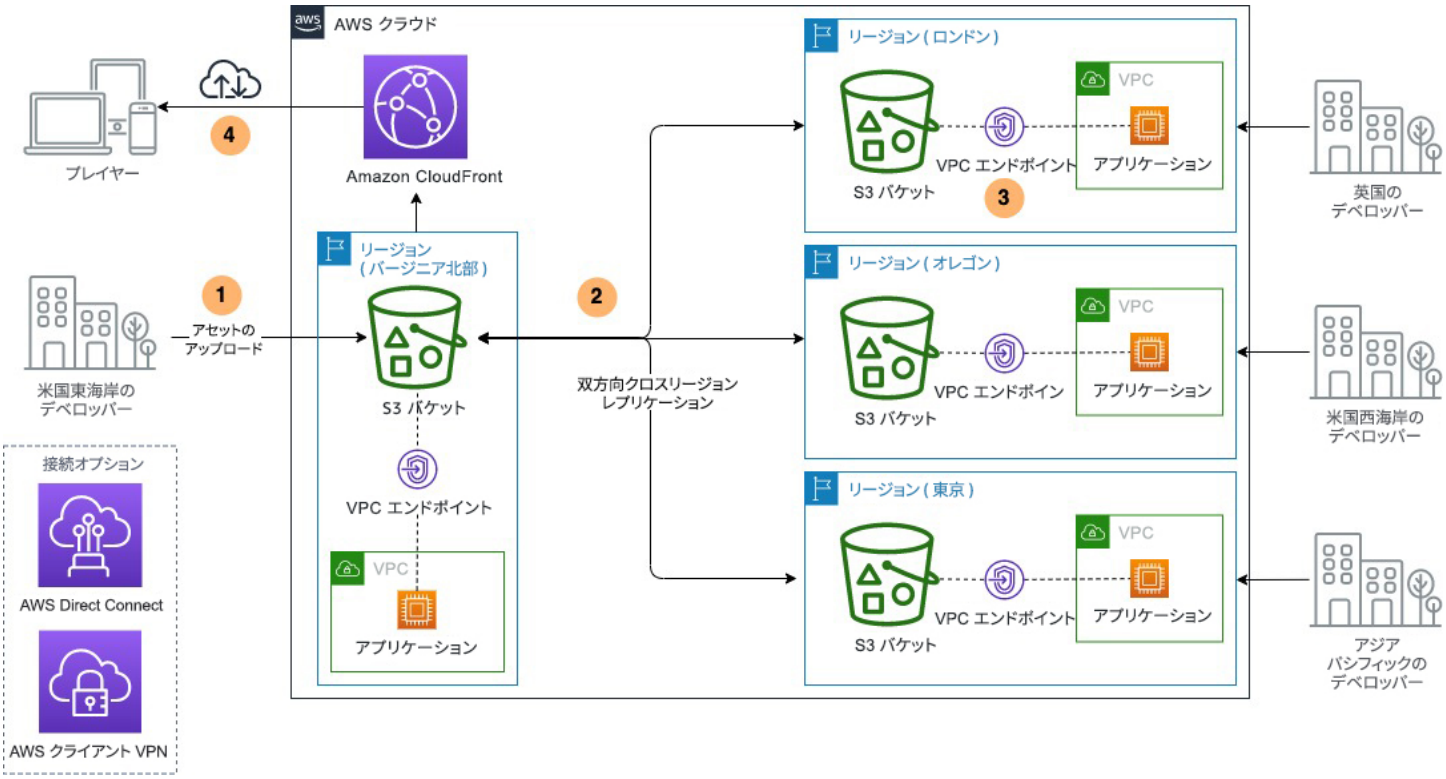
内部トラフィックをアプリケーションと同じアベイラビリティーゾーン内に留めることを優先します。ゲームバックエンドサービスのデータ転送を最適化するには、インスタンスを持つデータベースクラスターとキャッシュクラスターをリージョン内の複数のアベイラビリティーゾーンにデプロイして、アプリケーションサーバーと同じアベイラビリティーゾーンにあるインスタンスからのデータ読み取りを優先するようにアプリケーションを設定できます。この設定でも、アベイラビリティーゾーン間のデータレプリケーションに伴うデータ転送コストは発生しますが、アプリケーションでデータベースやキャッシュを多用するユースケース (読み取り量の多いワークロードなど) では、同じアベイラビリティーゾーン内にデータのローカルコピーを持つことで費用対効果が高まるため、この設定をお勧めします。

特定のリージョン内にあるデータに対して、他のリージョンにあるアプリケーションから定期的にアクセスする必要がある場合は、これらの他のリージョンにデータのコピーをレプリケートします。アプリケーションがリージョンをまたいでデータにアクセスするよりも、リージョン間でデータをレプリケートしてデータのローカルコピーに必要なだけ頻繁にアクセスできるようにする方が費用対効果が高くなります。リージョンをまたいだアクセスは、規模が大きくなると費用対効果が低くなり、パフォーマンスが低下し、リージョンをまたいで適切なセキュリティコントロールを提供するためのネットワーク設定が複雑になります。

例えば、ゲームバックエンドサービスをバージニア北部リージョンにデプロイし、プレイヤー集団に最も近い複数のリージョンにグローバルにゲームサーバーをデプロイして、ゲームプレイヤーのレイテンシーを短縮することができます。ゲームサーバーからアクセスする先のオブジェクトが、バージニア北部でホストしている Amazon ElastiCache for Redis の S3 バケットまたはキャッシュデータに保存されている場合は、ゲームサーバーがあるリージョンにキャッシュデータをレプリケートすると、これらのサーバーがデータを取得するための継続的なデータ転送コストを削減できるため、費用対効果が高くなります。AWS には、データのマルチリージョンレプリケーションを簡単に設定できる機能として、 Amazon Aurora Global Database、Amazon ElastiCache Global Datastore for Redis、 Amazon DynamoDB Global Tables などが用意してあります。Amazon S3 に保存しているオブジェクトに対して別のリージョンでホストしているアプリケーションから頻繁にアクセスする必要があるユースケースでは、 Amazon S3 クロスリージョンレプリケーション (CRR) を使用してコストを削減することを検討してください。CRR を使用すると、アプリケーションをデプロイした 1 つ以上のリージョンでホストしているバケットをレプリケート先として、オブジェクトのコピーを自動的にレプリケートしてコストを削減できます。この設定でも、オブジェクトを別のリージョンにレプリケートするコストは発生しますが、同じリージョン内のレプリケート先の S3 バケットからオブジェクトを取得できるようになるため、アプリケーションがリージョンをまたいで S3 からオブジェクトを取得するたびに発生するデータ転送コストはなくなります。

VPC エンドポイントを使用してサービスと統合し、NAT ゲートウェイを経由したデータトラフィックと処理に対する課金を低減することをお勧めします。同様に、パブリックサブネットでホストしている公開アプリケーションでは、トラフィックが NAT ゲートウェイを経由する必要がない場合があり、送信トラフィックをインターネットゲートウェイに直接送信するように設定して、NAT ゲートウェイが不要な場所でのデータ処理と転送のコストを回避できます。

次の図に示すアーキテクチャを使用すると、他の複数のリージョンでホストしている複数のアプリケーションから共有データセットに低レイテンシーでアクセスする必要がある場合に、アクセスコストを削減できます。



世界中のユーザーがレイテンシーの影響を受けやすいゲームコンテンツにアクセスする場合のコストの最適化

1. ゲーム開発チームが世界中に分散していて、Amazon S3 内の同一コンテンツのコピーにアクセスする場合があります。この場合、米国東海岸に所在するゲームデベロッパーは、コンテンツを Amazon S3 バケットに直接アップロードするか、このリージョンでホストしているアプリケーションからアップロードすることができます。
2. S3 クロスリージョンレプリケーション は、オブジェクトのコピーを他のリージョンのバケットにレプリケートするように設定されているため、これらのリージョンでホストしているアプリケーションは、リージョンをまたいでアクセスのリクエストを送信しなくても、ローカルリージョンからオブジェクトを取得できます。レプリケーションは双方向に設定できるため、他のいずれかのリージョンで行った更新は残りのリージョンにも反映できます。

3. VPC エンドポイント が VPC から Amazon S3 へのプライベートアクセスを提供するため、アプリケーションは NAT ゲートウェイ経由でトラフィックをルーティングする必要がありません。NAT ゲートウェイは、他の高スループットのアプリケーショントラフィックで使用されている場合、輻輳を引き起こすおそれがあります。他のグローバルスタジオ、リモートワーカー、請負業者などのゲーム開発チームは、各自が最も高いパフォーマンスを得られるリージョンに接続してデータベースのコピーにアクセスできます。スタジオの所在地やデータセンターとリージョンとの間に専用接続を設定するには、 Direct Connect を使用します。リモートワーカーに VPC への安全なリモートアクセスを提供するには、 クライアント VPN を使用します。
4. プレイヤーのゲームクライアントやその他のインターネットベースのアプリケーション が CloudFront と統合することで、S3 に保存したオブジェクトのコンテンツキャッシュを CloudFront が提供し、インターネット経由の静的および動的コンテンツのデータ転送コストを削減します。

Amazon S3 のマルチリージョンアクセスポイント を使用すると、S3 バケットをホストしていないリージョンでホストされているアプリケーションのアクセスパターンを簡素化できます。アプリケーションは、マルチリージョンアクセスポイントとやり取りし、レイテンシーが最も低いバケットの所在地を特定してリクエストを処理できます。マルチリージョンアクセスポイントには追加料金がかかります。

GAMECOST03 - ゲームインフラストラクチャのデータストレージコストは、どのようにして最適化していますか?

ゲームは大量のデータを生成する可能性があり、これらのデータを保存して、デベロッパー、プレイヤー、ゲーム自体が利用できるようにする必要があります。例えば、ゲームデベロッパーが常時生成する新しいソースコード、ゲームコンテンツ、アセットを保存したり、プレイヤーが生成する新しいユーザー生成コンテンツを処理したりする必要があります。また、ゲームクライアントやサーバーが生成するゲーム分析テレメトリをデータレイクに保存して分析チームが利用できるようにする必要があります。ゲームは構造化データも生成します。

GAMECOST_BP06 - 適切なストレージタイプを選択してコストを削減します。

生成して保存するデータの種類ごとに固有の特性があります。これらの特性を考慮して、ワークロードに使用する適切なストレージソリューションを決定する必要があります。

最も費用対効果の高いストレージクラスにオブジェクトデータを保存するには、 S3 オブジェクトのライフサイクル管理 を使用します。Amazon S3 には、複数の ストレージクラス と オブジェクトの

ライフサイクル管理 機能が用意されており、シンプルできめ細かなポリシーを簡単に設定して、ストレージ階層間のデータ移行を自動化し、コストを削減できます。すべてのデータをデフォルトで S3 Standard ストレージクラスに単に保存するのではなく、時間の経過とともに階層間でデータを自動的に移行するようにライフサイクル設定をセットアップすることを検討してください。あるいは、不明なアクセスパターンや変化するアクセスパターンには S3 Intelligent-Tiering ストレージクラスを使用します。また、S3 Intelligent-Tiering は、費用対効果の高い方法でデータを階層間で自動的に移行できます。ライフサイクルポリシーを手動で設定しなくてもコストを最適化できるため、デフォルトのストレージクラスとしてお勧めします。現時点では、 小規模で存続期間が短いオブジェクト向けの最適な選択肢となっています。Amazon S3 の一般的なユースケースには、ゲームアセット、静的コンテンツ、ゲームログ、データレイクストレージ、バックアップの保存があります。開発中に共有ファイルシステムをワークステーションにアタッチするなど、ファイルシステムが必要なユースケースでは、 Amazon Elastic File System (Amazon EFS) を使用してさまざまなストレージクラスを提供する ことを検討してください。Amazon EFS は、ファイルの追加や削除に応じて自動的に拡大/縮小するため、インフラストラクチャの管理は不要です。

需要と供給の一致

ゲームのワークロードに適用するコスト最適化の 需要と供給の管理 に関するベストプラクティスについては、Well-Architected Framework のホワイトペーパーを参照してください。また、ゲームインフラストラクチャの動的スケーリングに関するその他のベストプラクティスについては、このレンズの信頼性の柱と運用上の優秀性の柱を参照してください。

費用認識

GAMECOST04 - ゲーム環境のコストは、どのようにして測定しますか?

プレイヤーあたりのコスト、ゲーム機能、環境を理解し、プレイヤー数の経時的な変化、機能の追加や強化に伴う支出を管理および予測できるようにします。以下のベストプラクティスを参照し、さまざまなゲーム環境のコストを管理してください。

GAMECOST_BP07: プレイヤー、ゲーム機能、環境ごとにコストの割り出しを行います。

通常、ゲームサーバーのコストの割り出しは、ゲームバックエンドサービスよりも簡単に行うことができます。ゲームサーバーは、インスタンスごとに特定数の同時実行プレイヤーをホストできるように最適化されていることが多く、この数にはインスタンスを実行するコストで対応できるためです。ゲームバックエンドサービスの場合は、ゲームのコンポーネントを個別の機能に分離して個別の

論理リソースや物理リソースとして管理できるようにし、コストを分析しやすくすることをお勧めします。例えば、1 つのモノリシックアプリケーションを実装してゲームのバックエンドサービスをホストする方が簡単に思われますが、このパターンでは、リソースのコンピューティング、ネットワーク、ストレージのコストをすべてのサービスで共有するため、機能を追加するに従ってプレイヤーやゲーム機能ごとの総コストを割り出すことが困難になります。

ゲームバックエンドサービスには、サーバーレスアーキテクチャを採用することを検討してください。例えば、コンピューティングには Amazon API Gateway と Lambda または Fargate、メッセージングには Amazon SQS と Amazon SNS、オブジェクトストレージには Amazon S3、データベースストレージには Amazon DynamoDB をサービスとして使用します。これらのサービスでは、主にリクエスト量に応じた使用量ベースの料金設定を通じてコストをより細かく分類できますが、そうした製品例は多数あります。Lambda 関数、Fargate サービス、DynamoDB テーブル、S3 バケットなどの各リソースにはコスト配分タグを関連付けることができるため、これらのサービスのコストをゲーム機能名ごとに確認でき、サービスごとのコストが理解しやすくなります。

また、複数のゲーム開発環境を個別に管理して、異なる環境ごとにコストを割り出せるようにすることをお勧めします。通常、ゲームデベロッパーは、このゲーム業界レンズの運用上の優秀性の柱で説明しているように、開発、テスト、ステージング、本番稼働環境を個別に管理します。各環境は一般的に、スケーラビリティ、パフォーマンス、使用の要件が異なり、また、別々のチームが管理する場合があります。コストを管理するには、これらの環境を整理して、各環境のコストを適切にモニタリングおよび確認できるようにします。

詳細については、以下のドキュメントを参照してください。

- Building a serverless multi-player game that scales (スケールできるサーバーレスマルチプレイヤーゲームの構築)
- Standalone Game Session Servers with a Websockets-based backend (Websockets ベースのバックエンドを備えたスタンドアロンのゲームセッションサーバー)
- Standalone Game Session Servers with a Serverless backend (サーバーレスバックエンドを備えたスタンドアロンのゲームセッションサーバー)

長期的な最適化

長期的な最適化に関するガイダンスについては、 Well-Architected Framework のコスト最適化の柱を参照してください。

リソース

コスト最適化に関するその他のベストプラクティスについては、以下のリソースを参照してください。

ドキュメントとブログ

- NAT ゲートウェイのデータ転送料金を削減するにはどうすればよいですか?
- Agones 用の Amazon GameLift FleetIQ アダプター
- VPC で NAT ゲートウェイを通過するトラフィックの上位の要因を見つけるにはどうすればよいですか?
- グローバルゲームサーバーに適切なコンピューティング戦略を選択する
- Well-Architected ラボ - Cost effective resources (費用対効果の高いリソース)
- Amazon VPC CNI plugin increases pods per node limits (Amazon VPC CNI プラグインがノードあたりのポッド数の制限を緩和)
- Architecture Best Practices for Cost Optimization (コスト最適化に関するアーキテクチャのベストプラクティス)
- Reducing player wait time and right sizing compute allocation using Amazon SageMaker RL and Amazon EKS (Amazon Sagemaker RL と Amazon EKS を使用してプレイヤーの待ち時間を短縮し、適切なサイズのコンピューティングを割り当てる)
- Compute Optimizer
- Electronic Arts optimizes storage costs and operations using Amazon S3 Intelligent-Tiering and S3 Glacier (Electronic Arts が、Amazon S3 Intelligent-Tiering と S3 Glacier を使ってストレージコストとオペレーションを最適化)
- Escape unfriendly licensing practices by migrating Windows workloads to Linux (Windows ワークロードを Linux に移行して不親切なライセンス慣行から脱却する)
- Overview of Data Transfer Costs for Common Architectures (一般的なアーキテクチャでのデータ転送コストの概要)

ホワイトペーパー

- コスト最適化の柱 - Well-Architected Framework
- Amazon EC2 Reserved Instances and Other AWS Reservation Models (Amazon EC2 リザーブドインスタンスとその他の予約モデル)

- Right Sizing: Provisioning Instances to Match Workloads (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)

まとめ

ゲームは、世界中のプレイヤーにエンターテインメントエクスペリエンスを提供するように設計されており、一般的に予測不能で変わりやすい使用特性を持っています。ゲーム業界レンズでは、ゲームアーキテクチャを構成する一般的なタイプのシナリオについて説明し、クラウド内でゲームを構築および運用する際に考慮すべき一連の質問とベストプラクティスを示しています。このフレームワークをゲームアーキテクチャに適用することで、信頼性、安全性、効率性、費用対効果に優れたゲームをクラウド内で構築できるようになります。

寄稿者

このドキュメントの執筆寄稿者は以下のとおりです。

- Kyle Somers – Amazon Web Services、プリンシパルソリューションアーキテクト
- Hyobin An – Amazon Web Services、ソリューションアーキテクト
- Peter Chapman – Amazon Web Services、ソリューションアーキテクチャ、マネージャー
- Nirav Doshi – Amazon Web Services、プリンシパルテクニカルアカウントマネージャー
- Chris Finch – Amazon Web Services、シニア AMER デジタルユーザーエンゲージメントスペシャリストソリューションアーキテクト
- Jackie Jiang – Amazon Web Services、シニアソリューションアーキテクト
- Sungsoo Khim – Amazon Web Services、ソリューションアーキテクト
- Byungsu Kim – Amazon Web Services、ソリューションアーキテクト
- Minsuk Kim – Amazon Web Services、ソリューションアーキテクト
- Pawan Matta – Amazon Web Services、シニアソリューションアーキテクト
- Jinsung Park – Amazon Web Services、アソシエイトソリューションアーキテクト
- Tanya Rhodes – Amazon Web Services、シニアソリューションアーキテクト
- Bruce Ross – Amazon Web Services、Well-Architected、シニアメディアソリューションアーキテクト

ドキュメント履歴

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
初版発行	ホワイトペーパーの初回発行。	November 19, 2021
マイナーな更新	「定義」セクションを追加しました。	November 19, 2021

> **Note**
>
> RSS の更新をサブスクライブするには、使用しているブラウザで RSS プラグインが必要です。

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 の責任と義務は、AWS 契約により規定されます。本書は、AWS とお客様の間のいかなる契約の一部でもなく、またそれを変更するものでもありません。

AWS Glossary

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