aws

開発者ガイド

Amazon GameLift

Amazon GameLift: 開発者ガイド

Table of Contents

Amazon GameLift とは

Amazon GameLift を使用して、セッションベースのマルチプレイヤーゲームのため、クラウドに低コストの専有サーバーをデプロイ、運用、スケールできます。AWS グローバルコンピューティングインフラストラクチャに構築された Amazon GameLift を利用すると、高パフォーマンス、高信頼性のゲームサーバーを提供できるほか、世界中のプレイヤーの需要に合わせてリソースの使用を動的にスケーリングできます。

Amazon GameLift の使用

Amazon GameLift は次のユースケースやその他のユースケースをサポートします。

- 独自のカスタムマルチプレイヤーゲームサーバーを使用するか、または、準備のできているリアルタイムサーバーを使用してゲームをホストします。
- Amazon Elastic Compute Cloud (Amazon EC2) スポットインスタンスを使用して、低コストのホスティングリソースを実行します。
- ゲームに必要なホスティングリソースの量を、使用状況に基づいて自動的にスケーリングします。
- Amazon GameLift FleetIQ を使用して、Amazon EC2 のコンピューティングリソースをすべて 1 か所で管理できます。
- Amazon GameLift FlexMatch を使ってマルチプレイヤーでプレーヤーをマッチングします。
- Amazon GameLift Anywhere を使用して、ゲームサーバーとクライアントのビルドを繰り返しテストします。
- Amazon GameLift Anywhere では、独自のハードウェアを使用しつつもすべてを 1 か所で管理できます。

> **Tip**
>
> Amazon GameLift ゲームサーバーホスティングを試すには、「Amazon GameLift の開始方法」を参照してください。

Amazon GameLift ソリューションの使用を開始する

ゲーム開発者向けの Amazon GameLift ソリューション

- カスタムサーバー用の Amazon GameLift ホスティング。
- リアルタイムサーバーを使用した Amazon GameLiftホスティング
- Amazon EC2 でホスティングするための Amazon GameLift FleetIQ
- マッチメーキングのためのAmazon GameLift FlexMatch
- Amazon GameLift Anywhere のハードウェアホスティング

カスタムサーバー用の Amazon GameLift ホスティング。

Amazon GameLift は、独自のカスタムゲームサーバーをホストするのに必要な作業を代行します。自動スケーリング機能により、必要以上のリソースにお金を払う必要がなくなります。また、自動スケーリングを使用すると、新規プレイヤーが最小限の待ち時間でいつでもゲームに参加できるようになります。

Amazon GameLift のホスティング詳細については、「Amazon GameLift の仕組み」を参照してください。

主な特徴

- 自動スケーリング、マルチロケーションキュー、ゲームセッションプレイスメントなど、Amazon GameLift 管理機能を使用します。
- Amazon Linux または Windows Server オペレーティングシステムで実行するゲームサーバーをデプロイできます。
- ゲームセッションとプレイヤーセッションを管理できます。
- サーバープロセス用の、カスタマイズされたヘルストラッキングを設定して、問題を検出し、低レベルのパフォーマンスプロセスを解決します。
- Amazon GameLiftの AWS CloudFormation テンプレートを使用してゲームリソースを管理します。

リアルタイムサーバーを使用した Amazon GameLiftホスティング

リアルタイムサーバーを使用すれば、カスタム構築のゲームサーバーを必要としないゲームを立ち上げられます。この軽量サーバーソリューションは、ゲームに合わせて構成できるゲームサーバーを提供します。

Amazon GameLift をリアルタイムサーバーでホスティングする方法の詳細は「Amazon GameLift リアルタイムサーバーとのゲームの統合」を参照してください。

主な特徴

- 自動スケーリング、マルチロケーションキュー、ゲームセッションプレイスメントなど、Amazon GameLift 管理機能を使用します。
- Amazon GameLift ホスティングリソースを使用し、フリートの AWS コンピューティングハードウェアのタイプを選択します。
- ゲームのクライアントとサーバー間のやり取りのための完全なネットワークスタックを活用できます。
- カスタマイズ可能なサーバーロジックにより、ゲームサーバーのコア機能を利用できます。
- リアルタイム設定およびサーバーロジックに対してライブ更新を行えます。

Amazon EC2 でホスティングするための Amazon GameLift FleetIQ

Amazon GameLift FleetIQ を使用して、Amazon EC2 と Amazon EC2 Auto Scaling のホスティングリソースを直接します。これにより、Amazon GameLift が最適化され、安価で耐障害性の高いゲームホスティングを実現できるというメリットが得られます。このソリューションは、フルマネージド Amazon GameLift ソリューションで提供されるものよりも高い柔軟性を必要とするゲーム開発者向けに設計されています。

Amazon GameLift FleetIQ が Amazon EC2 および EC2 Auto Scaling と連携してゲームホスティングを行う方法については、Amazon GameLift FleetIQ 開発者ガイドを参照してください。

主な特徴

- FleetIQ アルゴリズムを使用してスポットインスタンスのバランスを最適化します。
- プレイヤールーティングの特徴を使って、ゲームサーバーのリソースを効率的に管理し、ゲームに参加する際に最適なプレイヤーエクスペリエンスを提供します。
- プレーヤーの使用状況に応じてホストする容量を自動的にスケールできます。
- 自分の AWS アカウント アカウントから Amazon EC2インスタンスを直接管理します。
- Windows、Linux、コンテナ、Kubernetes など、サポートされているゲームサーバー実行可能ファイル形式のいずれかを使用できます。

マッチメーキングのためのAmazon GameLift FlexMatch

FlexMatch を使用してカスタムルールセットを構築し、ゲームのマルチプレイヤーマッチを定義します。FlexMatch はルールセットを使用して各マッチで互換性のあるプレイヤーを比較し、プレイヤーに理想的なマルチプレイヤーエクスペリエンスを提供します。

FlexMatch の詳細については、「Amazon GameLift FlexMatch とは」を参照してください。

主な特徴

- マッチの作成速度とマッチの質のバランスを取ります。
- 定義された特性に基づいてプレーヤーやチームをマッチさせます。
- レイテンシーに基づいてプレイヤーをマッチに参加させるルールを定義します。

Amazon GameLift Anywhere のハードウェアホスティング

Amazon GameLift Anywhere を使用して、ご使用の環境のどこにあるハードウェアでも Amazon GameLift ゲームホスティングに統合します。Anywhere フリートおよび EC2 フリートをマッチメーカーキューとゲームセッションキューに統合し、マッチメーキングとゲームプレースメントをハードウェア全体で管理できます。

Anywhere でのテストの詳細については、「Amazon GameLift Anywhere フリートを使用して統合をテストする」を参照してください。Anywhere フリートのセットアップに関する詳細については、「Amazon GameLift のフリートのセットアップ」を参照してください。

主な特徴

- ゲームサーバーとクライアントのビルドを素早く繰り返しテストできます。
- Amazon GameLift のセットツールを使用して、独自のハードウェアにゲームをデプロイします。
- プレーヤーに最も近いハードウェアをどこでも使用できます。

Amazon GameLift へのアクセス

Amazon GameLift を使用するには、これらのツールを使用します。

Amazon GameLift SDK

Amazon GameLift SDK には、ゲームクライアント、ゲームサーバー、ゲームサービスから Amazon GameLift と通信するために必要なライブラリが用意されています。詳細については、「Amazon での開発サポート GameLift」を参照してください。

Amazon GameLift リアルタイムクライアント SDK

リアルタイムクライアント SDK を使用すると、ゲームクライアントは、リアルタイムサーバーに接続し、ゲームセッションに参加して、他のプレイヤーと同期することができます。[SDK] をダウンロードして、リアルタイムサーバークライアント API (C#) を使用した API コールの詳細はこちらを確認してください。

Amazon GameLift コンソール

ゲームのデプロイの管理、リソースの設定、プレイヤーの使用状況とパフォーマンスメトリクスの追跡を行うには、Amazon GameLift の AWS Management Console を使用します。Amazon GameLift コンソールには、AWS Command Line Interface (AWS CLI) でプログラムによりリソースを管理する代替 GUI が用意されています。

AWS CLI

Amazon GameLift API を含む、AWS SDKへの呼び出しを行うには、このコマンドラインツールを使用します。AWS CLI の使用の詳細については、AWS Command Line Interface ユーザーガイドの「AWS CLIの開始方法」を参照してください。

Amazon GameLift の料金

Amazon GameLift では、インスタンスについては使用時間ごとに課金され、帯域幅については転送されたデータ量によって課金されます。Amazon GameLift の課金および料金の詳細な一覧については、「Amazon GameLift の料金表」を参照してください。

Amazon GameLift でのゲームのホスティングまたはマッチメーキングのコストの計算については、AWS Pricing Calculator の使用方法について説明している「Amazon GameLift の価格見積もりの生成」を参照してください。

Amazon GameLift の仕組み

このトピックでは、ゲーム ホスティングのコアコンポーネントについて説明し、Amazon GameLift がマルチプレイヤー ゲームサーバーをプレイヤーに利用できるようにする方法について説明します。

Amazon GameLift でホスティングするのにゲームの準備はできていますか? Amazon GameLift マネージドホスティングロードマップ を確認してください。

主要コンポーネント

Amazon GameLift を設定してゲームをホストするには次のコンポーネントを扱います。マネージド Amazon を使用したゲームアーキテクチャ GameLift の図は、これらのコンポーネント間の関係を視覚化したものです。

- ゲームサーバーは、フリートで実行されるゲームのサーバーソフトウェアです。ゲームサーバーのビルドまたはスクリプトを Amazon GameLift にアップロードし、Amazon GameLift に通知します。Amazon GameLift Anywhere または Amazon GameLift FleetIQ を使用する場合は、ゲームサーバービルドをコンピューティングリソースに直接アップロードします。
- [ゲームセッション] とは、プレイヤーと一緒に進行中のゲームです。有効期間やプレイヤー数など、ゲームセッションの基本特性を定義します。次に、プレイヤーはゲームサーバーに接続してゲームセッションに参加します。
- ゲームクライアントは、プレイヤーのデバイスで実行されるゲームのソフトウェアです。ゲームクライアントは、Amazon GameLift サービスから受信する接続情報に基づいて、バックエンドサービスを通してゲームサーバーに接続してゲームセッションに参加します。
- バックエンドサービスは、Amazon GameLift に関連するタスクを処理する追加のカスタムサービスです。ベストプラクティスとして、Amazon GameLift とのすべてのゲームクライアント通信をバックエンドサービスが処理します。

ゲームサーバーのホスティング

Amazon GameLift を使用すると、マネージド Amazon GameLift、Amazon GameLift FleetIQ、Amazon GameLift Anywhere の 3 つの方法でゲームサーバーをホストできます。Amazon GameLift FleetIQ の詳細については、「Amazon GameLift FleetIQ とは」を参照してください。

フリートは、ゲームのニーズに合わせて設計できます。フリートの設計に関する詳細については、「Amazon GameLift フリート設計ガイド」を参照してください。

マネージド Amazon GameLift

マネージド Amazon GameLift を使用すると、インスタンスと呼ばれる Amazon GameLift 仮想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できます。インスタンスのフリートを作成し、それらをデプロイしてゲームサーバー (カスタムゲームサーバーまたはリアルタイムサーバー) を実行して、ホスティングリソースを設定します。

Amazon GameLift Anywhere

Amazon GameLift Anywhere を使用すると、管理するコンピューティング環境でゲームサーバーをホストできます。コンピューティングを参照する Anywhere フリートを作成して、ホスティングリソースを設定します。

フリートのエイリアス

エイリアスは、フリート間で転送できる指定先であり、一般的なフリートの口ケーションを指定する便利な方法です。エイリアスを使用すると、ゲームクライアントを変更せずに、ゲームクライアントをあるフリートから別のフリートに切り替えることができます。コンテンツを指定するターミナルエイリアスを作成することもできます。

ゲームセッションの実行

フリートにゲームサーバービルドをデプロイし、Amazon GameLift で各インスタンスでゲームサーバープロセスを起動すると、フリートでゲームセッションをホストできるようになります。ゲームクライアントサービスがバックエンドサービスまたは Amazon GameLift にプレイスメントをリクエストを送信すると、新しいゲームセッションがスタートされます。

ゲームセッションプレイスメントと FleetIQ アルゴリズム

キューは FleetIQ アルゴリズムを使用して、新しいゲームセッションをホストする利用可能なゲームサーバーを選択します。ゲームセッションプレイスメントの主な構成要素は Amazon GameLift のゲームセッションキューです。ゲームセッションキューに、そのキューがゲームセッションを配置できる場所を決定するフリートのリストを割り当てます。ゲームセッションのキューの詳細と、ゲーム用にそれらを設計する方法については、「ゲームセッションキューの設計」を参照してください。

ゲームへのプレーヤーのConnection

ゲームセッションプレイスメントプロセスのパートとして、キューまたはゲームセッションは新しいゲームセッションをスタートするように選択したゲームサーバーに求めます。ゲームサーバーは、プロンプトに応答し、プレイヤー接続を受け入れる準備ができたら Amazon GameLift に報告します。その後、Amazon GameLift は接続情報をバックエンドサ q ービスまたはゲームクライアントサービスに送信します。ゲームクライアントはこの情報を使用してゲームセッションに直接Connectし、ゲームに参加します。

フリートの容量のスケーリング

フリートがアクティブになりゲームセッションをホスティングできるようになると、プレイヤーの需要に合わせてフリートキャパシティを調整できます。すべての新規プレイヤーが素早くゲームを見つ

けるられるようにすることと、アイドル状態のリソースを浪費することの間でバランスを図ることをお勧めします。

Amazon GameLift は非常に効果的な自動スケーリングツールを提供しています。また、フリートのキャパシティをマニュアルで設定することもできます。詳細については、「Amazon GameLift ホスティングキャパシティのスケーリング」を参照してください。

Auto Scaling

Amazon GameLift では 2 つの自動スケーリングの方法が用意されています。

- ターゲットベースの自動スケーリング
- ルールベースのポリシーによる自動スケーリング

追加のスケーリング機能

- [ゲームセッション保護] - Amazon GameLift がアクティブなプレイヤーをホストしているゲームセッションをスケールダウン中に終了するのを防ぎます。
- [Scaling limits](スケーリング制限) – フリートのインスタンス数に下限と上限を設定することで全体的なインスタンスの使用量をコントロールします。
- [自動スケーリングの停止] – 自動スケーリングポリシーを変更または削除せずに、フリートロケーションレベルで自動スケーリングを停止します。
- [スケーリングメトリクス] – フリートのキャパシティおよびスケーリングイベントを追跡します。

Amazon GameLift のモニタリング

フリートが起動すると、Amazon GameLift は、デプロイしたゲームサーバーのパフォーマンスをモニタリングするに役立つさまざまな情報を収集します。この情報を使って、リソースの使用の最適化、問題のトラブルシューティング、プレイヤーのゲーム内での行動の仕方の確認できます。Amazon GameLift は以下を収集します。

- フリート、場所、ゲームセッション、プレイヤーセッションに関する詳細
- 使用状況メトリクス
- サーバープロセスの健全性
- ゲームセッションログ

Amazon GameLift でのモニタリングに関する詳細は、「Amazon のモニタリング GameLift」を参照してください。

他の AWS リソースの使用

ゲームサーバーやアプリケーションは他の AWS リソースと通信できます。たとえば、プレイヤーの認証やソーシャル ネットワークのために一連のウェブサービスを使用する場合があります。ゲームサーバーが AWS アカウント が管理する AWS リソースにアクセスするには、Amazon Amazon GameLift が AWS リソースにアクセスすることを明示的に許可します。

Amazon GameLift には、このタイプのアクセスを管理するためのオプションがいくつかあります。詳細については、「フリートの他の AWS リソースと通信する」を参照してください。

プレイヤーがゲームに接続する方法

[game session](ゲームセッション)は Amazon GameLift で実行するゲームのインスタンスです。ゲームをプレイするには、既存のゲームセッションを見つけて参加するか、新しいゲームセッションを作成して参加します。プレイヤーはゲームセッションの [プレイヤーセッション] を作成することで参加します。ゲームセッションがプレイヤーに対して開いている場合、Amazon GameLift はプレイヤーのためにスロットを予約し、接続情報を提供します。その後、プレイヤーはゲームセッションに接続して、予約されたスロットを要求できます。

ゲームセッションとプレイヤーセッションのカスタムゲームサーバーでの作成と管理の詳細については、「ゲームクライアント GameLift に Amazon を追加する」を参照してください。プレイヤーをリアルタイムサーバーに接続する方法については、「リアルタイムサーバー用のゲームクライアントの統合」を参照してください。

Amazon GameLift には、ゲームセッションとプレイヤーセッションに関連するさまざまな特徴があります。

複数の口ケーション間で使用可能なリソースのうち、最適なリソースでゲームセッションをホストする

> Amazon GameLift が新しいゲームセッションをホストするためのリソースを選択する方法を設定する際、複数のオプションから選択できます。複数の口ケーションでフリートを実行している場合、口ケーションに関係なく任意のフリートに新しいゲームセッションを配置するゲームセッションキューを設計できます。

ゲームセッションに対するプレイヤーのアクセスコントロール

現在接続しているプレイヤーの数に関係なく、新しいプレイヤーからの参加リクエストを許可または拒否するようにゲームセッションを設定します。

カスタムゲームとプレイヤーデータを使用する

ゲームセッションオブジェクトとプレイヤーセッションオブジェクトにカスタムデータを追加します。Amazon GameLift は、新しいゲームをスタートするときゲームセッションデータをゲームサーバーに渡します。Amazon GameLift は、プレイヤーがゲームセッションに接続すると、プレイヤーセッションデータをゲームサーバーに渡します。

使用可能なゲームセッションのフィルタとソート

セッション検索とソートを使用して、将来のプレイヤーに最適なマッチングを探したり、使用可能なゲームセッションのリストからプレイヤーが選択できるようにします。セッション検索とソートを使用して、セッションの特性に基づいてゲームセッションを検索します。独自のカスタムゲームデータに基づいて検索およびソートすることもできます。

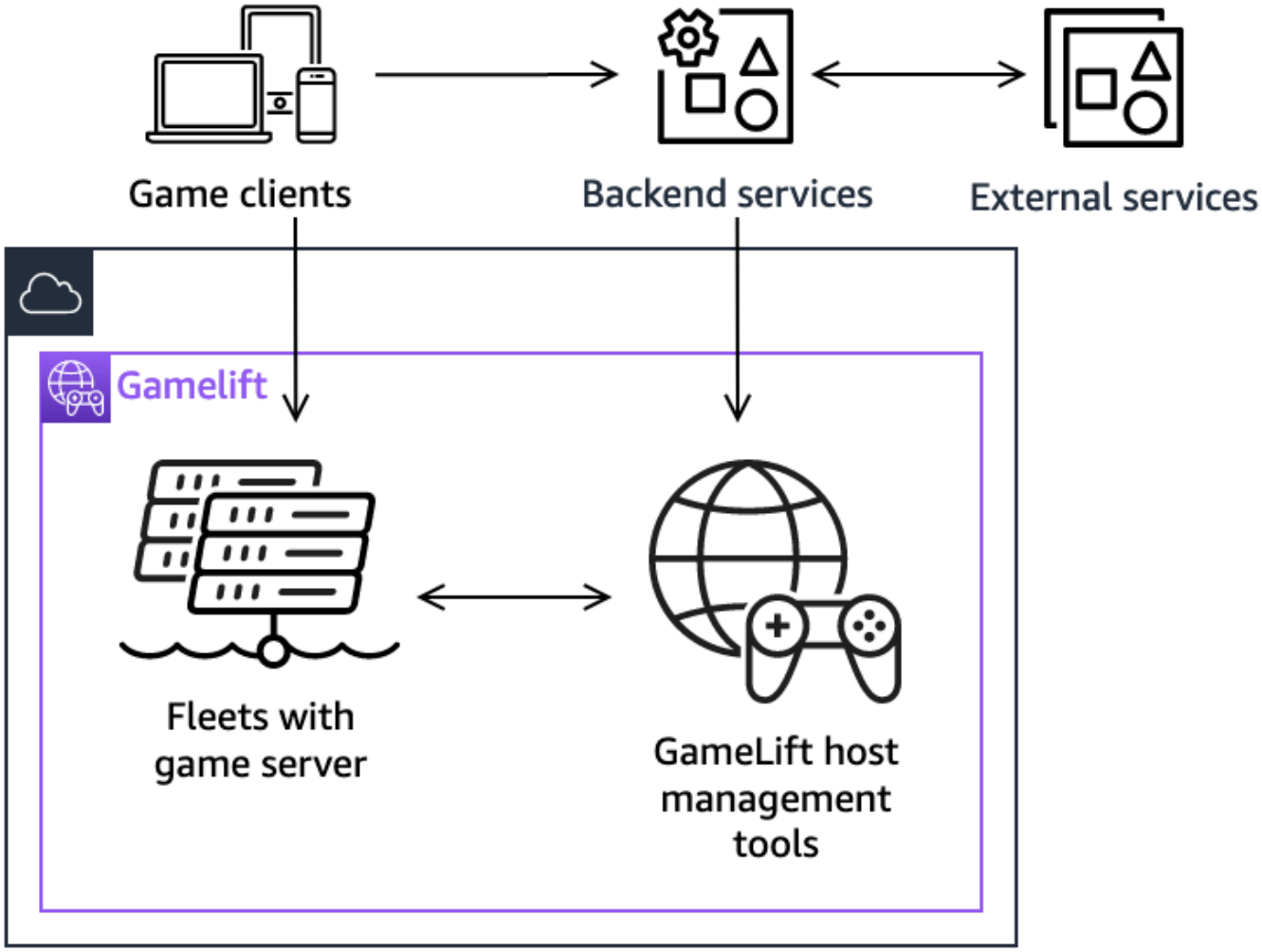
ゲームおよびプレイヤー使用状況データを追跡する

完了したゲームセッションに対して自動的にログが保存されます。ゲームサーバーに Amazon GameLift を統合するときに、ログストレージをセットアップできます。詳細については、「Amazon GameLift でのサーバーメッセージの口グ記録」を参照してください。

セッションのメタデータ、設定、プレイヤーセッションデータなど、ゲームセッションの詳細情報を表示するには、Amazon GameLift コンソールを使用します。詳細については、ゲームおよびプレイヤーセッションのデータを表示する および メトリクス を参照してください。

マネージド Amazon を使用したゲームアーキテクチャ GameLift

次の図は、 マネージド Amazon GameLift ソリューションを使用してホストされるゲームアーキテクチャの主要なコンポーネントを示しています。



アーキテクチャの主要コンポーネントには次が含まれています。

ゲームクライアント

Amazon でホストされているゲームに参加するには GameLift、ゲームクライアントが最初に利用可能なゲームセッションを見つける必要があります。ゲームクライアントは、既存のゲームセッションを検索したり、マッチメーキングをリクエストしたり、バックエンドサービス GameLift を介して Amazon と通信して新しいゲームセッションを開始したりします。バックエンドサービスは Amazon にリクエストを行い GameLift、それに応じてサービスがゲームセッション情報を受信し、それをゲームクライアントに中継します。その後、ゲームクライアントはゲームサーバーに接続します。詳細については、「Amazon 用のゲームの準備 GameLift」を参照してください。

バックエンドサービス

バックエンドサービスは、 AWS SDK で Amazon GameLift サービス API オペレーションを呼び出す GameLift ことで、ゲームクライアントと Amazon 間の通信を処理します。また、プレイヤーの認証と認可、インベントリ、通貨管理など、ゲーム固有の他のタスクにもバックエンドサービスを使用することができます。詳細については、「ゲームクライアントサービスを設計する」を参照してください。

外部サービス

ゲームがサブスクリプションメンバーシップの確認などの目的で外部サービスに依存している場合があります。外部サービスは、バックエンドサービスと Amazon を介してゲームサーバーに情報を渡すことができます GameLift。

ゲームサーバー

ゲームサーバーソフトウェアを Amazon にアップロード GameLift すると GameLift、Amazon はそれをホスティングマシンにデプロイしてゲームセッションをホストし、プレイヤー接続を受け入れます。ゲームサーバーは Amazon と通信 GameLift してゲームセッションを開始し、新しく接続されたプレイヤーを検証し、ゲームセッション、プレイヤー接続、利用可能なリソースのステータスを報告します。

カスタムゲームサーバー GameLift は、Amazon GameLift Server SDK を使用して Amazon と通信します。ゲームクライアントは、バックエンドサービス GameLift を介して Amazon から接続の詳細を受信した後、ゲームサーバーに直接接続します。詳細については、「カスタムゲームサーバーとゲームを統合する」を参照してください。

リアルタイムサーバーは、カスタムスクリプトを実行するゲームサーバーです。ゲームに参加するとき、ゲームクライアントは Realtime Client SDK を使用してリアルタイムサーバーに直接接続します。詳細については、「Amazon GameLift リアルタイムサーバーとのゲームの統合」を参照してください。

ホスト管理ツール

ホスティングリソースをセットアップおよび管理するとき、ゲームの所有者はホスティング管理ツールを使用して、ゲームサーバーのビルドまたはスクリプト、フリート、マッチメーキング、キューを管理します。 AWS SDK とコンソールの Amazon GameLift ツールセットには、ホスティングリソースを管理するための複数の方法が用意されています。個別のゲームサーバーにリモートでアクセスしてトラブルシューティングを行うこともできます。

セットアップ

Amazon GameLift を使用してマルチプレイヤーゲームをホストするように AWS アカウント をセットアップするためのヘルプを提供します。

> **Tip**
>
> Amazon GameLift ゲームサーバーホスティングを試すには、「Amazon GameLift の開始方法」を参照してください。

トピック

のセットアップ AWS アカウント

Amazon の使用を開始するには GameLift、 を作成してセットアップします AWS アカウント。 AWS アカウントの作成には料金はかかりません。このセクションでは、アカウントの作成、ユーザーの設定、アクセス許可の設定について順を追って説明します。

トピック

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [アカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

のセキュリティ保護 AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「ルートユーザーとしてサインインする」を参照してください。
2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、「IAM ユーザーガイド」の AWS アカウント 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。
2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、「 AWS サインインユーザーガイド」の AWS 「 アクセスポータルへのサインイン」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。
2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの参加」を参照してください。

Amazon のユーザーアクセス許可を管理する GameLift

Amazon GameLift リソースの必要に応じて、追加のユーザーを作成するか、既存のユーザーにアクセス許可を拡張します。ベストプラクティス (IAM におけるセキュリティのベストプラクティス) と

して、すべてのユーザーに最小特権のアクセス権限を適用します。アクセス許可構文のガイダンスについては、「Amazon GameLift の IAM アクセス許可の例」を参照してください。

以下の手順を使用して、 AWS アカウントのユーザーの管理方法に基づいてユーザーアクセス許可を設定します。

アクセス権限を付与するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM Identity Center :

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。

- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については、「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

IAM ユーザーと作業する場合、ベストプラクティスとして、個々のユーザーではなく、常にロールまたはユーザーグループにアクセス許可を付与してください。

ユーザーにプログラムによるアクセスをセットアップする

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 にアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 ユーザーガイド」の AWS CLI 「 を使用するための の設定 AWS IAM Identity CenterAWS Command Line Interface 」を参照してください。 • AWS SDKs 、ツール、AWS APIs「 SDK とツールのリファレンスガイド」の「IAM Identity Center 認証」を参照してください。AWS SDKs
IAM	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、、AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールのリファレンスガイド」の「長期的な認証情報を使

プログラマチックアクセス権を必要とするユーザー	目的	方法
		用した認証」を参照してください。 AWS SDKs • AWS APIsユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

アクセスキーを使用する場合は、AWS 「アクセスキーを管理するためのベストプラクティス」を参照してください。

ゲームへのプログラムによるアクセスをセットアップする

ほとんどのゲームは、 AWS SDKs GameLift を使用して Amazon と通信するためにバックエンドサービスを使用します。例えば、(ゲームクライアントに代わって) バックエンドサービスを使用して、ゲームセッションをリクエストしたり、プレイヤーをゲームに参加させたり、その他のタスクを実行します。これらのサービスには、Amazon GameLift サービス APIsへの呼び出しを認証するためのプログラムによるアクセスとセキュリティ認証情報が必要です。

Amazon では GameLift、 AWS Identity and Access Management (IAM) でプレイヤーユーザーを作成して、このアクセスを管理します。次のいずれかのオプションを使用して、プレイヤーのユーザーアクセス許可を管理します。

- プレイヤーユーザーアクセス許可を持つ IAM ロールを作成し、必要に応じてプレイヤーユーザーがそのロールを引き受けられるようにします。バックエンドサービスには、Amazon にリクエストを行う前に、このロールを引き受けるコードが含まれている必要があります GameLift。セキュリティのベストプラクティスに従い、ロールは限定的で一時的なアクセスを提供します。ロールは、AWS リソース (IAM ロール) または の外部 AWS （ IAM Roles Anywhere) で実行されているワークロードに使用できます。
- プレイヤーユーザーアクセス許可を持つ IAM ユーザーグループを作成して、そのプレイヤーユーザーをグループに追加します。このオプションでは、プレイヤーユーザーに長期的な認証情報が提供されます。これは、Amazon との通信時にバックエンドサービスが保存および使用する必要があります GameLift。

アクセス許可ポリシーの構文については、「プレイヤーユーザーアクセス許可の例」を参照してください。

ワークロードで使用するアクセス許可の管理の詳細については、「IAM アイデンティティ: IAM の一時認証情報」を参照してください。

Amazon GameLift の IAM アクセス許可の例

これらの例の構文を使用して、Amazon GameLift リソースにアクセスする必要があるユーザーに AWS Identity and Access Management (IAM) アクセス許可を設定します。ユーザーアクセス許可の管理に関する詳細は、「Amazon のユーザーアクセス許可を管理する GameLift」を参照してください。IAM アイデンティティセンター以外のユーザーのアクセス許可を管理する場合、ベストプラクティスとして、個々のユーザーではなく、常に IAM ロールまたはユーザーグループにアクセス許可をアタッチしてください。

Amazon GameLift FleetIQ をスタンドアロンソリューションとして使用している場合は、「Amazon GameLift FleetIQ 用に AWS アカウント をセットアップする」を参照してください。

管理者アクセス許可の例

これらの例では、ユーザーは Amazon GameLift ゲームホスティングリソースを管理できます。

Example Amazon GameLift リソースのアクセス許可の構文

次の例では、すべての Amazon GameLift リソースにアクセスを拡張しています。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "gamelift:*",
    "Resource": "*"
  }
}
```

Example デフォルトでは有効になっていないリージョンをサポートする Amazon GameLift リソースアクセス許可の構文

次の例では、デフォルトで有効になっていないすべての Amazon GameLift リソースと AWS リージョンへのアクセスを拡張します。デフォルトで有効になっていないリージョンとそれを有効にする

方法についての詳細は、AWS 全般のリファレンス の「AWS リージョン の管理」を参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "ec2:DescribeRegions",
      "gamelift:*"
    ],
    "Resource": "*"
  }
}
```

Example Amazon GameLift リソースと **`PassRole`** ア クセス許可の構文

次の例では、すべての Amazon GameLift リソースへのアクセスを拡張し、ユーザーが IAM サービスロールを Amazon GameLift に渡せるようにします。「Amazon の IAM サービスロールを設定する GameLift」で説明されているように、サービスロールを使用すると、Amazon GameLift がユーザーに代わって他のリソースやサービスにアクセスする能力が制限されます。例えば、`CreateBuild` リクエストに応答する場合、Amazon GameLift は Amazon S3 バケット内のビルドファイルにアクセスする必要があります。`PassRole` アクションの詳細については、IAM ユーザーガイドの「IAM: IAM ロールを特定の AWS サービスに渡す」

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "gamelift:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "gamelift.amazonaws.com"
        }
      }
```

```
    }
  ]
}
```

プレイヤーユーザーアクセス許可の例

これらの例により、バックエンドサービスまたはその他のエンティティが Amazon GameLift API に API コールを行うことができます。ゲームセッション、プレイヤーセッション、マッチメーキングを管理する一般的なシナリオを網羅しています。詳細については、「ゲームへのプログラムによるアクセスをセットアップする」を参照してください。

Example ゲームセッションプレイスメントアクセス許可の構文

次の例では、ゲームセッションプレイスメントキューを使用してゲームセッションを作成し、プレイヤーセッションを管理する Amazon GameLift API へのアクセスを拡張します。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "PlayerPermissionsForGameSessionPlacements",
    "Effect": "Allow",
    "Action": [
      "gamelift:StartGameSessionPlacement",
      "gamelift:DescribeGameSessionPlacement",
      "gamelift:StopGameSessionPlacement",
      "gamelift:CreatePlayerSession",
      "gamelift:CreatePlayerSessions",
      "gamelift:DescribeGameSessions"
    ],
    "Resource": "*"
  }
}
```

Example マッチメーキングアクセス許可の構文

次の例では、FlexMatch のマッチメーキングアクティビティを管理する Amazon GameLift API へのアクセスを拡張しています。FlexMatch は、新規または既存のゲームセッションのプレイヤーをマッチングし、Amazon GameLift でホストされているゲームのゲームセッションプレイスメントを開始します。FlexMatch の詳細については、「GameLift FlexMatch とは」を参照してください。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "PlayerPermissionsForGameSessionMatchmaking",
    "Effect": "Allow",
    "Action": [
      "gamelift:StartMatchmaking",
      "gamelift:DescribeMatchmaking",
      "gamelift:StopMatchmaking",
      "gamelift:AcceptMatch",
      "gamelift:StartMatchBackfill",
      "gamelift:DescribeGameSessions"
    ],
    "Resource": "*"
  }
}
```

Example ゲームセッションの手動プレイスメントアクセス許可の構文

次の例では、指定されたフリートでゲームセッションとプレイヤーセッションを手動で作成する Amazon GameLift API へのアクセスを拡張しています。このシナリオは、プレイヤーが利用可能なゲームセッションのリストから選択して参加できるゲーム (「リスト & ピック」方式) など、プレイスメントキューを使用しないゲームをサポートします。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "PlayerPermissionsForManualGameSessions",
    "Effect": "Allow",
    "Action": [
      "gamelift:CreateGameSession",
      "gamelift:DescribeGameSessions",
      "gamelift:SearchGameSessions",
      "gamelift:CreatePlayerSession",
      "gamelift:CreatePlayerSessions",
      "gamelift:DescribePlayerSessions"
    ],
    "Resource": "*"
  }
}
```

Amazon の IAM サービスロールを設定する GameLift

Amazon の一部 GameLift の機能では、所有するAWSリソースへの制限付きアクセスを拡張する必要があります。そのためには、AWS Identity and Access Management (IAM) ロールを作成します。IAM ロールは、特定の許可があり、アカウントで作成できるもう 1 つの IAM アイデンティティです。IAM ロールは、ID が AWS で実行できることとできないことを決定する許可ポリシーを持つ AWS ID であるという点で IAM ユーザーと似ています。ただし、ユーザーは 1 人の特定の人に一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。また、ロールには標準の長期認証情報 (パスワードやアクセスキーなど) も関連付けられません。代わりに、ロールを引き受けると、ロールセッション用の一時的なセキュリティ認証情報が提供されます。

このトピックでは、Amazon GameLift マネージドフリートで使用できるロールを作成する方法について説明します。Amazon GameLift FleetIQ を使用して Amazon Elastic Compute Cloud (Amazon EC2) インスタンスでのゲームホスティングを最適化する場合は、「Amazon GameLift FleetIQ 用にをセットアップするAWS アカウント」を参照してください。

次の手順では、カスタムアクセス許可ポリシーと Amazon がロールを GameLift 引き受けることを許可する信頼ポリシーを使用してロールを作成します。

カスタム IAM ロールを作成する

ステップ 1: 許可ポリシーを作成する。

JSON ポリシーエディタでポリシーを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。

   初めて [ポリシー] を選択する場合には、[管理ポリシーにようこそ] ページが表示されます。[今すぐ始める] を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. [ポリシーエディタ] セクションで、[JSON] オプションを選択します。
5. JSON ポリシードキュメントを入力するか貼り付けます。IAM ポリシー言語の詳細については、「IAM JSON ポリシーリファレンス」を参照してください。
6. ポリシーの検証中に生成されたセキュリティ警告、エラー、または一般的な警告を解決してから、[Next] (次へ) を選択します。

> **Note**
>
> いつでも [Visual] と [JSON] エディタオプションを切り替えることができます。ただし、[Visual] エディタで [次] に変更または選択した場合、IAM はポリシーを再構成して visual エディタに合わせて最適化することがあります。詳細については、『IAM ユーザーガイド』の「ポリシーの再構成」を参照してください。

7. （オプション）AWS Management Console でポリシーを作成または編集するときに、AWS CloudFormation テンプレートで使用できる JSON または YAML ポリシーテンプレートを生成できます。

   これを行うには、ポリシーエディタでアクション を選択し、テンプレートの生成 CloudFormation を選択します。AWS CloudFormation の詳細については、『AWS CloudFormation ユーザーガイド』の「AWS Identity and Access Management リソースタイプリファレンス」を参照してください。
8. ポリシーにアクセス権限を追加し終えたら、[次へ] を選択します。
9. [確認と作成] ページで、作成するポリシーの [ポリシー名] と [説明] (オプション) を入力します。[このポリシーで定義されているアクセス許可] を確認して、ポリシーによって付与されたアクセス許可を確認します。
10. (オプション) タグをキー - 値のペアとしてアタッチして、メタデータをポリシーに追加します。IAM でのタグの使用に関する詳細については、『IAM ユーザーガイド』の「IAM リソースにタグを付ける」を参照してください。
11. [Create Policy (ポリシーを作成)] をクリックして、新しいポリシーを保存します。

ステップ 2: Amazon が引き受け GameLift ることができるロールを作成します。

1. IAM コンソールのナビゲーションペインで、[ロール]、[ロールを作成] を選択します。
2. [信頼されたエンティティを選択] ページで、[カスタム信頼ポリシー] オプションを選択します。これを選択すると、[カスタム信頼ポリシー] エディタが開きます。
3. デフォルトの JSON 構文を次の構文に置き換え、[次へ] を選択して続行します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "gamelift.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

4. [アクセス許可を追加] ページで、ステップ 1 で作成したアクセス許可ポリシーを探して選択します。[次へ] を選択して続行します。
5. [名前、確認、および作成] ページで、作成するロールの [ポリシー名] と [説明] (オプション) を入力します。[信頼エンティティ] と [追加されたアクセス許可] を確認します。
6. [ロールを作成] を選択して新しいロールを保存します。

アクセス許可ポリシーの構文

- Amazon がサービスロール GameLift を引き受けるアクセス許可

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "gamelift.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- デフォルトでは有効になっていない AWS リージョンへのアクセス許可

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
```

```
            "gamelift.amazonaws.com",
            "gamelift.ap-east-1.amazonaws.com",
            "gamelift.me-south-1.amazonaws.com",
            "gamelift.af-south-1.amazonaws.com",
            "gamelift.eu-south-1.amazonaws.com"
          ]
        },
        "Action": "sts:AssumeRole"
      }
    ]
}
```

Amazon での開発サポート GameLift

Amazon GameLift には、マネージドゲームホスティングソリューションで使用できる一連の SDKs が用意されています。Amazon GameLift SDKs を使用して、Amazon GameLift ホスティングサービスとやり取りする必要があるマルチプレイヤーゲームサーバー、ゲームクライアント、ゲームサービスに必要な機能を追加します。

Amazon GameLift SDK のバージョンと SDK の互換性に関する最新情報については、「」を参照してくださいAmazon GameLift リリースノート。

カスタム ゲームサーバーの場合

Amazon サーバー SDK を使用して 64 ビットのカスタムゲーム GameLift サーバーを作成してデプロイします。サーバー SDK と統合され、ホスティング用にデプロイされたゲームサーバーは、Amazon GameLift サービスと通信してゲームセッションを開始および管理できます。ゲームサーバー SDK の統合に関する情報については、「Amazon 用のゲームの準備 GameLift」を参照してください。

開発用オペレーティングシステム

- Windows
- Linux

サポートされているプログラミング言語

Amazon GameLift は、以下の言語用のサーバー SDK を提供しています。 Server SDKs。バージョン固有の情報については、各パッケージに含まれている Readme ファイルを参照してください。

- C++ サーバー SDK
  - SDK リファレンス
  - SDK 統合
- C# サーバー SDK (バージョンは.NET 4 と.NET 6 をサポートしている場合があります)
  - SDK リファレンス
  - SDK 統合
- Go
  - SDK リファレンス
  - SDK 統合

サポートされるゲームエンジン

C++、C#、または Go ライブラリをサポートするすべてのエンジンで、言語固有の SDK を使用できます。さらに、Amazon GameLift は、次のゲームエンジンプラグインを提供しています: Amazon GameLift プラグインをダウンロードする

- Unity
  - Unity 用 C# サーバー SDK プラグインは、Unity パッケージマネージャーを使用してインストールできる構築済みのライブラリを備えた軽量プラグインです。このプラグインは Windows および Mac OS 用の 2020.3 LTS、2021.3 LTS、2022.3 LTS の Unity バージョンで使用します。Unity の.NET Frameworkと.NET Standard (.NET Standard 2.1 と .NET 4.x) の.NET Standard プロファイルをサポートしています。
    - Unity プロジェクトに Amazon GameLift を統合する
  - Unity 2021.3 LTS および 2022.3 LTS 用のスタンドアロンプラグインは、Unity 用に構築された C# SDK ライブラリと、ホスティング用の Amazon GameLift リソースを設定およびデプロイするための GUI 要素を備えたフル機能のプラグインです。
    - サーバー SDK 5.x 用 Unity ガイド用 Amazon GameLift プラグイン
    - C# 用 Amazon GameLift サーバー SDK リファレンス
- Unreal Engine
  - Unreal 用 C++ サーバー SDK プラグインは、C++ Unreal ソースコードで構成される軽量プラグインで、これをライブラリに組み込むことで、アンリアルエンジンバージョン 4、5、5.1 で使用できます。
    - Amazon GameLift を Unreal Engine プロジェクトに統合する
    - Amazon GameLift Unreal Engine サーバー SDK 5.x リファレンス

- Unreal Engine 5.0、5.1、および 5.2 用のスタンドアロンプラグインは、Unreal サーバー SDK ライブラリおよび AWS SDK 用の C++ を備えたフル機能のプラグインです。プラグインは Unreal エディタにインストールされ、ホスティング用の Amazon GameLift リソースを設定およびデプロイするための UI 要素とサポート資料が含まれています。
  - Unreal Engine 用 Amazon GameLift プラグインとのゲームの統合
  - Amazon GameLift Unreal Engine サーバー SDK 5.x リファレンス

[Game server operating systems] (ゲーム サーバーオペレーティングシステム)

Amazon GameLift Server SDK を使用して、次のプラットフォームで実行するゲームサーバーを構築します。

- Windows Server 2016
- Amazon Linux 2023
- Amazon Linux 2 (AL2)
- Windows Server 2012 (Windows 2012 の Amazon GameLift FAQ を参照)
- Amazon Linux (AL1) (AL1 の Amazon GameLift FAQ」を参照)

カスタムクライアントサービスの場合

Amazon GameLift API で AWS SDK を使用して 64 ビットのカスタムクライアントサービスを作成します。この SDK により、クライアントサービスはゲームセッションを管理し、Amazon でホストされているゲームにプレイヤーを参加させることができます GameLift。開始するには、 AWS SDK をダウンロードします。Amazon で SDK を使用する方法の詳細については GameLift、「Amazon GameLift API リファレンス」を参照してください。

リアルタイムサーバー

マルチプレイヤーゲームをホストするためにリアルタイムサーバーを設定およびデプロイします。ゲームクライアントがリアルタイムサーバーに接続できるようにするには、Amazon GameLift Realtime Client SDK を使用します。ゲームクライアントは、この SDK を使用して、リアルタイムサーバー、およびサーバーに接続する他のゲームクライアントとメッセージを交換します。開始するには、Amazon GameLift Realtime Client SDK をダウンロードします。設定情報については、「リアルタイムサーバー用のゲームクライアントの統合」を参照してください。

[SDK support] (SDK サポート)

リアルタイムクライアント SDK には、以下の言語の出典が含まれています。

- C# (.NET)

[Development environments] (開発環境)

次のサポートされているこれらの開発オペレーティングシステムおよびゲームエンジンに必要なソースから SDK を構築します。

- [オペレーティングシステム] – Windows、Linux、Android、iOS。
- [Game engines] (ゲームエンジン) - Unity、C# ライブラリをsupportするエンジン。

[Game server operating systems] (ゲーム サーバーオペレーティングシステム)

リアルタイムサーバーを以下のプラットフォームを実行するホストリソースにデプロイできます。

- Amazon Linux
- Amazon Linux 2

ゲームホスティングコストを管理する

AWS 請求書にはゲームホスティングコストが反映されます。当月の推定請求額と前月の最終請求額は、請求コンソール https://console.aws.amazon.com/billing/ で確認できます。AWS コストの管理に役立つツールとリソースの詳細については、「AWS Billing ユーザーガイド」を参照してください。このガイドは、リソース消費の確認、将来の使用の確立、およびスケーリングニーズの判断に役立ちます。

特に、Amazon GameLift サービスのコスト管理に役立つ以下のヒントを検討してください。

請求アラートを作成して、使用状況をモニタリングする

AWS 無料利用枠の使用状況アラートを設定して、使用量が Amazon GameLift およびその他の の無料利用枠の制限に近づいたり、超えたりしたときに通知しますAWS のサービス。使用レベルに基づいてアラートがアクションを実行するように設定できます。例えば、無料利用枠の上限に達したときに、予算を自動的にゼロに設定できます。

使用量がカスタムしきい値に達したときに通知を受け取るように Amazon CloudWatch 請求アラートを設定することもできます。

詳細については、「AWS Billingユーザーガイド」のこれらのトピックを参照してください。

- AWS 無料利用枠の使用状況の追跡
- 請求アラート設定

Amazon GameLift フリートあたりのコストを追跡する

AWS コスト配分タグを使用して、 GameLift Amazon EC2 フリートやその他の EC2 リソースに基づいてゲームホスティングコストを整理および追跡します。フリートに個別またはグループごとにタグ付けることで、割り当てられたタグに基づいてコストを分類するコスト配分レポートを作成できます。このタイプのレポートを使うと、フリートがホスティングコストにどの程度寄与しているかを特定できます。タグは、AWS Cost Explorerの表示をフィルタリングするために使用することもできます。

詳細については、以下のトピックを参照してください。

- 「AWS Billing ユーザーガイド」の「AWSコスト配分タグの使用」
- 「AWS Cost Management ユーザーガイド」の「AWS Cost Explorer を用いてコストを分析する」

未使用のフリートキャパシティをゼロに設定します。

フリートが使用されていないときでも、ゲームセッションをホストしているフリートには引き続きコストがかかる可能性があります。不必要な料金が発生しないように、使用していないときはフリートをスケールダウンします。自動スケーリングを使用する場合は、このアクティビティを停止し、フリートのキャパシティを手動で設定してください。

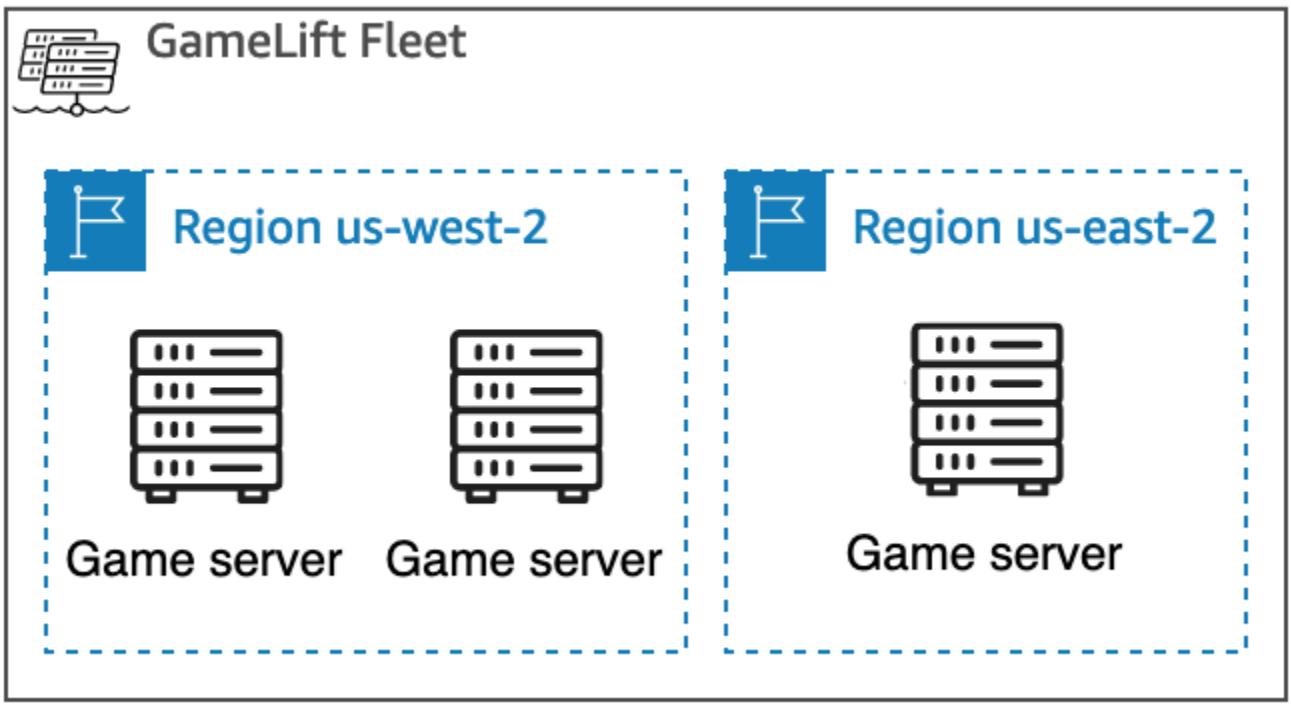
Amazon GameLift ホスティングロケーション

Amazon GameLift は、複数の AWS リージョンおよび Local Zones で使用できます。ロケーションの完全なリストについては、「」の「Amazon GameLift エンドポイントとクォータ」を参照してくださいAWS 全般のリファレンス。

Amazon GameLift ホスティング

Amazon GameLift フリートを作成すると、Amazon は現在の にフリートのリソース GameLift を作成しますAWS リージョン。Amazon はこのリージョンをフリートのホームリージョン と GameLift 呼び出します。フリートを管理するには、そのホームリージョンからアクセスします。

マルチロケーションフリートは、フリートのホームリージョンに加えて、他のロケーションにインスタンスをデプロイできます。マルチロケーションフリートでは、ロケーションごとにキャパシティを個別に管理でき、ロケーションごとにゲームセッションを配置できます。マルチロケーションフリートは、Amazon が GameLift サポートする任意のリージョンまたはローカルゾーンにリモートロケーションを持つことができます。次の図は、2 つのリージョンにリソースを持つマルチロケーションフリートを示しています。この図では、us-west-2 リージョンに 2 つのゲームサーバーが含まれ、us-east-2 リージョンには 1 つのゲームサーバーがあります。



デフォルトでは有効になっていないリージョンのインスタンスでマルチロケーションフリートを使用する場合は、それらのリージョンを自分の AWS アカウント リージョンで有効にする必要があります。また、Amazon GameLift 管理者ポリシーで ec2:DescribeRegionsアクションを許可する必要があります。デフォルトで有効になっていないリージョンとそれを有効にする方法についての詳細は、AWS 全般のリファレンス の「AWS リージョン の管理」を参照してください。デフォルトで有効になっていないリージョンのポリシーの例については、「管理者アクセス許可の例」を参照してください。

> **Important**
>
> デフォルトで有効になっていないリージョンを使用するには、AWS アカウント でそのリージョンを有効にします。

- 2022 年 2 月 28 日より前に作成した、有効化されていないリージョンのあるフリートは影響を受けません。
- 新しいマルチロケーションフリートを作成したり、既存のマルチロケーションフリートを更新するには、まず、使用することを選択したリージョンをすべて有効にします。

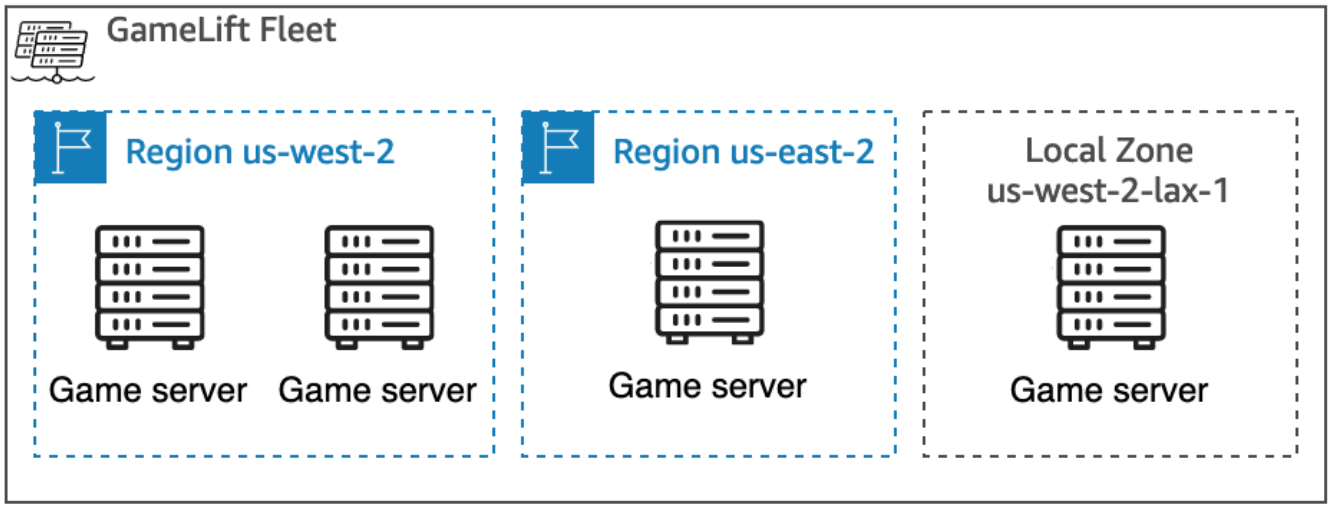
ゲームセッション配置では、Amazon が GameLift サポートする任意の場所にゲームセッションキューを作成できます。Amazon GameLift は、キューを作成した場所からゲームセッションを配置します。

ローカルゾーン

ローカルゾーンは、ユーザーに近い場所に位置する AWS リージョン の拡張です。ローカルゾーンは、インターネットへの独自の接続を持ちます。また、ローカルゾーンは AWS Direct Connect をサポートしているため、ローカルゾーンで作成されたリソースは、低レイテンシーの通信でローカルユーザーにサービスを提供できます。詳細については、AWS Local Zonesを参照してください。

ローカルゾーンのコードは、そのリージョンコードの後に、物理的な場所を示す識別子が続きます。例えば、`us-west-2-lax-1` ローカルゾーンはロサンゼルスにあります。使用可能なローカルゾーンのリストについては、「利用可能な Local Zones」を参照してください。

Amazon は、フリート用に選択した各ロケーションでゲームを GameLift ホストします。次の図は、`us-west-2` リージョンに 2 台のゲームサーバー、`us-east-2` リージョンに 1 台のゲームサーバー、`us-west-2-lax-1` ローカルゾーンに 1 台のゲームサーバーがあるフリートを示しています。



利用可能な Local Zones

次の表に、利用可能なローカルゾーンとそれらの物理的なロケーションを示します。

ローカルゾーン	ロケーション (都市圏名)	
us-east-1-atl-1	アトランタ	
us-east-1-chi-1	シカゴ	
us-east-1-dfw-1	ダラス	
us-east-1-iah-1	ヒューストン	
us-east-1-mci-1	カンザスシティ	
us-west-2-den-1	デンバー	
us-west-2-lax-1	ロサンゼルス	
us-west-2-phx-1	フェニックス	

Amazon GameLift Anywhere

Amazon GameLift Anywhere を使用して独自のハードウェアでフリートを作成し、Amazon を使用してゲームビルド、スクリプト、ゲームサーバー、クライアントを管理できます GameLift。Amazon GameLift Anywhere は、Amazon が GameLift サポートするすべての リージョンで利用できます。Anywhere フリートの作成とゲームサーバー統合のテストの詳細については、「Amazon GameLift Anywhere フリートを作成する」と「Amazon GameLift Anywhere フリートを使用して統合をテストする」を参照してください。

Amazon GameLift Anywhere では、Amazon GameLift 統合ゲームサーバーのホストに使用しているハードウェアの物理的な場所を表すカスタムロケーションを作成します。

Amazon GameLift FlexMatch

では FlexMatch、マッチ生成ゲームセッションを Amazon GameLift がサポートする任意の場所でホストできます。実際のマッチメーキングアクティビティは、マッチメーカーリソースの作成AWS リージョンを選択した で行われます。Amazon は一致リクエストをマッチメーカーに GameLift ルー

ティングし、その場所で処理します。Amazon の詳細については GameLift FlexMatch、「Amazon とは」を参照してください GameLift FlexMatch。

AWS リージョン FlexMatch リソースをサポートする

中国 GameLift での Amazon

Sinnet が運営する中国 (北京) リージョン、または NWCD が運営する中国 (寧夏) リージョンの GameLift リソースに Amazon を使用する場合は、別の AWS (中国) アカウントが必要です。中国リージョンでは、一部の機能が使用できないことに注意してください。これらのリージョン GameLift での Amazon の使用の詳細については、以下のリソースを参照してください。

- 中国の Amazon Web Services
- Amazon GameLift (中国での Amazon Web Services の開始方法)

Amazon GameLift の開始方法

Amazon GameLift を自分のゲームに使用する前に、次の例を試してみることをお勧めします。カスタムゲームサーバーの例は、Amazon GameLift コンソールでゲームホスティングの使用体験を提供します。リアルタイムサーバーの例では、リアルタイムサーバーを使用してホスティングできるようにゲームを準備する方法がわかります。

自分のゲーム用に Amazon GameLift を使用開始するには、「Amazon GameLift マネージドホスティングロードマップ」を参照してください。

カスタムゲームサーバーの例

この例は、Amazon GameLift でのライブカスタムゲームを紹介します。この例では、次の手順を具体的に説明します。

- サンプルのゲームビルドの作成。
- ゲームサーバーを実行するフリートの作成。
- サンプルゲームクライアントからゲームサーバーへの接続。
- フリートとゲームセッションのメトリクスの確認。

これらのステップを実行した後、複数のゲームクライアントを立ち上げてゲームをプレイし、ホスティングデータを生成できます。その後、Amazon GameLift コンソールで、ホスティングリソースを表示したり、メトリクスを追跡したり、ホスティングキャパシティをスケールする方法を試すことができます。

使用を開始するには、 [Amazon GameLift コンソール] にサインインします。

リアルタイムサーバーのサンプルゲーム

このサンプルは、Mega Frog Race という名前の完全なマルチプレイヤーゲームで、ソースコードが含まれています。このサンプルは、ゲームクライアントをリアルタイムサーバーと統合する方法を示しています。また、このサンプルゲームは、FlexMatch などの他の Amazon GameLift の機能を試す開始ポイントとして使用することもできます。

実践的なチュートリアルについては、GameTech ブログ記事 AWS の「数行の JavaScript のみでマルチプレイヤーモバイルゲーム用にサーバーを作成する」を参照してください。

Mega Frog Race のソースコードについては、GitHub リポジトリ」を参照してください。

ソースコードには次の部分が含まれます。

- ゲーム クライアント – Unity で作成した C++ ゲームクライアントのソースコード。ゲームクライアントはゲームセッション接続情報を取得し、サーバーに接続して、他のプレイヤーとアップデートを交換します。
- バックエンドサービス – Amazon GameLift への直接 API コールを管理する AWS Lambda 関数のソースコード。
- リアルタイムスクリプト - ゲームのリアルタイムサーバーのフリートを設定するソーススクリプトファイル。このスクリプトには、リアルタイムサーバーが Amazon GameLift と通信し、ゲームをホストするために最低限必要な設定が含まれています。

Amazon GameLift マネージドホスティングロードマップ

このトピックは、セッションベースのマルチプレイヤーゲームのさまざまな Amazon GameLift ホスティングオプションから選択するのに役立ちます。このセクションの残りのトピックでは、Amazon GameLift をマネージドホスティングに使用する方法について説明します。

ゲームを本番環境にローンチする準備を始める前に、ローンチアンケートに記入して Amazon GameLift チームとの連携を開始してください。

トピック

ホスティングオプションを選択する

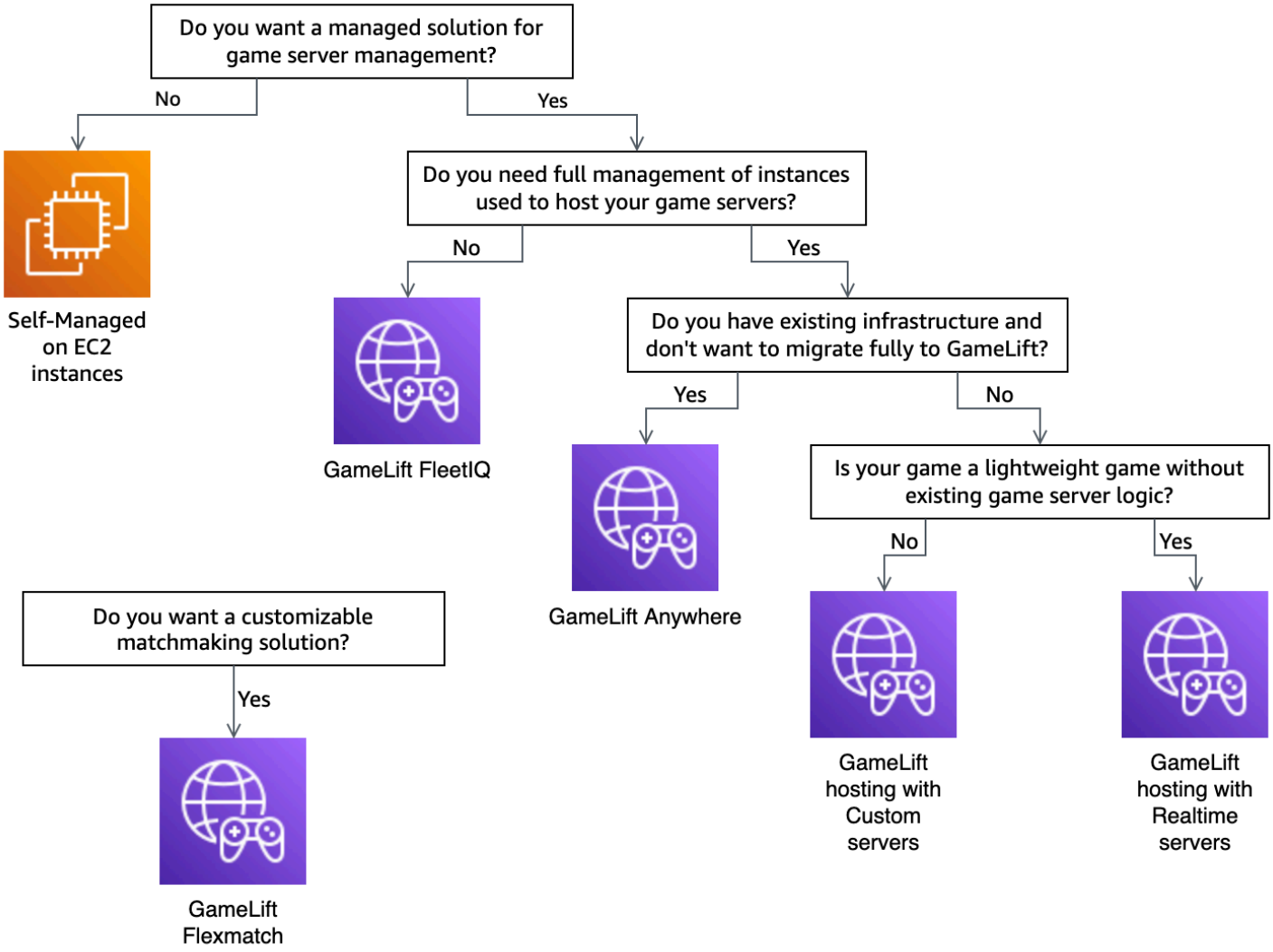
次のフローチャートは、各自のユースケースに合った Amazon GameLift ソリューションを見つけるための質問をまとめたものです。

1. ゲームサーバー管理用のマネージドソリューションが必要ですか？
   - はい – ステップ 2 に進んでください。
   - いいえ – Amazon EC2 インスタンス上のセルフマネージドゲームサーバーを検討してください。
2. ゲームサーバーをホストするインスタンスを完全に制御する必要がありますか？
   - はい – Amazon GameLift FleetIQ を検討してください。
   - いいえ – ステップ 3 に進んでください。
3. Amazon GameLift で使用したい既存のインフラストラクチャはありますか？
   - はい – Amazon GameLift Anywhere を検討してください。

- いいえ – ステップ 4 に進んでください。

4. 既存のゲームサーバーロジックがなくてもゲームは軽量ですか？

   - はい – リアルタイムサーバーを検討してください。
   - いいえ – カスタムサーバーを検討してください。



フローチャートに記載されている Amazon GameLift ホスティングオプションの一部について、さらに詳しく説明します。

Amazon GameLift Anywhere

Amazon GameLift を使用すると、Amazon GameLift Anywhere 管理ツールの利点を活かして、独自のハードウェアでゲームをホストできます。Anywhere フリートを使用してゲームサーバーを

繰り返しテストすることもできます。詳細については、「Amazon GameLift Anywhere フリートを作成する」を参照してください。

マネージド Amazon GameLift

マネージド Amazon GameLift ホスティングには次の 2 つのオプションがあります。

カスタムサーバー – Amazon GameLift は、ゲームサーバーのバイナリを実行するカスタムサーバーをホストします。

リアルタイムサーバー – Amazon GameLift は軽量ゲームサーバーをホストします。

Amazon GameLift FleetIQ

フローチャートでは、リフトアンドシフト移行とは、ゲームアーキテクチャに変更を加えることができない場合の移行を指します。Amazon GameLift FleetIQ を使用すると、既存のデプロイを変更する必要が少なくなり、フリート管理用の Amazon GameLift ツールも提供されます。Amazon ES の詳細については、Amazon GameLift FleetIQ デベロッパーガイドを参照してください。

Amazon GameLift Anywhere またはマネージド Amazon GameLift を使用することに決めた場合は、Amazon GameLift 用にゲームを準備する に進みます。

Amazon GameLift 用にゲームを準備する

このトピックでは、マルチプレイヤーゲームをマネージド Amazon GameLift ホスティングと統合するための準備手順について説明します。ゲームを準備するには、ゲームと Amazon GameLift 間の通信をアクティブ化する必要があります。

カスタムゲームサーバーを準備する

ゲームセッションの開始したり、その他のタスクを実行するには、Amazon GameLift にそのステータスについて通知可能である必要があります。Amazon GameLift との通信を有効にするには、ゲームサーバープロジェクトにコードを追加します。詳細については、「カスタムゲームサーバーとゲームを統合する」を参照してください。

1. GameLiftAmazon GameLift でホストするためにカスタムゲームサーバーを準備します。

   - Amazon GameLift Server SDK を取得して、任意のプログラム言語とゲームエンジンを使用して構築します。

- ゲームサーバープロジェクトにコードを追加して、Amazon GameLift との通信を有効にします。

2. ゲームクライアントを Amazon GameLift がホストするゲームセッションに接続するように準備します。

   - AWS SDK をバックエンドサービスとゲームクライアントプロジェクトに追加します。詳細については、「クライアントサービス用に Amazon GameLift SDK をダウンロードする」を参照してください。
   - ゲームセッションで情報を取得する機能を追加して新しいゲームセッションを配置し、ゲームセッションのプレイヤー用の容量を予約します。
   - (オプション) プレイヤーのマッチメーキングには FlexMatch を使用してください。詳細については、「Amazon GameLift ホスティングと FlexMatch の統合」を参照してください。

リアルタイムサーバーを準備する

Amazon GameLift リアルタイムサーバーは、ゲームに合わせて構成できるソリューション軽量を提供します。リアルタイムサーバーは、Amazon GameLift がゲームサーバーに提供するのと同じ利点を提供しますが、ゲームサーバーのカスタマイズ性が低下します。

Amazon GameLift でホスティングするためのリアルタイムスクリプトを作成します。

リアルタイムスクリプトには、サーバー構成とオプションのカスタムゲームロジックが含まれています。リアルタイムサーバーは、ゲームセッションを開始および停止し、プレイヤーの接続を受け入れ、Amazon GameLift との通信と、ゲーム内のプレイヤー間の通信を管理するように設計されています。ゲーム用にカスタムサーバーロジックを追加するためのフックもあります。リアルタイムサーバーは Node.js と JavaScript を使用します。詳細については、リアルタイムスクリプトの作成 および Amazon GameLift との統合をテストする を参照してください。

Amazon GameLift との統合をテストする

Amazon GameLift は、ゲームサーバーをテストする際の高速イテレーションをサポートしています。このトピックでは、利用可能なテストの種類について説明します。

カスタムゲームサーバー

Amazon GameLift を使用して、ご使用の環境のどこにあるハードウェアでも Amazon GameLift ゲームホスティングアーキテクチャに統合します。Amazon GameLift Anywhere はハードウェア

を Anywhere フリートの Amazon GameLift に登録するので、ユーザーは自分のローカル開発用コンピュータを使用してテストできます。Amazon GameLift Anywhere でのテストの詳細については、「Amazon GameLift Anywhere フリートを使用して統合をテストする」を参照してください。Amazon GameLift Anywhere を使用してオンプレミスソリューションでゲームをホストする方法の詳細については、「Amazon GameLift 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。

Realtime サーバー

リアルタイムサーバーでは、いつでもスクリプトを更新できます。リアルタイムスクリプトを更新すると、Amazon GameLift は数分以内に新しいバージョンをホスティングリソースに配信します。Amazon GameLift が新しいスクリプトをデプロイすると、新しいゲームセッションはすべて新しいスクリプトバージョンを使用します。Amazon GameLift が新しいスクリプトをデプロイしたら、すぐにテストを開始できます。リアルタイムサーバーの詳細については、「Amazon GameLift リアルタイムサーバーとのゲームの統合」を参照してください。

Amazon GameLift リソースを計画してデプロイする

グローバルな Amazon GameLift リソースのデプロイを計画するには、次のヒントを使用してください。Amazon GameLift でゲームをホストできる場所については、「Amazon GameLift ホスティングロケーション」を参照してください。

Amazon GameLift リソースをデプロイするには、次のタスクを実行します。

- ゲームサーバーをパッケージ化して Amazon GameLift またはハードウェアにアップロードします。サーバーを Amazon GameLift にアップロードする場合、サーバーはフリートのホームの AWS リージョン にのみアップロードします。Amazon GameLift は、サーバーをフリート内の他のロケーションに自動的に分散します。詳細については、「Amazon GameLift のビルドとスクリプトのアップロード」を参照してください。
- ゲーム用の Amazon GameLift フリートを設計してデプロイします。使用するコンピューティングリソースの種類、デプロイ先のリージョン、キューの使用有無などのオプションを決めます。詳細については、「Amazon GameLift フリート設計ガイド」を参照してください。
- キューを作成して、新しいゲームセッションプレイスメントとスポットインスタンス戦略を管理します。詳細については、「ゲームセッションキューの設計」を参照してください。
- 自動スケーリングを有効にして、予測されるユーザーの需要に対してフリートのホスティングキャパシティを管理します。詳細については、「Amazon GameLift ホスティングキャパシティのスケーリング」を参照してください。

- ゲームに FlexMatch マッチメーキングルールを使用します。詳細については、「Amazon GameLift ホスティングと FlexMatch の統合」を参照してください。

Amazon GameLift リソースを自動的にデプロイする

Amazon GameLift リソースのグローバルデプロイを効率化するために、Infrastructure as Code (IaC) を使用してリソースを定義することをお勧めします。Amazon GameLift は AWS CloudFormation テンプレートをサポートしているため、デプロイ固有の構成に関してパラメータをテンプレートに設定できます。

AWS CloudFormation スタックのデプロイを管理するには、AWS CodePipeline などの継続的インテグレーションおよび継続的デリバリー (CI/CD) ツールとサービスを使用することもお勧めします。これにより、ゲームサーバーバイナリが構築されるたびに、自動的にデプロイするか、承認によりデプロイできます。

以下は、CI/CD ツールまたはサービスを使用して自動化できる新しいゲームサーバーバージョンの Amazon GameLift リソースデプロイの一般的なステップです。

- ゲームサーバーバイナリの構築とテスト
- Amazon GameLift またはハードウェアにバイナリをアップロードする。
- 新しいビルドに新しいフリートをデプロイする。
- 新しいフリートをデプロイした後、Amazon GameLift キューから以前のバージョンフリートを削除し、新しいフリートに置き換える。
- 以前のバージョンのフリートがすべてのゲームセッションを正常に終了したら、それらのフリートの AWS CloudFormation スタックを削除する。

AWS Cloud Development Kit (AWS CDK) を使用して Amazon GameLift リソースを定義することもできます。AWS CDKの詳細については、AWS Cloud Development Kit (AWS CDK)デベロッパーガイドを参照してください。

ゲームクライアントサービスを設計する

プレイヤーを認証し、Amazon GameLift API と通信するゲームクライアントサービスを実装することをお勧めします。カスタムゲームクライアントサービスを実装することで、次のことが可能になります。

- プレイヤーの認証をカスタマイズします。

- Amazon GameLift がゲームセッションにマッチさせて開始する方法を制御します。
- クライアントを信頼する代わりに、マッチメーキングのスキルレーティングなどのプレイヤー属性には、プレーヤーデータベースを使用します。

ゲームクライアントサービスを使用すると、ゲームクライアントが Amazon GameLift API と直接やり取りすることによって発生するセキュリティリスクも軽減されます。

プレーヤーの認証

Amazon Cognito およびプレーヤーセッション ID を使用してゲームクライアントを認証できます。プレイヤー ID のライフサイクルとプロパティを管理するには、Amazon Cognito ユーザープールを使用します。

必要に応じて、カスタム ID ソリューションを構築してそれを AWS にホストできます。API Gateway で、カスタム認証ロジックに Lambda オーソライザーも使用できます。

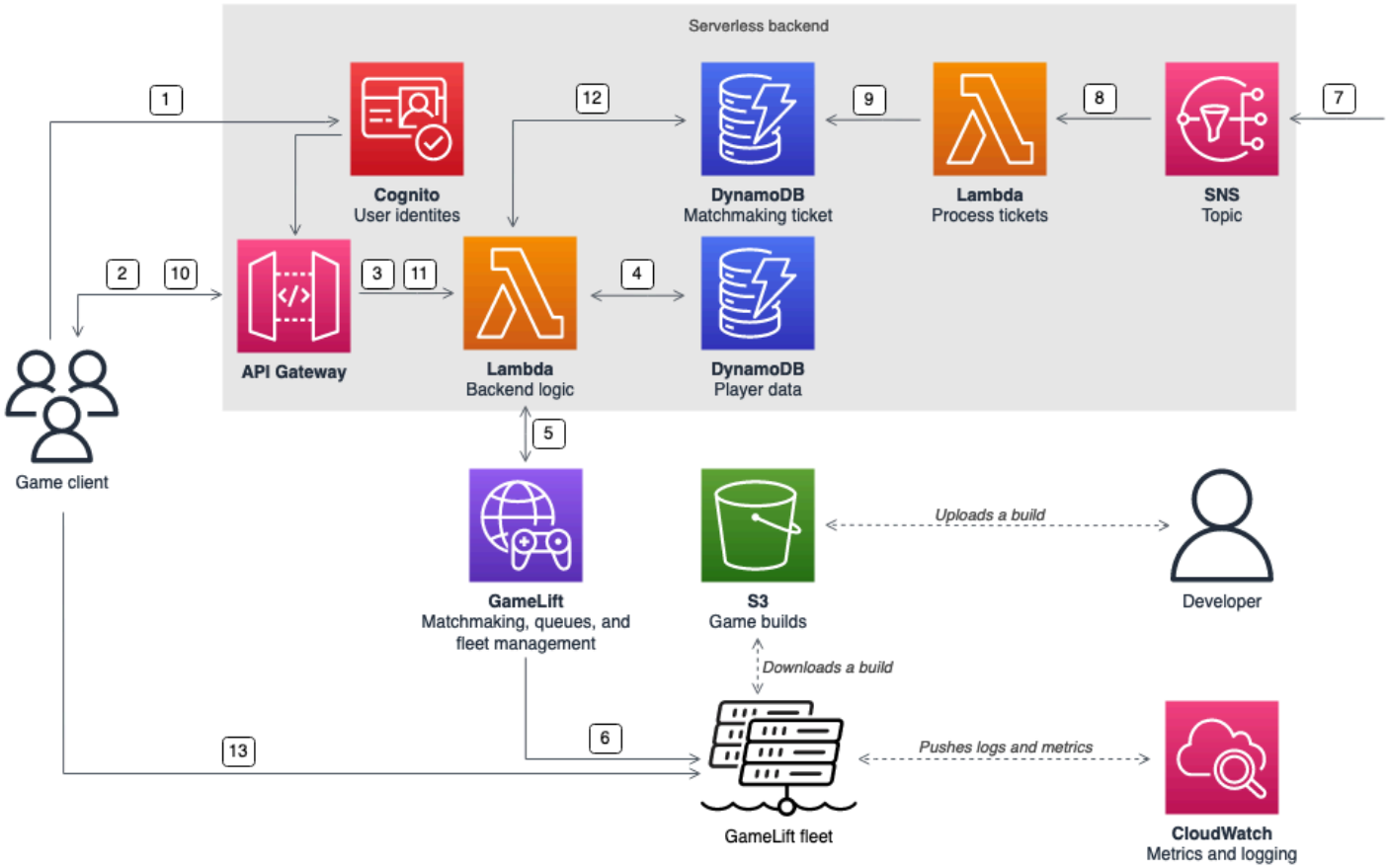
その他のリソース:

- アイデンティティプール (フェデレーティッド ID) を使用する (Amazon Cognito 開発者ガイド)
- ユーザープールの開始方法 (Amazon Cognito 開発者ガイド)
- Amazon Cognito でゲームサーバーのプレイヤー認証を設定する方法 (ゲームブログの AWS)

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

サーバーレスクライアントサービスアーキテクチャを使用すると、バックエンドは Amazon GameLift API に直接アクセスするのではなく、高度にスケーラブルなデータベースからマッチメーキングチケットのステータスを表示することができます。

次の図は、Amazon GameLift フリートで実行されているゲームにプレイヤーをマッチングさせる AWS のサービス で構築されたサーバーレスバックエンドを示しています。次のリストには、図内の各番号付きの吹き出しについての説明が含まれます。この例を試すには、GitHub の「AWS のマルチプレイヤーセッションベースのゲームホスティング」を参照してください。



1. ゲームクライアントは Amazon Cognito アイデンティティプールから Amazon Cognito ユーザー ID をリクエストします。
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、Amazon API Gateway API を介してゲームセッションを要求します。
3. API Gateway は AWS Lambda 関数を呼び出します。
4. Lambda 関数は、Amazon DynamoDB テーブルからプレイヤーデータをリクエストします。この関数は、リクエストコンテキストデータに Amazon Cognito ID を提供します。
5. Lambda 関数は Amazon GameLift FlexMatch マッチメーキングを通じてマッチをリクエストします。
6. FlexMatch は、適切なレイテンシーを持つプレイヤーグループをマッチさせた後、Amazon GameLift キューを通じてゲームセッションのリクエストエストします。キューには、1 つ以上の AWS リージョン ロケーションが登録されているフリートがあります。
7. Amazon GameLift がフリートのいずれかの口ケーションにセッションを配置すると、Amazon GameLift は Amazon Simple Notification Service (Amazon SNS) トピックにイベント通知を送信します。

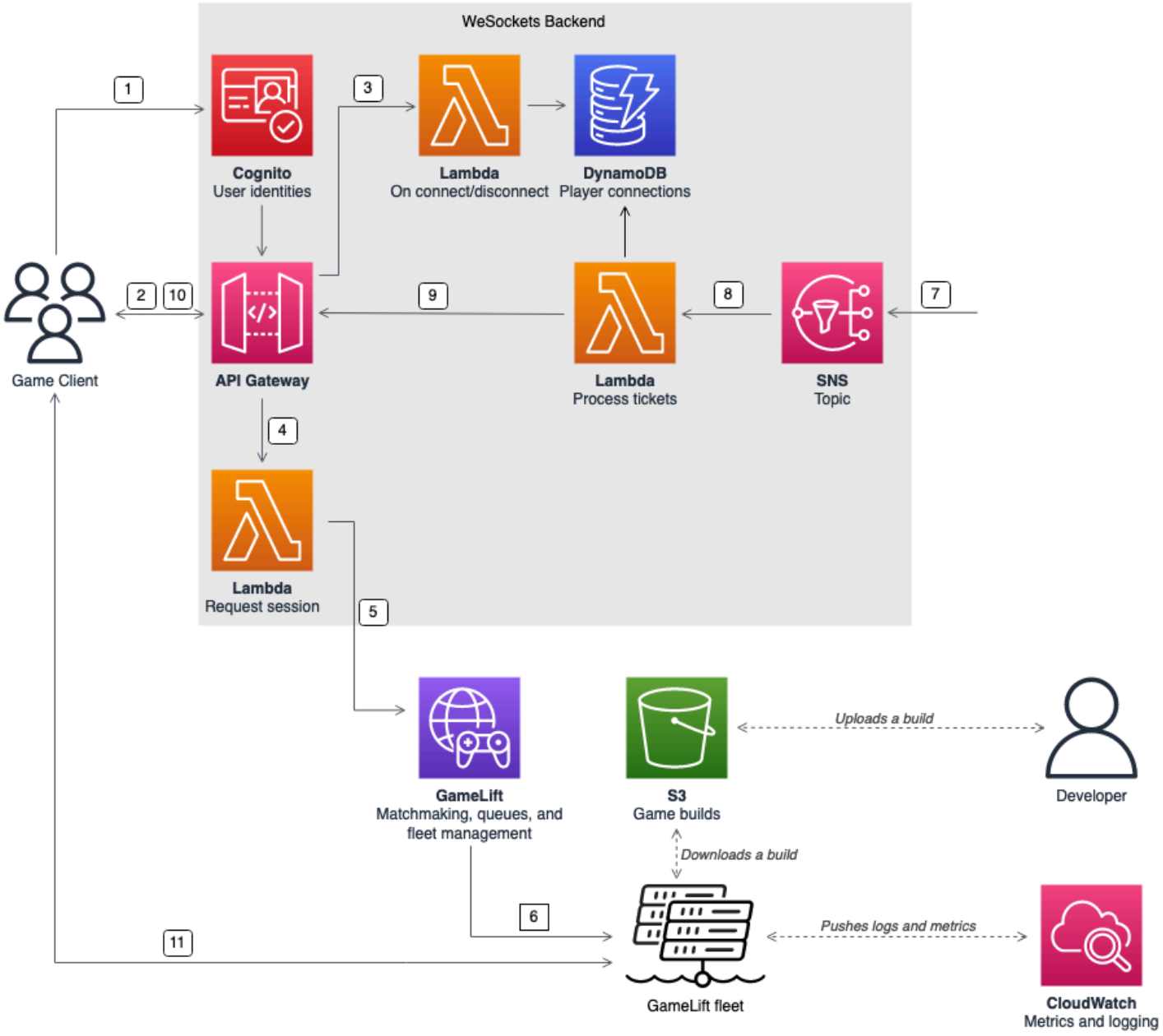
8. Lambda 関数は Amazon SNS イベントを受け取り、それを処理します。
9. マッチメーキングチケットが `MatchmakingSucceeded` イベントの場合、Lambda 関数はゲームサーバーのポートと IP アドレスと一緒に結果を DynamoDB に書き込みます。
10. ゲームクライアントは API Gateway に対して署名付きリクエストを行い、特定の間隔でマッチメーキングチケットのステータスを表示します。
11. API Gateway は、マッチメーキングチケットのステータスをチェックする Lambda 関数を使用します。
12. Lambda 関数は DynamoDB テーブルをチェックして、チケットが成功したかどうかを確認します。成功すると、この関数はゲームサーバーのポートと IP アドレスをプレーヤーセッション ID と一緒にクライアントに送り返します。チケットが成功しなかった場合、この関数は、マッチがまだ準備できていないことを確認するレスポンスを送信します。
13. ゲームクライアントは、バックエンドサービスが提供するポートと IP アドレスを使用して、TCP または UDP によってゲームサーバーに接続します。次に、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、次に Amazon GameLift サーバー SDK を使用して ID を検証します。

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

この Amazon API Gateway WebSocket ベースのアーキテクチャを使用すると、WebSockets でマッチメーキングリクエストを行い、サーバーが開始するメッセージを使用して、マッチメーキングの完了のプッシュ通知を送信することができます。このアーキテクチャにより、クライアントとサーバー間の双方向通信が可能になり、パフォーマンスが向上します。

API Gateway WebSock API の使用に関する詳細については、「WebSocket API の使用」を参照してください。

次の図は、API Gateway および他の AWS のサービス を使用して Amazon GameLift フリートで実行されているゲームにプレイヤーをマッチングする WebSocket ベースのバックエンドアーキテクチャを示しています。次のリストには、図内の各番号付きの吹き出しについての説明が含まれます。



1. ゲームクライアントは Amazon Cognito アイデンティティプールから Amazon Cognito ユーザー ID をリクエストします。
2. ゲーム クライアントは Amazon Cognito 認証情報を使用して、API Gateway API への WebSocket 接続に署名します。
3. API Gateway は AWS Lambda 関数を接続に呼び出します。この関数は接続情報を Amazon DynamoDB テーブルに保存します。
4. ゲームクライアントは、WebSocket 接続で API Gateway API を介して Lambda 関数にメッセージを送信し、セッションをリクエストします。

5. Lambda 関数はメッセージを受信した後、Amazon GameLift FlexMatch マッチメーキングを通じてマッチをリクエストします。
6. FlexMatch がプレイヤーグループをマッチさせた後、FlexMatch は 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。
7. Amazon GameLift がフリートのいずれかのロケーションにセッションを配置すると、Amazon GameLift は Amazon Simple Notification Service (Amazon SNS) トピックにイベント通知を送信します。
8. Lambda 関数は Amazon SNS イベントを受け取り、それを処理します。
9. マッチメーキングチケットが `MatchmakingSucceeded` イベントの場合、Lambda 関数は DynamoDB から正しいプレイヤー接続をリクエストします。次に、この関数は WebSocket 接続で API Gateway API を介してゲームクライアントにメッセージを送信します。このアーキテクチャでは、ゲームクライアントはマッチメーキングのステータスを積極的にポーリングしません。
10. ゲームクライアントは、WebSocket 接続を介してゲームサーバーのポートと IP アドレスをプレイヤーセッション ID と共に受け取ります。
11. ゲームクライアントは、バックエンドサービスが提供するポートと IP アドレスを使用して、TCP または UDP によってゲームサーバーに接続します。次に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 ID を検証します。

Amazon GameLift のメトリックスとロギングをセットアップする

Amazon GameLift ゲームサーバーとリソースから収集されたデータを使用して、異常を発見できます。メトリクスを使ってパフォーマンスを改善することもできます。

Amazon GameLift で注意すべき主な分野は次のとおりです。

- Amazon GameLift サービスメトリクス – Amazon GameLift は、ゲームサーバー、フリート、キュー、FlexMatch などのリソースに関する Amazon CloudWatch メトリクスを提供します。これらのメトリクスは Amazon GameLift コンソールと CloudWatch コンソールで確認できます。CloudWatch の Amazon GameLift メトリクスの詳細については、「Amazon CloudWatch で Amazon GameLift をモニタリングする」を参照してください。
- ゲームサーバーメトリクス – Amazon GameLift はゲームサーバーメトリクスにアクセスできません。ただし、CloudWatch エージェントを使用して、ゲームサーバーから CloudWatch に直接カスタムメトリクスを送信できます。フリート AWS Identity and Access Management (IAM) ロールと

AWS SDK を使用して、CloudWatch に直接メトリクスを送信することもできます。メトリクスの設定方法の例については、GitHub の「AWS でのマルチプレイヤーセッションベースのゲームホスティング」を参照してください。このレポジトリには、シンプルな C# StatsD クライアントのサンプルの CloudWatch エージェント設定とコードが含まれています。

- ゲームサーバーログ – ゲームサーバーでゲームサーバーのログファイルを設定するには、Amazon GameLift サーバー SDK 設定を使用します。Amazon CloudWatch Logs をリアルタイムのログ管理ソリューションとして使用したり、CloudWatch エージェントを使用してログを設定することもできます。詳細については、「Amazon GameLift でのサーバーメッセージの log 記録」を参照してください。

ゲーム起動のチェックリスト

これらのチェックリストを使用して、ゲームのデプロイ後のフェーズを検証できます。チェックリストで [重要] とマークされている項目は、本番の起動に不可欠です。

トピック

オンボーディング

次のチェックリストを使用して、Amazon GameLift ホスティングのゲームのオンボーディングに関する項目を追跡します。[Critical]（重要）とマークされた項目は、本番起動開始に不可欠です。

- [重要] Amazon GameLift コンソールで Amazon GameLift オンボーディングアンケートに記入します。
- [重要] ゲームクライアントがゲームサーバーとやり取りするためのバックエンドサービスを設計して実装します。
- [重要] 他の AWS リソースにアクセスするために Amazon GameLift サーバーインスタンスに提供するAWS Identity and Access Management (IAM) ロールを作成します。
- [重要] FlexMatch とキューの他の AWS リージョン へのフェイルオーバーを設計および実装します。

- ゲームのキューとフリートの構造を考慮して、ターゲットロケーションへのフリートのロールアウトを計画します。
- AWS CloudFormation と AWS Cloud Development Kit (AWS CDK) を備えたコードとしての Infrastructure as Code (IaC) を使用してデプロイを自動化します。
- Amazon CloudWatch と Amazon Simple Storage Service (Amazon S3) を使用して、ログと分析情報を収集します。

テスト

Amazon GameLift ホスティングを使用してゲームを開発する際のテスト項目を追跡するには、次のチェックリストを使用してください。[Critical] （重要）とマークされた項目は、本番起動開始に不可欠です。

- [重要] ローンチアンケートに回答し、記入済みのアンケートを Amazon GameLift ローンチチームに送信します。リリースアンケートは [Amazon GameLift コンソール] にあります。
- [重要] Amazon GameLift サービスクォータやその他の AWS のサービス クォータのリクエストを増やして、本番環境のニーズに合わせてライブ環境をスケールアップできるようにします。
- [重要] ライブフリートで開いているポートが、サーバーが使用できるポートの範囲と一致していることを確認してください。
- [重要] RDP ポート 3389 と SSH ポート 22 を閉じます。
- ゲームの DevOps 管理計画を作成します。Amazon CloudWatch Logs または Amazon CloudWatch カスタムメトリクスを使用している場合は、サーバーフリートの重大または重要な問題に対するアラームを定義します。障害をシミュレートし、ランブックをテストします。
- 最大使用時にインスタンスで実行されているサーバーの数を検証し、サーバーインスタンスタイプの機能内であることを確認します。
- 最初は、より保守的な値にスケーリングポリシーを調整し、必要と思われるよりも少し多めのアイドルキャパシティを提供します。後でコストを最適化できます。20% のアイドルキャパシティを持つターゲットベースのスケーリングポリシーの使用を検討します。
- FlexMatch レイテンシールールを使用して、同じ AWS リージョン に地理的に近いプレイヤーをマッチングします。負荷テストクライアントからの合成レイテンシー データを使用して、これがロード時にどのように動作するかをテストします。
- プレイヤー認証とゲームセッションインフラストラクチャに負荷テストを実施して、需要に合わせて効果的にスケーリングできるかどうかを確認します。
- サーバーが数日間稼働したままの場合でも、接続を受け入れることができることを検証します。

- AWS Support プランレベルを「ビジネス」または「エンタープライズ」に引き上げて、AWS が問題や障害発生時に対応できるようにします。

起動する

次のチェックリストを使用して、Amazon GameLift でホストされているゲームのローンチに関する項目を追跡します。[Critical]（重要）とマークされた項目は、本番起動開始に不可欠です。

- [重要] スケールダウンでアクティブなゲームセッションを停止しないように、すべてのライブフリートで、フリート保護ポリシーをフルプロテクションに設定します。
- [重要] 少なくともピーク時に予想される需要に対応するために、フリートの最大サイズを十分高い値に設定します。予期せぬ需要に備えて、最大サイズを 2 倍にすることをお勧めします。
- 開発チーム全員にローンチイベントに参加してもらい、ローンチルームでゲームのローンチをモニタリングするように奨励しましょう。
- プレイヤーのレイテンシーとプレイヤーエクスペリエンスをモニタリングします。

ローンチ後

次のチェックリストを使用して、Amazon GameLift でホストされているゲームのローンチ後に関する項目を追跡します。

- スケーリングルールを調整してアイドルキャパシティを最小限に抑えます。
- [FlexMatch ルール] を変更するか、レイテンシー要件に基づいてロケーションを追加します。
- パフォーマンス効率がフリートコストに直接影響するため、サーバーの実行可能ファイルを最適化します。同じインフラストラクチャでより多くのゲームセッションを実行するには、インスタンスあたりのサーバープロセスの数を増やします。
- 分析データを使用して、継続的な開発を促進し、プレイヤーエクスペリエンスとゲームの寿命を向上させ、収益を最適化します。

Amazon 用のゲームの準備 GameLift

Amazon でホストするマルチプレイヤーゲームを準備するには GameLift、ゲームと Amazon 間の通信を設定します GameLift。このセクションのトピックでは、ゲームを Amazon GameLift、カスタムゲームサーバー、リアルタイムサーバーと統合し、 でマッチメーキングを追加するための詳細なヘルプを提供します FlexMatch。

トピック

- カスタムゲームサーバーとゲームを統合する
- Amazon GameLift リアルタイムサーバーとのゲームの統合
- Unity 用 Amazon GameLift プラグインとのゲームの統合
- Unreal Engine 用 Amazon GameLift プラグインとのゲームの統合
- Amazon GameLift インスタンスのフリートデータを取得する
- FlexMatch のマッチメーキングの追加

カスタムゲームサーバーとゲームを統合する

Amazon GameLift には、マルチプレイヤーゲームとカスタムゲームサーバーを Amazon GameLift で動作するように準備するための必要なツールがすべて用意されています。Amazon GameLift SDK には、ゲームクライアントとサーバーが Amazon GameLift と通信するために必要なライブラリが含まれています。SDK の詳細、および入手できる場所については、「Amazon での開発サポート GameLift」を参照してください。

このセクションのトピックでは、Amazon GameLift にデプロイする前に、ゲームクライアントとゲームサーバーに Amazon GameLift の機能を追加する方法について詳しく説明しています。ゲームを Amazon GameLift で起動して実行するための完全なロードマップについては、「Amazon GameLift マネージドホスティングロードマップ」を参照してください。

トピック

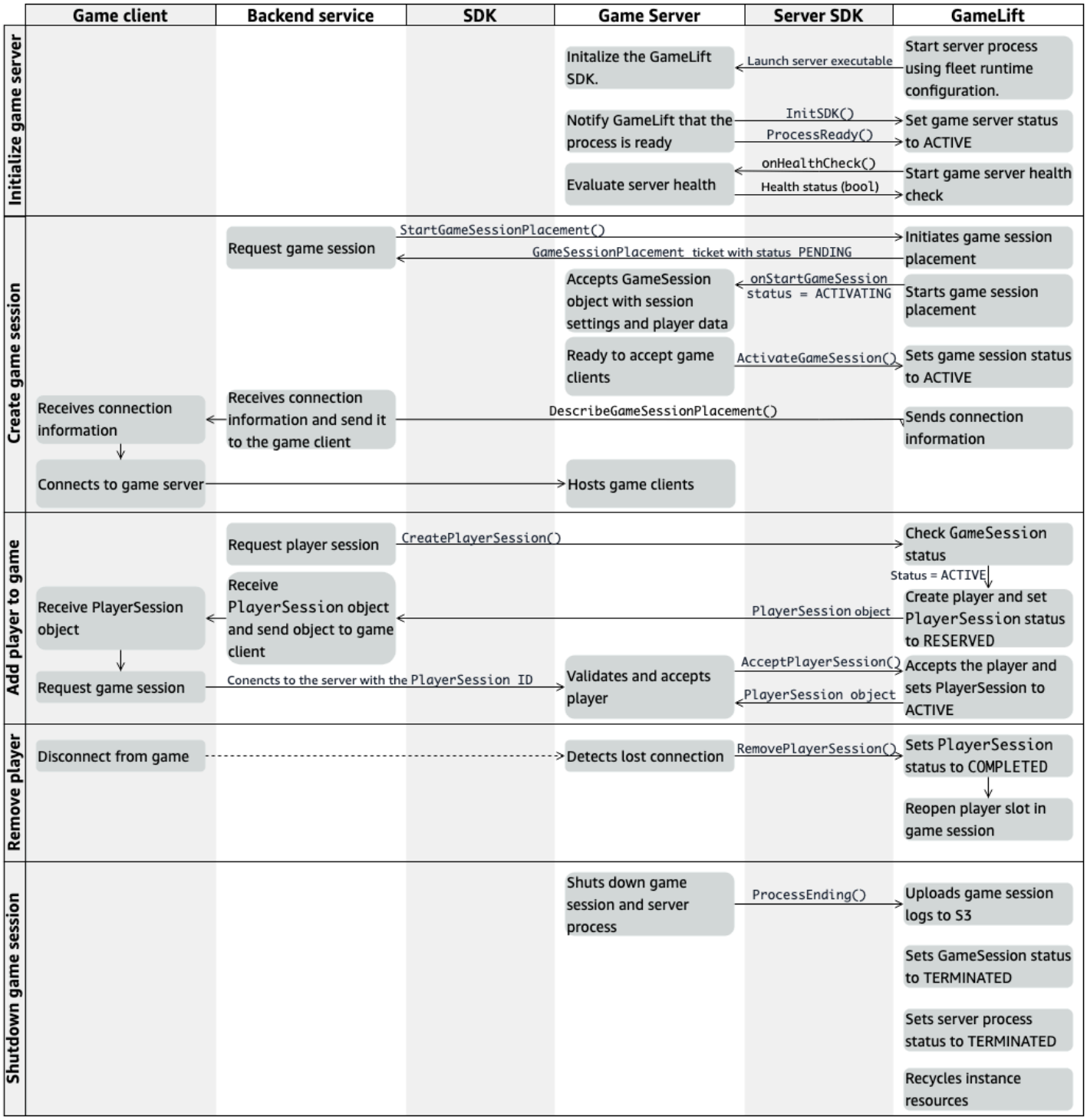
- Amazon GameLift とゲームサーバークライアントのやり取り
- ゲームサーバーを Amazon GameLift に統合する
- ゲーム クライアントを Amazon GameLift に統合する
- ゲームエンジンと Amazon GameLift
- Amazon GameLift Anywhere フリートを使用して統合をテストする

- Amazon GameLift Local を使用して統合をテストする

Amazon GameLift とゲームサーバークライアントのやり取り

このトピックでは、ゲームクライアント、バックエンドサービス、ゲームサーバー、および Amazon GameLift の間のやり取りについて説明します。

次の図は、ゲームクライアント、バックエンドサービス、Amazon GameLift SDK、マネージド EC2 ゲームサーバー、Amazon GameLift サーバー SDK、および Amazon GameLift 間の相互のやり取りを示しています。表示されているやり取りの詳細については、このページの後続のセクションを参照してください。



ゲームサーバーを初期化する

以下の手順では、ゲームサーバーでゲームセッションをホストする準備をするときに発生するやり取りについて説明します。

1. Amazon GameLift は、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスで新しいサーバー実行ファイルを起動します。
2. ゲームサーバーは以下を呼び出します。

   a. `InitSDK()` はサーバー SDK を初期化します。

   b. `ProcessReady()` はゲームセッションの準備状態、接続情報、およびゲームセッションログファイルのロケーションを通知します。

   次に、サーバープロセスは、Amazon GameLift からのコールバックを待機します。
3. Amazon GameLift はサーバープロセスのステータスを `ACTIVE` に更新して、ゲームセッションプレイスメントを有効にします。
4. Amazon GameLift が `onHealthCheck` コールバックの呼び出しを開始し、サーバープロセスがアクティブである間に定期的に呼び出しを継続します。サーバープロセスは、正常か正常でないかを 1 分以内に報告することができます。

ゲームセッションを作成する

ゲームサーバーを初期化した後、プレイヤーをホストするゲームセッションを作成すると、次のやり取りが発生します。

1. バックエンドサービスが SDK オペレーション `StartGameSessionPlacement()` を呼び出します。
2. Amazon GameLift はステータスが `PENDING` の新しい `GameSessionPlacement` チケットを作成し、バックエンドサービスに返します。
3. バックエンドサービスがキューからプレイスメントチケットのステータスを取得します。詳細については、「ゲームセッション配置のイベン通知を設定」を参照してください。
4. Amazon GameLift は、適切なフリートを選択して `0` のゲームセッションでフリート内のアクティブなサーバープロセスを検索することで、ゲームセッションプレイスメントを開始します。Amazon GameLift がサーバープロセスを見つけると、Amazon GameLift は次の操作を行います。

   a. ゲームセッション設定と、プレイスメントリクエストからのプレイヤーデータのある `GameSession` オブジェクトを `ACTIVATING` ステータスで作成します。

b. サーバープロセスで onStartGameSession コールバックを呼び出します。Amazon GameLift は、サーバープロセスがゲームセッションをセットアップする可能性があることを示す GameSession オブジェクトを渡します。

c. サーバープロセスのゲームセッション数を 1 に変更します。

5. サーバープロセスは、onStartGameSession コールバック関数を実行します。サーバープロセスはプレーヤーの接続の準備ができたら ActivateGameSession() を呼び出し、プレーヤーの接続を待機します。

6. Amazon GameLift は、サーバープロセスの接続情報について GameSession オブジェクトを更新します。(この情報には、ProcessReady() で報告されたポート設定が含まれます)。Amazon GameLift もステータスを ACTIVE に変更します。

7. バックエンドサービスが DescribeGameSessionPlacement() を呼び出して、更新されたチケットステータスを検出します。次に、バックエンドサービスは接続情報を使用してゲームクライアントをサーバープロセスに接続し、ゲームセッションに参加します。

ゲームにプレイヤーを追加する

このシーケンスでは、既存のゲームセッションにプレーヤーを追加するプロセスを説明します。プレイヤーセッションは、ゲームセッション配置リクエストのパートとしてリクエストすることもできます。

1. バックエンドサービスはゲームセッション ID を使用してクライアント API CreatePlayerSession() オペレーションを呼び出します。

2. Amazon GameLift は、ゲームセッションのステータスを確認し (ACTIVE である必要あります)、ゲームセッションで開いているプレイヤースロットを探します。スロットを利用できる場合、Amazon GameLift は以下を実行します。

   a. 新しい PlayerSession オブジェクトを作成し、ステータスを RESERVED に設定します。

   b. バックエンドサービスのリクエストに PlayerSession オブジェクトで応答します。

3. バックエンドサービスは、プレイヤーセッション ID を使用してゲームクライアントをサーバープロセスに直接接続します。

4. サーバーは、サーバー API オペレーション AcceptPlayerSession() を呼び出してプレイヤーセッション ID を検証します。検証された場合、Amazon GameLift は PlayerSession オブジェクトをサーバープロセスに渡します。サーバープロセスは接続を受け入れるか、または拒否します。

5. Amazon GameLift は次のいずれかを実行します。

a. 接続が受け入れられると、Amazon GameLift は PlayerSession ステータスを ACTIVE に設定します。

b. バックエンドサービスの元の CreatePlayerSession() 呼び出しに 60 秒以内に応答がない場合、Amazon GameLift は、PlayerSession のステータスを TIMEDOUT に変更し、ゲームセッションのプレーヤースロットを再び開きます。

プレイヤーを削除する

新しいプレイヤーが参加できるスペースを作るためにプレイヤーをゲームセッションから削除するとき、次のようなやり取りが発生します。

1. プレイヤーがゲームから接続解除します。
2. サーバーは失われた接続を検出し、サーバー API オペレーション RemovePlayerSession() を呼び出します。
3. Amazon GameLiftは、PlayerSession のステータスを COMPLETED に変更し、ゲームセッションのプレーヤースロットを再び開きます。

ゲームセッションをシャットダウンする

この一連のやり取りは、サーバープロセスが現在のゲームセッションをシャットダウンしたときに発生します。

1. サーバーがゲームセッションとサーバーをシャットダウンします。
2. サーバーが ProcessEnding() を Amazon GameLift に対して呼び出します。
3. Amazon GameLift は以下を行います。

   a. Amazon Simple Storage Service (Amazon S3) にゲーム セッションログをアップロードします。

   b. GameSession ステータスが TERMINATED に変わります。

   c. サーバープロセスのステータスを TERMINATED に変更します。

   d. インスタンスリソースをリサイクルします。

ゲームサーバーを Amazon GameLift に統合する

カスタムゲームサーバーが Amazon GameLift インスタンスにデプロイされて実行されると、Amazon GameLift (および場合によってはその他のリソース) とやり取りできる必要があります。このセクションでは、ゲームサーバーソフトウェアを Amazon GameLift と統合する方法について説明します。

> **Note**
>
> これらの手順では、AWS アカウント を作成しており、既存のゲーム サーバープロジェクトが存在することを前提としています。

このセクションのトピックでは、次の統合タスクの処理方法について説明します。

- Amazon GameLift とゲームサーバー間の通信を確立します。
- ゲームクライアントとゲームサーバー間のセキュアな接続を確立するために、TLS 証明書を生成して使用します。
- ゲームサーバーソフトウェアが他の AWS リソースとやり取りするためのアクセス許可を提供します。
- ゲームサーバープロセスが、実行されているフリートに関する情報を取得できるよう許可します。

トピック

Amazon GameLift をゲームサーバーに追加する

各ゲームサーバープロセスは Amazon GameLift が開始するイベントに応答できる必要があるため、カスタムゲームサーバーは Amazon GameLift と通信する必要があります。また、ゲームサーバーは、Amazon GameLift にサーバープロセスのステータスとプレーヤーの接続について知らせる必要があります。ゲームサーバー、バックエンドサービス、ゲームクライアント、Amazon GameLift が連携してゲームホスティングを管理する方法の詳細については、「Amazon GameLift とゲームサーバークライアントのやり取り」を参照してください。

Amazon GameLift と連携するゲームサーバーを準備するには、ゲームサーバープロジェクトに Amazon GameLift Server SDKを追加し、このトピックで説明されている機能を構築します。Server

SDK は複数の言語で用意されています。Amazon GameLift サーバー SDK の詳細については、「Amazon での開発サポート GameLift」を参照してください。

Server SDK API リファレンス

- C++ 用 Amazon GameLift サーバー SDK 5.x リファレンス
- C# と Unity 用 Amazon GameLift サーバー SDK 5.x リファレンス
- Amazon GameLift Unreal Engine サーバー SDK 5.x リファレンス

サーバープロセスを初期化する

Amazon GameLift との通信を確立し、サーバープロセスがゲームセッションをホストする準備ができたことを報告するコードを追加します。このコードは、Amazon GameLift コードの前に実行する必要があります。

1. `InitSdk()` を呼び出して Amazon GameLift API クライアントを初期化します。Amazon GameLift Anywhere コンピューティングリソースでサーバープロセスを初期化するには、次の `ServerParameters` で `InitSdk()` を呼び出します。
   - ゲームサーバーへの接続に使用されるウェブソケットの URL。
   - ゲームサーバーのホストに使用されるプロセスの ID。
   - ゲームサーバープロセスをホスティングするコンピューティングの ID。
   - Amazon GameLift Anywhere コンピューティングを含む GameLift フリートの ID。
   - Amazon GameLift オペレーション `GetComputeAuthToken` によって生成された認証トークン。

   > **Note**
   >
   > Amazon GameLift マネージド Amazon EC2 インスタンス上のゲームサーバーを初期化するには、デフォルト `InitSDK()` (C++) (C#) (Unreal) コンストラクター (パラメータなし) を使用して ServerParameters を作成します。Amazon GameLift がコンピューティング環境をセットアップし、お客様に代わって Amazon GameLift に自動的に接続します。

2. サーバープロセスが、ゲームセッションを Amazon GameLift にホストする準備ができたことを通知します。以下の情報を使用して、`ProcessReady()` (C++) (C#) (Unreal) を呼び出します。

(ProcessReady() は、サーバープロセスごとに 1 回だけ呼び出す必要があることに注意してください)。

- サーバープロセスが使用するポート番号。バックエンドサービスはゲームクライアントにポート番号と IP アドレスを提供し、サーバープロセスに接続してゲームセッションに参加します。
- ゲームセッションログなど、Amazon GameLift に保持させたいファイルのロケーション。サーバープロセスはゲームセッション中にこれらのファイルを生成します。これらは、サーバープロセスが実行されているインスタンスに一時的に保存され、インスタンスがシャットダウンすると失われます。リストしたファイルはすべて Amazon GameLift にアップロードされます。これらのファイルにアクセスするには、[Amazon GameLift コンソール] 経由または Amazon GameLift API オペレーション GetGameSessionLogUrl() を呼び出します。
- Amazon GameLift がサーバープロセスに呼び出すことができるコールバック関数の名前。ゲームサーバーは、これらの関数を実装する必要があります。詳細については、「 (C++) (C#) (Unreal) 」を参照してください。
  - (オプション) onHealthCheck – Amazon GameLift は、この関数を定期的に呼び出して、サーバーからヘルスステータスレポートをリクエストします。
  - onStartGameSession – Amazon GameLift はクライアントのリクエスト CreateGameSession() に応じ、この関数を呼び出します。
  - onProcessTerminate – Amazon GameLift はサーバープロセスを強制的に停止し、正常にシャットダウンさせます。
  - (オプション) onUpdateGameSession – Amazon GameLift は、更新されたゲームセッションオブジェクトをゲームサーバーに配信するか、マッチバックフィルリクエストでステータスの更新を提供します。FlexMatch バックフィル機能にはこのコールバックが必要です。

また、ゲームサーバーをセットアップして、所有または管理する AWS リソースに安全にアクセスできます。詳細については、「フリートの他の AWS リソースと通信する」を参照してください。

(オプション) サーバープロセスの健全性を報告する

コールバック関数 onHealthCheck() を実装するコードをゲームサーバーに追加します。Amazon GameLift はこのコールバックメソッドを定期的に呼び出してヘルスメトリクスを収集します。このコールバック関数を実装する場合は、以下を実行します：

- サーバープロセスのヘルスステータスを確認します。例えば、外部の依存関係で エラーが発生した場合にサーバープロセスを異常と報告できます。
- 状態評価を完了し、60 秒以内にコールバックに応答します。Amazon GameLift サービスがその時間内にレスポンスを受け取らない場合、自動的にサーバープロセスを異常とみなします。
- ブール値を返します (正常の場合は true、異常の場合は false)。

ヘルスチェックコールバックを実装していないと、サーバーが応答しない場合を除き、Amazon GameLift はサーバープロセスを正常とみなします。

Amazon GameLift はサーバープロセスの健全性を利用して、異常のあるプロセスを終了し、リソースをクリアにします。サーバープロセスが異常と報告され続ける場合や、ヘルスチェックに 3 回連続して応答しない場合、Amazon GameLift はプロセスをシャットダウンして新しいプロセスをスタートすることがあります。Amazon GameLift は、フリートのサーバープロセスの健全性に関するメトリクスを収集します。

(オプション) TLS 証明書を取得する

TLS 証明書の生成が有効になっているフリートでサーバーが実行されている場合は、TLS 証明書を取得してゲーム　クライアントとのセキュリティ保護ありの接続を確立し、クライアントサーバー通信を暗号化できます。証明書のコピーがインスタンスに保存されます。ファイルの口ケーションを取得するには、`GetComputeCertificate()` (C++) (C#) (Unreal) を呼び出します。

ゲームセッションをスタートする

コールバック関数 `onStartGameSession` を実装するコードを追加します。Amazon GameLift はこのコールバックを呼び出して、サーバー上でゲームセッションを開始します。

`onStartGameSession`関数は[GameSession] (ゲームセッション) オブジェクトを入力パラメータとして取得します。このオブジェクトには、最大プレイヤー数などの主なゲームセッション情報が含まれます。また、ゲームデータとプレイヤーデータを含めることもできます。この関数の実装は、以下のタスクを実行する必要があります。

- `GameSession`プロパティに基づいて、新しいゲームセッションを作成するためのアクションを開始します。少なくとも、ゲームサーバーは、サーバープロセスに接続するときにゲームクライアントがリファレンスするゲームセッション ID を関連付ける必要があります。
- 必要に応じて、ゲームデータとプレイヤーデータを処理します。このデータは、`GameSession` オブジェクトにあります。

- 新しいゲームセッションでプレイヤーを受け入れる準備が整うと、Amazon GameLift に通知します。サーバー API オペレーション `ActivateGameSession()` (C++) (C#) (Unreal) を呼び出します。呼び出しの成功に応じて、Amazon GameLift はゲームセッションのステータスを `ACTIVE` に変更します。

(オプション) 新しいプレイヤーを検証する

プレイヤーセッションのステータスを追跡する場合は、新しいプレイヤーがゲームサーバーに接続するときに、プレイヤーを検証するコードを追加します。Amazon GameLift は、現在のプレイヤーと利用可能なゲームセッションスロットを追跡します。

検証のために、ゲームセッションへのアクセス許可を要求しているゲームクライアントには、プレイヤーセッション ID を含める必要があります。プレイヤーが StartGameSessionPlacement() or StartMatchmaking() を使ってゲームに参加するように要求すると、Amazon GameLift は自動的にこの ID を生成します。プレイヤーセッションでは、ゲームセッションにオープンスロットを予約します。

ゲームサーバープロセスがゲームクライアント接続要求を受信すると、`AcceptPlayerSession()` (C++) (C#) (Unreal) をプレイヤーセッション ID で呼び出します。それに応じて、Amazon GameLift は、プレイヤーセッション ID がゲームセッションで予約されているオープンスロットに対応していることを確認します。Amazon GameLift がプレイヤーセッション ID を検証すると、サーバープロセスは接続を受け入れます。これで、プレイヤーはゲームセッションに参加できます。Amazon GameLift がプレイヤーセッション ID を検証しない場合、サーバープロセスは接続を拒否します。

(オプション) プレイヤーセッションの終了を報告する

プレイヤーセッションのステータスを追跡している場合は、プレイヤーがゲームセッションの終了を Amazon GameLift に通知するコードを追加します。このコードは、サーバープロセスが接続中断を検出するたびに実行する必要があります。Amazon GameLift はこの通知を使ってゲームセッションにおける現在のプレイヤーと使用可能なスロットを追跡します。

切断された接続を処理するには、対応するプレイヤーセッション ID でサーバー API オペレーション `RemovePlayerSession()` (C++) (C#) (Unreal) への呼び出しをコードに追加します。

ゲームセッションを終了する

サーバープロセスのシャットダウンシーケンスに、ゲームセッションの終了を Amazon GameLift に通知するコードを追加します。ホスティングリソースをリサイクルして更新するために、Amazon GameLift はゲームセッションの完了後にサーバープロセスをシャットダウンします。

サーバープロセスのシャットダウンコードの開始時に、サーバー API オペレーション `ProcessEnding()` (C++) (C#) (Unreal) を呼び出します。この呼び出しは、サーバープロセスがシャットダウンしていることを Amazon GameLift に通知します。Amazon GameLift は、ゲームセッションステータスとサーバープロセスステータスを `TERMINATED` に変更します。`ProcessEnding()` を呼び出した後は、プロセスが安全にシャットダウンされます。

サーバープロセスのシャットダウン通知に応答する

Amazon GameLift からの通知に応答してサーバープロセスをシャットダウンするコードを追加します。Amazon GameLift は、サーバープロセスが一貫して正常でないと報告する場合、またはサーバープロセスが実行されているインスタンスが終了している場合に、この通知を送信します。Amazon GameLift は、キャパシティのスケールダウンイベントの一環として、またはスポットインスタンスの中断に応じてインスタンスを停止する場合があります。

シャットダウン通知を処理するには、ゲームサーバーコードに次の変更を加えます。

- コールバック関数 `onProcessTerminate()` を実装します。この関数は、サーバープロセスをシャットダウンするコードを呼び出します。Amazon GameLift がこのオペレーションを呼び出すと、スポットインスタンスの中断により 2 分間の通知が提供されます。この通知は、サーバープロセスにプレイヤーを適切に切断してゲーム状態データを保存し、他のクリーンアップタスクを実行する時間を提供します。
- ゲームサーバーのシャットダウンコードからサーバー API オペレーション `GetTerminationTime()` (C++) (C#) (Unreal) を呼び出します。Amazon GameLift がサーバープロセスを中止する呼び出しを発行した場合、`GetTerminationTime()` は予想終了時間を返します。
- ゲームサーバーシャットダウンコードの開始時に、サーバー API オペレーション `ProcessEnding()` (C++) (C#) (Unreal) を呼び出します。この呼び出しは、サーバープロセスがシャットダウン中であることを Amazon GameLift に通知し、次に Amazon GameLift はサーバープロセスのステータスを `TERMINATED` に変更します。`ProcessEnding()` を呼び出した後は、プロセスが安全にシャットダウンされます。

フリートの他の AWS リソースと通信する

Amazon GameLift フリートにデプロイするゲームサーバービルドを作成する場合、ゲームビルド内のアプリケーションが、所有する他の AWS リソースと直接かつセキュアに通信できるようにしたい場合があります。Amazon GameLift はゲームホスティングフリートを管理するため、Amazon GameLift にこれらのリソースとサービスへの制限付きアクセスを許可する必要があります。

シナリオの例には次のようなものがあります。

- Amazon CloudWatch エージェントを使用し、マネージド EC2 フリートと Anywhere フリートからメトリクス、ログ、トレースを収集します。
- Amazon CloudWatch Logsにインスタンス ログデータを送信します。
- データファイルを格納する Amazon Simple Storage Service (Amazon S3) バケットを取得します。
- Amazon DynamoDB データベースや他のデータストレージサービスに保存されているゲームデータ (ゲームモードやインベントリなど) を動的に読み取りおよび書き込みします。
- Amazon Simple Queue Service (Amazon SQS) を使用し、シグナルをインスタンスに直接送信します。
- Amazon Elastic Compute Cloud (Amazon EC2) にデプロイされ、実行しているカスタムリソースにアクセスします。

Amazon GameLift は、アクセスを確立するための方法として以下をサポートしています。

- IAM ロールを使用して AWS リソースにアクセスする
- VPC ピアリングを使用して AWS リソースにアクセスする

IAM ロールを使用して AWS リソースにアクセスする

IAM ロールを使用してリソースにアクセスできるユーザーを指定し、そのアクセスに制限を設定します。信頼できる関係者はロールを「引き受ける」ことで、リソースとのやりとりを許可する一時的なセキュリティ認証情報を取得できます。関係者がリソースに関連する API リクエストを行う場合は、認証情報を含める必要があります。

IAM ロールで制御されるアクセスを設定するには、以下のタスクを実行します。

1. IAM ロールを作成する
2. 認証情報を取得するようにアプリケーションを変更する
3. IAM ロールをフリートに関連付ける。

IAM ロールを作成する

このステップでは、AWS リソースへのアクセスを制御する一連のアクセス許可と、Amazon GameLift にそのロールのアクセス許可を使用する権限を付与する信頼ポリシーを含む IAM ロールを作成します。

IAM rオールのセットアップ方法については「Amazon の IAM サービスロールを設定する GameLift」を参照してください。アクセス許可かポリシーを作成するときは、アプリケーションが連携する必要のある特定のサービス、リソース、アクションを選択します。ベストプラクティスとして、アクセス許可の範囲をできる限り制限してください。

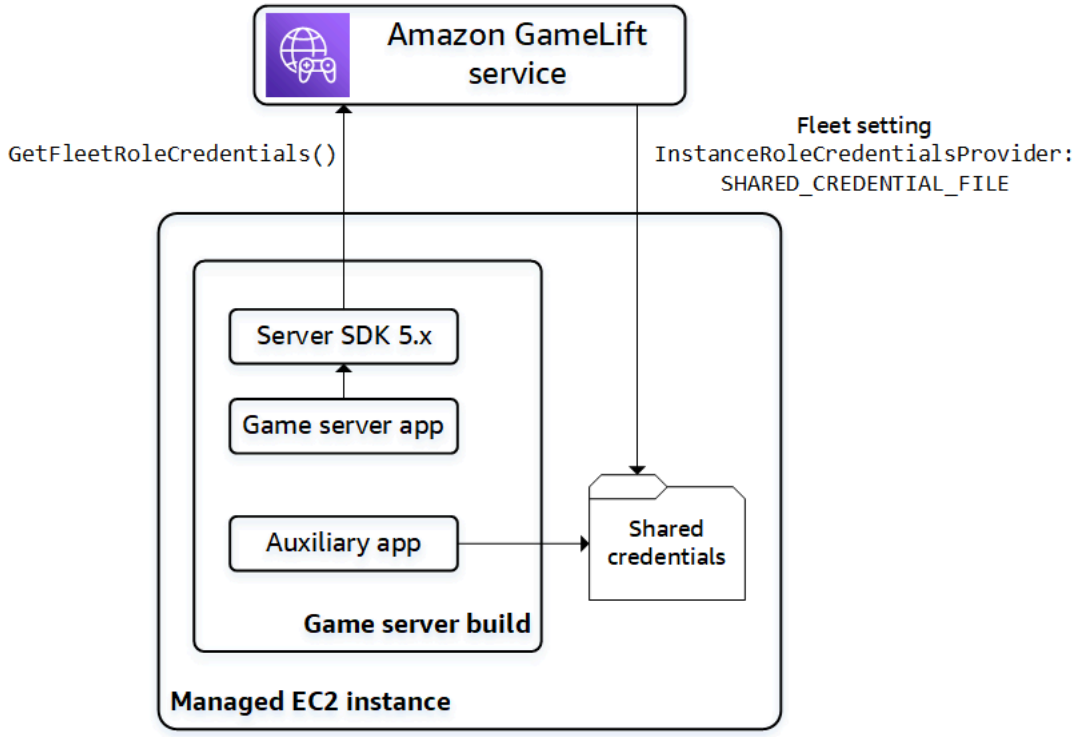
ロールを作成した後、ロールの Amazon リソースネーム (ARN) をメモします。フリート作成時にはロール ARN が必要です。

認証情報を取得するようにアプリケーションを変更する

このステップでは、IAM ロールのセキュリティ認証情報を取得し、それらを AWS リソースとのやり取りに使用するようにアプリケーションを設定します。以下の表を参照して、(1) アプリケーションのタイプ、(2) ゲームが Amazon GameLift との通信に使用するサーバー SDK バージョンに基づいてアプリケーションを変更する方法を決定してください。

	ゲームサーバーアプリケーション	その他のアプリケーション
サーバー SDK バージョン 5.x の使用	ゲームサーバーコードからサーバー SDK メソッド `GetFleetRoleCredentials()` を呼び出します。	アプリケーションにフリートインスタンス上の共有ファイルから認証情報を取得するコードを追加します。
バージョン 4 以前のサーバー SDK の使用	ロール ARN で AWS Security Token Service (AWS STS) `AssumeRole` を呼び出します。	ロール ARN で AWS Security Token Service (AWS STS) `AssumeRole` を呼び出します。

サーバー SDK 5.x と統合されたゲームの場合、この図は、デプロイされたゲームビルドのアプリケーションが IAM ロールの認証情報を取得する方法を示しています。



`GetFleetRoleCredentials()` (サーバー SDK 5.x) を呼び出す

Amazon GameLift サーバー SDK 5.x とすでに統合されているゲームサーバーコードで、`GetFleetRoleCredentials` (C++) (C#) (Unreal) を呼び出して、一時的な認証情報のセットを取得します。認証情報の有効期限が切れたら、`GetFleetRoleCredentials` をもう一度呼び出して認証情報を更新できます。

共有の認証情報 (サーバー SDK 5.x) を使用する

サーバー SDK 5.x を使用するゲームサーバービルドでデプロイされる非サーバーアプリケーションの場合は、共有ファイルに保存されている認証情報を取得して使用するコードを追加します。Amazon GameLift は、フリートインスタンスごとに認証情報プロファイルを生成します。認証情報はインスタンス上のすべてのアプリケーションで使用できます。Amazon GameLift は一時的な認証情報を継続的に更新します。

フリート作成時に共有の認証情報ファイルを生成するようにフリートを設定する必要があります。

共有の認証情報ファイルを使用する必要がある各アプリケーションで、次のようにファイルの場所とプロファイル名を指定します。

Windows の場合:

```
[credentials]
```

```
shared_credential_profile= "FleetRoleCredentials"
shared_credential_file= "C:\\Credentials\\credentials"
```

Linux :

```
[credentials]
shared_credential_profile= "FleetRoleCredentials"
shared_credential_file= "/local/credentials/credentials"
```

例: Amazon GameLift フリートインスタンスのメトリクスを収集するように CloudWatch エージェントをセットアップする

Amazon CloudWatch エージェントを使用して Amazon GameLift フリートからメトリクス、ログ、トレースを収集する場合は、このメソッドを使用して、エージェントがアカウントにデータを送信することを許可します。このシナリオでは、以下のステップを行います。

1. CloudWatch エージェント `config.json` ファイルを取得または書き込みます。
2. 前述のように、エージェント用の `common-config.toml` ファイルを更新して、認証情報ファイル名とプロファイル名を特定します。
3. ゲームサーバーのビルドインストールスクリプトを設定して、CloudWatch エージェントをインストールして開始します。

`AssumeRole()` (サーバー SDK 4) を使用する

IAM ロールを引き受けるコードをアプリケーションに追加し、AWS リソースとやり取りするための認証情報を取得します。サーバー SDK 4 以前の Amazon GameLift フリートインスタンスで実行されるすべてのアプリケーションで IAM ロールを引き受けることができます。

アプリケーションコードでは、AWS リソースにアクセスする前に、アプリケーションが AWS Security Token Service (AWS STS) `AssumeRole` API オペレーションを呼び出し、ロール ARN を指定する必要があります。このオペレーションは、アプリケーションによる AWS リソースへのアクセスを承認する一時的な認証情報のセットを返します。詳細については、IAM ユーザーガイドの「AWS リソースを使用した一時的なセキュリティ認証情報の使用」を参照してください。

IAM ロールをフリートに関連付ける。

IAM ロールを作成し、ゲームサーバービルドのアプリケーションを更新してアクセス認証情報を取得して使用したら、フリートをデプロイできます。新しいフリートを設定する場合は、以下のパラメータを設定します。

- InstanceRoleArn – このパラメータを IAM ロールの ARN に設定します。
- InstanceRoleCredentialsProvider – Amazon GameLift に各フリートインスタンスの共有の認証情報ファイルを生成するように求めるには、このパラメータを SHARED_CREDENTIAL_FILE に設定します。

これらの値は、フリートを作成するときに設定する必要があります。これらは後で更新できません。

VPC ピアリングを使用して AWS リソースにアクセスする

Amazon Virtual Private Cloud (Amazon VPC) ピア接続を使用して、Amazon GameLift インスタンスで実行されているアプリケーションと別の AWS リソースとの間で通信することができます。VPC は、ユーザーが定義する仮想プライベートネットワークであり、AWS アカウント を通じて管理される一連のリソースを含みます。Amazon GameLift フリートごとに専用の VPC があります。VPC ピアリングを使用すると、フリート用の VPC と他の AWS リソース用の VPC との間に、直接ネットワーク接続を確立できます。

Amazon GameLift は、ゲームサーバー用の VPC ピアリング接続を設定するプロセスを合理化します。ピアリングリクエストを処理し、ルートテーブルを更新し、必要に応じて接続を設定します。ゲームサーバーの VPC ピア接続を設定する方法の手順については、「Amazon GameLift の VPC ピアリング」を参照してください。

ゲーム クライアントを Amazon GameLift に統合する

このセクションのトピックでは、バックエンドサービスに追加できるマネージド Amazon GameLift 機能について説明します。バックエンドサービスは次のタスクを処理します。

- Amazon GameLift からのアクティブなゲームセッションについての情報をリクエストする。
- 既存のゲームセッションにプレイヤーを参加させます。
- 新しいゲームセッションを作成し、そのセッションにプレーヤーを参加させる。
- 既存のゲームセッションに関するメタデータを変更する。

ゲームクライアントが、Amazon GameLift および Amazon GameLift で実行されるゲームサーバーと連携する方法の詳細は、「Amazon GameLift とゲームサーバークライアントのやり取り」を参照してください。

前提条件

- AWS アカウント。

- Amazon GameLift にアップロードされたゲームサーバービルド。
- ゲームをホストするためのフリート。

トピック

- ゲームクライアント GameLift に Amazon を追加する
- プレイヤー ID を生成する

ゲームクライアント GameLift に Amazon を追加する

Amazon GameLift をゲームセッション情報を必要とするゲームコンポーネントに統合し、新しいゲームセッションを作成し、ゲームにプレイヤーを追加します。ゲームのアーキテクチャに応じて、この機能は、プレイヤーの認証、マッチメーキング、ゲームセッションプレイスメントなどのタスクを処理するバックエンドサービスに配置されています。

> **Note**
>
> Amazon が GameLift ホストするゲームのマッチメーキングを設定する方法の詳細については、「Amazon GameLift FlexMatch デベロッパーガイド」を参照してください。

バックエンドサービス GameLift で Amazon をセットアップする

Amazon GameLift クライアントを初期化し、キー設定を保存するコードを追加します。このコードは、Amazon に依存するコードの前に実行する必要があります GameLift。

1. クライアントをセットアップします。既定のクライアント設定を使用するか、カスタムクライアント設定オブジェクトを作成します。詳細については、AWS::Client::ClientConfiguration「 (C++)」またはAmazonGameLiftConfig「 (C#)」を参照してください。

   クライアント設定は、Amazon への問い合わせ時に使用するターゲットリージョンとエンドポイントを指定します GameLift。リージョンは、使用するデプロイ済みのリソース (フリート、キュー、マッチメーカー) のセットを識別します。デフォルトのクライアント設定では、米国東部 (バージニア北部) リージョンが指定されます。その他のリージョンを使用するには、カスタム設定を作成します。
2. Amazon GameLift クライアントを初期化します。デフォルトのクライアント設定またはカスタムクライアント設定で、Aws:GameLift:::GameLiftClient() (C++) または (AmazonGameLiftClient) (C#) を使用します。

3. 各プレイヤーに一意の識別子を生成するメカニズムを追加します。詳細については、「プレイヤー ID を生成する」を参照してください。
4. 以下の情報を収集して保存します。

   - ターゲットフリート — 多くの Amazon GameLift API リクエストではフリートを指定する必要があります。このためには、ターゲットフリートを指し示すフリート ID またはエイリアス ID のいずれかを使用します。ベストプラクティスとして、バックエンドサービスを更新しなくてもプレイヤーをあるフリートから別のフリートに切り替えられるようにフリートエイリアスを使用してください。
   - [ターゲットキュー] – ゲームでマルチフリートキューを使用して新しいゲームセッションを配置する場合は、使用するキューを指定します。
   - AWS 認証情報 – Amazon へのすべての呼び出しは、ゲームをホスト AWS アカウント する の認証情報を提供する GameLift 必要があります。これらの認証情報は、「ゲームへのプログラムによるアクセスをセットアップする」で説明されているようにプレイヤーユーザーを作成して取得します。プレイヤーユーザーのアクセスを管理する方法に応じて、次の操作を行います。
     - ロールを使用してプレイヤーユーザーのアクセス許可を管理する場合は、Amazon GameLift API を呼び出す前にロールを引き受けるコードを追加します。ロールを引き受けるリクエストは、一時的なセキュリティ認証情報のセットが返します。詳細については、「IAM ユーザーガイド」の「IAM ロール (AWS API) への切り替え」を参照してください。
     - 長期的なセキュリティ認証情報がある場合は、保存されている認証情報を検索して使用するようにコードを設定します。「AWS SDK およびツールリファレンスガイド」の「長期認証情報を使用して認証する」を参照してください。認証情報の保存については、 (C++) および (.NET) のAWS API リファレンスを参照してください。
     - 一時的なセキュリティ認証情報がある場合は、「IAM ユーザーガイド」の「AWS SDK での一時的なセキュリティ認証情報の使用」で説明されているように、 AWS Security Token Service (AWS STS) を使用して認証情報を定期的に更新するコードを追加します。古い認証情報の有効期限が切れる前に、コードから新しい認証情報をリクエストする必要があります。

ゲームセッションを取得する

使用可能なゲームセッションを検出し、ゲームセッション設定とメタデータを管理するコードを追加します。

アクティブなゲームセッションを検索する

SearchGameSessions を使用して、特定のゲームセッション、すべてのアクティブなセッション、または一連の検索条件に一致するセッションに関する情報を取得します。この呼び出しは、検索リクエストに一致するアクティブなゲームセッションごとに GameSession オブジェクトを返します。

プレイヤーが参加できるアクティブなゲームセッションのフィルタリングされたリストを取得するには、検索条件を使用します。たとえば、次のようにセッションをフィルタリングできます。

- 空きがないゲームセッションを除外する: `CurrentPlayerSessionCount = MaximumPlayerSessionCount`。
- セッションが実行されている時間の長さに基づいてゲームセッションを選択する: `CreationTime` を評価する。
- カスタムゲームプロパティに基づいてゲームセッションを検索する: `gameSessionProperties.gameMode = "brawl"`。

ゲームセッションを管理する

ゲームのセッション情報を取得または更新するには、次のいずれかのオペレーションを使用します。

- DescribeGameSessionDetails() - ゲームセッション情報に加えて、ゲームセッションの保護ステータスを取得します。
- UpdateGameSession() – 必要に応じてゲームセッションのメタデータと設定を変更します。
- GetGameSessionLogUrl – 保存されたゲームセッションログにアクセスします。

ゲームセッションを作成する

デプロイ済みフリートで新しいゲームセッションを起動し、それらのセッションをプレイヤーが使用できるようにするコードを追加します。ゲームセッションを作成するには、ゲームを複数のリージョンにデプロイするか、単一のリージョン AWS リージョン にデプロイするかに応じて、2 つのオプションがあります。

マルチフリートキューにゲームセッションを作成する

StartGameSessionPlacement を使用して、新しいゲームセッションのリクエストをキューに配置します。この操作を使用するには、キューを作成します。これにより、Amazon が新しいゲームセッション GameLift を配置する場所が決まります。キューの詳細とそれらの使用方法については、「ゲームセッションプレイスメント用の Amazon GameLiftt キューのセットアップ」を参照してください。

ゲームセッションプレイスメントを作成するときは、使用するキューの名前、ゲームセッション名、同時実行プレイヤーの最大数、オプションのゲームプロパティのセットを指定します。また、オプションで、ゲームセッションに自動的に参加するプレイヤーのリストを指定することもできます。関連するリージョンのプレイヤーレイテンシーデータを含めると、Amazon GameLift はこの情報を使用して、プレイヤーに最適なゲームプレイエクスペリエンスを提供するフリートに新しいゲームセッションを配置します。

ゲームセッション配置は非同期プロセスです。リクエスト送信後は、それが成功するかタイムアウトするかです。を使用して、いつでもリクエストをキャンセルすることもできますStopGameSessionPlacement。プレイスメントリクエストのステータスを確認するには、 を呼び出しますDescribeGameSessionPlacement。

特定のフリートにゲームセッションを作成する

CreateGameSession を使用して、指定したフリートに新しいセッションを作成します。この同期オペレーションは、フリートに新しいゲームセッションをホストするための使用可能なリソースがあるかどうかによって、成功か失敗かが決まります。Amazon が新しいゲームセッション GameLift を作成し、GameSessionオブジェクトを返したら、プレイヤーをそのセッションに参加させることができます。

このオペレーションを使用するときは、フリート ID またはエイリアス ID、セッション名、そのゲームの同時実行プレイヤーの最大数を指定します。オプションで、一連のゲームプロパティを含めることができます。ゲームプロパティは、キーと値のペアの配列で定義されます。

Amazon GameLift リソース保護機能を使用して 1 人のプレイヤーが作成できるゲームセッションの数を制限する場合は、ゲームセッション作成者のプレイヤー ID を指定します。

ゲームセッションにプレイヤーを参加させる

アクティブなゲームセッションにプレイヤースロットを予約し、ゲームクライアントをゲームセッションに接続するコードを追加します。

1. ゲームセッションにプレイヤースロットを予約する

   プレイヤースロットを予約するには、ゲームセッションに新しいプレイヤーセッションを作成します。プレイヤーセッションの詳細については、「プレイヤーがゲームに接続する方法」を参照してください。

   新しいプレイヤーセッションを作成するには、2 つの方法があります。

- を使用してStartGameSessionPlacement、新しいゲームセッションで 1 人以上のプレイヤーのスロットを予約します。
- CreatePlayerSession またはゲームセッション ID を使用して、1 人以上のプレイヤーCreatePlayerSessionsのプレイヤースロットを予約します。

Amazon は GameLift まず、ゲームセッションが新しいプレイヤーを受け入れ、利用可能なプレイヤースロットがあることを確認します。成功すると、Amazon はプレイヤーのスロット GameLift を予約し、新しいプレイヤーセッションを作成し、PlayerSessionオブジェクトを返します。このオブジェクトには、ゲームクライアントがゲームセッションに接続するために必要な DNS 名、IP アドレス、ポートが含まれます。

プレイヤーセッションリクエストには、各プレイヤーの一意の ID が含まれている必要があります。詳細については、「プレイヤー ID を生成する」を参照してください。

プレイヤーセッションには、一連のカスタムプレイヤーデータを含めることができます。このデータは新しく作成されたプレイヤーセッションオブジェクトに保存され、DescribePlayerSessions() を呼び出すことで取得できます。 GameLift また、プレイヤーがゲームセッションに直接接続すると、Amazon はこのオブジェクトをゲームサーバーに渡します。複数のプレイヤーセッションをリクエストするときは、プレイヤー ID にマッピングされたプレイヤーデータ文字列をプレイヤーごとにリクエストで指定します。

2. ゲームセッションに接続する

ゲームクライアントにコードを追加し、ゲームセッションの接続情報を含む `PlayerSession` オブジェクトを取得します。この情報を使用して、サーバーへの直接接続を確立します。

- 指定したポートと、サーバープロセスに割り当てられた DNS 名または IP アドレスを使用して接続できます。
- フリートに対して TLS 証明書の生成が有効になっている場合、DNS 名とポートを使用して接続します。
- ゲームサーバーが新規プレイヤーの接続を検証する場合は、プレイヤーセッション ID を参照します。

接続後、ゲームクライアントとサーバープロセスは Amazon を関与させることなく直接通信します GameLift。サーバーは Amazon GameLift との通信を維持し、プレイヤーの接続ステータス、ヘルスステータスなどを報告します。ゲームサーバーが新規プレイヤーを検証する場合、プレイヤーセッション ID がゲームセッション内の予約済みスロットと一致することを検証し、プ

レイヤーの接続を受け入れるか拒否します。プレイヤーが切断されると、サーバープロセスが接続中断を報告します。

ゲームセッションプロパティを使用する

ゲームクライアントは、ゲームプロパティを使用してゲームセッションにデータを渡すことができます。ゲームプロパティは、ゲームサーバーが追加、読み取り、一覧表示、変更できるキーと値のペアです。新しいゲームセッションを作成するとき、または後でゲームセッションがアクティブになったときに、ゲームプロパティを渡すことができます。ゲームセッションには、最大 16 個のゲームプロパティを含めることができます。ゲームプロパティは削除できません。

例えば、ゲームには、、、Novice、Easy、 Intermediateの難易度レベルがありますExpert。プレイヤーは を選択しEasy、ゲームを開始します。ゲームクライアント GameLift は、前のセクションで説明したCreateGameSessionように、 StartGameSessionPlacementまたは を使用して Amazon に新しいゲームセッションを要求します。リクエストでは、クライアントは を渡します{"Key": "Difficulty", "Value":"Easy"}。

リクエストに応じて、Amazon は指定されたゲームプロパティを含むGameSessionオブジェクト GameLift を作成します。 GameLift 次に、Amazon は利用可能なゲームサーバーに新しいゲームセッションを開始し、GameSessionオブジェクトを渡すよう指示します。ゲームサーバーは、Difficultyの を使用してゲームセッションを開始しますEasy。

詳細はこちら

- GameProperty データ型
- SearchGameSessions() の例
- UpdateGameSession() GameProperties パラメータ

プレイヤー ID を生成する

Amazon GameLift では、ゲームセッションに接続されるプレイヤーを表すためにプレイヤーセッションを使用します。Amazon GameLift は、Amazon GameLift と統合されたゲームクライアントを使用してプレーヤーがゲームセッションに接続するたびに、プレーヤーセッションを作成します。プレイヤーがゲームを離れると、プレイヤーセッションは終了します。Amazon GameLift はプレイヤーセッションを再利用しません。

以下のサンプルコードでは、一意のプレイヤー ID をランダムに生成します。

```
bool includeBrackets = false;
bool includeDashes = true;
string playerId = AZ::Uuid::CreateRandom().ToString<string>(includeBrackets,
 includeDashes);
```

プレイヤーセッションの詳細については、「ゲームおよびプレイヤーセッションのデータを表示する」を参照してください。

ゲームエンジンと Amazon GameLift

C++ または C# ライブラリをサポートするほとんどの主要なゲームエンジン (Amazon Lumberyard、Unreal Engine、Unity など) で、マネージド Amazon GameLift サービスを使用することができます。ゲームに必要なバージョンを構築します。構築手順および最小要件については、各バージョンの README ファイルを参照してください。利用可能な Amazon GameLift SDK、サポートされている開発プラットフォームおよびオペレーティングシステムの詳細に関しては、ゲームサーバー用の「Amazon での開発サポート GameLift」を参照してください。

このトピックに記載されているエンジン固有の情報に加えて、Amazon GameLift をゲームサーバー、クライアント、サービスと統合する際に役立つその他のヘルプは、以下のトピックを参照してください。

- Amazon GameLift マネージドホスティングロードマップ – Amazon GameLift をゲームに正常に統合してホスティングリソースをセットアップするための 6 ステップのワークフローです。
- Amazon GameLift をゲームサーバーに追加する – Amazon GameLift をゲームサーバーに統合する詳しい手順です。
- ゲームクライアント GameLift に Amazon を追加する – ゲームセッションの作成やプレイヤーのゲームへの参加を含む、ゲームクライアントまたはサービスに統合する詳しい手順です。

O3DE

ゲームサーバー

[GameLift サーバー SDK for C++]を使用して、Amazon GameLift でホストできるようにゲームサーバーを準備します。必要な機能をゲームサーバーに統合する方法については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

ゲームクライアントとゲームサービス

ゲームクライアントやゲームサービスが Amazon GameLift のサービスとの連動を有効にし、利用可能なゲームセッションの検索や新規作成、ゲームへの参加者の追加などを行うことができます。コアクライアント機能については、[AWSSDK for C++]（SDK for C++ ）で説明されています。Amazon GameLift を O3DE ゲームプロジェクトに統合するには、「Amazon GameLift を O3DE ゲームクライアントとサーバーに追加する」と「ゲームクライアント GameLift に Amazon を追加する」を参照してください。

Unreal Engine

ゲームサーバー

Amazon GameLift でホストするゲームサーバーを準備するために、[Unreal Engine 用のサーバー] をプロジェクトに追加し、必要なサーバー機能を実装します。Unreal Engine プラグインのセットアップと Amazon GameLift コードの追加については、「Amazon GameLift を Unreal Engine プロジェクトに統合する」を参照してください。

ゲームクライアントとゲームサービス

ゲームクライアントやゲームサービスが Amazon GameLift のサービスとの連動を有効にし、利用可能なゲームセッションの検索や新規作成、ゲームへの参加者の追加などを行うことができます。コアクライアント機能については、[AWSSDK for C++]（SDK for C++ ）で説明されています。Amazon GameLift を Unreal Engine ゲームプロジェクトに統合するには、「ゲームクライアント GameLift に Amazon を追加する」を参照してください。

Unity

ゲームサーバー

Amazon GameLift でホストするゲームサーバーを準備するために、[Amazon GameLift Server SDK for C#] をプロジェクトに追加し、必要なサーバー機能を実装します。Unity での設定と Amazon GameLift コードの追加については、「Unity プロジェクトに Amazon GameLift を統合する」を参照してください。

ゲームクライアントとゲームサービス

ゲームクライアントやゲームサービスが Amazon GameLift のサービスとの連動を有効にし、利用可能なゲームセッションの検索や新規作成、ゲームへの参加者の追加などを行うことができます。コアクライアント機能については、AWS SDK for .NET で説明されています。Amazon GameLift を Unity ゲームプロジェクトに統合するには、「ゲームクライアント GameLift に Amazon を追加する」を参照してください。

他のエンジン

ゲームサーバーとクライアントで利用可能な Amazon GameLift SDK リストについては、「the section called “開発サポート”」を参照してください。

Amazon GameLift を O3DE ゲームクライアントとサーバーに追加する

オープンソースのクロスプラットフォームのリアルタイム 3D エンジンである O3DE を使用して、ゲームやシミュレーションなど、高性能でインタラクティブなエクスペリエンスを作成できます。O3DE レンダラーとツールはモジュラーフレームワークにまとめられており、好みの開発ツールで変更や拡張が可能です。

モジュラーフレームワークでは、標準のインターフェイスとアセットを備えたライブラリを含む Gem を使用します。独自の Gem を選択し、要件に基づいて追加する機能を選択します。

Amazon GameLift Gem には次の機能があります。

Amazon GameLift 統合

O3DE ネットワークレイヤーを拡張し、Multiplayer Gem を Amazon GameLift 専用サーバーソリューションと連携させるためのフレームワーク。Gem は Amazon GameLift サーバー SDK と (Amazon GameLift サービス自体を呼び出す) AWS SDK クライアントの両方との統合を可能にします。

ビルドとパッケージ管理

専用サーバービルドと AWS Cloud Development Kit (AWS CDK) (AWS CDK) アプリケーションをパッケージ化し、オプションでアップロードしてリソースをセットアップおよび更新する手順。

Amazon GameLift Gem のセットアップ

このセクションの手順に従って、O3DE で Amazon GameLift Gem をセットアップしてください。

前提条件

- Amazon GameLift 用の AWS アカウントをセットアップする 詳細については、「のセットアップ AWS アカウント」を参照してください。
- O3DE の AWS 認証情報をセットアップします。詳細については、「AWS 認証情報の設定」を参照してください。

- AWS CLI と AWS CDK をセットアップします。詳細については、「AWS Command Line Interface」および「AWS Cloud Development Kit (AWS CDK)」を参照してください。

Amazon GameLift Gem とその依存関係をオンにする

1. [プロジェクトマネージャー] を開きます。
2. プロジェクトの下のメニューを開き、[プロジェクト設定を編集...] を選択します。
3. [Gem を設定] を選択します。
4. Amazon GameLift Gem と以下の依存 Gem をオンにします。
   - [AWS Core Gem] – O3DE で AWS のサービス を使用するフレームワークを提供します。
   - [Multiplayer Gem] – ネットワークフレームワークを拡張することでマルチプレイヤー機能を提供します。

Amazon GameLift Gem 静的ライブラリを含める

1. `Gem::AWSGameLift.Server.Static` をプロジェクトサーバーターゲットの `BUILD_DEPENDENCIES` として含めます。

```
ly_add_target(
    NAME YourProject.Server.Static STATIC
    ...
    BUILD DEPENDCIES
        PUBLIC
            ...
        PRIVATE
            ...
            Gem::AWSGameLift.Server.Static
)
```

2. `AWSGameLiftService` をプロジェクトサーバーのシステムコンポーネントに必須に設定します。

```
void
 YourProjectServerSystemComponent::GetRequiredServices(AZ::ComponentDescriptor::DependencyA
 required)
{
    ...
    required.push_back(AZ_CRC_CE("AWSGameLiftServerService"));
```

```
    ...
}
```

3. (オプション) C++ で Amazon GameLift サービスリクエストを行うには、クライアントターゲットの BUILD_DEPENDENCIES に Gem::AWSGameLift.Client.Static を含めます。

```
ly_add_target(
    NAME YourProject.Client.Static STATIC
    ...
    BUILD_DEPENDENCIES
    PUBLIC
        ...
    PRIVATE
        ...
        Gem::AWSCore.Static
        Gem::AWSGameLift.Client.Static
}
```

ゲームと専用サーバーを統合する

[セッション管理統合] を使用して、ゲーム内および専用ゲームサーバー内のゲームセッションを管理します。FlexMatch をサポートするには、「FlexMatch の統合」を参照してください。

Amazon GameLift を Unreal Engine プロジェクトに統合する

このトピックでは、Unreal Engine 用の Amazon GameLift C++ サーバー SDK プラグインをセットアップし、ゲームプロジェクトに統合する方法について説明します。

その他のリソース:

- Unreal ダウンロードサイト用のサーバー SDK プラグイン
- Amazon GameLift Unreal Engine サーバー SDK 5.x リファレンス
- the section called “開発サポート”

前提条件

続行する前に、以下の前提条件を必ず確認してください。

前提条件

- Unreal Engine を実行できるコンピューター。Unreal Engine の要件の詳細については、Unreal Engine の「ハードウェアとソフトウェアの仕様」ドキュメントを参照してください。
- Microsoft Visual Studio 2019 バージョン。
- CMake バージョン 3.1 以降
- Python バージョン 3.6 以降。
- PATH 上で使用可能な Git クライアント。
- Epic のゲームアカウント。Unreal Engine の公式ウェブサイトでアカウントをサインナップしてください。
- Unreal Engine アカウント GitHub に関連付けられたアカウント。詳細については、Unreal Engine ウェブサイトの「 での Unreal Engine ソースコードへのアクセス GitHub」を参照してください。

> **Note**
>
> Amazon GameLift は現在、Unreal Engine の次のバージョンをサポートしています。
>
> - 4.22
> - 4.23
> - 4.24
> - 4.25
> - 4.26
> - 4.27
> - 5.1.0
> - 5.1.1
> - 5.2
> - 5.3

Unreal Engine をソースから構築する

Epic ランチャーからダウンロードした Unreal Engine エディタの標準バージョンでは、Unreal クライアントアプリケーションのビルドのみを使用できます。Unreal サーバーアプリケーションを構築するには、Unreal Engine Github リポジトリを使用して、ソースから Unreal Engine をダウンロード

して構築する必要があります。詳細については、Unreal Engine ドキュメンテーションのウェブサイトの「ソースからの Unreal Engine の構築」チュートリアルを参照してください。

> **Note**
>
> まだアクセスしていない場合は、「 での Unreal Engine ソースコードへのアクセス GitHub」の指示に従って、 GitHub アカウントを Epic Games アカウントにリンクします。

Unreal Engine ソースを開発環境にクローンするには

1. Unreal Engine ソースをお好きなブランチの開発環境にクローンます。

   ```
   git clone https://github.com/EpicGames/UnrealEngine.git
   ```

2. ゲームの開発に使用しているバージョンのタグを確認してください。例えば、次の例では Unreal Engine バージョン 5.1.1 をチェックします。

   ```
   git checkout tags/5.1.1-release -b 5.1.1-release
   ```

3. ローカルリポジトリのルートフォルダに移動します。ルートフォルダに移動したら、`Setup.bat` ファイルを実行します。
4. ルートフォルダで、`GenerateProjectFiles.bat` ファイルも実行します。
5. 前のステップのファイルを実行すると、Unreal Engine ソリューションファイル、`UE5.sln` が作成されます。Visual Studio を開き、Visual Studio エディタで `UE5.sln` ファイルを開きます。
6. Visual Studio で [表示] メニューを開き、[ソリューションエクスプローラー] オプションを選択します。Unreal プロジェクトノードのコンテキストメニューが開きます。[ソリューションエクスプローラー] ウィンドウで `UE5.sln` ファイル (単に `UE5` としてリストされる場合があります) を右クリックし、[構築] を選択して Development Editor Win64 ターゲットで Unreal プロジェクトを構築します。

   > **Note**
   >
   > ビルドが完了するまで最大 1 時間かかることがあります。

ビルドが完了すると、Unreal Development Editor を開いてプロジェクトを作成またはインポートすることができます。

Unreal プロジェクトをプラグイン用に設定します。

Unreal Engine 用 Amazon GameLift サーバー SDK プラグインをゲームサーバープロジェクト用に準備するには、次の手順に従います。

プラグイン用にプロジェクトを設定するには

1. Visual Studio を開いた状態で、[ソリューションエクスプローラー] ペインに移動し、UE5 ファイルを選択して Unreal プロジェクトのコンテキストメニューを開きます。コンテキストメニューで、[スタートアッププロジェクトとして設定] オプションを選択します。
2. Visual Studio ウィンドウの上部にある [デバッグを開始] (緑色の矢印) を選択します。

   このアクションは Unreal Editor の新しいソースビルドインスタンスを起動します。Unreal Editor の使用に関する詳細は、Unreal Engine ドキュメントのウェブサイトにある「Unreal Editor インターフェイス」を参照してください。
3. Unreal Editor は Unreal プロジェクトとゲームプロジェクトを含む別の Visual Studio ウィンドウを開くので、開いた Visual Studio ウィンドウを閉じます。
4. Unreal Editor で、以下のいずれかを実行します。
   - Amazon と統合する既存の Unreal プロジェクトを選択します GameLift。
   - 新しい プロジェクトを作成します。Unreal 用 Amazon GameLift プラグインを試すには、Unreal エンジンの 3 人称テンプレートを試してください。このテンプレートの詳細については、Unreal Engine ドキュメンとのウェブサイトにある「Third Person テンプレート」を参照してください。

     または、以下の設定で新しいプロジェクトを設定します。

     - C++
     - スターターコンテンツを使用
     - Desktop
     - プロジェクト名 このトピックの例では、プロジェクトに `GameLiftUnrealApp` という名前を付けました。
5. Visual Studio の [ソリューションエクスプローラー] で、Unreal プロジェクトの場所に移動します。Unreal `Source` フォルダーで、*`Your-application-name`*`.Target.cs` という名前のファイルを検索します。

   例: `GameLiftUnrealApp.Target.cs`。

6. このファイルのコピーを作成し、*Your-application-name*Server.Target.cs という名前を付けます。
7. 新しいファイルを開き、以下の変更を加えます。
   - class と constructor をファイル名と一致するように変更します。
   - Type を TargetType.Game から TargetType.Server に変更します。
   - 最終的なファイルは、以下の例のようになります。

```
public class GameLiftUnrealAppServerTarget : TargetRules
 {
     public GameLiftUnrealAppServerTarget(TargetInfo Target) : base(Target)
     {
         Type = TargetType.Server;
         DefaultBuildSettings = BuildSettingsVersion.V2;
         IncludeOrderVersion = EngineIncludeOrderVersion.Unreal5_1;
         ExtraModuleNames.Add("GameLiftUnrealApp");
     }
 }
```

これで、プロジェクトが Amazon GameLift サーバー SDK プラグインを受け入れるように設定されました。

次のタスクは、Unreal 用 C++ サーバー SDK ライブラリを構築して、プロジェクトにインポートできるようにすることです。

Unreal 用 C++ サーバー SDK ライブラリを構築するには

1. Unreal 用の Amazon GameLift C++ サーバー SDK プラグインをダウンロードします。

> **Note**
>
> SDK をデフォルトのダウンロードディレクトリに置くと、パスが 260 文字の制限を超えるため、ビルドが失敗する可能性があります。例：C:\Users\Administrator\Downloads\GameLift-SDK-Release-06_15_2023\GameLift-Cpp-ServerSDK-5.0.4 C:\GameLift-Cpp-ServerSDK-5.0.4 など、SDK を別のディレクトリに移動することをおすすめします。

2. OpenSSL をダウンロードし、インストールします。OpenSSL のダウンロードについて詳しくは、Github の「OpenSSL のビルドとインストール」ドキュメントをご覧ください。

   詳細については、OpenSSL の「Windows プラットフォームに関する注意事項」ドキュメントを参照してください。

   > **Note**
   >
   > Amazon GameLift サーバー SDK の構築に使用する OpenSSL のバージョンは、ゲームサーバーをパッケージ化するために Unreal が使用する OpenSSL のバージョンと一致する必要があります。バージョン情報は、Unreal インストールディレクトリ にあります...Engine\Source\ThirdParty\OpenSSL。

3. ライブラリをダウンロードしたら、Unreal Engine 用 C++ サーバー SDK ライブラリを構築します。

   ダウンロードした SDK の GameLift-Cpp-ServerSDK-*<version>* ディレクトリで、-DBUILD_FOR_UNREAL=1 パラメータを指定してコンパイルし、サーバー SDK を構築します。次の例は、cmake を使用してコンパイルする方法を示しています。

   ターミナルで以下のコマンドを実行します。

   ```
   mkdir cmake-build
   cmake.exe -G "Visual Studio 17 2022" -DCMAKE_BUILD_TYPE=Release -S . -B ./cmake-build -DBUILD_FOR_UNREAL=1 -A x64
   cmake.exe --build ./cmake-build --target ALL_BUILD --config Release
   ```

   Windows ビルドでは、out\gamelift-server-sdk\Release フォルダに次のバイナリファイルが作成されます。

   - cmake-build\prefix\bin\aws-cpp-sdk-gamelift-server.dll
   - cmake-build\prefix\bin\aws-cpp-sdk-gamelift-server.lib

   2 つのライブラリファイルを Amazon GameLift Unreal Engine プラグインパッケージの ThirdParty\GameLiftServerSDK\Win64フォルダにコピーします。

Amazon GameLift プラグインをサンプルプロジェクトにインポートするには、次の手順に従います。

Amazon GameLift プラグインをインポートする

1. 前の手順でプラグインから抽出したGameLiftServerSDKフォルダを見つけます。
2. ゲームプロジェクトのルートフォルダPluginsで を見つけます。(フォルダが存在しない場合は、そこに作成します。)
3. GameLiftServerSDK フォルダを にコピーしますPlugins。

   これにより、Unreal プロジェクトはプラグインを表示できます。
4. Amazon GameLift サーバー SDK プラグインをゲームの .uproject ファイルに追加します。

   この例では、アプリは GameLiftUnrealApp と呼ばれるため、ファイルは GameLiftUnrealApp.uproject になります。
5. .uproject ファイルを編集してプラグインをゲームプロジェクトに追加します。

```
"Plugins": [
    {
        "Name": "GameLiftServerSDK",
        "Enabled": true
    }
]
```

6. ゲームの がプラグイン ModuleRules に依存していることを確認します。.Build.cs ファイルを開き、Amazon GameLiftServerSDK の依存関係を追加します。このファイルは *Your-application-name*/Source/*/Your-application-name/* の下にあります。

   例えば、チュートリアルのファイルパスは ../GameLiftUnrealApp/Source/GameLiftUnrealApp/GameLiftUnrealApp.Build.cs です。
7. PublicDependencyModuleNames のリストの最後に "GameLiftServerSDK" を追加します。

```
using UnrealBuildTool;
using System.Collections.Generic;
public class GameLiftUnrealApp : ModuleRules
  {
    public GameLiftUnrealApp(TargetInfo Target)
    {
      PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject",
 "Engine", "InputCore", "GameLiftServerSDK" });
      bEnableExceptions = true;
    }
```

```
}
```

これで、プラグインはアプリケーションで動作するようになります。次のセクションに進み、Amazon GameLift 機能をゲームに統合します。

Unreal プロジェクトに Amazon GameLift サーバーコードを追加する

Unreal Engine 環境を設定してセットアップし、ゲームサーバーを Amazon と統合できるようになりました GameLift。このトピックで説明するコードは、Amazon GameLift サービスに必要な呼び出しを行います。また、Amazon GameLift サービスからのリクエストに応答する一連のコールバック関数も実装されています。各関数とコードの機能について詳しくは、「サーバープロセスの初期化」を参照してください。このコードで使用されている SDK アクションとデータ型についての詳細は、「Unreal Engine 用 Amazon GameLift サーバー SDK リファレンス」を参照してください。

Amazon でゲームサーバーを初期化するには GameLift、次の手順を使用します。

> **Note**
>
> 次のセクションで提供される Amazon GameLift固有のコードは、`WITH_GAMELIFT`プリプロセッサフラグの使用によって異なります。このフラグは、以下の両方の条件が満たされる場合にのみ適用されます。
>
> - `Target.Type == TargetRules.TargetType.Server`
> - プラグインは Amazon GameLift サーバー SDK バイナリを見つけました。
>
> これにより、Unreal Server ビルドのみが Amazon GameLiftのバックエンド API を呼び出すようになります。また、ゲームが生成する可能性のあるさまざまな Unreal ターゲットすべてに対して正しく実行されるコードを記述できます。

ゲームサーバーを Amazon と統合する GameLift

1. Visual Studio で、アプリケーションの `.sln` ファイルを開きます。この例では、`GameLiftUnrealApp.sln` ファイルはルートフォルダーにあります。
2. ソリューションを開いた状態で、アプリケーションの *`Your-application-name`*`GameMode.h` ファイルを見つけます。例えば、`GameLiftUnrealAppGameMode.h` などです。
3. ヘッダーファイルを次のサンプルコードに合わせて変更します。必ずGameLiftUnrealApp「」を独自のアプリケーション名に置き換えてください。

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/GameModeBase.h"
#include "GameLiftServerSDK.h"
#include "GameLiftUnrealAppGameMode.generated.h"

DECLARE_LOG_CATEGORY_EXTERN(GameServerLog, Log, All);

UCLASS(minimalapi)
class AGameLiftUnrealAppGameMode : public AGameModeBase
{
    GENERATED_BODY()

public:
    AGameLiftUnrealAppGameMode();

protected:
    virtual void BeginPlay() override;

private:
    // Process Parameters needs to remain in scope for the lifetime of the app
    FProcessParameters m_params;

    void InitGameLift();
};
```

4. 関連するソースファイルの *Your-application-name*GameMode.cpp を開きます。この例では: GameLiftUnrealAppGameMode.cpp ですが、次のサンプルコードに合うようにコードを変更してください。必ずGameLiftUnrealApp「」を独自のアプリケーション名に置き換えてください。

   このサンプルは、「Amazon をゲームサーバーに追加する」で説明されているように GameLift、Amazon との統合に必要なすべての要素を追加する方法を示しています。 GameLift これには、以下が含まれます。

   - Amazon GameLift API クライアントの初期化。
   - コールバック関数を実装して、、OnStartGameSession、 OnProcessTerminateなどの Amazon GameLift サービスからのリクエストに応答しますonHealthCheck。

- 指定されたポートで ProcessReady() を呼び出して、ゲームセッションをホストする準備 GameLiftservice ができたら Amazon に通知します。

```
#include "GameLiftUnrealAppGameMode.h"
#include "GameLiftUnrealAppCharacter.h"

#include "UObject/ConstructorHelpers.h"

DEFINE_LOG_CATEGORY(GameServerLog);

AGameLiftUnrealAppGameMode::AGameLiftUnrealAppGameMode()
{
    // set default pawn class to our Blueprinted character
    static ConstructorHelpers::FClassFinder<APawn> PlayerPawnBPClass(TEXT("/Game/
ThirdPerson/Blueprints/BP_ThirdPersonCharacter"));
    if (PlayerPawnBPClass.Class != NULL)
    {
        DefaultPawnClass = PlayerPawnBPClass.Class;
    }
}

void AGameLiftUnrealAppGameMode::BeginPlay()
{
#if WITH_GAMELIFT
    InitGameLift();
#endif
}

void AGameLiftUnrealAppGameMode::InitGameLift()
{
    UE_LOG(GameServerLog, Log, TEXT("Initializing the GameLift Server"));

    //Getting the module first.
    FGameLiftServerSDKModule* gameLiftSdkModule =
 &FModuleManager::LoadModuleChecked<FGameLiftServerSDKModule>(FName("GameLiftServerSDK"));

    //Define the server parameters for a GameLift Anywhere fleet. These are not
 needed for a GameLift managed EC2 fleet.
    FServerParameters serverParameters;

    //AuthToken returned from the "aws gamelift get-compute-auth-token" API. Note
 this will expire and require a new call to the API after 15 minutes.
```

```
   if (FParse::Value(FCommandLine::Get(), TEXT("-authtoken="),
serverParameters.m_authToken))
   {
       UE_LOG(GameServerLog, Log, TEXT("AUTH_TOKEN: %s"),
*serverParameters.m_authToken)
   }

   //The Host/compute-name of the GameLift Anywhere instance.
   if (FParse::Value(FCommandLine::Get(), TEXT("-hostid="),
serverParameters.m_hostId))
   {
       UE_LOG(GameServerLog, Log, TEXT("HOST_ID: %s"), *serverParameters.m_hostId)
   }

   //The Anywhere Fleet ID.
   if (FParse::Value(FCommandLine::Get(), TEXT("-fleetid="),
serverParameters.m_fleetId))
   {
       UE_LOG(GameServerLog, Log, TEXT("FLEET_ID: %s"),
*serverParameters.m_fleetId)
   }

   //The WebSocket URL (GameLiftServiceSdkEndpoint).
   if (FParse::Value(FCommandLine::Get(), TEXT("-websocketurl="),
serverParameters.m_webSocketUrl))
   {
       UE_LOG(GameServerLog, Log, TEXT("WEBSOCKET_URL: %s"),
*serverParameters.m_webSocketUrl)
   }

   //The PID of the running process
   serverParameters.m_processId = FString::Printf(TEXT("%d"),
GetCurrentProcessId());
   UE_LOG(GameServerLog, Log, TEXT("PID: %s"), *serverParameters.m_processId);

   //InitSDK establishes a local connection with GameLift's agent to enable
further communication.
   //Use InitSDK(serverParameters) for a GameLift Anywhere fleet.
   //Use InitSDK() for a GameLift managed EC2 fleet.
   gameLiftSdkModule->InitSDK(serverParameters);

   //Implement callback function onStartGameSession
   //GameLift sends a game session activation request to the game server
   //and passes a game session object with game properties and other settings.
```

```
    //Here is where a game server takes action based on the game session object.
    //When the game server is ready to receive incoming player connections,
    //it invokes the server SDK call ActivateGameSession().
    auto onGameSession = [=](Aws::GameLift::Server::Model::GameSession gameSession)
    {
        FString gameSessionId = FString(gameSession.GetGameSessionId());
        UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"),
*gameSessionId);
        gameLiftSdkModule->ActivateGameSession();
    };

    m_params.OnStartGameSession.BindLambda(onGameSession);

    //Implement callback function OnProcessTerminate
    //GameLift invokes this callback before shutting down the instance hosting this
game server.
    //It gives the game server a chance to save its state, communicate with
services, etc.,
    //and initiate shut down. When the game server is ready to shut down, it
invokes the
    //server SDK call ProcessEnding() to tell GameLift it is shutting down.
    auto onProcessTerminate = [=]()
    {
        UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
        gameLiftSdkModule->ProcessEnding();
    };

    m_params.OnTerminate.BindLambda(onProcessTerminate);

    //Implement callback function OnHealthCheck
    //GameLift invokes this callback approximately every 60 seconds.
    //A game server might want to check the health of dependencies, etc.
    //Then it returns health status true if healthy, false otherwise.
    //The game server must respond within 60 seconds, or GameLift records 'false'.
    //In this example, the game server always reports healthy.
    auto onHealthCheck = []()
    {
        UE_LOG(GameServerLog, Log, TEXT("Performing Health Check"));
        return true;
    };

    m_params.OnHealthCheck.BindLambda(onHealthCheck);

    //The game server gets ready to report that it is ready to host game sessions
```

```
    //and that it will listen on port 7777 for incoming player connections.
    m_params.port = 7777;

    //Here, the game server tells GameLift where to find game session log files.
    //At the end of a game session, GameLift uploads everything in the specified
    //location and stores it in the cloud for access later.
    TArray<FString> logfiles;
    logfiles.Add(TEXT("GameLift426Test/Saved/Logs/GameLift426Test.log"));
    m_params.logParameters = logfiles;

    //The game server calls ProcessReady() to tell GameLift it's ready to host game
 sessions.
    UE_LOG(GameServerLog, Log, TEXT("Calling Process Ready"));
    gameLiftSdkModule->ProcessReady(m_params);
}
```

5. 開発エディターと開発サーバーの両方のターゲットタイプのゲームプロジェクトを構築します。

> **Note**
>
> ソリューションを再構築する必要はありません。代わりに、アプリ名と一致する Games フォルダの下にプロジェクトだけを構築します。そうしないと、Visual Studio は、UE5 プロジェクト全体を再構築します。これには、最大で 1 時間かかる場合があります。

6. 両方のビルドが完了したら、Visual Studio を閉じてプロジェクトの `.uproject` ファイル Unreal Editor で開きます。
7. Unreal Editor で、ゲームのサーバービルドをパッケージ化します。ターゲットを選択するには、プラットフォーム 、Windows に移動し、***Your-application-name*Server**を選択します。
8. サーバーアプリケーションの構築プロセスを開始するには、[プラットフォーム]、[Windows] に移動し、[パッケージプロジェクト] を選択します。ビルドが完了すると、実行ファイルが作成される必要があります。この例の場合、ファイル名は `GameLiftUnrealAppServer.exe` です。
9. Unreal Editor でサーバーアプリケーションをビルドすると、2 つの実行ファイルが生成されます。1 つはゲームビルドフォルダのルートにあり、実際のサーバー実行ファイルのラッパーとして機能します。

   サーバービルドで Amazon GameLift フリートを作成するときは、実際のサーバー実行可能ファイルをランタイム設定の起動パスとして渡すことをお勧めします。例えば、ゲームビルドフォルダには、ルートに `GameLiftFPS.exe` ファイルがあり、別のファイルが

\GameLiftFPS\Binaries\Win64\GameLiftFPSServer.exe にある場合があります。フリートを作成するときは、ランタイム設定の起動パスとして C:\GameLiftFPS\Binaries\Win64\GameLiftFPSServer.exe を使用することをおすすめします。

10. ゲームサーバーがゲームクライアントと通信できるように、Amazon GameLift フリートで必要な UDP ポートを必ず開いてください。デフォルトでは、Unreal Engine は 7777 ポートを使用します。詳細については、「Amazon GameLift サービス API リファレンスガイドUpdateFleetPortSettings」の「」を参照してください。
11. ゲームビルド用の install.bat ファイルを作成します。このインストールスクリプトは、ゲームビルドが Amazon GameLift フリートにデプロイされるたびに実行されます。サンプル install.bat ファイルは次のとおりです。

```
VC_redist.x64.exe /q
UE5PrereqSetup_x64.exe /q
```

Unreal Engine の一部のバージョンでは、代わりに を にinstall.batする必要があります。

```
VC_redist.x64.exe /q
UEPrereqSetup_x64.exe /q
```

> **Note**
>
> <>PrereqSetup_x64.exe ファイルへのファイルパスは Engine\Extras\Redist\en-us です。

12. これで、ゲームビルドをパッケージ化して Amazon にアップロードできます GameLift。

ゲームビルドでパッケージ化する OpenSSL のバージョンは、ゲームエンジンがゲームサーバーの構築時に使用したバージョンと一致する必要があります。ゲームサーバーのビルドには、必ず正しい OpenSSL バージョンをパッケージ化してください。Windows OS の場合、OpenSSL 形式は .dll です。

> **Note**
>
> OpenSSL DLL をゲームサーバービルドにパッケージ化します。 DLLs ゲームサーバーの構築時に使用したのと同じバージョンの OpenSSL をパッケージ化してください。
>
> - libssl-1_1-x64.dll

```
libcrypto-1_1-x64.dll
```

依存関係をゲームサーバーの実行可能ファイルとともに zip ファイルのルートにパッケージ化します。例えば、`openssl-lib` dll は `.exe` ファイルと同じディレクトリにある必要があります。

次のステップ

Unreal Engine 環境を設定してセットアップし、Amazon GameLift をゲームに統合できるようになりました。

Amazon GameLift をゲームに追加する方法の詳細については、以下を参照してください。

- Amazon GameLift をゲームサーバーに追加する
- Unreal Engine 用 Amazon GameLift サーバー SDK リファレンス

ゲームのテストに関する手順については、Amazon GameLift Anywhere フリートを使用して統合をテストする を参照してください。

Unity プロジェクトに Amazon GameLift を統合する

このトピックでは、Amazon GameLift の Unity 用の C# サーバー SDK プラグインをセットアップし、ゲームプロジェクトに統合する方法について説明します。

その他のリソース:

- Amazon GameLift Server SDK のダウンロードサイト
- C# と Unity 用 Amazon GameLift サーバー SDK 5.x リファレンス
- the section called “開発サポート”

前提条件

Unity 用の Amazon GameLift C# サーバー SDK プラグインを使用するには、以下のコンポーネントが必要です。

- プラグインがサポートする開発環境と Unity Editor バージョン (「Amazon での開発サポート GameLift」を参照)。Unity バージョンについての情報は、Unity ドキュメントの「Unity のシステム要件」を参照してください。
- Unity パッケージ用の Amazon GameLift サーバー SDK プラグイン。このパッケージには C# 用のサーバー SDK 5+ が含まれています。パッケージは、「Amazon GameLift の開始方法」のサイトからダウンロードできます。
- サードパーティのスコープ設定されたレジストリ UnityNuGet。このツールはサードパーティ DLL を管理します。詳細については、UnityNuGet GitHub リポジトリを参照してください。

UnityNuGet のセットアップ

ゲームプロジェクトに UnityNuget をセットアップしていない場合は、以下の手順に従って Unity パッケージマネージャーを使用してツールをインストールしてください。または、NuGet CLI を使用して DLL を手動でダウンロードすることもできます。詳細については、Unity 用の Amazon GameLift C# サーバー SDK README を参照してください。

UnityNuGet を ゲームプロジェクトに統合するには

1. Unity Editor でプロジェクトを開いた状態で、メインメニューに移動し、[編集]、[プロジェクト設定] を選択します。オプションから [パッケージマネージャー] セクションを選択し、[スコープ設定レジストリ] グループを開きます。
2. [+] ボタンを選択し、UnityNuGet スコープ設定レジストリに以下の値を入力します。

   ```
   Name: Unity NuGet
   URL: https://unitynuget-registry.azurewebsites.net
   Scope(s): org.nuget
   ```

3. Unity 2021 バージョンのユーザー:

   UnityNuget をセットアップしたら、Unity コンソールにエラーが表示されていないか確認します。これらのエラーは、NuGet パッケージ内の厳密な名前付きアセンブリのバインディングリダイレクトが Unity プロジェクト内のパスに正しく解決されない場合に発生します。この問題を解決するには、Unity のアセンブリバージョンの検証を設定します。

   a. Unity Editor でメインメニューに移動し、[編集]、[プロジェクト設定] を選択し、[プレイヤー] セクションを開きます。

   b. [アセンブリバージョン検証] オプションの選択を解除します。

プラグインをインストールする

以下の手順を使用して Unity 用 Amazon GameLift C# サーバー SDK プラグインをインストールし、log4net 口ギングを設定します。

プラグインをインストールするには

1. Unity Editor でプロジェクトを開いた状態で、メインメニューに移動し、[ウィンドウ」、[パッケージマネージャー] を選択します。
2. [+] ボタンを選択して新しいパッケージを追加します。[tarball からパッケージを追加] オプションを選択します。
3. [ディスクでパッケージを選択] で、Unity ダウンロードファイル用の Amazon GameLift C# サーバー SDK プラグインを探し、Amazon GameLift Server SDK `.tgz` ファイルを選択します。[開く] を選択してプラグインをインストールします。

Amazon GameLift サーバー SDK は log4net フレームワークを使用してログメッセージを出力します。デフォルトではサーバービルドのターミナルにメッセージを出力するように設定されていますが、Unity ではファイル口ギングのサポートを追加するための設定が必要です。Amazon GameLift サーバー SDK パッケージ内にあるサンプルをインポートして、このサポートをプロジェクトに追加できます。以下の手順に従って、サンプルを追加し、log4net を設定します。

log4net をファイル出力用に設定するには

1. Unity Editor でプロジェクトを開いた状態で、メインメニューに移動し、[ウィンドウ」、[パッケージマネージャー] を選択します。
2. ドロップダウンメニューから [パッケージ: プロジェクト内] を選択し、パッケージのリストから [Amazon GameLift サーバー SDK] を選択します。これによりパッケージの詳細が開きます。
3. パッケージの詳細で、[サンプル] グループオプションを選択し、[インポート] を押します。
4. `log4net.config` ファイルと付属の `LoggingConfiguration.cs` スクリプトによって構成が自動的に実行され、プロジェクトの `Assets/Samples` フォルダーに設定されました。

> **Note**
>
> `log4net.config` ファイルをプロジェクト内の別のフォルダーに移動する必要がある場合は、スクリプト `LoggingConfiguration.cs` 内の config ファイルのファイルパスも新しいパスで更新する必要があります。詳細については、「log4net の設定に関する log4net のマニュアル」を参照してください。

詳細な手順とテストガイダンスについては、「プラグインのダウンロードにある README」を参照してください。

Amazon GameLift Anywhere フリートをテスト用にセットアップする

開発ワークステーションを Amazon GameLift Anywhere ホスティングフリートとしてセットアップして、Amazon GameLift 統合を繰り返しテストできます。この設定では、ワークステーションでゲームサーバープロセスを開始し、プレイヤー参加リクエストまたはマッチメーキングリクエストを Amazon GameLift に送信してゲームセッションを開始し、クライアントを新しいゲームセッションに接続できます。独自のワークステーションをホスティングサーバーとしてセットアップすると、Amazon GameLift とのゲーム統合のあらゆる側面をモニタリングできます。

ワークステーションの設定方法については、「Amazon GameLift Anywhere フリートを使用して統合をテストする」を参照して次の手順を完了してください。

1. ワークステーション用のカスタムロケーションを作成します。
2. 新しいカスタムロケーションを使用して Amazon GameLift Anywhere フリートを作成します。成功すると、このリクエストはフリート ID を返します。この値をメモしておきます。これは後で必要になります。
3. ワークステーションを新しい Anywhere フリートにコンピューティングとして登録します。一意のコンピューティング名を入力し、ワークステーションの IP アドレスを指定します。成功すると、このリクエストはサービス SDK エンドポイントを WebSocket URL の形式で返します。この値をメモしておきます。これは後で必要になります。
4. ワークステーションコンピューティング用の認証トークンを生成します。この短期間の認証には、トークンと有効期限が含まれます。ゲームサーバーは、これを使用して Amazon GameLift サービスとの通信を認証します。実行中のゲームサーバープロセスがアクセスできるように、認証をワークステーションのコンピューティングに保存します。

Amazon GameLift サーバーコードを Unity プロジェクトに追加する

ゲームサーバーは Amazon GameLift サービスと通信して、指示を受け取り、進行中のステータスを報告します。これを実現するには、Amazon GameLift サーバー SDK を使用するゲームサーバーコードを追加します。

提供されているコード例は、必要とされる基本的な統合要素を示しています。ここでは、`MonoBehavior` を使用して Amazon GameLift でゲームサーバーのシンプルな初期化について示します。この例では、ゲームサーバーがテスト用に Amazon GameLift Anywhere フリート上で動作していることを前提としています。これには以下のためのコードが含まれています。

- Amazon GameLift API クライアントを初期化する。このサンプルでは、Anywhere フリートとコンピューティングのサーバーパラメータを含む `InitSDK()` のバージョンを使用しています。前の Amazon GameLift Anywhere フリートをテスト用にセットアップする のトピックで定義した WebSocket URL、フリート ID、コンピューティング名 (ホスト ID)、および認証トークンを使用します。
- `OnStartGameSession`、`OnProcessTerminate`、`onHealthCheck` など、Amazon GameLift サービスからのリクエストに応答するコールバック関数を実装します。
- 指定されたポートで ProcessReady() を呼び出して、プロセスがゲームセッションをホストする準備ができたときに Amazon GameLift サービスに通知します。

このトピックで紹介するコードは、Amazon GameLift サービスとの通信を確立します。また、からのリクエストに応答する一連のコールバック関数も実装されています。各関数とコードの機能について詳しくは、「サーバープロセスの初期化」を参照してください。このコードで使用されている SDK アクションとデータ型について詳細は、「C# 用 Amazon GameLift サーバー SDK リファレンス」を参照してください。

このサンプルは、「Amazon GameLift をゲームサーバーに追加する」で説明されているように、必要な要素をすべて追加する方法を示しています。これには、以下が含まれます。

Amazon GameLift 機能を追加する方法の詳細については、以下のトピックを参照してください。

- Amazon GameLift をゲームサーバーに追加する
- C# 用 Amazon GameLift サーバー SDK リファレンス

```
using System.Collections.Generic;
using Aws.GameLift.Server;
using UnityEngine;

public class ServerSDKManualTest : MonoBehaviour
{
    //This example is a simple integration that initializes a game server process
    //that is running on an Amazon GameLift Anywhere fleet.
    void Start()
    {
        //Identify port number (hard coded here for simplicity) the game server is
 listening on for player connections
        var listeningPort = 7777;
```

```
        //WebSocketUrl from RegisterHost call
        var webSocketUrl = "wss://us-west-2.api.amazongamelift.com";

        //Unique identifier for this process
        var processId = "myProcess";

        //Unique identifier for your host that this process belongs to
        var hostId = "myHost";

        //Unique identifier for your fleet that this host belongs to
        var fleetId = "myFleet";

        //Authorization token for this host process
        var authToken = "myAuthToken";

        //Server parameters are required for a GameLift Anywhere fleet.
        //They are not required for a GameLift managed EC2 fleet.
        ServerParameters serverParameters = new ServerParameters(
            webSocketUrl,
            processId,
            hostId,
            fleetId,
            authToken);

        //InitSDK establishes a local connection with an Amazon GameLift agent
        //to enable further communication.
        var initSDKOutcome = GameLiftServerAPI.InitSDK(serverParameters);
        if (initSDKOutcome.Success)
        {
            //Implement callback functions
            ProcessParameters processParameters = new ProcessParameters(
            //Implement OnStartGameSession callback
                (gameSession) => {
                    //GameLift sends a game session activation request to the game
server
                    //with game session object containing game properties and other
settings.
                    //Here is where a game server takes action based on the game
session object.
                    //When the game server is ready to receive incoming player
connections,
                    //it invokes the server SDK call ActivateGameSession().
                    GameLiftServerAPI.ActivateGameSession();
                },
```

```
                (updateGameSession) => {
                    //GameLift sends a request when a game session is updated (such as
for
                    //FlexMatch backfill) with an updated game session object.
                    //The game server can examine matchmakerData and handle new
incoming players.
                    //updateReason explains the purpose of the update.
                },
                () => {
                    //Implement callback function OnProcessTerminate
                    //GameLift invokes this callback before shutting down the instance
hosting this game server.
                    //It gives the game server a chance to save its state, communicate
with services, etc.,
                    //and initiate shut down. When the game server is ready to shut
down, it invokes the
                    //server SDK call ProcessEnding() to tell GameLift it is shutting
down.
                    GameLiftServerAPI.ProcessEnding();
                },
                () => {
                    //Implement callback function OnHealthCheck
                    //GameLift invokes this callback approximately every 60 seconds.
                    //A game server might want to check the health of dependencies,
etc.
                    //Then it returns health status true if healthy, false otherwise.
                    //The game server must respond within 60 seconds, or GameLift
records 'false'.
                    //In this example, the game server always reports healthy.
                    return true;
                },
                //The game server gets ready to report that it is ready to host game
sessions
                //and that it will listen on port 7777 for incoming player connections.
                listeningPort,
                new LogParameters(new List<string>()
                {
                    //Here, the game server tells GameLift where to find game session
log files.
                    //At the end of a game session, GameLift uploads everything in the
specified
                    //location and stores it in the cloud for access later.
                    "/local/game/logs/myserver.log"
                }));
```

```
            //The game server calls ProcessReady() to tell GameLift it's ready to host
 game sessions.
            var processReadyOutcome =
 GameLiftServerAPI.ProcessReady(processParameters);
            if (processReadyOutcome.Success)
            {
                print("ProcessReady success.");
            }
            else
            {
                print("ProcessReady failure : " +
 processReadyOutcome.Error.ToString());
            }
        }
        else
        {
            print("InitSDK failure : " + initSDKOutcome.Error.ToString());
        }
    }

    void OnApplicationQuit()
    {
        //Make sure to call GameLiftServerAPI.ProcessEnding() and
 GameLiftServerAPI.Destroy() before terminating the server process.
        //These actions notify Amazon GameLift that the process is terminating and
 frees the API client from memory.
        GenericOutcome processEndingOutcome = GameLiftServerAPI.ProcessEnding();
        GameLiftServerAPI.Destroy();
        if (processEndingOutcome.Success)
        {
            Environment.Exit(0);
        }
        else
        {
            Console.WriteLine("ProcessEnding() failed. Error: " +
 processEndingOutcome.Error.ToString());
            Environment.Exit(-1);
        }
    }
}
```

その他のリソース

以下のリソースを使用してゲームサーバーをテストし、機能を拡張します。

- 開発マシンを Amazon GameLift Anywhere フリートとしてセットアップし、ローカルテストに使用します。「カスタムサーバー統合のテスト」を参照してください。
- ゲームサーバーを構築し、そのビルドを Amazon GameLift にアップロードします。「カスタムゲームサーバービルドを Amazon GameLift にアップロードする」を参照してください。
- ゲームサーバービルドを Amazon GameLift マネージド EC2 フリートにデプロイします。「新しい Amazon GameLift フリートを作成する」を参照してください。

Amazon GameLift Anywhere フリートを使用して統合をテストする

Amazon GameLift Anywhere フリートを使用して、Amazon GameLift とのゲームの統合を繰り返し構築してテストすることができます。Amazon GameLift サービスに接続する独自のハードウェアを Anywhere フリートとしてセットアップし、その上にゲームサーバーをインストールして実行します。テストアプリを使用して、ゲームセッションの開始/停止、プレイヤー接続の追跡、マッチメーキングのバックフィルの処理などのシナリオを実行します。Anywhere フリートがあれば、必要に応じてゲームサーバーのビルドを更新し、ホスティングアクティビティを完全に可視化できます。

Amazon GameLift Server SDK バージョン 5 以降と統合されたゲームで Amazon GameLift Anywhere フリートを使用できます。

トピック

- 初期開発
- ゲームサーバーでのイテレーション

初期開発

ゲームを開発し、Amazon GameLift サーバー SDK と統合しようとしています。統合をテストするには、ゲームサーバービルドの新しい統合を Amazon GameLift にアップロードし、フリートを作成できます。また、開発用ラップトップで Anywhere フリートを使用すると、開発とテストの繰り返しをより効率的に行うことができます。

以下の手順に従って Anywhere フリートを作成し、Amazon GameLift コンソールまたは AWS Command Line Interface () AWS CLI を使用してラップトップでゲームセッションを開始します。

Console

1. [Amazon GameLift コンソール] を開きます。
2. ナビゲーションペインの [ホスティング] で [ロケーション] を選択します。
3. [ロケーションを作成する] を選択します。
4. [ロケーションを作成する] ダイアログボックスで、次の操作を行います。
   a. [ロケーション名] を入力します。これにより、Amazon GameLift Anywhere フリートでゲームを実行するために使用するコンピューティングリソースの場所にラベル付けします。カスタムロケーション名は custom- で始まる必要があります。
   b. [Create] (作成) を選択します。
5. Anywhere フリートを作成するには、以下の手順を実行します。
   a. ナビゲーションペインの [ホスティング] で [フリート] を選択します。
   b. [フリート] ページで、[フリートの作成] を選択します。
   c. [コンピューティングタイプを選択] ステップで [Anywhere] を選択し、[次へ] を選択します。
   d. [フリートの詳細を定義] ステップで、新しいフリートを定義します。詳細については、「新しい Amazon GameLift フリートを作成する」を参照してください。
   e. [ロケーションを選択] ステップで、作成したカスタムロケーションを選択します。
   f. 残りのフリート作成ステップを完了して、Anywhere フリートを作成します。
6. 作成したフリートにラップトップをコンピューティングリソースとして登録します。`register-compute` コマンド (または `RegisterCompute` API オペレーション) を使用します。前のステップで作成した `fleet-id` を含め、`compute-name` とラップトップの `ip-address` を追加します。

```
aws gamelift register-compute \
    --compute-name DevLaptop \
    --fleet-id fleet-1234 \
    --ip-address 10.1.2.3 \
    --location custom-location-1
```

出力例:

```
Compute {
    FleetId = fleet-1234,
```

```
    ComputeName = DevLaptop,
    Status = ACTIVE,
    IpAddress = 10.1.2.3,
    GameLiftServiceSdkEndpoint = wss://12345678.execute-api.amazonaws.com/,
    Location = custom-location-1
}
```

7. ゲームサーバーのデバッグセッションを開始します。

   a. 作成したフリート内のラップトップの認証トークンを取得します。get-compute-auth-token コマンド (または GetComputeAuthToken API オペレーション) を使用します。

   ```
   aws gamelift get-compute-auth-token \
       --fleet-id fleet-1234 \
       --compute-name DevLaptop
   ```

   出力例:

   ```
   ComputeAuthToken {
     FleetId = fleet-1234,
     ComputeName = DevLaptop,
     AuthToken = abcdefg123,
     ExpirationTime = 1897492857.11
   }
   ```

   b. ゲームサーバーの実行ファイルのデバッグインスタンスを実行します。デバッグインスタンスを実行するには、ゲームサーバーが InitSDK() を呼び出す必要があります。プロセスがゲームセッションをホストする準備ができた後、ゲームサーバーは ProcessReady() を呼び出します。

8. ゲームセッションを作成して、Amazon GameLift Anywhere との初めての統合をテストします。create-game-session コマンド (または CreateGameSession API オペレーション) を使用します。フリートのカスタムロケーションを指定します。

```
aws gamelift create-game-session \
    --fleet-id fleet-1234 \
    --name DebugSession \
    --maximum-player-session-count 2 \
    --location custom-location-1
```

出力例:

```
GameSession {
    FleetId = fleet-1234,
    GameSessionId = 1111-1111,
    Name = DebugSession,
    IpAddress = 10.1.2.3,
    Port = 1024,
    ...
}
```

Amazon GameLift は、登録されたサーバープロセスに onStartGameSession() メッセージを送信します。メッセージには、ゲームプロパティ、ゲームセッションデータ、マッチメーカーデータなど、ゲームセッションに関する前のステップの GameSession オブジェクトが含まれます。

9. ゲームサーバーにロジックを追加して、サーバープロセスが onStartGameSession() メッセージに ActivateGameSession() で応答するようにします。オペレーションは、サーバーがゲームセッション作成メッセージを受信して受け入れたという確認を Amazon GameLift に送信します。詳細については、「Amazon GameLift サーバー SDK リファレンス」を参照してください。

これで、ゲームサーバーでゲームセッションが実行され、テストして繰り返し使用できます。ゲームサーバーでイテレーションを行う方法については、次のセクションに進みます。

AWS CLI

1. create-location コマンド (または CreateLocation API オペレーション) を使用してカスタムロケーションを作成します。カスタムロケーションは、Amazon GameLift が Anywhere フリートでゲームを実行するために使用するハードウェアの場所にラベル付けします。

```
aws gamelift create-location \
    --location-name custom-location-1
```

出力例:

```
{
    Location {
```

```
        LocationName = custom-location-1
    }
}
```

2. create-fleet コマンド (または CreateFleet API オペレーション) を使用して、カスタムロケーションで Anywhere フリートを作成します。Amazon GameLift は、お客様のホームリージョンとお客様が指定したカスタムロケーションにフリートを作成します。

```
aws gamelift create-fleet \
    --name LaptopFleet \
    --compute-type ANYWHERE \
    --locations "location=custom-location-1"
```

出力例:

```
Fleet {
    Name = LaptopFleet,
    ComputeType = ANYWHERE,
    FleetId = fleet-1234,
    Status = ACTIVE
    ...
}
```

3. 作成したフリートにラップトップをコンピューティングリソースとして登録します。register-compute コマンド (または RegisterCompute API オペレーション) を使用します。前のステップで作成した fleet-id を含め、compute-name とラップトップのパブリック ip-address を追加します。

```
aws gamelift register-compute \
    --compute-name DevLaptop \
    --fleet-id fleet-1234 \
    --ip-address 10.1.2.3 \
    --location custom-location-1
```

出力例:

```
Compute {
    FleetId = fleet-1234,
    ComputeName = DevLaptop,
    Status = ACTIVE,
    IpAddress = 10.1.2.3,
```

```
    GameLiftServiceSdkEndpoint = wss://12345678.execute-api.amazonaws.com/,
    Location = custom-location-1
}
```

4. ゲームサーバーのデバッグセッションを開始します。

   a. 作成したフリート内のラップトップの認証トークンを取得します。get-compute-auth-token コマンド (または GetComputeAuthToken API オペレーション) を使用します。

   ```
   aws gamelift get-compute-auth-token \
       --fleet-id fleet-1234 \
       --compute-name DevLaptop
   ```

   出力例:

   ```
   ComputeAuthToken {
     FleetId = fleet-1234,
     ComputeName = DevLaptop,
     AuthToken = abcdefg123,
     ExpirationTime = 1897492857.11
   }
   ```

   b. ゲームサーバーの実行ファイルのデバッグインスタンスを実行します。デバッグインスタンスを実行するには、ゲームサーバーが InitSDK() を呼び出す必要があります。プロセスがゲームセッションをホストする準備ができた後、ゲームサーバーは ProcessReady() を呼び出します。

5. ゲームセッションを作成して、Amazon GameLift Anywhere との初めての統合をテストします。create-game-session コマンド (または CreateGameSession API オペレーション) を使用します。

   ```
   aws gamelift create-game-session \
       --fleet-id fleet-1234 \
       --name DebugSession \
       --maximum-player-session-count 2
   ```

   出力例:

   ```
   GameSession {
       FleetId = fleet-1234,
   ```

```
    GameSessionId = 1111-1111,
    Name = DebugSession,
    IpAddress = 10.1.2.3,
    Port = 1024,
    ...
}
```

Amazon GameLift は、登録されたサーバープロセスに onStartGameSession() メッセージを送信します。メッセージには、ゲームプロパティ、ゲームセッションデータ、マッチメーカーデータなど、ゲームセッションに関する前のステップの GameSession オブジェクトが含まれます。

6. ゲームサーバーにロジックを追加して、サーバープロセスが onStartGameSession() メッセージに ActivateGameSession() で応答するようにします。オペレーションは、サーバーがゲームセッション作成メッセージを受信して受け入れたという確認を Amazon GameLift に送信します。詳細については、「Amazon GameLift サーバー SDK リファレンス」を参照してください。

これで、ゲームサーバーでゲームセッションが実行され、テストして繰り返し使用できます。ゲームサーバーでイテレーションを行う方法については、次のセクションに進みます。

ゲームサーバーでのイテレーション

このユースケースでは、ゲームサーバーをセットアップしてテストし、バグを発見したシナリオを考えてみましょう。Amazon GameLift Anywhere を使用すると、コードを繰り返し処理できるため、Amazon EC2 Fleet を使用するという面倒なセットアップを回避できます。

1. 可能であれば、既存の GameSession をクリーンアップします。ゲームサーバーがクラッシュしたり、ProcessEnding() を呼び出さない場合、Amazon GameLift はゲームサーバーがヘルスチェックの送信を停止した後で GameSession をクリーンアップします。
2. ゲームサーバーのコードを変更し、コンパイルして、次のテストに備えます。
3. 以前の Anywhere フリートはまだアクティブで、ラップトップはフリート内のコンピューティングリソースとして登録されたままです。テストを再開するには、新しいデバッグインスタンスを作成します。

    a. 作成したフリート内のラップトップの認証トークンを取得します。get-compute-auth-token コマンド (または GetComputeAuthToken API オペレーション) を使用します。

```
aws gamelift get-compute-auth-token \
    --fleet-id fleet-1234 \
    --compute-name DevLaptop
```

出力例:

```
ComputeAuthToken {
  FleetId = fleet-1234,
  ComputeName = DevLaptop,
  AuthToken = hijklmnop456,
  ExpirationTime = 1897492857.11
}
```

b. ゲームサーバーの実行ファイルのデバッグインスタンスを実行します。デバッグインスタンスを実行するには、ゲームサーバーが InitSDK() を呼び出す必要があります。プロセスがゲームセッションをホストする準備ができた後、ゲームサーバーは ProcessReady() を呼び出します。

4. これで、フリートに使用可能なサーバープロセスができました。ゲームセッションを作成し、次のテストを実行します。create-game-session コマンド (または CreateGameSession API オペレーション) を使用します。

```
aws gamelift create-game-session \
    --fleet-id fleet-1234 \
    --name SecondDebugSession \
    --maximum-player-session-count 2
```

Amazon GameLift は、登録されたサーバープロセスに onStartGameSession() メッセージを送信します。メッセージには、ゲームプロパティ、ゲームセッションデータ、マッチメーカーデータなど、ゲームセッションに関する前のステップの GameSession オブジェクトが含まれます。

5. ゲームサーバーにロジックを追加して、サーバープロセスが onStartGameSession() メッセージに ActivateGameSession() で応答するようにします。オペレーションは、サーバーがゲームセッション作成メッセージを受信して受け入れたという確認を Amazon GameLift に送信します。詳細については、「Amazon GameLift サーバー SDK リファレンス」を参照してください。

ゲームサーバーのテストが終了した後も、引き続き Amazon GameLift をフリートとゲームサーバーの管理に使用できます。詳細については、「Amazon GameLift Anywhere フリートを作成する」を参照してください。

Amazon GameLift Local を使用して統合をテストする

> **Note**
>
> バージョン 4.x 以前のバージョンの Amazon GameLift サーバー SDK を使用している場合は、このテスト手順を使用してください。サーバー SDK パッケージには、Amazon GameLift Local の互換バージョンが含まれています。サーバー SDK バージョン 5.x を使用している場合は、Amazon GameLift Anywhere フリートでのローカルテストについて Amazon GameLift Anywhere フリートを使用して統合をテストする を参照してください。

Amazon GameLift Local を使用して、ローカルデバイスでマネージド Amazon GameLift サービスの限定バージョンを実行し、それに対してゲーム統合をテストします。このツールは、ゲーム統合で反復開発を行う場合に便利です。代わりの方法 - 新しい各構築を Amazon GameLift にアップロードし、ゲームをホストするようフリートを設定する場合は、毎回 30 分以上の時間がかかる場合があります。

Amazon GameLift Local では、以下の点を確認できます。

- ゲームサーバーが Server SDK と正常に統合されており、新しいゲームセッションのスタート、新しいプレイヤーの承諾、ヘルスおよびステータスの報告を行うために、Amazon GameLift サービスと正しく通信している。
- ゲームクライアントが正常に AWS SDK for Amazon GameLift と統合されており、既存のゲームセッションでの情報の取得、新しいゲームセッションのスタート、プレイヤーのゲームへの参加、ゲームセッションへの接続を行うことができる。

Amazon GameLift Local は、マネージド Amazon GameLift サービスの自己完結型バージョンを開始するコマンドラインツールです。Amazon GameLift Local には、サーバープロセスの初期化、ヘルスチェック、API コールおよびレスポンスの実行イベントログも用意されています。Amazon GameLift Local は、Amazon GameLift 向けの AWS SDK アクションのサブセットを認識します。呼び出しは、AWS CLI またはゲームクライアントから行うことができます。すべての API アクションは、Amazon GameLift ウェブサービスでの実行と同じようにローカルで実行されます。

各サーバープロセスは 1 つのゲームセッションのみをホストする必要があります。ゲームセッションは、Amazon GameLift Local へのConnectに使用する実行可能ファイルです。ゲームセッションが完了したら、`GameLiftServerSDK::ProcessEndning` を呼び出して、プロセスを終了します。Amazon GameLift Local でローカルでテストする場合、複数のサーバープロセスをスタートできます。各プロセスは Amazon GameLift Local に接続します。その後、サーバープロセスごとに 1 つのゲームセッションを作成できます。ゲームセッションが終了すると、ゲームサーバープロセスは終了します。その後、別のサーバープロセスをマニュアルでスタートする必要があります。

Amazon GameLift ローカルは、以下の API をサポートしています。

- CreateGameSession
- CreatePlayerSession
- CreatePlayerSessions
- DescribeGameSessions
- DescribePlayerSessions

Amazon GameLift Local のセットアップ

Amazon GameLift Local は、[サーバー SDK] でバンドルされた実行可能 `.jar` ファイルとして提供されます。Windows または Linux で実行でき、Amazon GameLift でサポートされる任意の言語で使用できます。

Local を実行する前に、以下のものもインストールされている必要があります。

- Amazon GameLift Server SDK バージョン 3.1.5 から 4.x のビルド。
- Java 8

ゲームサーバーのテスト

ゲームサーバーのみをテストする場合、AWS CLI を使用して Amazon GameLift Local サービスへのゲームクライアント呼び出しをシミュレートできます。これにより、ゲームサーバーが正常に実行されていることと、以下の点が確認されます。

- ゲームサーバーが正しく起動し、Amazon GameLift サーバー SDK を初期化します。
- 起動プロセスのパートとして、サーバーでゲームセッションをホストする準備ができていることをゲームサーバーが Amazon GameLift に通知します。
- ゲームサーバーが、実行中 1 分ごとにヘルスステータスを Amazon GameLift に送信します。

- ゲームサーバーがリクエストに応答して新しいゲームセッションを開始する。

1. Amazon GameLift Local を開始します。

   コマンドプロンプトウィンドウを開き、*GameLiftLocal.jar* ファイルがあるディレクトリに移動して実行します。デフォルトでは、Local はポート 8080 でゲームクライアントからのリクエストをリッスンします。別のポート番号を指定するには、以下の例に示すように -p パラメータを使用します。

   ```
   java -jar GameLiftLocal.jar -p 9080
   ```

   Local が起動すると、2 つのローカルサーバー (ゲームサーバーをリッスンしているサーバーとゲームクライアントまたは AWS CLI をリッスンしているサーバー) が起動したことがログに示されます。ログは、ゲームコンポーネントとの通信など、2 つのローカルサーバーでのアクティビティを報告し続けます。

2. ゲームサーバーを起動します。

   Amazon GameLift が統合されたゲームサーバーをローカルでスタートします。ゲームサーバーのエンドポイントを変更する必要はありません。

   Local のコマンドプロンプトウィンドウでは、ゲームサーバーが Amazon GameLift Local サービスに接続されていることがログメッセージに示されます。これは、ゲームサーバーが Amazon GameLift サーバー SDK を正常に初期化したこと (`InitSDK()` を使用) を意味します。示されたログパスを使用して `ProcessReady()` を呼び出し、成功した場合はゲームセッションをホストする準備が完了します。ゲームサーバーの実行中、Amazon GameLift はゲームサーバーから各ヘルス ステータスレポートを記録します。次のログメッセージの例は、正常に統合されたゲームサーバーを示しています。

   ```
   16:50:53,217  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - SDK
    connected: /127.0.0.1:64247
   16:50:53,217  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - SDK pid is 17040,
    sdkVersion is 3.1.5 and sdkLanguage is CSharp
   16:50:53,217  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - NOTE: Only SDK
    versions 3.1.5 and above are supported in GameLiftLocal!
   16:50:53,451  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - onProcessReady
    received from: /127.0.0.1:64247 and ackRequest requested? true
   16:50:53,543  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - onProcessReady
    data: logPathsToUpload: "C:\\game\\logs"
   logPathsToUpload: "C:\\game\\error"
   ```

```
port: 1935

16:50:53,544  INFO || - [HostProcessManager] nioEventLoopGroup-3-1 - Registered new
 process true, true,
16:50:53,558  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - onReportHealth
 received from /127.0.0.1:64247 with health status: healthy
```

考えられるエラーおよび警告メッセージには次のものがあります。

- Error: "ProcessReady did not find a process with pID: *<process ID>*! Was InitSDK() invoked?"
- Warning: "Process state already exists for process with pID: *<process ID>*! Is ProcessReady(...) invoked more than once?"

3. AWS CLI の開始

   ゲームサーバーが ProcessReady( ) を正常に呼び出したら、クライアント呼び出しを開始できます。別のコマンドプロンプトウィンドウを開き、AWS CLI ツールを開始します。デフォルトで、AWS CLI では Amazon GameLift ウェブ サービスエンドポイントが使用されます。これは、次のリクエスト例に示すように、--endpoint-url パラメータを使用して各リクエストで Local エンドポイントによって上書きする必要があります。

   ```
   AWS gamelift describe-game-sessions --endpoint-url http://localhost:9080  --fleet-
   id fleet-123
   ```

   AWS CLIコマンドプロンプトウィンドウでは、AWS gameliftコマンドを使用すると、AWS CLI[Command Reference] (コマンドリファレンス) に記載されているように応答が返されます。

4. ゲームセッションの作成。

   AWS CLI を使用して、CreateGameSession() リクエストを送信します。リクエストは予期される構文に従う必要があります。Local では、FleetId パラメーターを任意の文字列 (^fleet-\S+) に設定できます。

   ```
   AWS gamelift create-game-session --endpoint-url http://localhost:9080 --maximum-
   player-session-count 2 --fleet-id
       fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d
   ```

Local のコマンドプロンプトウィンドウでは、Amazon GameLift Local がゲームサーバーに onStartGameSession コールバックを送信したことがログメッセージに示されます。ゲームセッションが正常に作成された場合、ゲームサーバーは ActivateGameSession を呼び出すことで応答します。

```
13:57:36,129  INFO || - [SDKInvokerImpl]
        Thread-2 - Finished sending event to game server to start a game session:
        arn:aws:gamelift:local::gamesession/
fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d/gsess-ab423a4b-b827-4765-
aea2-54b3fa0818b6.
        Waiting for ack response.13:57:36,143  INFO || - [SDKInvokerImpl]
        Thread-2 - Received ack response: true13:57:36,144  INFO || -
        [CreateGameSessionDispatcher] Thread-2 - GameSession with id:
        arn:aws:gamelift:local::gamesession/
fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d/gsess-ab423a4b-b827-4765-
aea2-54b3fa0818b6
        created13:57:36,227  INFO || - [SDKListenerImpl]
        nioEventLoopGroup-3-1 - onGameSessionActivate received
 from: /127.0.0.1:60020 and ackRequest
        requested? true13:57:36,230  INFO || - [SDKListenerImpl]
        nioEventLoopGroup-3-1 - onGameSessionActivate data: gameSessionId:
        "arn:aws:gamelift:local::gamesession/
fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d/gsess-abcdef12-3456-7890-abcd-
ef1234567890"
```

AWS CLI ウィンドウで、Amazon GameLift はゲームセッション ID を含むゲームセッションオブジェクトで応答します。新しいゲームセッションのステータスが [Activating] になっていることに注目してください。ゲームサーバーが ActivateGameSession を呼び出すとステータスは [Active] に変わります。変更されたステータスを確認するには、AWS CLI で DescribeGameSessions() を呼び出します。

```
{
    "GameSession": {
      "Status": "ACTIVATING",
      "MaximumPlayerSessionCount": 2,
      "FleetId": "fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d",
      "GameSessionId": "arn:aws:gamelift:local::gamesession/
fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d/gsess-abcdef12-3456-7890-abcd-
ef1234567890",
      "IpAddress": "127.0.0.1",
```

```
      "Port": 1935
    }
}
```

ゲームサーバーおよびクライアントのテスト

プレイヤーのゲームへの接続など、ゲームの完全統合を確認するには、ゲームサーバーとクライアントの両方をローカルで実行することができます。これにより、ゲーム クライアントから Amazon GameLift Local へのプログラムによる呼び出しをテストできます。次のアクションを確認できます。

- ゲームクライアントが、ゲームセッションの作成、既存のゲームセッションでの情報の取得、プレイヤーセッションの作成など、Amazon GameLift Local サービスへの AWS SDK リクエストを正常に行っています。
- プレイヤーがゲームセッションに参加しようとしたときにゲームサーバーがプレイヤーを正しく検証する。プレイヤーが検証されると、ゲームサーバーはプレイヤーデータ (実装されている場合) を取得できます。
- プレイヤーがゲームを終了すると、ゲームサーバーが接続中断を報告する。
- ゲームサーバーがゲームセッションの終了を報告する。

1. Amazon GameLift Local を開始します。

   コマンドプロンプトウィンドウを開き、*GameLiftLocal.jar* ファイルがあるディレクトリに移動して実行します。デフォルトでは、Local はポート 8080 でゲームクライアントからのリクエストをリッスンします。別のポート番号を指定するには、以下の例に示すように -p パラメーターを使用します。

   ```
   ./gamelift-local -p 9080
   ```

   Local が起動すると、2 つのローカルサーバー (ゲームサーバーをリッスンしているサーバーとゲームクライアントまたは AWS CLI をリッスンしているサーバー) が起動したことがログに示されます。

2. ゲームサーバーを起動します。

   Amazon GameLift が統合されたゲームサーバーをローカルでスタートします。メッセージログの詳細については、「ゲームサーバーのテスト」を参照してください。

3. Local のゲームクライアントを設定して起動します。

Amazon GameLift Local サービスでゲーム クライアントを使用するには、「バックエンドサービス GameLift で Amazon をセットアップする」で説明されているように、ゲームクライアントのセットアップに以下の変更を加える必要があります。

- `http://localhost:9080` などの Local エンドポイントをポイントするように、`ClientConfiguration` オブジェクトを変更します。
- ターゲットフリートの ID 値を設定します。Local では、実際のフリート ID は必要ありません。ターゲットフリートを、`fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d` などの任意の有効な文字列 (`^fleet-\S+`) に設定します。
- AWS 認証情報を設定します。Local では、実際の AWS 認証情報は必要ありません。アクセスキーとシークレットキーを任意の文字列に設定できます。

Local のコマンドプロンプトウィンドウで、ゲーム クライアントをスタートすると、`GameLiftClient` を初期化し、Amazon GameLift サービスと正常に通信していることがログメッセージに示されるはずです。

4. Amazon GameLift サービスへのゲーム クライアント呼び出しをテストします。

   ゲームクライアントが以下の API 呼び出しのすべてまたはいずれかを正常に行っていることを確認します。

   - CreateGameSession()
   - DescribeGameSessions()
   - CreatePlayerSession()
   - CreatePlayerSessions()
   - DescribePlayerSessions()

   Local のコマンドプロンプトウィンドウでは、`CreateGameSession()` を呼び出した場合のみログメッセージが生成されます。ログメッセージは、Amazon GameLift Local がゲームサーバーにゲームセッションのスタートを求め (`onStartGameSession` コールバック)、ゲームサーバーが呼び出したときに `ActivateGameSession` の取得が成功した場合に表示されます。AWS CLI ウィンドウでは、説明されているとおり、すべての API 呼び出しによりレスポンスまたはエラーメッセージが生成されます。

5. ゲームサーバーが新しいプレイヤー接続を検証していることを確認します。

ゲームセッションとプレイヤーセッションを作成したら、ゲームセッションへの直接接続を確立します。

Local のコマンドプロンプトウィンドウでは、ゲームサーバーが AcceptPlayerSession() リクエストを送信して新しいプレイヤー接続を検証したことが表示されます。AWS CLI を使用して DescribePlayerSessions() を呼び出した場合、プレイヤーセッションステータスが Reserved から Active に変わります。

6. ゲームサーバーがゲームおよびプレイヤーのステータスを Amazon GameLift サービスに報告していることを確認します。

   Amazon GameLift がプレイヤーの要求を管理してメトリクスを正常にレポートするには、ゲームサーバーは各種ステータスを Amazon GameLift にレポートする必要があります。Local が以下のアクションに関連するイベントを記録していることを確認します。AWS CLI を使用してステータス変更を追跡することもできます。

   - [プレイヤーがゲームセッションから切断します] – Amazon GameLift Local ログメッセージには、ゲームサーバーが RemovePlayerSession() を呼び出すことが示されています。DescribePlayerSessions() への AWS CLI 呼び出しには、Active から Completed へのステータス変更が反映されます。さらに、DescribeGameSessions() を呼び出して、ゲームセッションの現在のプレイヤー数が 1 人減少したことを確認することもできます。
   - [ゲームセッションが終了します] – Amazon GameLift Local ログメッセージには、ゲームサーバーが TerminateGameSession() を呼び出すことが示されます。

     > **Note**
     >
     > 以前のガイダンスでは、ゲームセッションを終了するときに TerminateGameSession() を呼び出していました。このメソッドは Amazon GameLift サーバー SDK v4.0.1 では非推奨です。「ゲームセッションを終了する」を参照してください。

   - [サーバープロセスが終了しています]– Amazon GameLift Local ログメッセージには、ゲームサーバーが ProcessEnding() を呼び出すことが示されます。DescribeGameSessions() への AWS CLI 呼び出しには、Active から Terminated (または Terminating) へのステータス変更が反映されます。

Local でのバリエーション

Amazon GameLift Local を使用する場合は、次の点に留意してください。

- Amazon GameLift ウェブサービスとは異なり、Local はサーバーのヘルスステータスを追跡せず、`onProcessTerminate` コールバックを開始しません。Local はゲームサーバーのヘルスレポートの記録を停止するだけです。
- AWS SDK への呼び出しの場合、フリート ID は検証されないため、パラメータ要件を満たす任意の文字列値 (`^fleet-\S+`) にすることができます。
- Local で作成されたゲームセッション ID の構造は異なります。ここに示すように、文字列 `local` が含まれています。

```
arn:aws:gamelift:local::gamesession/fleet-123/gsess-56961f8e-
db9c-4173-97e7-270b82f0daa6
```

Amazon GameLift リアルタイムサーバーとのゲームの統合

このトピックでは、マネージド Amazon GameLift および Realtime サーバーソリューションの概要に関して説明します。概要では、このソリューションがどのような場合にゲームに適しているか、リアルタイムサーバーがどのようにマルチプレイヤーゲームをサポートしているかについて説明します。

ゲームを稼働させるための完全なロードマップについては、「Amazon GameLift マネージドホスティングロードマップ」を参照してください。

> **Tip**
>
> Amazon GameLift ゲームサーバーホスティングを試すには、「Amazon GameLift の開始方法」を参照してください。

リアルタイムサーバーとは

リアルタイムサーバーは、マルチプレイヤーゲームで使用するために Amazon GameLift が提供する軽量ですぐに使用できるゲームサーバーです。リアルタイムサーバーを使うと、カスタムゲームサーバーの開発、テスト、デプロイのプロセスが不要になります。このソリューションにより、ゲームの完成までにかかる時間と労力を最小限に抑えることができます。

主な特徴

- ゲームのクライアントとサーバー間のやり取りのための完全なネットワークスタック
- ゲームサーバーの主な機能。
- カスタマイズ可能なサーバーロジック
- リアルタイム設定およびサーバーロジックへライブ更新
- FlexMatch マッチメーキング
- ホスティングリソースの柔軟な制御

フリートを作成し、設定スクリプトを提供することで、リアルタイムサーバーをセットアップします。リアルタイムサーバーの作成とゲームクライアントの準備方法の詳細については、「リアルタイムサーバーを準備する」を参照してください。

リアルタイムサーバーがゲームセッションを管理する方法

リアルタイムスクリプトに組み込んで、ゲームセッション管理用のカスタムロジックを追加できます。サーバー固有のオブジェクトへのアクセスや、コールバックを使用したイベント駆動型ロジックの追加、イベント以外のシナリオに基づくロジックの追加を行うコードを記述できます。

リアルタイムクライアントとリアルタイムサーバーのやり取りの方法

ゲームセッション中、ゲームクライアントはバックエンドサービスを通じてリアルタイムサーバーにメッセージを送信することでやり取りします。その後、バックエンドサービスはメッセージをゲームクライアント間で中継して、アクティビティ、ゲームの状態、関連するゲームデータを交換します。

さらに、ゲームロジックを Realtime スクリプトに追加することで、クライアントとサーバーのやり取り方法をカスタマイズできます。カスタムゲームロジックにより、リアルタイムでイベント駆動型レスポンスを開始するコールバックを実装できます。

通信プロトコル

リアルタイムサーバーと接続されたゲームクライアントは、2 つのチャネル (確実な配信のための TCP 接続と迅速な配信のための UDP チャネル) を通して通信します。メッセージを作成するときは、メッセージのタイプに応じて使用するプロトコルを選択します。メッセージ配信はデフォルトで UDP に設定されています。UDP チャネルが利用できない場合、Amazon GameLift はフォールバックとして TCP を使用してメッセージを送信します。

メッセージの内容

メッセージの内容は、必須のオペレーションコード (opCode) とオプションのペイロードの 2 つの要素で構成されています。ペイロードがオペレーションコードに関連する追加のデータを提供し、メッセージの opCode は特定のプレイヤーの活動やゲームイベントを識別します。これらの要素はどちらも開発者定義です。ゲームクライアントは、受信したメッセージ内の opCode に基づいてアクションを実行します。

プレイヤーグループ

Realtime サーバーはプレイヤーのグループを管理する機能を提供します。デフォルトでは、Amazon GameLift はゲームに接続しているすべてのプレイヤーを「すべてのプレイヤー」グループに配置します。さらに、開発者は自分のゲームに他のグループを設定でき、プレイヤーは同時に複数のグループのメンバーになることができます。グループメンバーは、メッセージを送信したり、グループのすべてのプレーヤーとゲームデータを共有できます。グループの用途の 1 つは、プレイヤーチームを設定し、チームのコミュニケーションを管理することです。

TLS 証明書を使用したリアルタイムサーバー

リアルタイムサーバーでは、サーバー認証とデータパケット暗号化がリアルタイムサーバー用に組み込まれています。TLS 証明書の生成を有効にすると、これらのセキュリティ機能が有効になります。ゲームクライアントが Realtime サーバーに接続しようとすると、サーバーは TLS 証明書で自動的に応答します。この証明書はクライアントが検証します。Amazon GameLift は暗号化を TCP (Websocket) 通信の TLS および UDP トラフィックの DTLS を使って処理します。

リアルタイムサーバーのカスタマイズ

リアルタイムサーバーはステートレスな中継サーバーとして機能します。リアルタイムサーバーは、ゲームに接続されているゲームクライアント間でメッセージのパケットとゲームデータを中継します。ただし、リアルタイムサーバーはメッセージの評価、データの処理、ゲームプレイロジックの実行は行いません。このように使用されると、各ゲームクライアントはゲーム状態の独自のビューを維持し、リレーサーバーを介して他のプレイヤーに更新情報を提供します。各ゲームクライアントは、これらの更新を組み込み、独自のゲーム状態を調整します。

また、リアルタイムスクリプト機能を追加することによって、サーバーをカスタマイズすることもできます。例えば、ゲームロジックでは、サーバー権限によるゲームの状態の表示により、ステートフルなゲームを構築できます。

Amazon GameLift では一連のサーバー側のコールバックが リアルタイムスクリプト用に定義されています。これらのコールバックを実装して、イベント駆動型の機能をサーバーに追加します。例えば、以下のことが可能です。

- ゲームクライアントがサーバーに接続を試みるときにプレイヤーを認証します。
- リクエスト時にプレイヤーがグループに参加できるかどうかを確認します。
- 特定のプレイヤーからのメッセージまたはターゲットプレイヤーへのメッセージの配信タイミングを決定するか、それに応じて追加の処理を実行します。
- プレイヤーがグループを離れるか、サーバーから切断されたときに、すべてのプレイヤーに通知します。
- ゲームセッションオブジェクトまたはメッセージオブジェクトの内容を表示してデータを使用します。

リアルタイムサーバーのデプロイと更新

リアルタイムサーバーの主な利点は、いつでもスクリプトを更新できることです。スクリプトを更新すると、Amazon GameLift は数分以内に新しいバージョンをすべてのホスティングリソースに配信します。Amazon GameLift が新しいスクリプトをデプロイすると、その時点以降に作成された新しいゲームセッションはすべて新しいスクリプトバージョンを使用します。(既存のゲームセッションは引き続き元のバージョンを使用します)。

ゲームとリアルタイムサーバーの統合の開始方法

- リアルタイムサーバー用のゲームクライアントの統合
- リアルタイムスクリプトの作成

リアルタイムサーバー用のゲームクライアントの統合

このトピックでは、Amazon GameLift がホストするゲームセッションに参加できるようにゲームクライアントを準備する方法について説明します。

ゲームクライアントを準備するために必要なタスクは 2 セットあります。

- 既存のゲームに関する情報の取得、マッチメーキングのリクエスト、新しいゲームセッションの開始、およびプレイヤー用のゲームセッションスロットの予約を行うようにゲームクライアントを設定します。
- ゲームクライアントがリアルタイムサーバーでホストされているゲームセッションに参加してメッセージを交換できるようにします。

ゲームセッションとプレイヤーセッションを検索または作成します。

ゲームセッションを作成または開始し、FlexMatch マッチメーキングをリクエストし、プレイヤーセッションを作成してゲーム内のプレイヤーのスペースを予約するようにゲームクライアントを設定します。ベストプラクティスは、バックエンドサービスを作成しておき、それを使用して、ゲームクライアントのアクションによって Amazon GameLift サービスがトリガーされたときに、そのサービスに直接リクエストを送信することです。その後、バックエンドサービスは該当するレスポンスをゲームクライアントに中継します。

1. AWS SDK をゲームクライアントに追加し、Amazon GameLift クライアントを初期化して、フリートおよびキューでホスティングリソースを使用するように設定します。AWS SDK はいくつかの言語で使用可能です。Amazon GameLift SDK を参照してくださいカスタムクライアントサービスの場合。
2. バックエンドサービスに GameLift 機能を追加します。詳細な手順については、「ゲームクライアント GameLift に Amazon を追加する」を参照してください。そしてFlexMatch マッチメーキングを追加してください。ベストプラクティスは、ゲームセッションの配置を使用して新しいゲームセッションを作成することです。この方法では、GameLift の新しいゲームセッションをすばやくインテリジェントに配置できるだけでなく、プレイヤーのレイテンシーデータを使用してゲームのラグを最小限に抑えることができます。最低限、バックエンドサービスは新しいゲームセッションをリクエストし、それに応じてゲームセッションデータを処理できる必要があります。また場合によっては、既存のゲームセッションを検索して情報を取得し、プレイヤーセッションをリクエストして、既存のゲームセッションで効果的にプレイヤースロットが予約されるようにする機能を追加する必要があります。
3. 接続情報をゲームクライアントに返します。バックエンドサービスに応じて、ゲームセッションオブジェクトとプレイヤーセッションオブジェクトを受け取る Amazon GameLift サービスにリクエストします。これらのオブジェクトには、Realtime Server で実行されているゲームセッションにゲームクライアントが接続するために必要な情報、特に接続の詳細 (IP アドレスおよびポート) とプレイヤーセッション ID が含まれています。

リアルタイムサーバーでゲームに接続する

ゲームクライアントがリアルタイムサーバー上のホスト型ゲームセッションに直接接続し、サーバーや他のプレイヤーとメッセージを交換できるようにします。

1. リアルタイムクライアント SDK を入手して構築し、それをゲームクライアントプロジェクトに追加します。SDK の要件およびクライアントライブラリの構築方法については、README ファイルを参照してください。

2. 使用するクライアント/サーバー接続のタイプをクライアント設定で指定して Client() を呼び出します。

> **Note**
>
> TLS 証明書の生成が有効になっているセキュリティ保護ありのフリートで実行されている Realtime サーバーに接続する場合は、セキュリティ保護ありの接続タイプを指定する必要があります。

3. ゲームクライアントに次の機能を追加します。詳細については、「リアルタイムサーバー API (C#) リファレンス」を参照してください。
   - ゲームに接続する/ゲームから切断する
     - Connect()
     - Disconnect()
   - ターゲットの受信者にメッセージを送信する
     - SendMessage()
   - メッセージを受信して処理する
     - OnDataReceived()
   - プレイヤーグループに参加する/プレイヤーグループから離脱する
     - JoinGroup()
     - RequestGroupMembership()
     - LeaveGroup()
4. 必要に応じてクライアントコールバック用のイベントハンドラを設定します。「リアルタイムサーバークライアント API (C#) リファレンス: 非同期コールバック」を参照してください。

TLS 証明書の生成が有効になっているリアルタイムフリートを使用する場合、サーバーは TLS 証明書を使用して自動的に認証されます。TCP および UDP トラフィックは、トランスポートレイヤーセキュリティを提供するために、転送時に暗号化されます。TCP トラフィックは TLS 1.2 を使用して暗号化され、UDP トラフィックは DTLS 1.2 を使用して暗号化されます。

ゲームクライアントの例

基本的なリアルタイムクライアント (C#)

この例は、基本的なゲームクライアントとリアルタイムクライアント SDK (C#) の統合を示しています。図に示すように、この例では、リアルタイムクライアントオブジェクトを初期化し、イベントハンドラを設定します。またクライアントサイドコールバックを実装し、リアルタイムサーバーに接続して、メッセージを送信して切断します。

```
using System;
using System.Text;
using Aws.GameLift.Realtime;
using Aws.GameLift.Realtime.Event;
using Aws.GameLift.Realtime.Types;

namespace Example
{
    /**
     * An example client that wraps the GameLift Realtime client SDK
     *
     * You can redirect logging from the SDK by setting up the LogHandler as such:
     * ClientLogger.LogHandler = (x) => Console.WriteLine(x);
     *
     */
    class RealTimeClient
    {
        public Aws.GameLift.Realtime.Client Client { get; private set; }

        // An opcode defined by client and your server script that represents a custom
 message type
        private const int MY_TEST_OP_CODE = 10;

        /// Initialize a client for GameLift Realtime and connect to a player session.
        /// <param name="endpoint">The DNS name that is assigned to Realtime server</
param>
        /// <param name="remoteTcpPort">A TCP port for the Realtime server</param>
        /// <param name="listeningUdpPort">A local port for listening to UDP traffic</
param>
        /// <param name="connectionType">Type of connection to establish between client
 and the Realtime server</param>
        /// <param name="playerSessionId">The player session ID that is assigned to the
 game client for a game session </param>
```

```
       /// <param name="connectionPayload">Developer-defined data to be used during
client connection, such as for player authentication</param>
      public RealTimeClient(string endpoint, int remoteTcpPort, int listeningUdpPort,
ConnectionType connectionType,
                    string playerSessionId, byte[] connectionPayload)
       {
           // Create a client configuration to specify a secure or unsecure connection
type
           // Best practice is to set up a secure connection using the connection type
RT_OVER_WSS_DTLS_TLS12.
           ClientConfiguration clientConfiguration = new ClientConfiguration()
     {
               // C# notation to set the field ConnectionType in the new instance of
ClientConfiguration
         ConnectionType = connectionType
     };

           // Create a Realtime client with the client configuration
           Client = new Client(clientConfiguration);

           // Initialize event handlers for the Realtime client
           Client.ConnectionOpen += OnOpenEvent;
           Client.ConnectionClose += OnCloseEvent;
           Client.GroupMembershipUpdated += OnGroupMembershipUpdate;
           Client.DataReceived += OnDataReceived;

           // Create a connection token to authenticate the client with the Realtime
server
           // Player session IDs can be retrieved using AWS SDK for GameLift
           ConnectionToken connectionToken = new ConnectionToken(playerSessionId,
connectionPayload);

           // Initiate a connection with the Realtime server with the given connection
information
           Client.Connect(endpoint, remoteTcpPort, listeningUdpPort, connectionToken);
       }

       public void Disconnect()
       {
           if (Client.Connected)
           {
               Client.Disconnect();
           }
       }
```

```
        public bool IsConnected()
        {
            return Client.Connected;
        }

        /// <summary>
        /// Example of sending to a custom message to the server.
        ///
        /// Server could be replaced by known peer Id etc.
        /// </summary>
        /// <param name="intent">Choice of delivery intent i.e. Reliable, Fast etc. </
param>
        /// <param name="payload">Custom payload to send with message</param>
        public void SendMessage(DeliveryIntent intent, string payload)
        {
            Client.SendMessage(Client.NewMessage(MY_TEST_OP_CODE)
                .WithDeliveryIntent(intent)
                .WithTargetPlayer(Constants.PLAYER_ID_SERVER)
                .WithPayload(StringToBytes(payload)));
        }

        /**
         * Handle connection open events
         */
        public void OnOpenEvent(object sender, EventArgs e)
        {
        }

        /**
         * Handle connection close events
         */
        public void OnCloseEvent(object sender, EventArgs e)
        {
        }

        /**
         * Handle Group membership update events
         */
        public void OnGroupMembershipUpdate(object sender, GroupMembershipEventArgs e)
        {
        }

        /**
```

```
         *  Handle data received from the Realtime server
         */
        public virtual void OnDataReceived(object sender, DataReceivedEventArgs e)
        {
            switch (e.OpCode)
            {
                // handle message based on OpCode
                default:
                    break;
            }
        }

        /**
         * Helper method to simplify task of sending/receiving payloads.
         */
        public static byte[] StringToBytes(string str)
        {
            return Encoding.UTF8.GetBytes(str);
        }

        /**
         * Helper method to simplify task of sending/receiving payloads.
         */
        public static string BytesToString(byte[] bytes)
        {
            return Encoding.UTF8.GetString(bytes);
        }
    }
}
```

リアルタイムスクリプトの作成

ゲームにリアルタイムサーバーを使用するには、リアルタイムサーバーのフリートを設定してオプションでカスタマイズするためのスクリプト (JavaScript コードの形式) を提供する必要があります。このトピックでは、リアルタイムスクリプトを作成するための主なステップについて説明します。スクリプトの準備ができたら、Amazon GameLift サービスにアップロードし、それを使用してフリートを作成します (「リアルタイムサーバースクリプトを Amazon GameLift にアップロードします」を参照)。

リアルタイムサーバーで使用するスクリプトを準備するには、リアルタイムスクリプトに以下の機能を追加します。

ゲームセッションのライフサイクルを管理する (必須)

最低限、リアルタイムスクリプトに Init() 関数を含めて、この関数で、リアルタイムサーバーがゲームセッションの開始準備をするようにします。新しいゲームセッションを引き続きフリートで開始できるように、ゲームセッションを終了する方法も提供することを強くお勧めします。

Init() コールバック関数が呼び出されると、リアルタイムセッションオブジェクトを渡されます。このセッションオブジェクトには、リアルタイムサーバーのインターフェイスが含まれています。このインターフェイスの詳細については、「リアルタイムサーバーインターフェイス」を参照してください。

ゲームセッションを正常に終了するには、スクリプトはリアルタイムサーバーの session.processEnding 関数も呼び出す必要があります。この関数には、セッションをいつ終了するかを決定するための何らかのメカニズムが必要です。スクリプト例のコードは、プレイヤーの接続を確認し、指定された時間内にセッションに接続しているプレイヤーがいなければゲームセッションの終了をトリガーする、シンプルなメカニズムを示しています。

基本的な設定のリアルタイム -- サーバープロセスの開始と終了 -- 基本的にステートレスな中継サーバーとして動作します。リアルタイムサーバーは、ゲームに接続されているゲームクライアント間でメッセージとゲームデータを中継しますが、データを処理したりロジックを実行したりするための独立したアクションは行いません。ゲームのニーズに応じて、ゲームイベントなどのメカニズムによってトリガーされるゲームロジックをオプションで追加することができます。

サーバー側のゲームロジックを追加する (オプション)

オプションでリアルタイムスクリプトにゲームロジックを追加できます。たとえば、以下のいずれかまたはすべてを実行できます。スクリプト例のコードは実例を提供しています。「Amazon GameLift リアルタイムサーバースクリプトリファレンス」を参照してください。

- イベント駆動型のロジックを追加する。クライアントサーバーイベントに応答するためのコールバック関数を実装します。コールバックの完全なリストについては、「リアルタイムサーバーのスクリプトコールバック」を参照してください。
- サーバーにメッセージを送信してロジックを起動する。ゲームクライアントからサーバーに送信されるメッセージ専用に一連のオペレーションコードを作成し、受信を処理するための関数を追加します。コールバック onMessage で gameMessage を使用してメッセージの内容を解析します (「gameMessage.opcode」を参照)。
- 他の AWS リソースにアクセスできるようにゲームロジックを有効にします。詳細については、「フリートの他の AWS リソースと通信する」を参照してください。

- ゲームロジックが実行されているインスタンスのフリート情報へのアクセスを許可します。詳細については、「Amazon GameLift インスタンスのフリートデータを取得する」を参照してください。

リアルタイムサーバースクリプト例

この例は、リアルタイムサーバーをデプロイするために必要な基本スクリプトといくつかのカスタムロジックを示しています。また、必要な Init() 関数を含み、プレイヤーの接続のない期間に基づいてゲームセッションの終了をトリガーするタイマーメカニズムを使用しています。さらに、コールバックの実装など、カスタムロジック用のフックもいくつか含まれています。

```
// Example Realtime Server Script
'use strict';

// Example override configuration
const configuration = {
    pingIntervalTime: 30000,
    maxPlayers: 32
};

// Timing mechanism used to trigger end of game session. Defines how long, in
 milliseconds, between each tick in the example tick loop
const tickTime = 1000;

// Defines how to long to wait in Seconds before beginning early termination check in
 the example tick loop
const minimumElapsedTime = 120;

var session;                        // The Realtime server session object
var logger;                         // Log at appropriate level
 via .info(), .warn(), .error(), .debug()
var startTime;                      // Records the time the process started
var activePlayers = 0;              // Records the number of connected players
var onProcessStartedCalled = false; // Record if onProcessStarted has been called

// Example custom op codes for user-defined messages
// Any positive op code number can be defined here. These should match your client
 code.
const OP_CODE_CUSTOM_OP1 = 111;
const OP_CODE_CUSTOM_OP1_REPLY = 112;
const OP_CODE_PLAYER_ACCEPTED = 113;
const OP_CODE_DISCONNECT_NOTIFICATION = 114;
```

```
// Example groups for user-defined groups
// Any positive group number can be defined here. These should match your client code.
// When referring to user-defined groups, "-1" represents all groups, "0" is reserved.
const RED_TEAM_GROUP = 1;
const BLUE_TEAM_GROUP = 2;

// Called when game server is initialized, passed server's object of current session
function init(rtSession) {
    session = rtSession;
    logger = session.getLogger();
}

// On Process Started is called when the process has begun and we need to perform any
// bootstrapping.  This is where the developer should insert any code to prepare
// the process to be able to host a game session, for example load some settings or set
 state
//
// Return true if the process has been appropriately prepared and it is okay to invoke
 the
// GameLift ProcessReady() call.
function onProcessStarted(args) {
    onProcessStartedCalled = true;
    logger.info("Starting process with args: " + args);
    logger.info("Ready to host games...");

    return true;
}

// Called when a new game session is started on the process
function onStartGameSession(gameSession) {
    // Complete any game session set-up

    // Set up an example tick loop to perform server initiated actions
    startTime = getTimeInS();
    tickLoop();
}

// Handle process termination if the process is being terminated by GameLift
// You do not need to call ProcessEnding here
function onProcessTerminate() {
    // Perform any clean up
}
```

```
// Return true if the process is healthy
function onHealthCheck() {
    return true;
}

// On Player Connect is called when a player has passed initial validation
// Return true if player should connect, false to reject
function onPlayerConnect(connectMsg) {
    // Perform any validation needed for connectMsg.payload, connectMsg.peerId
    return true;
}

// Called when a Player is accepted into the game
function onPlayerAccepted(player) {
    // This player was accepted -- let's send them a message
    const msg = session.newTextGameMessage(OP_CODE_PLAYER_ACCEPTED, player.peerId,
                                           "Peer " + player.peerId + " accepted");
    session.sendReliableMessage(msg, player.peerId);
    activePlayers++;
}

// On Player Disconnect is called when a player has left or been forcibly terminated
// Is only called for players that actually connected to the server and not those
 rejected by validation
// This is called before the player is removed from the player list
function onPlayerDisconnect(peerId) {
    // send a message to each remaining player letting them know about the disconnect
    const outMessage = session.newTextGameMessage(OP_CODE_DISCONNECT_NOTIFICATION,
                                                  session.getServerId(),
                                                  "Peer " + peerId + " disconnected");
    session.getPlayers().forEach((player, playerId) => {
        if (playerId != peerId) {
            session.sendReliableMessage(outMessage, playerId);
        }
    });
    activePlayers--;
}

// Handle a message to the server
function onMessage(gameMessage) {
    switch (gameMessage.opCode) {
      case OP_CODE_CUSTOM_OP1: {
        // do operation 1 with gameMessage.payload for example sendToGroup
```

```
        const outMessage = session.newTextGameMessage(OP_CODE_CUSTOM_OP1_REPLY,
 session.getServerId(), gameMessage.payload);
        session.sendGroupMessage(outMessage, RED_TEAM_GROUP);
        break;
      }
    }
}

// Return true if the send should be allowed
function onSendToPlayer(gameMessage) {
    // This example rejects any payloads containing "Reject"
    return (!gameMessage.getPayloadAsText().includes("Reject"));
}

// Return true if the send to group should be allowed
// Use gameMessage.getPayloadAsText() to get the message contents
function onSendToGroup(gameMessage) {
    return true;
}

// Return true if the player is allowed to join the group
function onPlayerJoinGroup(groupId, peerId) {
    return true;
}

// Return true if the player is allowed to leave the group
function onPlayerLeaveGroup(groupId, peerId) {
    return true;
}

// A simple tick loop example
// Checks to see if a minimum amount of time has passed before seeing if the game has
 ended
async function tickLoop() {
    const elapsedTime = getTimeInS() - startTime;
    logger.info("Tick... " + elapsedTime + " activePlayers: " + activePlayers);

    // In Tick loop - see if all players have left early after a minimum period of time
 has passed
    // Call processEnding() to terminate the process and quit
    if ( (activePlayers == 0) && (elapsedTime > minimumElapsedTime)) {
        logger.info("All players disconnected. Ending game");
        const outcome = await session.processEnding();
        logger.info("Completed process ending with: " + outcome);
```

```
        process.exit(0);
    }
    else {
        setTimeout(tickLoop, tickTime);
    }
}

// Calculates the current time in seconds
function getTimeInS() {
    return Math.round(new Date().getTime()/1000);
}

exports.ssExports = {
    configuration: configuration,
    init: init,
    onProcessStarted: onProcessStarted,
    onMessage: onMessage,
    onPlayerConnect: onPlayerConnect,
    onPlayerAccepted: onPlayerAccepted,
    onPlayerDisconnect: onPlayerDisconnect,
    onSendToPlayer: onSendToPlayer,
    onSendToGroup: onSendToGroup,
    onPlayerJoinGroup: onPlayerJoinGroup,
    onPlayerLeaveGroup: onPlayerLeaveGroup,
    onStartGameSession: onStartGameSession,
    onProcessTerminate: onProcessTerminate,
    onHealthCheck: onHealthCheck
};
```

Unity 用 Amazon GameLift プラグインとのゲームの統合

このセクションのトピックでは、Unity 用 Amazon GameLift プラグインと、それを使用して Amazon でホストするためのマルチプレイヤーゲームプロジェクトを準備する方法について説明します GameLift。プラグインのガイド付きワークフローを使用して Unity 開発環境で完全に作業し、Amazon でホストするための基本要件を完了します GameLift。

Amazon GameLift は、ゲームデベロッパーがセッションベースのマルチプレイヤーゲーム専用のゲームサーバーを管理およびスケーリングできるようにするフルマネージドサービスです。Amazon GameLift ホスティングの詳細については、「」を参照してくださいAmazon GameLift の仕組み。

- サーバー SDK 5.x 用 Unity ガイド用 Amazon GameLift プラグインバージョン 2.0.0 の はサーバー SDK 5.x と連携し、Amazon をサポートします GameLift Anywhere。

- サーバー SDK 4.x 用 Unity ガイド用 Amazon GameLift プラグインバージョン 1.0.0 の は、サーバー SDK 4.x 以前で動作します。このバージョンでは、統合テストに Amazon GameLift Local を使用します。

サーバー SDK 5.x 用 Unity ガイド用 Amazon GameLift プラグイン

Amazon GameLift には、Amazon と連携するようにマルチプレイヤーゲームサーバーを準備するためのツールが用意されています GameLift。Unity 用 Amazon GameLift プラグインを使用すると、Amazon を Unity ゲームプロジェクト GameLift に統合し、Amazon との統合をテストし GameLift Anywhere、クラウドホスティング用に Amazon GameLift リソースをデプロイすることが容易になります。

このプラグインは、 AWS CloudFormation テンプレートを使用して、一般的なゲームシナリオのホスティングソリューションをデプロイします。これらのソリューションは、提供されているとおりに使用するか、ゲームの必要に応じてカスタマイズします。

トピック

- プラグインについて
- プラグインワークフロー
- Unity 用プラグインをインストールする
- AWS ユーザープロファイルを設定します。
- Amazon によるローカルテスト用にゲームをセットアップする GameLift Anywhere
- マネージド EC2 フリートでゲームをクラウドホスティングにデプロイします。

プラグインについて

Unity 用プラグインは、Unity マルチプレイヤーゲームを Amazon と統合してホストするための効率的な開始方法を提供します GameLift。プラグイン機能と構築済みコンポーネントを活用して、ゲームをすばやく起動して実行できます。

プラグインは Unity エディタにツールと機能を追加します。ガイド付きワークフローを使用して Amazon GameLift をゲームプロジェクトに統合し、ローカルでテストしてから、ゲームサーバーを Amazon GameLift クラウドホスティングにデプロイします。

プラグインの事前構築済みのホスティングソリューションを使用してゲームをデプロイします。ローカルワークステーションをホストとして Amazon GameLift Anywhere フリートを設定します。クラ

ウドホスティングでは、プレイヤーのレイテンシー、ゲームセッションの可用性、コストをさまざまな方法でバランスよく配分する 2 つの一般的なデプロイシナリオから選択します。1 つのシナリオには、シンプルな FlexMatchマッチメーカーとルールセットが含まれます。これらのシナリオを使用して、本番環境に対応した基本的なホスティングソリューションを導入し、必要に応じて最適化およびカスタマイズします。

このプラグインには以下のコンポーネントが含まれています。

- Unity エディタのプラグインモジュール。プラグインをインストールすると、新しいメインメニュー項目で Amazon GameLift の機能にアクセスできます。
- クライアント側の機能を備えた Amazon GameLift サービス API 用の C# ライブラリ。
- Amazon GameLift サーバー SDK (バージョン 5.x) 用の C# ライブラリ。
- アセットやシーンを含むサンプルゲームコンテンツ。構築可能なマルチプレイヤーゲームがない場合 GameLift でも、Amazon を試すことができます。
- プラグインがゲームサーバーをホスティングのためにクラウドにデプロイするときに使用する AWS CloudFormationテンプレートとして提供されるソリューション設定。

プラグインワークフロー

次のステップでは、Unity 用 Amazon GameLift プラグインとゲームプロジェクトを統合してデプロイするための一般的なアプローチについて説明します。これらのステップを完了するには、Unity エディタとゲームコードを使用します。

1. AWS アカウントにリンクし、Amazon を使用するアクセス許可を持つ有効なアカウントユーザーのアクセス認証情報を提供するユーザープロファイルを作成します GameLift。
2. ゲームプロジェクトにサーバーコードを追加して、実行中のゲームサーバーと Amazon GameLift サービスとの通信を確立します。
3. ゲームクライアントがゲームセッションを開始または参加し、ゲームサーバーに接続 GameLift するためのリクエストを Amazon に送信できるようにするクライアントコードをゲームプロジェクトに追加します。
4. Anywhere ワークフローを使用して、ローカルワークステーションをゲームサーバーの Anywhere ホストとして設定します。ゲームサーバーとクライアントをローカルで起動し、ゲームセッションに接続して、統合をテストします。
5. EC2 ホスティングワークフローを使用して、統合されたゲームサーバーをアップロードし、クラウドホスティングソリューションをデプロイします。ゲームサーバーの準備ができたら、ゲー

ムクライアントをローカルで起動し、ゲームセッションに接続してログインし、ゲームをプレイします。

プラグインで作業するときは、AWS リソースを作成して使用します。これらのアクションにより、使用中の AWS アカウントに料金が発生する可能性があります。を初めて使用する場合AWS、アクションは AWS 無料利用枠の対象になる場合があります。

Unity 用プラグインをインストールする

このセクションでは、Unity プロジェクトにプラグインを追加する方法について説明します。プラグインをインストールすると、Unity エディタでプロジェクトを開いたときにプラグイン機能を使用できます。

開始する前に

Unity 用 Amazon GameLift プラグインを使用するために必要なものは次のとおりです。

- Unity for Windows 2022 LTS または Unity for MacOS
- Unity ダウンロード用の Amazon GameLift プラグイン。〔ダウンロードサイト〕 ダウンロードには 2 つのパッケージが含まれています。
  - Unity 用 Amazon GameLift スタンドアロンプラグイン
  - Unity 用 Amazon GameLift C# サーバー SDK
- Microsoft Visual Studio 2019 以降。
- C# ゲームコードを持つマルチプレイヤーゲームプロジェクト。
- サードパーティーのスコープレジストリ UnityNuGet。このツールはサードパーティ DLL を管理します。詳細については、 UnityNuGetGitHub リポジトリを参照してください。

プラグインをゲームプロジェクトに追加する

Unity エディタとゲームプロジェクトファイルで作業して、次のタスクを完了します。

ステップ 1: ゲームプロジェクト UnityNuGet に追加する

ゲームプロジェクト用に UnityNuGet を設定していない場合は、次の手順で Unity パッケージマネージャーを使用してツールをインストールします。または、 NuGet CLI を使用して DLLsを手動でダウンロードできます。詳細については、「Amazon GameLift C# server SDK for Unity」を参照してくださいREADME。

1. Unity エディタでプロジェクトを開いた状態で、メインメニューに移動し、編集、プロジェクト設定 を選択します。オプションから [パッケージマネージャー] セクションを選択し、[スコープ設定レジストリ] グループを開きます。
2. + ボタンを選択し、 UnityNuGet スコープ付きレジストリに次の値を入力します。

```
Name: Unity NuGet
URL: https://unitynuget-registry.azurewebsites.net
Scope(s): org.nuget
```

Unity 2021 バージョンのユーザー:

を設定したら UnityNuGet、Unity コンソールに`Assembly Version Validation`エラーが表示されていないか確認します。これらのエラーは、 NuGet パッケージ内の強く名前が付けられたアセンブリのバインドリダイレクトが Unity プロジェクト内のパスに正しく解決されない場合に発生します。この問題を解決するには、Unity のアセンブリバージョンの検証を設定します。

1. Unity エディタで、メインメニューに移動し、編集、プロジェクト設定 を選択し、プレイヤー セクションを開きます。
2. [アセンブリバージョン検証] オプションの選択を解除します。

ステップ 2: プラグインと C# サーバー SDK パッケージを追加する

1. 両方のパッケージを含む Unity ダウンロード用の Amazon GameLift プラグインを解凍します。
2. Unity Editor でプロジェクトを開いた状態で、メインメニューに移動し、Window、Package Manager を選択します。
3. [+] ボタンを選択して新しいパッケージを追加します。[tarball からパッケージを追加] オプションを選択します。
4. 「ディスク上のパッケージの選択」で、Unity ダウンロードファイル用の Amazon GameLift C# Server SDK プラグインを見つけて、 `com.amazonaws.gameliftserver.sdk-<version>.tgz` ファイルを選択します。[開く] を選択してプラグインをインストールします。
5. 「ディスク上のパッケージの選択」で、Unity ダウンロードファイル用の Amazon GameLift スタンドアロンプラグインを探し、ファイル を選択します`com.amazonaws.gamelift-<version>.tgz`。[開く] を選択してプラグインをインストールします。
6. スタンドアロンプラグインがプロジェクトに追加されていることを確認します。Unity エディタウィンドウに戻ります。新しい Amazon GameLift メニューボタンのメインメニューを確認します。

ステップ 3: サンプルゲームをインポートする (オプション)

Unity 用プラグインには、ゲームプロジェクトに追加できるシーンを含む一連のサンプルゲームアセットが付属しています。サンプルゲームをインポートすると、Amazon でシンプルなマルチプレイヤーゲームをテスト、構築、デプロイするための迅速な道のりが得られます GameLift。サンプルゲームはすでに Amazon GameLift SDKsと完全に統合されているため、統合タスクをスキップして残りのワークフロータスクを完了できます。

サンプルゲームを使用すると、ローカルにホストされた Amazon GameLift Anywhere フリートをわずか数分でセットアップして参加させることができます。ゲームを Amazon GameLift にデプロイし、クラウドホストのライブゲームに 1 時間以内に参加できます。

サンプルゲームをインポートするには：

1. Unity Editor でゲームプロジェクトを開いた状態で、Amazon GameLift メニューに移動し、サンプルゲーム、サンプルゲームのインポートを選択します。
2. ファイルがインポートされたら、Amazon GameLift メニューに再度移動し、サンプルゲーム、初期化設定 を選択します。このステップでは、ゲームクライアントとサーバーを構築するためのプロジェクトを設定します。

インストールが完了すると、ゲームプロジェクトに 2 つの新しいシーンが追加されます。また、アセットを含むいくつかの追加のプロジェクトGameLiftClientSettingsアセットも表示されます。

サンプルの UI とゲームプレイの詳細については、サンプルゲームの readme を参照してください。

AWS ユーザープロファイルを設定します。

プラグインをインストールしたら、プロファイルを設定して有効な AWS アカウントユーザーにリンクします。複数のプロファイルを保持できますが、一度にアクティブにできるプロファイルは 1 つに限られます。プラグインで作業するときはいつでも、使用するプロファイルを選択してください。

複数のプロファイルを管理することで、異なるホスティングシナリオ間で切り替えることができます。例えば、同じ AWS 認証情報で異なる AWS リージョンのプロファイルを設定することができます。または、異なる AWS アカウントや異なるユーザー/アクセス許可のセットでプロファイルを設定することもできます。

> **Note**
>
> ワークステーションに CLI AWS をインストールし、プロファイルがすでに設定されている場合、Amazon GameLift プラグインはそれを検出して既存のプロファイルとして一覧表示で

きます。プラグインは [default] という名前のプロファイルを自動的に選択します。既存のプロファイルを使用するか、新しいプロファイルを作成できます。

AWS プロファイルを設定するには

1. Unity エディタのメインメニューで Amazon GameLift を選択し、AWSアカウントプロファイルの設定を選択します。このアクションにより、プラグインウィンドウが開きます。AWS ユーザープロファイルのページを開きます。
2. プラグインが既存のプロファイルを検出した場合、プロファイルを作成するように求められません。既存のプロファイルを選択するか、別のプロファイルを追加を選択して新しいプロファイルを作成します。
3. プラグインによって既存のプロファイルが検出されない場合、作成するよう求められます。新しいプロファイルは、新しいアカウントでも既存の AWS アカウントでも作成できます。

> **Note**
>
> AWS マネジメントコンソールを使用して新しい AWS アカウントを作成し、適切なアクセス許可セットを持つユーザーを作成または更新する必要があります。

プロファイルをセットアップするには、次の情報が必要です。

- AWS アカウント。新規 AWS アカウントを作成する必要がある場合、プロンプトに従ってアカウントを作成します。詳細については、「AWS アカウントを作成する」を参照してください。
- Amazon GameLift およびその他の必要な AWSのサービスを使用するアクセス許可を持つ AWS アカウントユーザー。Amazon アクセス GameLift 許可を持つ AWS Identity and Access Management (IAM) ユーザーを設定する手順のセットアップ AWS アカウントについては、「」を参照してください。
- AWS ユーザーの認証情報。このユーザーには、長期的な認証情報を使用したプログラムによるアクセスも必要です。これらの認証情報は、AWS アクセスキー IDおよび AWS シークレットアクセスキーで構成されています。詳細については、「アクセスキーを取得する」を参照してください。
- AWS リージョン これは、ホスティング用の AWS リソースを作成する地理的なロケーションです。開発中は、レイテンシーを最小限に抑えるために、物理的な場所に近いリージョンを使

用することをお勧めします。サポートされている AWS リージョンのリストを参照してください。

4. プロファイルを選択または作成したら、プロファイルのブートストラップステータスを確認し、必要に応じてアクションを実行します。Amazon GameLift プラグイン機能を使用するには、すべてのプロファイルをブートストラップする必要があります。

プロファイルをブートストラップするには

ブートストラップは、選択したユーザープロファイルで使用する Amazon S3 バケットを指定します。Amazon S3 は、データとオブジェクトストレージのコアAWSサービスです。プロジェクト設定、ビルドアーティファクト、およびその他の依存関係を保存するために使用されるバケット。バケットは他のプロファイルとは共有されません。

> **Note**
>
> ブートストラップによって新しいAWSリソースが作成され、コストが発生する可能性があります。

1. プラグインウィンドウのAWSユーザープロファイルでプロファイルを表示するときは、使用するプロファイルを選択します。プロファイルがまだブートストラップされていない場合は、警告メッセージが表示されます。
2. [プロファイルをブートストラップする] セクションで、ドロップダウンリストからプロファイルを選択し、ブートストラップのステータスを確認します。ステータスがバケットが存在しないことを示している場合は、「ブートストラッププロファイル」ボタンを選択します。バケット名を新しいバケット名に設定したり、アクセスできる既存のバケットを入力したり、自動生成された名前を維持したりできます。
3. ブートストラップのステータスが「アクティブ」に変わるまでお待ちください。これは数分かかることがあります。ステータスが「アクティブ」の場合、プロファイルを使用してプラグイン機能を操作できます。

Amazon によるローカルテスト用にゲームをセットアップする GameLift Anywhere

このワークフローでは、Amazon GameLift 機能のクライアントとサーバーのゲームコードを追加し、 プラグインを使用してローカルワークステーションをテストゲームサーバーホストとして指定します。統合タスクが完了したら、プラグインを使用してゲームクライアントとサーバーのコンポーネントを構築します。

Amazon GameLift Anywhere ワークフローを開始するには：

- Unity エディタのメインメニューで Amazon GameLift を選択し、Anywhere でホストを選択します。このアクションにより、@Anywhere フリートでゲームを設定するためのプラグインページが開きます。このページには、ゲームコンポーネントを統合、構築、起動するための 5 ステップのプロセスが表示されます。

プロファイルを設定する

このワークフローに従うときに使用したいプロファイルを選択します。選択したプロファイルは、ワークフローのすべてのステップに影響します。作成したすべてのリソースはプロファイルの AWS アカウントに関連付けられ、プロファイルのデフォルトの AWS リージョンに配置されます。プロファイルユーザーのアクセス許可によって、AWS リソースとアクションへのアクセスが決まります。

1. 使用可能なプロファイルのドロップダウンリストからプロファイルを選択します。まだプロファイルを持っていない場合、または新しいプロファイルを作成する場合は、Amazon GameLift メニューに移動し、AWSアカウントプロファイルの設定を選択します。
2. ブートストラップステータスが「アクティブ」でない場合は、[ブートストラップのプロファイル] を選択し、ステータスが「アクティブ」に変わるまで待ちます。

ゲームを Amazon と統合する GameLift

> **Note**
>
> サンプルゲームをインポートした場合は、このステップをスキップできます。サンプルゲームアセットには、必要なサーバーとクライアントコードが既に用意されています。

ワークフローのこのステップでは、ゲームプロジェクトのクライアントコードとサーバーコードを更新します。

- * ゲームサーバーは、ゲームセッションを開始するためのプロンプトを受信し、ゲームセッション接続情報を提供し、ステータスをレポートするために、Amazon GameLift サービスと通信できる必要があります。
- ゲームクライアントは、ゲームセッションに関する情報の取得、ゲームセッションへの参加または開始、ゲームに参加するための接続情報の取得を行える必要があります。

サーバーコードの統合

カスタムシーンで独自のゲームプロジェクトを使用している場合は、提供されたサンプルコードを使用して、ゲームプロジェクトに必要なサーバーコードを追加します。

1. ゲームプロジェクトファイルで、 `Assets/Scripts/Server`フォルダを開きます。存在しない場合は、作成します。
2. GitHub リポジトリ aws/amazon-gamelift-plugin-unity に移動し、パス を開きます`Samples~/SampleGame/Assets/Scripts/Server`。
3. ファイル GameLiftServer.cs. を見つけて、ゲームプロジェクトのサーバーフォルダにコピーします。サーバー実行可能ファイルを構築するときは、このファイルをビルドターゲットとして使用します。

サンプルコードには、Amazon GameLift C# サーバー SDK (バージョン 5) を使用するこれらの最低限必要な要素が含まれています。

- Amazon GameLift API クライアントを初期化します。Amazon GameLift Anywhere フリートには、サーバーパラメータを使用した`InitSDK()`呼び出しが必要です。これらの設定は、プラグインで使用するように自動的に設定されます。
- `OnStartGameSession`、、 `OnProcessTerminate`など、Amazon GameLift サービスからのリクエストに応答するために必要なコールバック関数を実装します`onHealthCheck`。
- 指定されたポート`ProcessReady()`で を呼び出し、サーバープロセスがゲームセッションをホストする準備ができたときに Amazon GameLift サービスに通知します。

サンプルサーバーコードをカスタマイズする場合は、次のリソースを参照してください。

- Amazon GameLift をゲームサーバーに追加する
- C# と Unity 用 Amazon GameLift サーバー SDK 5.x リファレンス

クライアントコードの統合

カスタムシーンで独自のゲームプロジェクトを使用している場合は、基本的な機能をゲームクライアントに統合する必要があります。また、プレイヤーがサインインしてゲームセッションに参加できるように、UI 要素を追加する必要があります。Amazon GameLift サービス APIs (AWS SDK 内) を使用して、ゲームセッション情報の取得、新しいゲームセッションの作成、既存のゲームセッションへの参加を行います。

Anywhere フリートを使用してローカルテスト用のクライアントを構築する場合、Amazon GameLift サービスに直接呼び出しを追加できます。クラウドホスティング用にゲームを開発する場合、またはプロダクションホスティングに Anywhere フリートを使用する予定の場合は、クライアント側のバックエンドサービスを作成して、ゲームクライアントと Amazon GameLift サービス間のすべての通信を処理する必要があります。

Amazon GameLift をクライアントコードに統合するには、以下のリソースをガイドとして使用します。

- クライアントをリポジトリ aws/ の GameLiftCoreApi GitHub クラスと統合しますamazon-gamelift-plugin-unity。このクラスは、プレイヤー認証とゲームセッション情報の取得のコントロールを提供します。
- GitHub リポジトリ aws/、 amazon-gamelift-plugin-unityで利用可能なサンプルゲーム統合を表示しますSamples~/SampleGame/Assets/Scripts/Client/GameLiftClient.cs。
- 「Unity ゲームクライアント GameLift に Amazon を追加する」の手順に従います。

Anywhere フリートに接続するゲームクライアントの場合、ゲームクライアントには次の情報が必要です。プラグインは、プラグインで作成したリソースを使用するようにゲームプロジェクトを自動的に更新します。

- FleetId - Anywhere フリートの一意の識別子。
- FleetLocation - Anywhere フリートのカスタムロケーション。
- AwsRegion - Anywhere フリートがホストされているAWSリージョン。これは、ユーザープロファイルで設定したリージョンです。
- ProfileName - AWS SDK for へのアクセスを許可するローカルマシン上の AWS認証情報プロファイル GameLift。ゲームクライアントは、これらの認証情報を使用して Amazon GameLift サービスへのリクエストを認証します。

> **Note**
>
> 認証情報プロファイルはプラグインによって生成され、ローカルマシンに保存されます。そのため、ローカルマシン (または同じプロファイルのマシン) でクライアントを実行する必要があります。

Anywhere フリートに接続する

このステップでは、使用する Anywhere フリートを指定します。Anywhere フリートは、ゲームサーバーをホスティングするための、どこにでも配置できる一連のコンピューティングリソースを定義します。

- 現在使用しているAWSアカウントに既存の Anywhere フリートがある場合は、フリート名ドロップダウンフィールドを開き、フリートを選択します。このドロップダウンには、現在アクティブなユーザープロファイルの AWS リージョン内の Anywhere フリートのみが表示されます。
- 既存のフリートがない場合、または新しいフリートを作成する場合は、新しい Anywhere フリートを作成してフリート名を指定します。

プロジェクトに Anywhere フリートを選択すると、Amazon はフリートのステータスがアクティブ広告にフリート ID が表示され GameLift ていることを確認します。このリクエストの進行状況は Unity エディタの出力ログで追跡できます。

コンピューティングを登録する

このステップでは、ローカルワークステーションを新しい Anywhere フリートのコンピューティングリソースとして登録します。

1. ローカルマシンのコンピューティング名を入力します。フリートに複数のコンピューティングを追加する場合、名前は一意でなければなりません。
2. コンピューティングの登録を選択します。このリクエストの進行状況は Unreal エディタの出力ログで追跡できます。

プラグインは、ローカルワークステーションの IP アドレスを localhost (127.0.0.1) に設定します。この設定では、ゲームクライアントとサーバーを同じマシンで実行することを前提としています。

このアクションに応じて、Amazon はコンピューティングに接続できる GameLift ことを確認し、新しく登録されたコンピューティングに関する情報を返します。

ゲームを起動する

このステップでは、ゲームコンポーネントを構築し、それらを起動してゲームをプレイします。以下のタスクを実行します。

1. ゲームクライアントを設定します。このステップでは、ゲームプロジェクトの`GameLiftClientSettings`アセットを更新するようにプラグインに促します。プラグイン

はこのアセットを使用して、ゲームクライアントが Amazon GameLift サービスに接続するために必要な特定の情報を保存します。

   a. サンプルゲームをインポートして初期化しなかった場合は、新しいGameLiftClientSettingsアセットを作成します。Unity エディタのメインメニューで、アセット、作成 GameLift、クライアント設定を選択します。プロジェクト GameLiftClientSettings で のコピーを複数作成すると、プラグインは自動的にこれを検出し、プラグインが更新するアセットを通知します。

   b. ゲームの起動 で、クライアントの設定: Anywhere 設定の適用 を選択します。このアクションは、先ほど設定した Anywhere フリートを使用するようにゲームクライアント設定を更新します。

2. ゲームクライアントを構築して実行します。

   a. 標準の Unity ビルドプロセスを使用して、クライアント実行可能ファイルを構築します。ファイル、ビルド設定 で、プラットフォームを Windows、Mac、Linux に切り替えます。サンプルゲームをインポートして設定を初期化すると、ビルドリストとビルドターゲットが自動的に更新されます。

   b. 新しく構築されたゲームクライアント実行可能ファイルの 1 つ以上のインスタンスを起動します。

3. Anywhere フリートでゲームサーバーを起動します。「サーバー: エディタでサーバーを起動する」を選択します。このタスクは、Unity エディタが開いている限り、クライアントが接続できるライブサーバーを起動します。

4. ゲームセッションを開始または参加します。ゲームクライアントインスタンスで、UI を使用して各クライアントをゲームセッションに参加させます。これを行う方法は、クライアントに機能を追加した方法によって異なります。

サンプルゲームクライアントを使用している場合、次の特性があります。

- プレイヤーログインコンポーネント。Anywhere フリートのゲームサーバーに接続する場合、プレイヤーの検証はありません。任意の値を入力してゲームセッションに参加できます。
- シンプルなゲーム結合 UI。クライアントがゲームに参加しようとすると、クライアントは使用可能なプレイヤースロットを持つアクティブなゲームセッションを自動的に検索します。利用可能なゲームセッションがない場合、クライアントは新しいゲームセッションをリクエストします。ゲームセッションが利用可能な場合、クライアントは利用可能なゲームセッションへの参加をリクエストします。複数の同時クライアントでゲームをテストする場合、最初のクライアントはゲームセッションを開始し、残りのクライアントは自動的に既存のゲームセッションに参加します。

- 4 つのプレイヤースロットを持つゲームセッション。最大 4 つのゲームクライアントインスタンスを同時に起動でき、同じゲームセッションに参加します。

サーバー実行可能ファイルからの起動 (オプション)

Anywhere フリートでテストするためのゲームサーバー実行可能ファイルを構築して起動できます。

1. 標準の Unity ビルドプロセスを使用してサーバー実行可能ファイルを構築します。ファイル、構築設定 で、プラットフォームを専用サーバーに切り替えて構築します。
2. Anywhere フリート ID とAWSリージョンget-compute-auth-tokenで AWS CLI コマンドを呼び出して、短期認証トークンを取得します。フリート ID は、フリートの作成時に Anywhere フリートに接続するに表示されます。アクティブなプロファイルを選択すると、AWSリージョンがプロファイルの設定に表示されます。

   ```
   aws gamelift get-compute-auth-token --fleet-id [your anywhere fleet ID] --region
    [your AWS region]
   ```

3. コマンドラインから新しく構築されたゲームサーバー実行可能ファイルを起動し、有効な認証トークンを渡します。

   ```
   my_project.exe --authToken [token]
   ```

マネージド EC2 フリートでゲームをクラウドホスティングにデプロイします。

このワークフローでは、 プラグインを使用して、Amazon が管理するクラウドベースのコンピューティングリソースでホストするゲームを準備します GameLift。Amazon GameLift 機能のクライアントとサーバーのゲームコードを追加し、ホスティングのためにサーバービルドを Amazon GameLift サービスにアップロードします。このワークフローが完了すると、クラウドで実行されているゲームサーバーと、それらに接続できる作業中のゲームクライアントが作成されます。

Amazon GameLift マネージド Amazon EC2 ワークフローを開始するには：

- Unity エディタのメインメニューで Amazon GameLift を選択し、マネージド EC2 でホストを選択します。このワークフローでは、ゲームコンポーネントを統合し、構築、デプロイ、起動するための 6 ステップのプロセスを紹介します。

プロファイルを設定する

このワークフローに従うときに使用したいプロファイルを選択します。選択したプロファイルは、ワークフローのすべてのステップに影響します。作成したすべてのリソースはプロファイルの AWS アカウントに関連付けられ、プロファイルのデフォルトの AWS リージョンに配置されます。プロファイルユーザーのアクセス許可によって、AWS リソースとアクションへのアクセスが決まります。

1. 使用可能なプロファイルのドロップダウンリストからプロファイルを選択します。まだプロファイルを持っていない場合、または新しいプロファイルを作成する場合は、Amazon GameLift メニューに移動し、AWSアカウントプロファイルの設定を選択します。
2. ブートストラップステータスが「アクティブ」でない場合は、[ブートストラップのプロファイル] を選択し、ステータスが「アクティブ」に変わるまで待ちます。

ゲームを Amazon と統合する GameLift

このタスクでは、ゲームプロジェクトのクライアントコードとサーバーコードを更新します。

- ゲームサーバーは、ゲームセッションの開始、ゲームセッション接続情報の提供、およびステータスのレポートを行うためのプロンプトを受信するために、Amazon GameLift サービスと通信できる必要があります。
- ゲームクライアントは、ゲームセッションに関する情報の取得、ゲームセッションへの参加または開始、ゲームに参加するための接続情報の取得を行える必要があります。

> **Note**
>
> サンプルゲームをインポートした場合は、このステップをスキップできます。サンプルゲームアセットには、必要なサーバーとクライアントコードが既に用意されています。

サーバーコードの統合

カスタムシーンで独自のゲームプロジェクトを使用する場合は、提供されたサンプルコードを使用して、ゲームプロジェクトに必要なサーバーコードを追加します。Anywhere フリートとのテスト用にゲームプロジェクトを統合した場合、このステップの手順はすでに完了しています。

1. ゲームプロジェクトファイルで、 `Assets/Scripts/Server`フォルダを開きます。存在しない場合は、作成します。

2. GitHub リポジトリ aws/amazon-gamelift-plugin-unity に移動し、パス を開きますSamples~/SampleGame/Assets/Scripts/Server。
3. ファイルを見つけGameLiftServer.csて、ゲームプロジェクトの Serverフォルダにコピーします。サーバー実行可能ファイルを構築するときは、このファイルをビルドターゲットとして使用します。

サンプルコードには、Amazon GameLift C# サーバー SDK (バージョン 5) を使用するこれらの最低限必要な要素が含まれています。

- Amazon GameLift API クライアントを初期化します。Amazon GameLift Anywhere フリートには、サーバーパラメータを使用した InitSDK () 呼び出しが必要です。これらの設定は、プラグインで使用するように自動的に設定されます。
- OnStartGameSession、、 OnProcessTerminateなど、Amazon GameLift サービスからのリクエストに応答するために必要なコールバック関数を実装しますonHealthCheck。
- 指定されたポートProcessReady()で を呼び出し、サーバープロセスがゲームセッションをホストする準備ができたときに Amazon GameLift サービスに通知します。

サンプルサーバーコードをカスタマイズする場合は、次のリソースを参照してください。

- Amazon GameLift をゲームサーバーに追加する
- C# と Unity 用 Amazon GameLift サーバー SDK 5.x リファレンス

クライアントコードの統合

クラウドベースのゲームサーバーに接続するゲームクライアントの場合、ゲームクライアントから直接呼び出すのではなく、クライアント側のバックエンドサービスを使用して Amazon GameLift サービスを呼び出すのがベストプラクティスです。

マネージド EC2 フリートでホストするプラグインワークフローでは、各デプロイシナリオに、以下のコンポーネントを含むビルド済みバックエンドサービスが含まれています。

- ゲームセッションのリクエストとゲームセッション情報の取得に使用される Lambda 関数と DynamoDB テーブルのセット。これらのコンポーネントは API ゲートウェイをプロキシとして使用します。
- 一意のプレイヤー IDs を生成し、プレイヤー接続を認証する Amazon Cognito ユーザープール。

これらのコンポーネントを使用するには、ゲームクライアントがバックエンドサービスにリクエストを送信して以下を実行する機能が必要です。

- AWS Cognito ユーザープールにプレイヤーユーザーを作成し、認証します。
- ゲームセッションに参加して接続情報を受け取ります。
- マッチメーキングを使用してゲームに参加します。

以下のリソースをガイドとして使用します。

- クライアントを GitHub リポジトリ aws/ amazon-gamelift-plugin-unityの GameLiftCoreApi クラスと統合します。このクラスは、プレイヤー認証とゲームセッション情報の取得のコントロールを提供します。
- サンプルゲーム統合を表示するには、 GitHub リポジトリ aws/amazon-gamelift-plugin-unity、 に移動しますSamples~/SampleGame/Assets/Scripts/Client/GameLiftClient.cs。
- Unity ゲームクライアント GameLift に Amazon を追加します。

デプロイシナリオの選択

このステップでは、この時点でデプロイしたいゲームホスティングソリューションを選択します。どのシナリオを使用しても、ゲームを複数デプロイすることができます。

- 単一リージョンフリート： アクティブなプロファイルのデフォルトAWSリージョンのホスティングリソースの単一のフリートにゲームサーバーをデプロイします。このシナリオは、AWS とのサーバー統合とサーバービルド設定をテストする出発点に適しています。次のリソースをデプロイします。
  - ゲームサーバービルドをインストールして実行中の AWS フリート (オンデマンド)。
  - プレイヤーが認証してゲームを開始するための Amazon Cognito ユーザープールとクライアント。
  - ユーザープールと API をリンクする API ゲートウェイオーソライザー。
  - プレイヤーから API ゲートウェイへの過剰な呼び出しを スロットリングする WebACL。
  - プレイヤーがゲームスロットをリクエストするための API ゲートウェイ + Lambda 関数。この関数は、何も利用できない場合に CreateGameSession( ) を呼び出します。
  - プレイヤーがゲームリクエストの接続情報を取得するための API ゲートウェイ + Lambda 関数。

- FlexMatch フリート： ゲームサーバーを一連のフリートにデプロイし、プレイヤー FlexMatch マッチを作成するためのルールを含むマッチメーカーを設定します。このシナリオでは、マルチフリートのマルチロケーション構造で低コストのスポットホスティングを使用し、高い可用性を実現します。このアプローチは、ホスティングソリューションのマッチメーカーコンポーネントの設計を開始する準備ができたときに役立ちます。このシナリオでは、このソリューションの基本リソースを作成し、必要に応じて後でカスタマイズできます。次のリソースをデプロイします。
  - FlexMatch プレイヤーのリクエストとフォームマッチを受け入れるためのマッチメーキング設定とマッチメーキングルールセット。
  - ゲームサーバービルドがインストールされ、複数のロケーションで稼働している 3 AWS つのフリート。バックアップとして 2 つのスポットフリートと 1 つのオンデマンドフリートが含まれます。
  - (実行可能性、コスト、プレイヤーレイテンシーなどに基づいて) 最適なホスティングリソースを見つけ、ゲームセッションを開始することで、提案されたマッチのリクエストに応える AWS ゲームセッションプレイスメントキュー。
  - プレイヤーが認証してゲームを開始するための Amazon Cognito ユーザープールとクライアント。
  - ユーザープールと API をリンクする API ゲートウェイオーソライザー。
  - プレイヤーから API ゲートウェイへの過剰な呼び出しを スロットリングする WebACL。
  - プレイヤーがゲームスロットをリクエストするための API ゲートウェイ + Lambda 関数。この関数は `StartMatchmaking()` を呼び出します。
  - プレイヤーがゲームリクエストの接続情報を取得するための API ゲートウェイ + Lambda 関数。
  - Amazon DynamoDB テーブルには、プレイヤーのマッチメーキングチケットとゲームセッション情報を保存できます。
  - SNS トピック + GameSessionQueue イベントを処理する Lambda 関数。

ゲームパラメータの設定

このステップでは、AWS にアップロードするゲームを記述します。

- ゲーム名： ゲームプロジェクトにわかりやすい名前を指定します。この名前はプラグイン内で使用されます。
- フリート名： マネージド EC2 フリートにわかりやすい名前を指定します。Amazon GameLift は、AWSコンソールでリソースを一覧表示するときに、この名前 (フリート ID とともに) を使用します。

- ビルド名： サーバービルドにわかりやすい名前を指定します。 AWSはこの名前を使用して、Amazon にアップロード GameLift され、デプロイに使用されるサーバービルドのコピーを参照します。
- 起動パラメータ： マネージド EC2 フリートインスタンスでサーバー実行可能ファイルを起動するときに実行するオプションの手順を入力します。最大長は 1024 文字です。
- ゲームサーバーフォルダ： サーバービルドを含むローカルフォルダへのパスを指定します。
- ゲームサーバーファイル： サーバー実行可能ファイル名を指定します。

デプロイシナリオ

このステップでは、選択したデプロイシナリオに基づいてゲームをクラウドホスティングソリューションにデプロイします。このプロセスには、AWS がサーバービルドの検証、ホスティングリソースのプロビジョニング、ゲームサーバーのインストール、サーバープロセスの起動、ゲームセッションをホストする準備が整うまで 40 分ほどかかる場合があります。

デプロイを開始するには、デプロイを選択します CloudFormation。ゲームホスティングの状況は、こちらで追跡できます。より詳細な情報については、AWS の AWS 管理コンソールにサインインしてイベント通知を確認することができます。必ず、プラグインのアクティブユーザープロファイルと同じアカウント、ユーザー、AWS リージョンを使用してサインインしてください。

デプロイが完了すると、AWS EC2 インスタンスにゲームサーバーがインストールされています。少なくとも 1 つのサーバープロセスが実行中で、ゲームセッションを開始する準備ができています。

ゲームクライアントを起動する

フリートが正常にデプロイされると、ゲームサーバーが実行され、ゲームセッションをホストできるようになります。これで、クライアントを構築して起動し、接続してゲームセッションに参加できます。

1. ゲームクライアントを設定します。このステップでは、ゲームプロジェクトの`GameLiftClientSettings`アセットを更新するようにプラグインに促します。プラグインはこのアセットを使用して、ゲームクライアントが Amazon GameLift サービスに接続するために必要な特定の情報を保存します。

    a. サンプルゲームをインポートして初期化していない場合は、新しい`GameLiftClientSettings`アセットを作成します。Unity エディタのメインメニューで、アセット、作成、 GameLift、クライアント設定を選択します。プロジェクト GameLiftClientSettings で のコピーを複数作成すると、プラグインは自動的にこれを検出し、プラグインが更新するアセットを通知します。

b. ゲームを起動 で、クライアントの設定: マネージド EC2 設定の適用 を選択します。このアクションは、デプロイしたマネージド EC2 フリートを使用するようにゲームクライアント設定を更新します。

2. ゲームクライアントを構築します。標準の Unity ビルドプロセスを使用して、クライアント実行可能ファイルを構築します。ファイル、ビルド設定で、プラットフォームを Windows、Mac、Linux に切り替えます。サンプルゲームをインポートして設定を初期化すると、ビルドリストとビルドターゲットが自動的に更新されます。
3. 新しく構築されたゲームクライアント実行可能ファイルを起動します。ゲームのプレイを開始するには、2～4 つのクライアントインスタンスを起動し、それぞれの UI を使用してゲームセッションに参加します。

サンプルゲームクライアントを使用している場合、次の特性があります。

- プレイヤーログインコンポーネント。Anywhere フリートのゲームサーバーに接続する場合、プレイヤーの検証はありません。任意の値を入力してゲームセッションに参加できます。
- シンプルなゲーム結合 UI。クライアントがゲームに参加しようとすると、クライアントは使用可能なプレイヤースロットを持つアクティブなゲームセッションを自動的に検索します。利用可能なゲームセッションがない場合、クライアントは新しいゲームセッションをリクエストします。ゲームセッションが利用可能な場合、クライアントは利用可能なゲームセッションへの参加をリクエストします。複数の同時クライアントでゲームをテストすると、最初のクライアントはゲームセッションを開始し、残りのクライアントは自動的に既存のゲームセッションに参加します。
- 4 つのプレイヤースロットを持つゲームセッション。最大 4 つのゲームクライアントインスタンスを同時に起動でき、同じゲームセッションに参加します。

サーバー SDK 4.x 用 Unity ガイド用 Amazon GameLift プラグイン

> **Note**
>
> このトピックでは、Unity 用 Amazon GameLift プラグインの以前のバージョンについて説明します。バージョン 1.0.0 (2021 年にリリース) では、Amazon GameLift サーバー SDK 4.x 以前を使用しています。サーバー SDK 5.x を使用し、Amazon をサポートするプラグインの最新バージョンのドキュメントについては、 GameLift Anywhere「」を参照してくださいサーバー SDK 5.x 用 Unity ガイド用 Amazon GameLift プラグイン。

Amazon GameLift には、Amazon で実行するようにマルチプレイヤーゲームサーバーを準備するためのツールが用意されています GameLift。Unity 用 Amazon GameLift プラグインを使用すると、Amazon を Unity ゲームプロジェクト GameLift に統合し、クラウドホスティング用に Amazon GameLift リソースを簡単にデプロイできます。Unity 用プラグインを使用して Amazon GameLift APIs、一般的なゲームシナリオ用にAWS CloudFormationテンプレートをデプロイします。

プラグインを設定したら、 で Amazon GameLift Unity サンプルを試すことができます GitHub。

トピック

- Amazon を Unity ゲームサーバープロジェクト GameLift と統合する
- Amazon を Unity ゲームクライアントプロジェクト GameLift と統合する
- プラグインをインストールしてセットアップする
- ローカルでゲームをテストする
- シナリオをデプロイする
- Unity GameLift で Amazon とゲームを統合する
- サンプルゲームをインポートして実行する

Amazon を Unity ゲームサーバープロジェクト GameLift と統合する

> **Note**
>
> このトピックでは、Unity 用 Amazon GameLift プラグインの以前のバージョンについて説明します。バージョン 1.0.0 (2021 年にリリース) では、Amazon GameLift サーバー SDK 4.x 以前を使用しています。サーバー SDK 5.x を使用し、Amazon をサポートするプラグインの最新バージョンのドキュメントについては、 GameLift Anywhere「」を参照してくださいサーバー SDK 5.x 用 Unity ガイド用 Amazon GameLift プラグイン。

このトピックは、Amazon でホストするカスタムゲームサーバーを準備するのに役立ちます GameLift。ゲームサーバーは、そのステータス GameLift を Amazon に通知し、プロンプトが表示されたらゲームセッションを開始および停止し、他のタスクを実行できる必要があります。詳細については、Amazon GameLift をゲームサーバーに追加するを参照してください。

前提条件

ゲームサーバーを統合する前に、次のタスクを完了します。

- Amazon の IAM サービスロールを設定する GameLift
- Unity 用プラグインをインストールする

新しいサーバープロセスのセットアップ

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

Amazon との通信を設定し GameLift 、サーバープロセスがゲームセッションをホストする準備ができていることを報告します。

1. `InitSDK()` を呼び出してサーバー SDK を初期化します。
2. サーバーがゲームセッションを受け入れる準備をするには、接続ポートとゲームセッションの場所の詳細を指定して `ProcessReady()` を呼び出します。`OnGameSession()`、、、など、Amazon GameLift サービスが呼び出すコールバック関数の名前を含めます`OnGameSessionUpdate()``OnProcessTerminate()``OnHealthCheck()`。Amazon がコールバックを提供するまでに数分かかる GameLift 場合があります。
3. Amazon は、サーバープロセスのステータスを に GameLift 更新します`ACTIVE`。
4. Amazon は`onHealthCheck`定期的に GameLift を呼び出します。

次のコード例は、Amazon でシンプルなサーバープロセスをセットアップする方法を示しています GameLift。

```
//initSDK
var initSDKOutcome = GameLiftServerAPI.InitSDK();

//processReady
// Set parameters and call ProcessReady
var processParams = new ProcessParameters(
    this.OnGameSession,
    this.OnProcessTerminate,
    this.OnHealthCheck,
    this.OnGameSessionUpdate,
    port,
    // Examples of log and error files written by the game server
```

```
    new LogParameters(new List<string>()
        {
            "C:\\game\\logs",
            "C:\\game\\error"
        })
);

var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);

// Implement callback functions
void OnGameSession(GameSession gameSession)
{
    // game-specific tasks when starting a new game session, such as loading map
    // When ready to receive players
    var activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}

void OnProcessTerminate()
{
    // game-specific tasks required to gracefully shut down a game session,
    // such as notifying players, preserving game state data, and other cleanup
    var ProcessEndingOutcome = GameLiftServerAPI.ProcessEnding();
}

bool OnHealthCheck()
{
    bool isHealthy;
    // complete health evaluation within 60 seconds and set health
    return isHealthy;
}
```

ゲームセッションをスタートする

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

ゲームの初期化が完了すると、ゲームセッションを開始できます。

1. コールバック関数 onStartGameSession を実装します。Amazon はこのメソッドを GameLift 呼び出して、サーバープロセスで新しいゲームセッションを開始し、プレイヤー接続を受信します。
2. ゲームセッションをアクティブ化するには、ActivateGameSession() を呼び出します。SDK の詳細については、「Amazon GameLift サーバー SDK (C#) リファレンス: アクション」を参照してください。

次のコード例は、Amazon でゲームセッションを開始する方法を示しています GameLift。

```
void OnStartGameSession(GameSession gameSession)
{
    // game-specific tasks when starting a new game session, such as loading map
    ...
    // When ready to receive players
    var activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}
```

ゲームセッションを終了する

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

ゲームセッションが終了した GameLift ときに Amazon に通知します。ベストプラクティスとして、ゲームセッションがホスティングリソースのリサイクルと更新を完了した後に、サーバープロセスをシャットダウンします。

1. Amazon からリクエストを受信 GameLift して を呼び出すonProcessTerminateには、 という名前の関数を設定しますProcessEnding()。
2. プロセスのステータスは TERMINATED に変わります。

次の例は、ゲームセッションのプロセスを終了する方法を示しています。

```
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();

if (processReadyOutcome.Success)
```

```
    Environment.Exit(0);

// otherwise, exit with error code
Environment.Exit(errorCode);
```

サーバービルドを作成して Amazon にアップロードする GameLift

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

ゲームサーバーを Amazon と統合したら GameLift、Amazon がゲームホスティング用に GameLift デプロイできるように、ビルドファイルをフリートにアップロードします。サーバーを Amazon にアップロードする方法の詳細については GameLift、「」を参照してくださいカスタムゲームサーバー構築を Amazon GameLift にアップロードする。

Amazon を Unity ゲームクライアントプロジェクト GameLift と統合する

> **Note**
>
> このトピックでは、Unity 用 Amazon GameLift プラグインの以前のバージョンについて説明します。バージョン 1.0.0 (2021 年にリリース) では、Amazon GameLift サーバー SDK 4.x 以前を使用しています。サーバー SDK 5.x を使用し、Amazon をサポートするプラグインの最新バージョンのドキュメントについては、 GameLift Anywhere「」を参照してくださいサーバー SDK 5.x 用 Unity ガイド用 Amazon GameLift プラグイン。

このトピックでは、バックエンドサービスを介して Amazon が GameLift ホストするゲームセッションに接続するためのゲームクライアントをセットアップする方法について説明します。Amazon GameLift APIs を使用して、マッチメーキングの開始、ゲームセッション配置のリクエストなどを行います。

バックエンドサービスプロジェクトにコードを追加して、Amazon GameLift サービスとの通信を許可します。バックエンドサービスは、 GameLift サービスとのすべてのゲームクライアント通信を処理します。バックエンドサービスの詳細については、「ゲームクライアントサービスを設計する」を参照してください。

バックエンドサーバーは次のゲームクライアントタスクを処理します。

- プレイヤーの認証をカスタマイズする。
- Amazon GameLift サービスからアクティブなゲームセッションに関する情報をリクエストします。
- 新しいゲームセッションを作成する。
- 既存のゲームセッションにプレーヤーを追加する。
- 既存のゲームセッションからプレイヤーを削除する。

トピック

前提条件

Amazon GameLift クライアントとのゲームサーバー通信を設定する前に、以下のタスクを完了してください。

- のセットアップ AWS アカウント
- Unity 用プラグインをインストールする
- Amazon を Unity ゲームサーバープロジェクト GameLift と統合する
- Amazon GameLift のフリートのセットアップ

ゲームクライアントを初期化する

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

ゲームクライアントを初期化するコードを追加する。起動時にこのコードを実行します。これは他の Amazon GameLift 関数に必要です。

1. AmazonGameLiftClient を初期化する。デフォルトのクライアント設定またはカスタム設定のいずれかで AmazonGameLiftClient を呼び出す。クライアントの設定方法に関する詳細については、「バックエンドサービス GameLift で Amazon をセットアップする」を参照してください。
2. ゲームセッションに接続するための一意のプレイヤー ID を生成する。詳細については、プレイヤー ID を生成するを参照してください。

   次の例は、Amazon GameLift クライアントをセットアップする方法を示しています。

```
public class GameLiftClient
{
    private GameLift gl;
    //A sample way to generate random player IDs.
    bool includeBrackets = false;
    bool includeDashes = true;
    string playerId = AZ::Uuid::CreateRandom().ToString<string>(includeBrackets,
 includeDashes);


    private Amazon.GameLift.Model.PlayerSession psession = null;
    public AmazonGameLiftClient aglc = null;

    public void CreateGameLiftClient()
    {
        //Access Amazon GameLift service by setting up a configuration.
        //The default configuration specifies a location.
        var config = new AmazonGameLiftConfig();
        config.RegionEndpoint = Amazon.RegionEndpoint.USEast1;

        CredentialProfile profile = null;
        var nscf = new SharedCredentialsFile();
        nscf.TryGetProfile(profileName, out profile);
        AWSCredentials credentials = profile.GetAWSCredentials(null);
        //Initialize GameLift Client with default client configuration.
        aglc = new AmazonGameLiftClient(credentials, config);

    }
}
```

特定のフリートにゲームセッションを作成する

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

デプロイ済みフリートで新しいゲームセッションを起動し、それらのセッションをプレイヤーが使用できるようにするコードを追加します。Amazon GameLift が新しいゲームセッションを作成して を返したらGameSession、そのセッションにプレイヤーを追加できます。

- 新しいゲームセッションをリクエストします。
  - ゲームでフリートを使用する場合は、フリートまたはエイリアス ID、セッション名、およびゲームの最大同時プレイヤー数を指定して CreateGameSession() を呼び出します。
  - ゲームがキューを使用している場合は、StartGameSessionPlacement() を呼び出します。

次の例は、ゲームセッションを作成する方法を示しています。

```
public Amazon.GameLift.Model.GameSession()
{
    var cgsreq = new Amazon.GameLift.Model.CreateGameSessionRequest();
    //A unique identifier for the alias with the fleet to create a game session in.
    cgsreq.AliasId = aliasId;
    //A unique identifier for a player or entity creating the game session
    cgsreq.CreatorId = playerId;
    //The maximum number of players that can be connected simultaneously to the game
 session.
    cgsreq.MaximumPlayerSessionCount = 4;

    //Prompt an available server process to start a game session and retrieves
 connection information for the new game session
    Amazon.GameLift.Model.CreateGameSessionResponse cgsres =
 aglc.CreateGameSession(cgsreq);
    string gsid = cgsres.GameSession != null ? cgsres.GameSession.GameSessionId : "N/
A";
    Debug.Log((int)cgsres.HttpStatusCode + " GAME SESSION CREATED: " + gsid);
    return cgsres.GameSession;
}
```

プレイヤーをゲームセッションに追加する

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

Amazon GameLift が新しいゲームセッションを作成してGameSessionオブジェクトを返したら、そのセッションにプレイヤーを追加できます。

1. 新しいプレイヤーセッションを作成して、ゲームセッションにプレイヤースロットを予約します。ゲームセッション ID と各プレイヤーの固有の ID を指定して、CreatePlayerSession または CreatePlayerSessions を使用します。
2. ゲーム セッションに接続します。PlayerSession オブジェクトを取得してゲームセッションの接続情報を取得します。この情報を使用して、サーバープロセスへの直接接続を確立できます。

    a. 指定したポートと、サーバープロセスの DNS 名または IP アドレスを使用します。
    b. フリートの DNS 名とポートを使用します。フリートに対して TLS 証明書の生成が有効になっている場合、DNS 名とポートが必要です。
    c. プレイヤーセッション ID を参照します。ゲームサーバーが新規プレイヤーの接続を検証する場合は、プレイヤーセッション ID が必要です。

以下の例は、ゲームセッションでプレイヤースポットを予約する方法を示しています。

```
public Amazon.GameLift.Model.PlayerSession
 CreatePlayerSession(Amazon.GameLift.Model.GameSession gsession)
{
    var cpsreq = new Amazon.GameLift.Model.CreatePlayerSessionRequest();
    cpsreq.GameSessionId = gsession.GameSessionId;
    //Specify game session ID.
    cpsreq.PlayerId = playerId;
    //Specify player ID.
    Amazon.GameLift.Model.CreatePlayerSessionResponse cpsres =
 aglc.CreatePlayerSession(cpsreq);
    string psid = cpsres.PlayerSession != null ? cpsres.PlayerSession.PlayerSessionId :
 "N/A";
    return cpsres.PlayerSession;
```

```
}
```

次のコードは、プレイヤーをゲームセッションに接続する方法を示しています。

```
public bool ConnectPlayer(int playerIdx, string playerSessionId)
{
    //Call ConnectPlayer with player ID and player session ID.
    return server.ConnectPlayer(playerIdx, playerSessionId);
}
```

ゲームセッションからプレイヤーを削除する

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

プレイヤーがゲームを離れると、そのプレイヤーをゲームセッションから削除できます。

1. プレイヤーがサーバープロセスから切断されたことを Amazon GameLift サービスに通知します。プレイヤーのセッション ID で RemovePlayerSession を呼び出します。
2. RemovePlayerSession が Success を返すことを確認します。次に、Amazon はプレイヤースロット GameLift を利用可能に変更します。このスロットは、Amazon が新しいプレイヤーに割り当てる GameLift ことができます。

以下の例は、プレイヤーセッションを削除する方法を示しています。

```
public void DisconnectPlayer(int playerIdx)
{
    //Receive the player session ID.
    string playerSessionId = playerSessions[playerIdx];
    var outcome = GameLiftServerAPI.RemovePlayerSession(playerSessionId);
    if (outcome.Success)
    {
        Debug.Log (":) PLAYER SESSION REMOVED");
    }
    else
    {
```

```
        Debug.Log(":(PLAYER SESSION REMOVE FAILED. RemovePlayerSession()
        returned " + outcome.Error.ToString());
    }
}
```

プラグインをインストールしてセットアップする

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

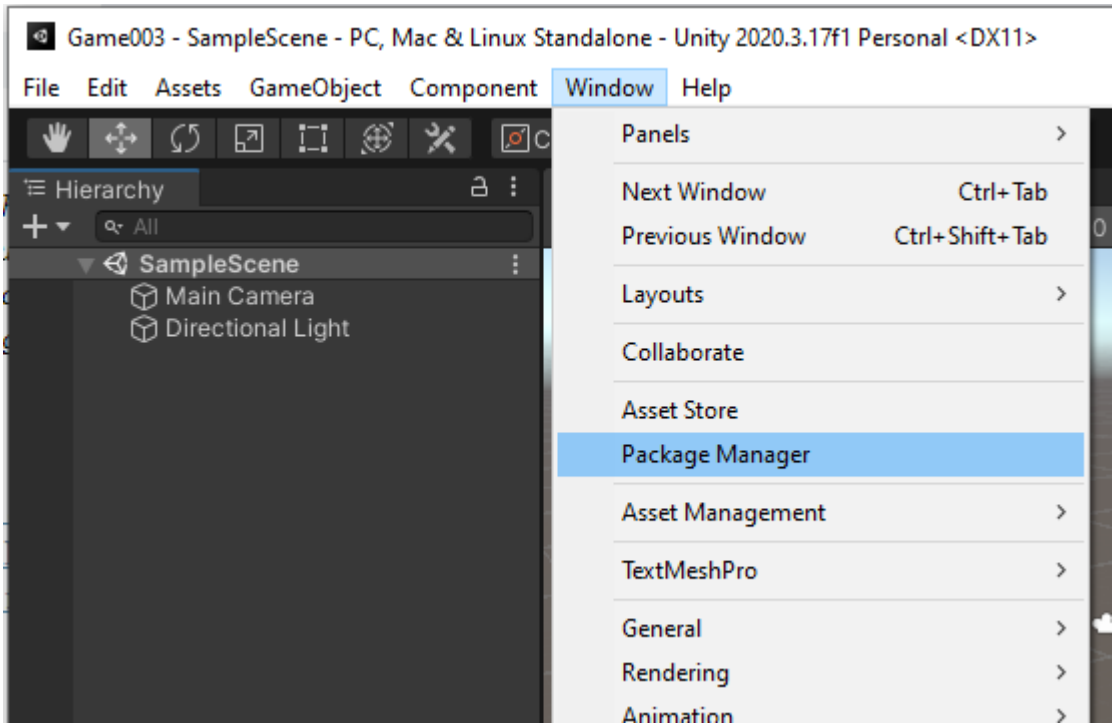
このセクションでは、Unity 用 Amazon GameLift プラグインのバージョン 1.0.0 をダウンロード、インストール、およびセットアップする方法について説明します。

前提条件

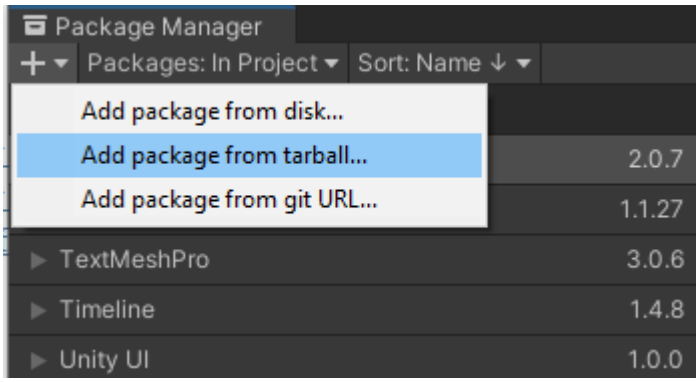
- Windows 2019.4 LTS 用 Unity、Windows 2020.3 LTS 用、または MacOS 用Unity
- Java の現在のバージョン
- .NET 4.x の現在のバージョン

Unity 用プラグインをダウンロードしてインストールするには

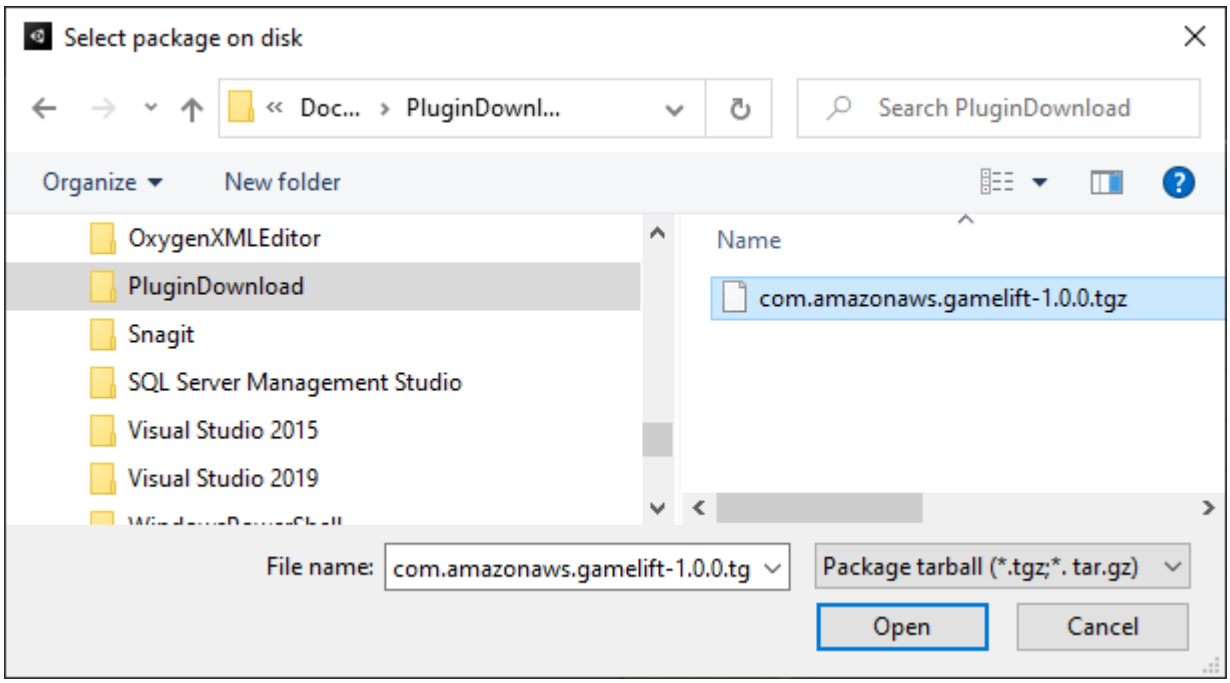
1. Unity 用 Amazon GameLift プラグインをダウンロードします。最新バージョンは、「Unity 用 Amazon GameLift プラグイン」リポジトリページにあります。[最新リリース] で [アセット] を選択し、`com.amazonaws.gamelift-version.tgz` ファイルをダウンロードします。
2. Unity を起動し、プロジェクトを選択します。
3. 上部のナビゲーションバーの [ウィンドウ] で、[パッケージマネージャー] を選択します。



4. [パッケージマネージャー] タブで [+] を選択し、[ターボールからパッケージを追加...] を選択します。



5. [ディスク上のパッケージを選択] ウィンドウで `com.amazonaws.gamelift` フォルダに移動し、ファイル `com.amazonaws.gamelift-`*`version`*`.tgz` を選択して [開く] を選択します。



6. Unity がプラグインをロードすると、Amazon GameLift は Unity メニューに新しい項目として表示されます。スクリプトのインストールと再コンパイルには数分かかる場合があります。Amazon GameLift プラグイン設定 タブが自動的に開きます。

7. SDK ペインで、Use .NET 4.x を選択します。

   構成すると、ステータスは [未設定] から [設定済み]に変更します。

ローカルでゲームをテストする

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

Amazon GameLift Local を使用して、ローカルデバイスで Amazon GameLift を実行します。Amazon GameLift Local を使用すると、ネットワーク接続なしでコードの変更を数秒で確認できます。

ローカルテストを設定する

1. Unity 用プラグインウィンドウで、[テスト] タブを選択します。
2. テストペインで、Amazon GameLift Local のダウンロードを選択します。Unity 用プラグインはブラウザウィンドウを開き、`GameLift_06_03_2021.zip` ファイルをダウンロードフォルダにダウンロードします。

   ダウンロードには、C# サーバー SDK、.NET ソースファイル、および Unity と互換性のある .NET コンポーネントが含まれます。
3. ダウンロードした `GameLift_06_03_2021.zip` ファイルを解凍します。
4. Amazon GameLift プラグイン設定ウィンドウで、Amazon GameLift Local Path を選択し、解凍したフォルダに移動し、ファイル を選択し`GameLiftLocal.jar`、 を開くを選択します。

   設定すると、ローカルテストのステータスが [未設定] から [設定済み] に変更します。
5. JRE のステータスを確認します。[未設定] の場合は、[JRE のダウンロード] を選択して、推奨された Java バージョンをインストールします。

   Java 環境がインストールされて設定されると、ステータスは [設定済み] に変更します。

ローカルゲームを実行する

1. Unity 用プラグインタブで、[テスト] タブを選択します。
2. [テスト] ペインで、[ローカルテスト UI を開く] を選択します。

3. [Local Testing] (ローカルテスト) ウィンドウで、[Server executable path](サーバー実行可能パス) を指定します。... を選択して、サーバーアプリケーションのパスと実行可能ファイル名を選択します。
4. [Local Testing] (ローカルテスト) ウィンドウで、GL ローカルポート を指定します。
5. [デプロイと実行] を選択し、サーバーをデプロイして実行します。
6. ゲームサーバーを停止するには、[停止] を選択するか、ゲームサーバーウィンドウを閉じます。

シナリオをデプロイする

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

シナリオでは、 AWS CloudFormation テンプレートを使用して、ゲームのクラウドホスティングソリューションをデプロイするために必要なリソースを作成します。このセクションでは、Amazon GameLift が提供するシナリオとその使用方法について説明します。

前提条件

シナリオをデプロイするには、Amazon GameLift サービスの IAM ロールが必要です。Amazon のロールを作成する方法については GameLift、「」を参照してくださいのセットアップ AWS アカウント。

各シナリオには、次のリソースへのアクセス許可が必要になります。

- Amazon GameLift
- Amazon S3
- AWS CloudFormation
- API Gateway
- AWS Lambda
- AWS WAFV2
- Amazon Cognito

シナリオ

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

Unity 用 Amazon GameLift プラグインには、次のシナリオが含まれます。

認証のみ

このシナリオでは、ゲームサーバー機能なしでプレイヤー認証を実行するゲームバックエンドサービスを作成します。このテンプレートによって以下のリソースがアカウントに作成されます。

- プレイヤー認証情報を保存する Amazon Cognito ユーザープール。
- Amazon API Gateway REST エンドポイントベースの AWS Lambda ハンドラーで、ゲームを開始したりゲーム接続情報を表示したりします。

単一リージョンフリート

このシナリオでは、単一の Amazon GameLift フリートでゲームバックエンドサービスを作成します。次のリソースを作成します。

- プレイヤーが認証してゲームを開始するための Amazon Cognito ユーザープール。
- フリートに空いているプレイヤースロットがある既存のゲームセッションを検索する AWS Lambda ハンドラー。空いているスロットが見つからない場合は、新しいゲームセッションを作成します。

キューとカスタムマッチメーカーを使ったマルチリージョンフリート

このシナリオでは、Amazon GameLift キューとカスタムマッチメーカーを使用して、待機プール内の最も古いプレイヤーをグループ化することでマッチを形成します。次のリソースを作成します。

- Amazon がメッセージ GameLift を発行する Amazon Simple Notification Service トピック。SNS トピックと通知に関する詳細については、「ゲームセッション配置のイベン通知を設定」を参照してください。
- プレイスメントとゲーム接続の詳細を通知するメッセージによって呼び出される Lambda 関数。

- プレイスメントとゲーム接続の詳細を保存する Amazon DynamoDB テーブル。`GetGameConnection` 呼び出しは、このテーブルから読み取り、ゲームクライアントに接続情報を返します。

キューとカスタムマッチメーカーを使ったスポットフリート

このシナリオでは、Amazon GameLift キューとカスタムマッチメーカーを使用してマッチを形成し、3 つのフリートを設定します。次のリソースを作成します。

- スポット使用不能に対する耐久性を提供する異なるインスタンスタイプを含む 2 つのスポットフリートです。
- 他のスポットフリートのバックアップとして機能するオンデマンドフリート。フリートの設計に関する詳細は、「Amazon GameLift フリート設計ガイド」を参照してください。
- サーバーの可用性とコストを低く抑えるための Amazon GameLift キュー。キューについての詳細とベストプラクティスに関しては、「ゲームセッションキューの設計」を参照してください。

FlexMatch

このシナリオでは FlexMatch、マネージドマッチメーキングサービスである を使用して、ゲームプレイヤーを一致させます。の詳細については FlexMatch、「Amazon とは GameLift FlexMatch」を参照してください。このシナリオでは、次のリソースを作成します。

- `StartGame` リクエストを受け取った後にマッチメーキングチケットを作成する Lambda 関数。
- FlexMatch 一致イベントをリッスンする別の Lambda 関数。

AWS アカウント に不要な課金が発生しないように、各シナリオで作成されたリソースは使用後に削除してください。対応する AWS CloudFormation スタックを削除します。

AWS 認証情報の更新

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

Unity 用 Amazon GameLift プラグインでは、シナリオをデプロイするためにセキュリティ認証情報が必要です。新しい認証情報を作成するか、既存の認証情報を使用できます。

認証情報の設定の詳細については、AWS 認証情報の理解と取得を参照してください。

AWS認証情報を更新するには

1. Unity では、Unity 用プラグインタブで、[デプロイ] タブを選択します。
2. [デプロイ] ペインで、AWS[認証情報] を選択します。
3. 新しい AWS 認証情報を作成するか、既存の認証情報を選択できます。
   - 認証情報を作成するには、[新しい認証情報プロファイルの作成] を選択し、[新しいプロファイル名]、AWS[アクセスキー ID]、AWS[シークレットキー] および AWS リージョン を指定します。
   - 既存の認証情報を選択するには、[既存の認証情報プロファイルの選択] を選択し、プロファイル名および AWS リージョン を選択します。
4. [AWS 認証情報の更新] ウィンドウで、[認証情報プロファイルの更新] を選択します。

アカウントブートストラップを更新する

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

ブートストラップの場所は、デプロイ中に使用される Amazon S3 バケットです。これは、ゲームサーバーのアセットやその他の依存関係を保存するために使用されます。バケットに選択する AWS リージョン は、シナリオのデプロイに使用するリージョンと同じである必要があります。

Amazon S3 バケットの詳細については、Amazon Simple ストレージサービスバケットの作成、設定、操作を参照してください。

アカウントのブートストラップの場所を更新するには

1. Unity では、Unity 用プラグインタブで、[デプロイ] タブを選択します。
2. [デプロイ] ペインで、[アカウントブートストラップを更新] を選択します。
3. [アカウントブートストラッピング] ウィンドウで、既存の Amazon S3 バケットを選択するか、新しい Amazon S3 バケットを作成します。
   - 既存のバケットを選択するには、[既存の Amazon S3 バケットを選択] を選択し、[更新] を選択して選択内容を保存します。

- [新しい Amazon S3 バケットの作成] を選択して、Amazon Amazon Simple Storage Service バケットを作成し、[ポリシー] を選択します。Amazon S3 バケットの有効期限がポリシーで指定されています。バケットを作成するには、[Create](作成) を選択します。

ゲームシナリオをデプロイする

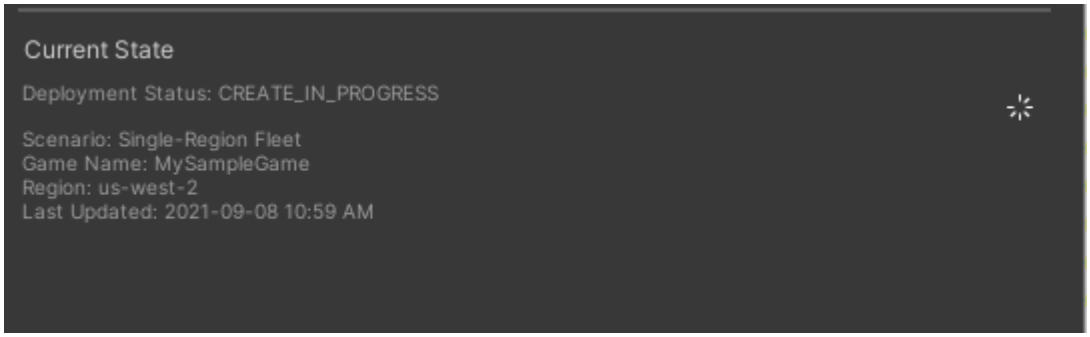
> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

シナリオを使用して、Amazon でゲームをテストできます GameLift。各シナリオでは、AWS CloudFormation テンプレートを使用して、必要なリソースを含むスタックを作成します。ほとんどのシナリオでは、ゲームサーバーの実行可能ファイルとビルドパスが必要です。シナリオをデプロイすると、Amazon GameLift はデプロイの一環としてゲームアセットをブートストラップの場所にコピーします。

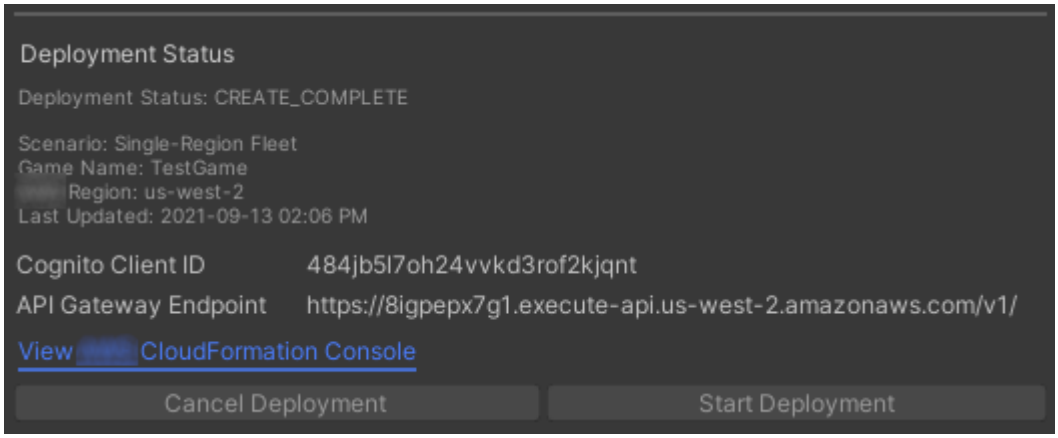
AWS認証情報と、シナリオをデプロイするためのAWS アカウント ブートストラップを設定する必要があります。

シナリオをデプロイするには

1. Unity では、Unity 用プラグインタブで、[デプロイ] タブを選択します。
2. [デプロイ] ペインで、[デプロイ UI を開く] を選択します。
3. [デプロイ] ウィンドウで、シナリオを選択します。
4. [チーム名] を入力します。また、名前は一意である必要があります。ゲーム名はシナリオをデプロイするときの AWS CloudFormation スタック名の一部です。
5. [ゲームサーバービルドフォルダパス] を選択します。構築フォルダのパスは、サーバーの実行可能ファイルと依存関係を含むフォルダを指します。
6. [ゲームサーバービルド .exe ファイルパス] を選択します。構築実行可能ファイルのパスは、ゲームサーバーの実行可能ファイルを指します。
7. [デプロイを開始] を選択してシナリオのデプロイを開始します。[デプロイ] ウィンドウの [現在の状態] で更新のステータスを確認できます。シナリオのデプロイには最大 30 分かかることがあります。



8. シナリオのデプロイが完了すると、[現在の状態] が更新され、ゲームにコピーして貼り付けられる Cognito クライアント ID および API Gateway エンドポイント が含まれます。



9. ゲーム設定を更新するには、Unity メニューで [クライアント接続設定に移動する] を選択します。これにより、[Inspector] タブが Unity 画面の右側に表示されます。
10. [ローカルテストモード] を選択解除します。
11. [API Gateway エンドポイント]と [Coginito クライアント ID] を入力します。シナリオのデプロイに使用したものと同じ AWS リージョン を選択します。その後、展開されたシナリオリソースを使用して、ゲームクライアントを再構築して実行できます。

シナリオによって作成されたリソースの削除

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

シナリオ用に作成されたリソースを削除するには、対応する AWS CloudFormation スタックを削除します。

シナリオによって作成されたリソースを削除するには

1. Unity デプロイ用の Amazon GameLift プラグインウィンドウで、AWS CloudFormationコンソールの表示を選択してAWS CloudFormationコンソールを開きます。
2. AWS CloudFormation コンソールで、[スタック] を選択し、デプロイ中に指定したゲーム名を含むスタックを選択します。
3. [削除] を選択して、スタックを削除します。スタックの削除には数分かかります。AWS CloudFormation がシナリオで使用されているスタックを削除すると、そのステータスは `ROLLBACK_COMPLETE` に変わります。

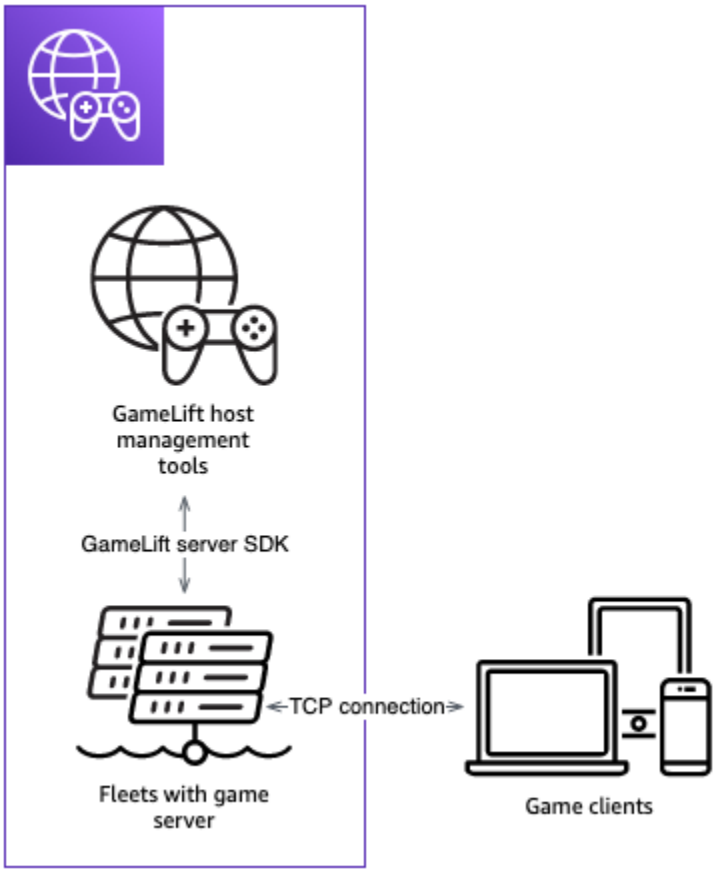
Unity GameLift で Amazon とゲームを統合する

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

次のタスクを完了 GameLift して、Unity ゲームを Amazon と統合します。

- Amazon を Unity ゲームサーバープロジェクト GameLift と統合する
- Amazon を Unity ゲームクライアントプロジェクト GameLift と統合する

次の図は、ゲームを統合するフローの例を示しています。この図では、ゲームサーバーを持つフリートが Amazon にデプロイされています GameLift。ゲームクライアントは、Amazon と通信するゲームサーバーと通信します GameLift。



サンプルゲームをインポートして実行する

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

Unity 用 Amazon GameLift プラグインには、ゲームを Amazon と統合するための基本を調べるために使用できるサンプルゲームが含まれています GameLift。このセクションでは、ゲームクライアントとゲームサーバーを構築し、Amazon GameLift Local を使用してローカルでテストします。

前提条件

- のセットアップ AWS アカウント
- プラグインをインストールしてセットアップする

サンプルゲームサーバーの構築と実行

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

サンプルゲームのゲームサーバーファイルをセットアップします。

1. Unity のメニューで Amazon GameLiftを選択し、サンプルゲームのインポート を選択します。
2. [サンプルゲームのインポート] ウィンドウで、[インポート] を選択して、ゲームとそのすべてのアセットと依存関係をインポートします。
3. ゲームサーバーを構築します。Unity のメニューで Amazon GameLiftを選択し、Windows サンプルサーバー構築設定の適用または MacOS サンプルサーバー構築設定の適用を選択します。ゲームサーバーの設定後、Unity はアセットを再コンパイルします。
4. Unity のメニューで [ファイル] を選択し、[ビルド] を選択します。[サーバービルド] を選択し、[ビルド] を選択して、サーバーファイル専用のビルドフォルダーを選択します。

   Unity はサンプル ゲーム サーバーを構築し、実行可能ファイルと必要なアセットを指定されたビルドフォルダに配置します。

サンプルゲームクライアントを構築して実行する

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

サンプルゲームのゲームクライアントファイルをセットアップします。

1. Unity のメニューで Amazon GameLiftを選択し、Windows サンプルクライアント構築設定の適用 または MacOS サンプルクライアント構築設定の適用 を選択します。ゲーム クライアントの設定後、Unity はアセットを再コンパイルします。
2. Unity のメニューで、[Go To Client Settings] (クライアント設定に移動) を選択します。これにより、[Inspector] タブを Unity 画面の右側に表示します。Amazon GameLift クライアント設定タブで、ローカルテストモード を選択します。

3. ゲーム クライアントを構築します。Unity のメニューで [ファイル] を選択します。[サーバービルド] がチェックされていないことを確認し、[ビルド] を選択し、クライアントファイル専用のビルドフォルダーを選択します。

   Unity はサンプルゲームクライアントを構築し、実行可能ファイルと必要なアセットを指定されたクライアントビルドフォルダーに配置します。
4. ゲームサーバーとクライアントは構築していません。次のステップでは、ゲームを実行し、それが Amazon とどのように相互作用するかを確認します GameLift。

サンプルゲームをローカルでテストする

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

Amazon GameLift Local を使用してインポートしたサンプルゲームを実行します。

1. ゲームサーバーを起動します。Unity では、Unity 用プラグインタブで、[デプロイ] タブを選択します。
2. [テスト] ペインで、[ローカルテスト UI を開く] を選択します。
3. [Local Testing] (ローカルテスト) ウィンドウで、[Game Server .exe File Path] (ゲームサーバー .exe ファイルパス) を指定します。パスには実行可能ファイル名を含める必要があります。例えば、「`C:/MyGame/GameServer/MyGameServer.exe`」と入力します。
4. [デプロイと実行] を選択します。Unity 用プラグインはゲームサーバーを起動し、Amazon GameLift Local ログウィンドウを開きます。ウィンドウには、ゲームサーバーと Amazon GameLift Local の間で送信されるメッセージなどのログメッセージが含まれます。
5. ゲーム クライアントを起動します。サンプルゲームクライアントでビルドの場所を見つけ、実行ファイルを選択します。
6. Amazon GameLift サンプルゲーム で、E メールとパスワードを入力し、ログイン を選択します。E メールとパスワードは検証も使用もされていません。
7. Amazon GameLift サンプルゲーム で、 の開始 を選択します。ゲームクライアントはゲームセッションを検索します。セッションが見つからない場合は、セッションを作成します。その後、ゲームクライアントはゲームセッションを開始します。ゲームのアクティビティはログで確認できます。

ゲームサーバーログのサンプル

```
...
2021-09-15T19:55:3495 PID:20728 Log :) GAMELIFT AWAKE
2021-09-15T19:55:3512 PID:20728 Log :) I AM SERVER
2021-09-15T19:55:3514 PID:20728 Log :) GAMELIFT StartServer at port 33430.
2021-09-15T19:55:3514 PID:20728 Log :) SDK VERSION: 4.0.2
2021-09-15T19:55:3556 PID:20728 Log :) SERVER IS IN A GAMELIFT FLEET
2021-09-15T19:55:3577 PID:20728 Log :) PROCESSREADY SUCCESS.
2021-09-15T19:55:3577 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
...
2021-09-15T19:55:3634 PID:20728 Log :) GAMELOGIC AWAKE
2021-09-15T19:55:3635 PID:20728 Log :) GAMELOGIC START
2021-09-15T19:55:3636 PID:20728 Log :) LISTENING ON PORT 33430
2021-09-15T19:55:3636 PID:20728 Log SERVER: Frame: 0 HELLO WORLD!
...
2021-09-15T19:56:2464 PID:20728 Log :) GAMELIFT SESSION REQUESTED
2021-09-15T19:56:2468 PID:20728 Log :) GAME SESSION ACTIVATED
2021-09-15T19:56:3578 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
2021-09-15T19:57:3584 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
2021-09-15T19:58:0334 PID:20728 Log SERVER: Frame: 8695 Connection accepted: playerIdx
 0 joined
2021-09-15T19:58:0335 PID:20728 Log SERVER: Frame: 8696 Connection accepted: playerIdx
 1 joined
2021-09-15T19:58:0338 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from playerIdx 0 Msg:
 CONNECT: server IP localhost
2021-09-15T19:58:0338 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from player 0:CONNECT:
 server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 CONNECT: player index 0
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from playerIdx 1 Msg:
 CONNECT: server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from player 1:CONNECT:
 server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 CONNECT: player index 1
```

Amazon GameLift Local ログのサンプル

```
12:55:26,000  INFO || - [SocketIOServer] main - Session store / pubsub factory used:
 MemoryStoreFactory (local session store only)
12:55:28,092  WARN || - [ServerBootstrap] main - Unknown channel option 'SO_LINGER' for
 channel '[id: 0xe23d0a14]'
12:55:28,101  INFO || - [SocketIOServer] nioEventLoopGroup-2-1 - SocketIO server
 started at port: 5757
```

```
12:55:28,101  INFO || - [SDKConnection] main - GameLift SDK server (communicates with
 your game server) has started on http://localhost:5757
12:55:28,120  INFO || - [SdkWebSocketServer] WebSocketSelector-20 - WebSocket Server
 started on address localhost/127.0.0.1:5759
12:55:28,166  INFO || - [StandAloneServer] main - GameLift Client server (listens for
 GameLift client APIs) has started on http://localhost:8080
12:55:28,179  INFO || - [StandAloneServer] main - GameLift server sdk http listener has
 started on http://localhost:5758
12:55:35,453  INFO || - [SdkWebSocketServer] WebSocketWorker-12 - onOpen
 socket: /?pID=20728&sdkVersion=4.0.2&sdkLanguage=CSharp and handshake /?
pID=20728&sdkVersion=4.0.2&sdkLanguage=CSharp
12:55:35,551  INFO || - [HostProcessManager] WebSocketWorker-12 - client connected with
 pID 20728
12:55:35,718  INFO || - [GameLiftSdkHttpHandler] GameLiftSdkHttpHandler-thread-0 -
 GameLift API to use: ProcessReady for pId 20728
12:55:35,718  INFO || - [ProcessReadyHandler] GameLiftSdkHttpHandler-thread-0 -
 Received API call for processReady from 20728
12:55:35,738  INFO || - [ProcessReadyHandler] GameLiftSdkHttpHandler-thread-0 -
 onProcessReady data: port: 33430
 12:55:35,739  INFO || - [HostProcessManager] GameLiftSdkHttpHandler-thread-0 -
 Registered new process with pId 20728
12:55:35,789  INFO || - [GameLiftSdkHttpHandler] GameLiftSdkHttpHandler-thread-0 -
 GameLift API to use: ReportHealth for pId 20728
12:55:35,790  INFO || - [ReportHealthHandler] GameLiftSdkHttpHandler-thread-0 -
 Received API call for ReportHealth from 20728
12:55:35,794  INFO || - [ReportHealthHandler] GameLiftSdkHttpHandler-thread-0 -
 ReportHealth data: healthStatus: true
 12:56:24,098  INFO || - [GameLiftHttpHandler] Thread-12 - API to use:
 GameLift.DescribeGameSessions
12:56:24,119  INFO || - [DescribeGameSessionsDispatcher] Thread-12 - Received API call
 to describe game sessions with input: {"FleetId":"fleet-123"}
12:56:24,241  INFO || - [GameLiftHttpHandler] Thread-12 - API to use:
 GameLift.CreateGameSession
12:56:24,242  INFO || - [CreateGameSessionDispatcher] Thread-12 - Received API call to
 create game session with input: {"FleetId":"fleet-123","MaximumPlayerSessionCount":4}
12:56:24,265  INFO || - [HostProcessManager] Thread-12 - Reserved process:
 20728 for gameSession: arn:aws:gamelift:local::gamesession/fleet-123/
gsess-59f6cc44-4361-42f5-95b5-fdb5825c0f3d
12:56:24,266  INFO || - [WebSocketInvoker] Thread-12 - StartGameSessionRequest:
 gameSessionId=arn:aws:gamelift:local::gamesession/fleet-123/
gsess-59f6cc44-4361-42f5-95b5-fdb5825c0f3d, fleetId=fleet-123, gameSessionName=null,
 maxPlayers=4, properties=[], ipAddress=127.0.0.1, port=33430, gameSessionData?=false,
 matchmakerData?=false, dnsName=localhost
```

```
12:56:24,564  INFO || - [CreateGameSessionDispatcher] Thread-12 - GameSession with
 id: arn:aws:gamelift:local::gamesession/fleet-123/gsess-59f6cc44-4361-42f5-95b5-
fdb5825c0f3d created
12:56:24,585  INFO || - [GameLiftHttpHandler] Thread-12 - API to use:
 GameLift.DescribeGameSessions
12:56:24,585  INFO || - [DescribeGameSessionsDispatcher] Thread-12 - Received API call
 to describe game sessions with input: {"FleetId":"fleet-123"}
12:56:24,660  INFO || - [GameLiftSdkHttpHandler] GameLiftSdkHttpHandler-thread-0 -
 GameLift API to use: GameSessionActivate for pId 20728
12:56:24,661  INFO || - [GameSessionActivateHandler] GameLiftSdkHttpHandler-thread-0 -
 Received API call for GameSessionActivate from 20728
12:56:24,678  INFO || - [GameSessionActivateHandler] GameLiftSdkHttpHandler-thread-0
 - GameSessionActivate data: gameSessionId: "arn:aws:gamelift:local::gamesession/
fleet-123/gsess-59f6cc44-4361-42f5-95b5-fdb5825c0f3d"
```

サーバープロセスをシャットダウンする

> **Note**
>
> このトピックでは、サーバー SDK 4.x 以前を使用する Unity バージョン 1.0.0 用の Amazon GameLift プラグインについて説明します。

サンプルゲームを終了したら、Unity のサーバーをシャットダウンします。

1. ゲーム クライアントで、[Quit] (終了) を選択するか、ウィンドウを閉じて、ゲーム クライアントを停止します。
2. Unityでは、[Local Testing] (ローカルテスト) ウィンドウで [Stop] (停止) を選択するか、ゲーム サーバー Windowsを閉じて、サーバーを停止します。

Unreal Engine 用 Amazon GameLift プラグインとのゲームの統合

このセクションのトピックでは、Unreal Engine (UE) 用 Amazon GameLift プラグインと、それを使用して Amazon でホストするマルチプレイヤーゲームプロジェクトを準備する方法について説明します GameLift。プラグインのガイド付きワークフローを使用して UE 開発環境で完全に作業し、Amazon でホストするための基本要件を完了します GameLift。

Amazon GameLift は、ゲームデベロッパーがセッションベースのマルチプレイヤーゲーム専用のゲームサーバーを管理およびスケーリングできるようにするフルマネージドサービスです。Amazon GameLift ホスティングの詳細については、「」を参照してくださいAmazon GameLift の仕組み。

トピック

プラグインについて

このプラグインは、Amazon GameLift ツールと機能を UE エディタに追加します。Amazon GameLift をゲームプロジェクトに統合し、テスト用のローカルホストとしてワークステーションを指定し、ゲームサーバーを Amazon GameLift クラウドホスティングにデプロイするためのプラグインのガイド付きワークフロー。

プラグインの事前構築済みのホスティングソリューションを使用してゲームをデプロイします。ローカルワークステーションをホストとして Amazon GameLift Anywhere フリートを設定します。クラウドホスティングの場合は、プレイヤーのレイテンシー、ゲームセッションの可用性、コストをさまざまな方法でバランスを取る 2 つの一般的なデプロイシナリオから選択します。1 つのシナリオには、シンプルな FlexMatchマッチメーカーとルールセットが含まれます。これらのソリューションを使用して、本番環境に対応したホスティング構造をすぐに導入し、必要に応じて最適化とカスタマイズを行います。

このプラグインには以下のコンポーネントが含まれています。

- UE エディター用のプラグインモジュール。プラグインをインストールすると、新しいメインメニューボタンで Amazon GameLift の機能にアクセスできます。
- クライアント側の機能を備えた Amazon GameLift サービス API 用の C++ ライブラリ。
- Amazon GameLift サーバー SDK (バージョン 5) 用の Unreal ライブラリ。
- テスト用のコンテンツ。スタートアップゲームマップと、サーバー統合のテストに使用する基本的なブループリントと UI 要素を含む 2 つのテストマップが含まれます。
- プラグインがホスティング用にゲームサーバーをデプロイする際に使用する、AWS CloudFormation テンプレート形式の編集可能な設定。

プラグインワークフロー

次のステップでは、Unreal Engine 用 Amazon GameLift プラグインとゲームプロジェクトを統合してデプロイするための一般的なアプローチについて説明します。これらのステップを完了するには、UE エディタとゲームコードを使って作業します。

1. AWS アカウントにリンクし、Amazon を使用するアクセス許可を持つ有効なアカウントユーザーのアクセス認証情報を提供するユーザープロファイルを作成します GameLift。
2. ゲームプロジェクトにサーバーコードを追加して、実行中のゲームサーバーと Amazon GameLift サービスとの通信を確立します。
3. ゲームプロジェクトにクライアントコードを追加して、ゲームクライアントが新しいゲームセッションを開始して接続 GameLift するためのリクエストを Amazon に送信できるようにします。
4. Anywhere ワークフローを使用して、ローカルワークステーションをゲームサーバーの Anywhere ホストとして設定します。プラグインを使用してゲームサーバーとクライアントをローカルで起動し、ゲームセッションに接続して、統合をテストします。
5. EC2 ホスティングワークフローを使用して、統合ゲームサーバーをアップロードし、クラウドホスティングソリューションをデプロイします。ゲームサーバーの準備ができたら、プラグインを使用してゲームクライアントをローカルで起動し、ゲームセッションに接続してゲームをプレイします。

プラグインで作業するときは、 AWSリソースを作成して使用します。これらのアクションでは、使用中のAWSアカウントに料金が発生する可能性があります。を初めて使用する場合AWS、これらのアクションは AWS無料利用枠 の対象になる場合があります。

Unreal 用プラグインをインストールする

このセクションでは、Unreal Engine プロジェクトにプラグインを追加する初期インストールタスクについて説明します。プラグイン機能は、Unreal Editor でプロジェクトを開いている場合に使用できます。

> **Note**
>
> Amazon GameLift プラグインは UE エディタの標準バージョンで使用できますが、ゲームサーバービルドをパッケージ化するときにソースビルドバージョンを使用する必要があります。

開始する前に

Unreal Engine 用 Amazon GameLift プラグインを使用するために必要なものは次のとおりです。

- Unreal Engine リリースパッケージ用の Amazon GameLift プラグイン。[ダウンロードサイト]。
- Microsoft Visual Studio 2019 以降。
- Unreal Engine エディタのソースビルドバージョン。マルチプレイヤーゲームのサーバーコンポーネントをパッケージ化するには、ソースビルドバージョンが必要です。その他の前提条件を含む詳細については、Unreal Engine のドキュメントを参照してください。
  - の Unreal Engine ソースコードにアクセスするには GitHub 、 と エピックゲームアカウントが必要です GitHub。
  - 「ソースからの Unreal Engine の構築」チュートリアル。
- C++ ゲームコードを使ったマルチプレイヤーゲームプロジェクト。ブループリントプロジェクトを使用している場合は、プロジェクトの C++ ソースコードを生成する方法に関する Unreal のドキュメントを参照してください。

プラグインをゲームプロジェクトに追加する

ゲームプロジェクトにプラグインを追加するには、次のタスクを実行します。

Amazon GameLift C++ サーバー SDK を構築する

1. Amazon GameLift plugin for Unreal Engine リリースパッケージを解凍して、2 つの zip ファイルを抽出します。

   - `amazon-gamelift-plugin-unreal-<>-sdk-<>.zip`
   - `GameLift-Cpp-ServerSDK-<>.zip.`

   これらのファイルを解凍します。
2. `GameLift-Cpp-ServerSDK-<>` フォルダを開き、プラットフォームの Linux または Microsoft Windows の手順を完了します。

Linux

1. 以下のコマンドを実行します。

```
mkdir out
cd out
cmake -DBUILD_FOR_UNREAL=1 ..
make
```

これらのコマンドは/lib/aws-cpp-sdk-gamelift-server.so、 ファイルを構築します。

2. /lib/aws-cpp-sdk-gamelift-server.so を amazon-gamelift-plugin-unreal/GameLiftPlugin/Source/GameliftServer/ThirdParty/GameLiftServerSDK/Linux/x86_64-unknown-linux-gnu/ ディレクトリにコピーします。

Microsoft Windows

1. 以下のコマンドを実行します。

```
mkdir out
cd out
cmake -G "Visual Studio 17 2022" -DBUILD_FOR_UNREAL=1 ..
msbuild ALL_BUILD.vcxproj /p:Configuration=Release
```

これらのコマンドは、次のバイナリファイルをビルドします。

- prefix\bin\aws-cpp-sdk-gamelift-server.dll
- prefix\lib\aws-cpp-sdk-gamelift-server.lib

2. ファイルを amazon-gamelift-plugin-unreal\GameLiftPlugin\Source\GameliftServer\ThirdParty\GameLiftServerSDK\Win64\ ディレクトリにコピーします。

ゲームプロジェクトファイルで作業し、以下のタスクを実行します。

1. プラグインをインストールします。

   a. ゲームプロジェクトのルートフォルダ (... > Unreal Projects/[project-name]/ など) を探します。Plugins フォルダが存在しない場合は、作成します。

   b. から解凍したamazon-gamelift-plugin-unrealフォルダに移動しますamazon-gamelift-plugin-unreal-<>-sdk-<>.zip。GameLiftPlugin フォルダからゲーム

プロジェクトディレクトリ内のPluginsフォルダにgamelift-plugin-unrealフォルダをコピーします。

2. プラグインを **.uproject** ファイルに追加します。

   a. ゲームプロジェクトのルートフォルダで、.uproject ファイルを開きます。

   b. ファイルを更新してGameLiftPlugin「」とWebBrowserWidget「」を Pluginsセクションに追加し、有効にします。次のコードは、「」というゲームの更新.uprojectファイルを示していますMyGame。

```
UnrealProjects > MyGame > MyGame.uproject
{
  ...
  "Plugins": [
    {
      "Name": "ModelingToolsEditorMode",
      "Enabled": true,
      "TargetAllowList": [ "Editor" ]
    },
    {
      "Name": "GameLiftPlugin",
      "Enabled": true
    },
    {
      "Name": "WebBrowserWidget",
      "Enabled": true
    }
  ]
}
```

3. プロジェクトの UE エディタのバージョンを変更します。

   あるエディタバージョン用のプロジェクトを作成した後に、別のバージョン (ソースビルドバージョンなど) に変更したい場合は、プロジェクトを更新する必要があります。

   ゲームプロジェクトのルートフォルダで .uproject ファイルを選択し、[Unreal Engine バージョンの切り替え] オプションを選択します。新しいエディタバージョンを選択します。

4. 更新内容でプロジェクトソリューションを再構築します。

a. プロジェクトルートフォルダで、ソリューション (`*.sln`) ファイルを探します。存在しない場合は、`.uproject` ファイルを選択して [Visual Studio プロジェクトファイルを生成する] オプションを選択します。

b. ソリューションファイルを開き、プロジェクトを構築または再構築します。

5. UE エディタでプラグインが有効になっていることを確認します。

> **Note**
>
> エディタを既に開いている場合は、新しいプラグインを認識する前にエディタを再起動しなければならない場合があります。

a. 選択した UE エディタでプロジェクトを開きます。

b. メインエディタのツールバーで、新しい Amazon GameLift メニューボタン [必要な画像] を確認します。

c. コンテンツブラウザで Amazon GameLift プラグインアセットを確認します。[表示オプション] 設定で [プラグインコンテンツを表示] オプションが選択されていることを確認します。

AWS ユーザープロファイルを設定します。

プラグインをインストールしたら、プロファイルを設定して有効な AWS アカウントユーザーにリンクします。複数のプロファイルを保持できますが、一度にアクティブにできるプロファイルは 1 つに限られます。プラグインで作業するときはいつでも、使用するプロファイルを選択してください。

複数のプロファイルを管理することで、異なるホスティングシナリオ間で切り替えることができます。例えば、同じ AWS 認証情報で異なる AWS リージョンのプロファイルを設定することができます。または、異なる AWS アカウントや異なるユーザー/アクセス許可のセットでプロファイルを設定することもできます。

> **Note**
>
> ワークステーションに CLI AWS をインストールし、プロファイルがすでに設定されている場合、Amazon GameLift プラグインはそれを検出して既存のプロファイルとして一覧表示できます。プラグインは `[default]` という名前のプロファイルを自動的に選択します。既存のプロファイルを使用するか、新しいプロファイルを作成できます。

AWS プロファイルを管理するには

1. Unreal エディタのメインツールバーで、Amazon GameLift メニューを選択し、AWSユーザープロファイルの設定を選択します。このアクションにより、プラグインの [プロジェクト設定] が開きます。[AWS ユーザープロファイル] セクションを展開します。
2. プラグインが既存のプロファイルを検出しない場合、プロファイルを作成するよう求められます。新しいプロファイルは、新しいアカウントでも既存の AWS アカウントでも作成できます。

   > **Note**
   >
   > AWS マネジメントコンソールを使用して新しい AWS アカウントを作成し、適切なアクセス許可セットを持つユーザーを作成または更新する必要があります。

   プロファイルをセットアップするには、次の情報が必要です。

   - AWS アカウント。新規 AWS アカウントを作成する必要がある場合、プロンプトに従ってアカウントを作成します。詳細については、「AWS アカウントを作成する」を参照してください。
   - Amazon GameLift およびその他の必要な AWSのサービスを使用するアクセス許可を持つ AWS ユーザー。Amazon アクセス GameLift 許可と長期的な認証情報を使用したプログラムによるアクセスを持つ AWS Identity and Access Management (IAM) ユーザーを設定する手順のセットアップ AWS アカウントについては、「」を参照してください。
   - AWS ユーザーの認証情報。これらの認証情報は、AWS アクセスキー IDおよび AWS シークレットアクセスキーで構成されています。詳細については、「アクセスキーを取得する」を参照してください。
   - AWS リージョン。これは、ホスティング用の AWS リソースを作成する地理的なロケーションです。開発中は、実際のロケーションに近い地域を使用することをおすすめします。サポートされている AWS リージョンのリストを参照してください。
3. プラグインが既存のプロファイルを検出した場合、プロファイルを作成するように求められません。プロファイルを更新したり、新しいプロファイルを作成したりする場合は、[プロファイルを管理する] を選択します。

プロファイルをブートストラップするには

Amazon GameLift プラグインで使用するには、すべてのプロファイルをブートストラップする必要があります。ブートストラッピングによって、プロファイル固有の Amazon S3 バケットが作成され

ます。プロジェクト設定、ビルドアーティファクト、およびその他の依存関係を保存するために使用されます。バケットは他のプロファイルとは共有されません。

ブートストラップには新しい AWS リソースの作成が必要で、コストが発生する可能性があります。

1. Unreal エディタのメインツールバーで、Amazon GameLift アイコンを選択し、AWSユーザープロファイルの設定を選択します。このアクションにより、プラグインの [プロジェクト設定] が開きます。[AWS ユーザープロファイル] セクションを展開します。
2. [プロファイルをブートストラップする] セクションで、ドロップダウンリストからプロファイルを選択し、ブートストラップのステータスを確認します。ステータスにバケットが存在しないことが示されている場合は、[ブートストラップしてプロファイルを作成] ボタンを選択し、選択したプロファイルの Amazon S3 バケットを作成します。
3. ブートストラップのステータスが「アクティブ」に変わるまでお待ちください。これは数分かかることがあります。

Amazon でのテスト用にゲームをセットアップする GameLift Anywhere

このワークフローでは、Amazon GameLift 機能のクライアントとサーバーのゲームコードを追加し、 プラグインを使用してローカルワークステーションをテストゲームサーバーホストとして指定します。統合タスクが完了したら、プラグインを使用してゲームクライアントとサーバーのコンポーネントを構築します。

Amazon GameLift Anywhere ワークフローを開始するには：

- Unreal エディタのメインツールバーで Amazon GameLift メニューを選択し、Anywhere でホストを選択します。このアクションにより [Anywhere のデプロイ] というプラグインページが開き、ゲームコンポーネントを統合、ビルド、起動するための 6 つのステップのプロセスが表示されます。

ステップ 1: プロフィールを設定する

このワークフローに従うときに使用したいプロファイルを選択します。選択したプロファイルは、ワークフローのすべてのステップに影響します。作成したすべてのリソースはプロファイルのAWSアカウントに関連付けられ、プロファイルのデフォルトAWSリージョンに配置されます。プロファイルユーザーのアクセス許可によって、AWSリソースとアクションへのアクセスが決まります。

1. 使用可能なプロファイルのドロップダウンリストからプロファイルを選択します。まだプロファイルを持っていない場合、または新しいプロファイルを作成する場合は、Amazon GameLift メニューに移動し、AWS「ユーザープロファイルの設定」を選択します。
2. ブートストラップステータスが「アクティブ」でない場合は、ブートストラッププロファイルを選択し、ステータスが「アクティブ」に変わるのを待ちます。

ステップ2: ゲームコードをセットアップする

このステップでは、クライアントとサーバーのコードに一連の更新を実施し、ホスティング機能を追加します。Unreal エディタのソースビルドバージョンをまだ設定していない場合、プラグインは手順とソースコードへのリンクを提供します。

このプラグインを使用すると、ゲームコードを統合する際にいくつかの便利な機能を活用できます。最小限の統合で基本的なホスティング機能を設定できます。より広範囲なカスタム統合を行うこともできます。このセクションでは、最小限の統合オプションについて説明します。プラグインに含まれているテストマップを使用して、クライアントの Amazon GameLift 機能をゲームプロジェクトに追加します。サーバー統合の場合は、提供されているコードサンプルを使用してプロジェクトのゲームモードを更新してください。

サーバーゲームモードを統合する

ゲームサーバーと Amazon GameLift サービス間の通信を可能にするサーバーコードをゲームに追加します。ゲームサーバーは、新しいゲームセッションを開始するなど GameLift、Amazon からのリクエストに応答できる必要があります。また、ゲームサーバーのヘルスとプレイヤー接続のステータスも報告できる必要があります。

1. コードエディタで、ゲームプロジェクトのソリューション (`.sln`) ファイルを開きます。通常はプロジェクトのルートフォルダにあります。例: `GameLiftUnrealApp.sln`。
2. ソリューションを開いた状態で、プロジェクトのゲームモードヘッダーファイル: `[project-name]GameMode.h` ファイルを探します。例: `GameLiftUnrealAppGameMode.h`。
3. ヘッダーファイルを次のサンプルコードに合わせて変更します。必ずGameLiftServer「」を独自のプロジェクト名に置き換えてください。これらの更新はゲームサーバー固有のものです。クライアントで使用するために、元のゲームモードファイルのバックアップコピーを作成することをお勧めします。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/GameModeBase.h"
#include "GameLiftServerGameMode.generated.h"

struct FProcessParameters;

DECLARE_LOG_CATEGORY_EXTERN(GameServerLog, Log, All);

UCLASS(minimalapi)
class AGameLiftServerGameMode : public AGameModeBase
{
    GENERATED_BODY()

public:
    AGameLiftServerGameMode();

protected:
    virtual void BeginPlay() override;

private:
    void InitGameLift();

private:
    TSharedPtr<FProcessParameters> ProcessParameters;
};
```

4. 関連するソースファイルの [project-name]GameMode.cpp ファイル (GameLiftUnrealAppGameMode.cpp など) を開きます。コードを次のサンプルコードに合わせて変更します。必ずGameLiftUnrealApp「」を独自のプロジェクト名に置き換えてください。これらの更新はゲームサーバー固有のものです。クライアントで使用するために、元のファイルのバックアップコピーを作成することをお勧めします。

   次のサンプルコードは、Amazon とのサーバー統合に最低限必要な要素を追加する方法を示しています GameLift。

   - Amazon GameLift API クライアントを初期化します。Amazon GameLift Anywhere フリートには、サーバーパラメータを使用したInitSDK()呼び出しが必要です。Anywhere フリートに接続すると、プラグインはサーバーパラメータをコンソール引数として保存します。サンプルコードはランタイム時に値にアクセスできます。

- OnStartGameSession、、 OnProcessTerminateなど、Amazon GameLift サービスからのリクエストに応答するために必要なコールバック関数を実装しますonHealthCheck。
- ゲームセッションをホストする準備ができたら、指定されたポートProcessReady()で を呼び出して Amazon GameLift サービスに通知します。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

#include "GameLiftServerGameMode.h"

#include "UObject/ConstructorHelpers.h"
#include "Kismet/GameplayStatics.h"

#if WITH_GAMELIFT
#include "GameLiftServerSDK.h"
#include "GameLiftServerSDKModels.h"
#endif

#include "GenericPlatform/GenericPlatformOutputDevices.h"

DEFINE_LOG_CATEGORY(GameServerLog);

AGameLiftServerGameMode::AGameLiftServerGameMode() :
    ProcessParameters(nullptr)
{
    // Set default pawn class to our Blueprinted character
    static ConstructorHelpers::FClassFinder<APawn> PlayerPawnBPClass(TEXT("/Game/
ThirdPerson/Blueprints/BP_ThirdPersonCharacter"));

    if (PlayerPawnBPClass.Class != NULL)
    {
        DefaultPawnClass = PlayerPawnBPClass.Class;
    }

    UE_LOG(GameServerLog, Log, TEXT("Initializing AGameLiftServerGameMode..."));
}

void AGameLiftServerGameMode::BeginPlay()
{
    Super::BeginPlay();
```

```
#if WITH_GAMELIFT
    InitGameLift();
#endif
}

void AGameLiftServerGameMode::InitGameLift()
{
#if WITH_GAMELIFT
    UE_LOG(GameServerLog, Log, TEXT("Calling InitGameLift..."));

    // Getting the module first.
    FGameLiftServerSDKModule* GameLiftSdkModule =
 &FModuleManager::LoadModuleChecked<FGameLiftServerSDKModule>(FName("GameLiftServerSDK"));

    //Define the server parameters for a GameLift Anywhere fleet. These are not
 needed for a GameLift managed EC2 fleet.
    FServerParameters ServerParametersForAnywhere;

    bool bIsAnywhereActive = false;
    if (FParse::Param(FCommandLine::Get(), TEXT("glAnywhere")))
    {
        bIsAnywhereActive = true;
    }

    if (bIsAnywhereActive)
    {
        UE_LOG(GameServerLog, Log, TEXT("Configuring server parameters for
 Anywhere..."));

        // If GameLift Anywhere is enabled, parse command line arguments and pass
 them in the ServerParameters object.
        FString glAnywhereWebSocketUrl = "";
        if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereWebSocketUrl="),
 glAnywhereWebSocketUrl))
        {
            ServerParametersForAnywhere.m_webSocketUrl =
 TCHAR_TO_UTF8(*glAnywhereWebSocketUrl);
        }

        FString glAnywhereFleetId = "";
        if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereFleetId="),
 glAnywhereFleetId))
        {
```

```
            ServerParametersForAnywhere.m_fleetId =
TCHAR_TO_UTF8(*glAnywhereFleetId);
        }

        FString glAnywhereProcessId = "";
        if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereProcessId="),
glAnywhereProcessId))
        {
            ServerParametersForAnywhere.m_processId =
TCHAR_TO_UTF8(*glAnywhereProcessId);
        }
        else
        {
            // If no ProcessId is passed as a command line argument, generate a
randomized unique string.
            ServerParametersForAnywhere.m_processId =
                TCHAR_TO_UTF8(
                    *FText::Format(
                        FText::FromString("ProcessId_{0}"),
                        FText::AsNumber(std::time(nullptr))
                    ).ToString()
                );
        }

        FString glAnywhereHostId = "";
        if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereHostId="),
glAnywhereHostId))
        {
            ServerParametersForAnywhere.m_hostId =
TCHAR_TO_UTF8(*glAnywhereHostId);
        }

        FString glAnywhereAuthToken = "";
        if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereAuthToken="),
glAnywhereAuthToken))
        {
            ServerParametersForAnywhere.m_authToken =
TCHAR_TO_UTF8(*glAnywhereAuthToken);
        }

        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_YELLOW);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Web Socket URL: %s"),
*ServerParametersForAnywhere.m_webSocketUrl);
```

```
        UE_LOG(GameServerLog, Log, TEXT(">>>> Fleet ID: %s"),
 *ServerParametersForAnywhere.m_fleetId);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Process ID: %s"),
 *ServerParametersForAnywhere.m_processId);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Host ID (Compute Name): %s"),
 *ServerParametersForAnywhere.m_hostId);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Auth Token: %s"),
 *ServerParametersForAnywhere.m_authToken);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }

    UE_LOG(GameServerLog, Log, TEXT("Initializing the GameLift Server..."));

    //InitSDK will establish a local connection with GameLift's agent to enable
 further communication.
    FGameLiftGenericOutcome InitSdkOutcome = GameLiftSdkModule-
>InitSDK(ServerParametersForAnywhere);
    if (InitSdkOutcome.IsSuccess())
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_GREEN);
        UE_LOG(GameServerLog, Log, TEXT("GameLift InitSDK succeeded!"));
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }
    else
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_RED);
        UE_LOG(GameServerLog, Log, TEXT("ERROR: InitSDK failed : ("));
        FGameLiftError GameLiftError = InitSdkOutcome.GetError();
        UE_LOG(GameServerLog, Log, TEXT("ERROR: %s"),
 *GameLiftError.m_errorMessage);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
        return;
    }

    ProcessParameters = MakeShared<FProcessParameters>();

    //When a game session is created, GameLift sends an activation request to the
 game server and passes along the game session object containing game properties
 and other settings.
    //Here is where a game server should take action based on the game session
 object.
    //Once the game server is ready to receive incoming player connections, it
 should invoke GameLiftServerAPI.ActivateGameSession()
```

```
    ProcessParameters->OnStartGameSession.BindLambda([=]
(Aws::GameLift::Server::Model::GameSession InGameSession)
    {
        FString GameSessionId = FString(InGameSession.GetGameSessionId());
        UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"),
 *GameSessionId);
        GameLiftSdkModule->ActivateGameSession();
    });

    //OnProcessTerminate callback. GameLift will invoke this callback before
 shutting down an instance hosting this game server.
    //It gives this game server a chance to save its state, communicate with
 services, etc., before being shut down.
    //In this case, we simply tell GameLift we are indeed going to shutdown.
    ProcessParameters->OnTerminate.BindLambda([=]()
    {
        UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
        GameLiftSdkModule->ProcessEnding();
    });

    //This is the HealthCheck callback.
    //GameLift will invoke this callback every 60 seconds or so.
    //Here, a game server might want to check the health of dependencies and such.
    //Simply return true if healthy, false otherwise.
    //The game server has 60 seconds to respond with its health status. GameLift
 will default to 'false' if the game server doesn't respond in time.
    //In this case, we're always healthy!
    ProcessParameters->OnHealthCheck.BindLambda([]()
    {
        UE_LOG(GameServerLog, Log, TEXT("Performing Health Check"));
        return true;
    });

    //GameServer.exe -port=7777 LOG=server.mylog
    ProcessParameters->port = FURL::UrlConfig.DefaultPort;
    TArray<FString> CommandLineTokens;
    TArray<FString> CommandLineSwitches;

    FCommandLine::Parse(FCommandLine::Get(), CommandLineTokens,
 CommandLineSwitches);

    for (FString SwitchStr : CommandLineSwitches)
    {
        FString Key;
```

```
        FString Value;

        if (SwitchStr.Split("=", &Key, &Value))
        {
            if (Key.Equals("port"))
            {
                ProcessParameters->port = FCString::Atoi(*Value);
            }
        }
    }

    //Here, the game server tells GameLift where to find game session log files.
    //At the end of a game session, GameLift uploads everything in the specified
    //location and stores it in the cloud for access later.
    TArray<FString> Logfiles;
    Logfiles.Add(TEXT("GameServerLog/Saved/Logs/GameServerLog.log"));
    ProcessParameters->logParameters = Logfiles;

    //The game server calls ProcessReady() to tell GameLift it's ready to host game
 sessions.
    UE_LOG(GameServerLog, Log, TEXT("Calling Process Ready..."));
    FGameLiftGenericOutcome ProcessReadyOutcome = GameLiftSdkModule-
>ProcessReady(*ProcessParameters);

    if (ProcessReadyOutcome.IsSuccess())
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_GREEN);
        UE_LOG(GameServerLog, Log, TEXT("Process Ready!"));
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }
    else
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_RED);
        UE_LOG(GameServerLog, Log, TEXT("ERROR: Process Ready Failed!"));
        FGameLiftError ProcessReadyError = ProcessReadyOutcome.GetError();
        UE_LOG(GameServerLog, Log, TEXT("ERROR: %s"),
 *ProcessReadyError.m_errorMessage);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }

    UE_LOG(GameServerLog, Log, TEXT("InitGameLift completed!"));
#endif
}
```

クライアントゲームマップを統合する

スタートアップゲームマップには、ゲームセッションをリクエストしたり、接続情報を使用してゲームセッションに接続するための基本コードが既に含まれているブループリントロジックと UI 要素が含まれています。このマップはそのまま使用することも、必要に応じて変更することもできます。スタートアップ時のゲームマップは、Unreal Engine が提供する Third Person テンプレートプロジェクトなどの他のゲームアセットと一緒に使用してください。これらのアセットはコンテンツブラウザで使用できます。これらを使用してプラグインのデプロイワークフローをテストしたり、ゲームのカスタムバックエンドサービスを作成するためのガイドとして使用したりできます。

このスタートアップマップには以下の特徴があります。

- Anywhere フリートとマネージド EC2 フリートの両方のロジックが含まれています。クライアントを実行するときに、どちらかのフリートに接続することを選択できます。
- クライアント機能には、ゲームセッションの検索 (SearchGameSessions())、新しいゲームセッションの作成 (CreateGameSession())、ゲームセッションの直接参加が含まれます。
- プロジェクトの Amazon Cognito ユーザープールから一意のプレイヤー ID を取得します (これはデプロイされた Anywhere ソリューションの一部です)。

スタートアップゲームマップを使用するには

1. UE エディタで、[プロジェクト設定、マップ & モード] ページを開き、[デフォルトマップ] セクションを展開します。
2. Editor Startup Map で、ドロップダウンリストからStartupMap「」を選択します。`... > Unreal Projects/[project-name]/Plugins/Amazon GameLift Plugin Content/Maps` にあるファイルを検索しなければいけない場合があります。
3. ゲームのデフォルトマップ では、ドロップダウンリストから同じStartupMap「」を選択します。
4. サーバーデフォルトマップ で、ThirdPersonMap「」を選択します。これはゲームプロジェクトに含まれるデフォルトのマップです。このマップはゲーム内の 2 人のプレイヤー向けに設計されています。
5. サーバーのデフォルトマップの詳細パネルを開きます。GameMode Override を「None」に設定します。

6. [デフォルトモード] セクションを展開し、[グローバルデフォルトサーバーゲームモード] をサーバー統合用に更新したゲームモードに設定します。

プロジェクトにこれらの変更を加えたら、ゲームコンポーネントを構築する準備が整います。

ゲームコンポーネントを構築する

1. 新しいサーバーとクライアントのターゲットファイルを作成する

   a. ゲームプロジェクトフォルダで、ソースフォルダに移動し、Target.cs ファイルを見つけます。

   b. [project-name]Editor.Target.cs ファイルを、[project-name]Client.Target.cs および [project-name]Server.Target.cs という名前の 2 つの新しいファイルにコピーします。

   c. 次に示すように、新しいファイルをそれぞれ編集して、クラス名とターゲットタイプの値を更新します。

```
UnrealProjects > MyGame > Source > MyGameClient.Target.cs
// Copyright Epic Games, Inc. All Rights Reserved.

using UnrealBuildTool;
using System.Collections.Generic;

public class MyGameClientTarget : TargetRules
{
  public MyGameClientTarget(TargetInfo Target) : base(Target)
  {
    Type = TargetType.Client;
     DefaultBuildSettings = BuildSettingsVersion.V2;
    IncludeOrderVersion = EngineIncludeOrderVersion.Unreal5_1;
     ExtraModuleNames.Add("MyGame");
  }
}
```

```
UnrealProjects > MyGame > Source > MyGameServer.Target.cs
// Copyright Epic Games, Inc. All Rights Reserved.

using UnrealBuildTool;
using System.Collections.Generic;
```

```
public class MyGameServerTarget : TargetRules
{
  public MyGameServerTarget(TargetInfo Target) : base(Target)
  {
    Type = TargetType.Server;
    DefaultBuildSettings = BuildSettingsVersion.V2;
    IncludeOrderVersion = EngineIncludeOrderVersion.Unreal5_1;
    ExtraModuleNames.Add("MyGame");
  }
}
```

2. .Build.cs ファイルを更新します。

   a. プロジェクトの .Build.cs ファイルを開きます。このファイルは UnrealProjects/[project name]/Source/[project name]/[project name].Build.cs にあります。

   b. 次のコードサンプルに示すように、ModuleRules クラスを更新します。

```
public class MyGame : ModuleRules
{
  public GameLiftUnrealApp(TargetInfo Target)
  {
    PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject",
 "Engine", "InputCore" });
    bEnableExceptions = true;

    if (Target.Type == TargetRules.TargetType.Server)
    {
      PublicDependencyModuleNames.AddRange(new string[]
 { "GameLiftServerSDK" });
      PublicDefinitions.Add("WITH_GAMELIFT=1");
    }
    else
    {
      PublicDefinitions.Add("WITH_GAMELIFT=0");
    }
  }
}
```

3. ゲームプロジェクトソリューションを再構築します。

4. Unreal Engine エディタのソースビルドバージョンでゲームプロジェクトを開きます。

5. クライアントとサーバーの両方で以下の操作を実行します。

    a. ターゲットを選択します。[プラットフォーム、Windows] に移動し、次のいずれかを選択します。

        - サーバー: [your-application-name]Server
        - クライアント: [your-application-name]Client

    b. ビルドを開始します [プラットフォーム、Windows、パッケージプロジェクト] に移動します。

パッケージ化処理を行うたびに、[your-application-name]Client.exe または [your-application-name]Server.exe という実行ファイルが生成されます。

プラグインで、ローカルワークステーションにクライアントとサーバーのビルド実行ファイルへのパスを設定します。

ステップ 3: Anywhere フリートに接続する

このステップでは、使用する Anywhere フリートを指定します。Anywhere フリートは、ゲームサーバーをホスティングするための、どこにでも配置できる一連のコンピューティングリソースを定義します。

- 現在使用しているAWSアカウントに既存の Anywhere フリートがある場合は、フリート名ドロップダウンフィールドを開き、フリートを選択します。このドロップダウンには、現在アクティブなユーザープロファイルの AWS リージョン内の Anywhere フリートのみが表示されます。
- 既存のフリートがない場合、または新しいフリートを作成する場合は、[新しい Anywhere フリートを作成する] を選択してフリート名を指定します。

プロジェクトに Anywhere フリートを選択すると、Amazon はフリートのステータスがアクティブ広告にフリート ID が表示され GameLift ていることを確認します。このリクエストの進行状況は、Unreal エディタの出力ログで追跡できます。

ステップ 4: ワークステーションを登録する

このステップでは、ローカルワークステーションを新しい Anywhere フリートのコンピューティングリソースとして登録します。

1. ローカルマシンのコンピューティング名を入力します。フリートに複数のコンピューティングを追加する場合、名前は一意でなければなりません。

2. ローカルマシンの IP アドレスを入力します。このフィールドのデフォルトは、マシンのパブリック IP アドレスです。ゲームクライアントとサーバーを同じマシンで実行している限り、localhost (127.0.0.1) を使用することもできます。
3. [コンピューティングを登録] を選択します。このリクエストの進行状況は、Unreal エディタの出力ログで追跡できます。

このアクションに応じて、Amazon はコンピューティングに接続できる GameLift ことを確認し、新しく登録されたコンピューティングに関する情報を返します。また、Amazon GameLift サービスとの通信を初期化するときにゲーム実行可能ファイルに必要なコンソール引数も作成します。

ステップ 5: 認証トークンを生成する

Anywhere コンピューティングで実行されているゲームサーバープロセスには、 GameLift サービスを呼び出すための認証トークンが必要です。プラグインからゲームサーバーを起動するたびに、プラグインは Anywhere フリートの認証トークンを自動的に生成して保存します。認証トークンの値はコマンドライン引数として保存され、サーバーコードはランタイム時にそれを取得できます。

このステップでは何もする必要はありません。

ステップ 6: ゲームを起動する

この時点で、Amazon を使用してローカルワークステーションでマルチプレイヤーゲームを起動してプレイするために必要なすべてのタスクが完了しました GameLift。

1. ゲームサーバーを起動します。ゲームサーバーは、ゲームセッションをホストする準備ができた GameLift ときに Amazon に通知します。
2. ゲームクライアントを起動し、新しい機能を使用して新しいゲームセッションを開始します。このリクエストは、新しいバックエンドサービス GameLift を介して Amazon に送信されます。それに応じて、Amazon GameLiftはローカルマシンで実行されているゲームサーバーを呼び出して、新しいゲームセッションを開始します。ゲームセッションがプレイヤーを受け入れる準備ができたら、Amazon はゲームクライアントがゲームセッションに参加するための接続情報 GameLift を提供します。

マネージド EC2 フリートでゲームをクラウドホスティングにデプロイします。

このワークフローでは、 プラグインを使用して、Amazon が管理するクラウドベースのコンピューティングリソースでホストするようにゲームを変更します GameLift。Amazon GameLift 機能のクラ

イアントとサーバーのゲームコードを追加し、サーバービルドを Amazon GameLift サービスにアップロードしてクラウドベースのリソースにデプロイします。このワークフローが完了すると、クラウド内のゲームサーバーに接続できるゲームクライアントが動作します。

Amazon GameLift マネージド Amazon EC2 ワークフローを開始するには：

- Unreal エディタのメインツールバーで Amazon GameLift メニューを選択し、マネージド EC2 でホストを選択します。このアクションにより [Amazon EC2 Fleet のデプロイ] というプラグインページが開き、ゲームコンポーネントを統合、ビルド、デプロイ、起動するための 6 つのステップのプロセスが表示されます。

ステップ 1: プロフィールを設定する

このワークフローに従うときに使用したいプロファイルを選択します。選択したプロファイルは、ワークフローのすべてのステップに影響します。作成したすべてのリソースはプロファイルのAWSアカウントに関連付けられ、プロファイルのデフォルトAWSリージョンに配置されます。プロファイルユーザーのアクセス許可によって、AWSリソースとアクションへのアクセスが決まります。

1. 使用可能なプロファイルのドロップダウンリストからプロファイルを選択します。まだプロファイルを持っていない場合、または新しいプロファイルを作成する場合は、Amazon GameLift メニューに移動し、AWS「ユーザープロファイルの設定」を選択します。
2. ブートストラップステータスが「アクティブ」でない場合は、ブートストラッププロファイルを選択し、ステータスが「アクティブ」に変わるのを待ちます。

ステップ2: ゲームコードをセットアップする

このステップでは、クライアントとサーバーのコードに一連の更新を実施し、ホスティング機能を追加します。Unreal エディタのソースビルドバージョンをまだ設定していない場合、プラグインは手順とソースコードへのリンクを提供します。

Anywhere フリートで使用するためにゲームを統合する場合は、ゲームコードを変更する必要はありません。スタートアップゲームマップを使用している場合は、EC2 デプロイでも機能します。

- ゲームコードをセットアップする (Anywhere)
- ゲームコンポーネントを構築する

ゲームサーバーを構築したら、以下のタスクを完了して Amazon にアップロードする準備をします GameLift。

サーバービルドをクラウドデプロイ用にパッケージ化するには

Unreal Editor がデフォルトでサーバービルドファイルをパッケージ化する `WindowsServer` フォルダに、次の内容を追加します。

1. プラグインのダウンロードに含まれるインストールスクリプトを `WindowsServer` フォルダーのルートにコピーします。`[project-name]/Plugins/Resources/CloudFormation/extra_server_resources/install.bat` ファイルを探します。Amazon GameLift はこのファイルを使用して、各 EC2 ホスティングリソースにサーバービルドをインストールします。
2. Visual Studio のインストールに含まれている `VC_redist.x64.exe` ファイルを `WindowsServer` フォルダのルートにコピーします。このファイルは一般に `C:/Program Files (x86)/Microsoft Visual Studio/2019/Professional/VC/Redist/MSVC/v142` にあります。
3. ゲームサーバービルドの OpenSSL DLL をフォルダ `WindowsServer/MyGame/Binaries/Win64` にコピーします。DLL がサーバービルドで使用されているバージョンと同じであることを確認してください。次のファイルをコピーします。

   - `libssl-3-x64.dll`
   - `libcrypto-3-x64.dll`

ステップ 3: デプロイシナリオを選択する

このステップでは、この時点でデプロイしたいゲームホスティングソリューションを選択します。どのシナリオを使用しても、ゲームを複数デプロイすることができます。

- 単一リージョンフリート: アクティブなプロファイルのデフォルトAWSリージョンのホスティングリソースの単一のフリートにゲームサーバーをデプロイします。このシナリオは、AWS とのサーバー統合とサーバービルド設定をテストする出発点に適しています。次のリソースをデプロイします。
  - ゲームサーバービルドをインストールして実行中の AWS フリート (オンデマンド)。
  - プレイヤーが認証してゲームを開始するための Amazon Cognito ユーザープールとクライアント。
  - ユーザープールと API をリンクする API ゲートウェイオーソライザー。
  - プレイヤーから API ゲートウェイへの過剰な呼び出しを スロットリングする WebACL。

- プレイヤーがゲームスロットをリクエストするための API ゲートウェイ + Lambda 関数。この関数は、何も利用できない場合に `CreateGameSession()` を呼び出します。
- プレイヤーがゲームリクエストの接続情報を取得するための API ゲートウェイ + Lambda 関数。

- FlexMatch フリート: ゲームサーバーを一連のフリートにデプロイし、プレイヤー FlexMatch マッチを作成するためのルールを含むマッチメーカーを設定します。このシナリオでは、マルチフリートのマルチロケーション構造で低コストのスポットホスティングを使用し、高い可用性を実現します。このアプローチは、ホスティングソリューションのマッチメーカーコンポーネントの設計を開始する準備ができたときに役立ちます。このシナリオでは、このソリューションの基本リソースを作成し、必要に応じて後でカスタマイズできます。次のリソースをデプロイします。
  - FlexMatch プレイヤーのリクエストとフォームマッチを受け入れるためのマッチメーキング設定とマッチメーキングルールセット。
  - ゲームサーバービルドがインストールされ、複数のロケーションで稼働している 3 AWS つのフリート。バックアップとして 2 つのスポットフリートと 1 つのオンデマンドフリートが含まれます。
  - (実行可能性、コスト、プレイヤーレイテンシーなどに基づいて) 最適なホスティングリソースを見つけ、ゲームセッションを開始することで、提案されたマッチのリクエストに応える AWS ゲームセッションプレイスメントキュー。
  - プレイヤーが認証してゲームを開始するための Amazon Cognito ユーザープールとクライアント。
  - ユーザープールと API をリンクする API ゲートウェイオーソライザー。
  - プレイヤーから API ゲートウェイへの過剰な呼び出しを スロットリングする WebACL。
  - プレイヤーがゲームスロットをリクエストするための API ゲートウェイ + Lambda 関数。この関数は `StartMatchmaking()` を呼び出します。
  - プレイヤーがゲームリクエストの接続情報を取得するための API ゲートウェイ + Lambda 関数。
  - Amazon DynamoDB テーブルには、プレイヤーのマッチメーキングチケットとゲームセッション情報を保存できます。
  - SNS トピック + GameSessionQueue イベントを処理する Lambda 関数。

ステップ 4: ゲームパラメーターを設定する

このステップでは、AWS にアップロードするゲームを記述します。

- サーバービルド名: ゲームサーバービルドにわかりやすい名前を指定します。 AWSは、この名前を使用して、アップロードされ、デプロイに使用されるサーバービルドのコピーを参照します。
- サーバービルド OS: サーバーが実行されるように構築されるオペレーティングシステムを入力します。これにより、AWS にゲームのホストに使用するコンピューティングリソースの種類が通知されます。
- ゲームサーバーフォルダ: ローカルサーバービルドフォルダのパスを指定します。
- ゲームサーバービルド: ゲームサーバーの実行可能ファイルのパスを指定します。
- ゲームクライアントパス: ゲームクライアント実行ファイルへのパスを指定します。
- クライアント設定出力: このフィールドは、AWS 設定を含むクライアントビルド内のフォルダを指す必要があります。次のロケーションを探してください: `[client-build]/[project-name]/Content/CloudFormation`。

ステップ 5: デプロイシナリオ

このステップでは、選択したデプロイシナリオに基づいてゲームをクラウドホスティングソリューションにデプロイします。このプロセスには、AWS がサーバービルドの検証、ホスティングリソースのプロビジョニング、ゲームサーバーのインストール、サーバープロセスの起動、ゲームセッションをホストする準備が整うまで 40 分ほどかかる場合があります。

デプロイを開始するには、デプロイを選択します CloudFormation。ゲームホスティングの状況は、こちらで追跡できます。より詳細な情報については、AWS の AWS 管理コンソールにサインインしてイベント通知を確認することができます。必ず、プラグインのアクティブユーザープロファイルと同じアカウント、ユーザー、AWS リージョンを使用してサインインしてください。

デプロイが完了すると、AWS EC2 インスタンスにゲームサーバーがインストールされています。少なくとも 1 つのサーバープロセスが実行中で、ゲームセッションを開始する準備ができています。

ステップ 6: クライアントの起動

この時点で、Amazon でホストされているマルチプレイヤーゲームの起動とプレイに必要なすべてのタスクが完了しました GameLift。ゲームをプレイするには、ゲームクライアントのインスタンスを起動します。

シングルフリートシナリオをデプロイした場合、1 人のプレイヤーで 1 つのクライアントインスタンスを開き、サーバーマップに入って動き回ることができます。ゲームクライアントのインスタンスをさらに開き、同じサーバーゲームマップに 2 人目のプレイヤーを追加します。

FlexMatch シナリオをデプロイした場合、ソリューションはプレイヤーがサーバーマップに入る前に、少なくとも 2 つのクライアントがゲームセッションの配置をキューに入れるのを待ちます。

Amazon GameLift インスタンスのフリートデータを取得する

カスタムゲームビルドまたはリアルタイムサーバースクリプトが、Amazon GameLift フリートに関する情報を必要とする場合があります。例えば、ゲーム構築またはスクリプトに次のコードが含まれる場合があります。

- フリートデータに基づいてアクティビティをモニタリングします。
- メトリクスをロールアップして、フリートデータでアクティビティを追跡します。(多くのゲームでは、このデータを LiveOps アクティビティに使っています。)
- マッチメーキング、追加のキャパシティスケーリング、テストなど、カスタムゲームサービスに関連するデータを提供します。

フリート情報は、次の場所の各インスタンスで JSON ファイルとして利用できます。

- Windows: `C:\GameMetadata\gamelift-metadata.json`
- Linux: `/local/gamemetadata/gamelift-metadata.json`

`gamelift-metadata.json` ファイルには、Amazon GameLift フリートリソースの属性が含まれています。

JSON ファイルの例:

```
{
    "buildArn":"arn:aws:gamelift:us-west-2:123456789012:build/
build-1111aaaa-22bb-33cc-44dd-5555eeee66ff",
    "buildId":"build-1111aaaa-22bb-33cc-44dd-5555eeee66ff",
    "fleetArn":"arn:aws:gamelift:us-west-2:123456789012:fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
    "fleetDescription":"Test fleet for Really Fun Game v0.8",
    "fleetId":"fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
    "fleetName":"ReallyFunGameTestFleet08",
    "fleetType":"ON_DEMAND",
    "instanceRoleArn":"arn:aws:iam::123456789012:role/S3AccessForGameLift",
    "instanceType":"c5.large",
    "serverLaunchPath":"/local/game/reallyfungame.exe"
```

```
}
```

FlexMatch のマッチメーキングの追加

Amazon GameLift FlexMatch を使用して、プレイヤーのマッチメーキング機能を Amazon GameLift でホストされているゲームに追加します。FlexMatch は、カスタム ゲームサーバーまたはリアルタイムサーバーで使用できます。

FlexMatch では、マッチメーキングサービスとカスタマイズ可能なルールエンジンをペアリングします。プレイヤーの属性とゲームモードに基づいて、ゲームに適したマッチング方法を設計します。FlexMatch は、ゲームを探している選手の評価、1 つ以上のチームとのマッチの形成、マッチをホストするためにゲームセッションのスタートのナットとボルトを管理します。

完全な FlexMatch サービスを使用するには、キューを使用してホスティングリソースを設定する必要があります。Amazon GameLift はキューを使用して、複数のリージョンとコンピューティングタイプのゲームに最適なホスティング場所を特定します。特に、Amazon GameLift キューは、ゲームクライアントによって提供されるレイテンシーデータを使用して、プレイ時にプレイヤーが可能な限り低いレイテンシーを経験するようにゲームセッションを配置できます。

マッチメーキングをゲームに統合する際の詳細なヘルプを含む FlexMatch の詳細については、「Amazon GameLift FlexMatch デベロッパーガイド」のトピックを参照してください。

- Amazon GameLift FlexMatch の仕組み
- [FlexMatch integration steps](FlexMatch 統合ステップ)

Amazon GameLift コンテナによるホスティングの管理

このドキュメントは、パブリックプレビューリリースの機能に関するものです。このドキュメントは変更される可能性があります。

Amazon GameLift は、ゲームサーバーホスティングのコンテナ化されたソリューションをサポートする完全なクラウドホスティングサービスを提供します。Amazon GameLift コンテナフリートでは、移植性、俊敏性、耐障害性などのコンテナの利点を活用できます。

主な特徴

Amazon GameLift コンテナフリートでは、次の機能を使用できます。

- Amazon GameLift ホスティングリソースでゲームサーバーソフトウェアを実行するための軽量コンテナを備えたカスタムコンテナアーキテクチャを開発します。
- Amazon GameLift Agent を含めて、コンテナ内のゲームサーバープロセスのライフサイクルを管理します。コンピューティング上のエージェントは、サーバープロセスをいつ、どのように開始するか、ゲームセッションホスティングのために維持する数に関する指示を実行します。
- Amazon が提供するリソースをカスタマイズ GameLift して、ゲームサーバーアプリケーションでコンテナイメージを構築します。提供された dockerfile を使用して Linux ベースのコンテナイメージを作成します。コンテナフリートのイメージを Amazon Elastic Container Registry (Amazon ECR) プライベートリポジトリに保存します。
- Amazon が GameLift サポートする AWS リージョン またはローカルゾーンにコンテナフリートリソースをデプロイすることで、低レイテンシーのプレイヤーエクスペリエンスを実現します。フリート管理を効率化するためのマルチロケーションコンテナフリートを作成します。Amazon GameLift ホスティングロケーション を参照してください。
- コンテナ化されたゲームホスティングソリューションを Amazon GameLift Anywhere フリートでテストします。Anywhere を使用して、Amazon GameLift SDK 統合やコンテナイメージ設定など、ソリューション開発をローカルでテストします。
- コンテナ固有のパフォーマンスメトリクスでゲームホスティングのパフォーマンスを追跡します。ハードウェアメトリクスを使用してフリートリソースのヘルス状態をモニタリングします。
- キューや FlexMatchマッチメーキングなどの Amazon GameLift ゲームセッション配置ツールを使用して、コンテナフリートでホストされている最適なゲームセッションにプレイヤーをマッチングします。

- Amazon の AWS CloudFormation テンプレートを使用してコンテナフリートリソースを管理します GameLift。

パブリックプレビュー中のコンテナフリートの使用

新しいコンテナフリート機能は、現在パブリックプレビュー中です。このフェーズでは、以下の Amazon GameLift 機能がサポートされています。

- コンテナフリートを使用して、Linux 用に構築されたゲームサーバーをホストします。コンテナフリートは Linux コンテナイメージを使用し`Amazon_Linux_2023`、サポートします。Windows コンテナはサポートされていません。
- ゲームサーバープロジェクトを Amazon GameLift サーバー SDK バージョン 5 以降とのみ統合します。以前のバージョンはサポートされていません。
- Amazon が GameLift サポートする Amazon EC2 オンデマンドインスタンスタイプのいずれかを使用します。現時点では、スポットフリートはサポートされていません。

Amazon でのコンテナの仕組み GameLift

このドキュメントは、パブリックプレビューリリースの機能に関するものです。このドキュメントは変更される可能性があります。

Amazon GameLift コンテナフリートは、コンテナ化されたアプリケーションをデプロイおよびスケーリングする方法を柔軟に行えるように設計されています。Amazon Elastic Container Service (Amazon ECS) を使用して、Amazon GameLift フリートのタスクのデプロイと実行を管理します。このトピックでは、Amazon GameLift マネージドフリートでコンテナを実行するための基本的な構造要素について説明し、一般的なアーキテクチャを説明し、いくつかの重要な概念の概要を説明します。

コンテナフリートのコンポーネント

フリート

コンテナフリートは、コンテナ化されたゲームサーバーを実行する Amazon によって管理 GameLiftされる Amazon EC2 インスタンスのコレクションです。フリートを作成するときは、コンテナアーキテクチャとゲームサーバーソフトウェアを各フリートインスタンスにデプロイする方法を設定します。コンテナフリートは、単一の AWS リージョン または複数の地理的場所にデ

プロイできます。Amazon GameLift 手動または自動スケーリングツールを使用して、ゲームセッションとプレイヤーをホストするためにコンテナフリートの容量をスケーリングできます。

インスタンス

Amazon EC2 インスタンスは、ゲームホスティングのコンピューティング性能を提供する仮想サーバーです。Amazon では GameLift、さまざまなインスタンスタイプから選択できます。インスタンスタイプごとに、CPU、メモリ、ストレージ、ネットワーク容量の組み合わせが異なります。

コンテナフリートを作成すると、Amazon は選択したインスタンスタイプとフリート設定に基づいてインスタンスを GameLift デプロイします。デプロイされた各フリートインスタンスは同じであり、コンテナ化されたゲームサーバーソフトウェアを同じ方法で実行します。フリート内のインスタンスの数によって、フリートのサイズとゲームホスティングの容量が決まります。

コンテナグループ

Amazon GameLift は、コンテナグループの概念を使用して、一連のコンテナを記述および管理します。コンテナグループは、コンテナ「タスク」または「ポッド」に似ています。各コンテナグループ内で、コンテナが使用可能な CPU とメモリリソースを共有する方法を定義できます。コンテナ間に依存関係を設定し、コンテナグループのライフサイクルを管理することもできます。

コンテナグループは、各フリートインスタンス間でレプリケートして、リソースの使用を最適化できます。次のように、コンテナグループのスケジューリング戦略を設定することで、レプリケーションを管理できます。

- レプリカコンテナグループは、ゲームサーバーアプリケーションとサポートソフトウェアを実行するコンテナを管理します。すべてのコンテナフリートは、レプリカコンテナグループを定義する必要があります。コンテナグループの要件と使用中のインスタンスタイプのリソースに応じて、レプリカグループが各フリートインスタンスに複数のコピーを持つ場合があります。レプリカグループ内のすべてのコンテナは、インスタンス全体で自動的に一緒にスケーリングされます。
- デーモンコンテナグループはオプションであり、モニタリングなどのバックグラウンドサービスやユーティリティプログラムの実行に役立ちます。ゲームサーバーソフトウェアは、デーモングループのプロセスに直接依存しません。デーモンコンテナグループはレプリケートされません。各フリートインスタンスには、デーモングループの最大 1 つのコピーがあります。つまり、デーモングループ内のコンテナは、レプリカグループ内のコンテナとともにフリートインスタンス全体にスケールされません。

コンテナフリートにはレプリカコンテナグループが 1 つ必要で、オプションで 1 つのデーモングループを持つことができます。

コンテナ

コンテナは、コンテナベースのアーキテクチャの最も基本的な要素です。ソフトウェア実行可能ファイルと依存ファイルを含むコンテナイメージで構成されます。Amazon で使用するコンテナを定義するときは GameLift、コンテナでソフトウェアを実行する方法を設定します。

コンテナフリート内の各コンテナグループには、「必須」と指定されたコンテナが 1 つ必要です。必須コンテナは、コンテナグループのライフサイクルを推進します。必須コンテナが失敗すると、コンテナグループ全体が再起動します。

コンテナタイプには以下が含まれます。

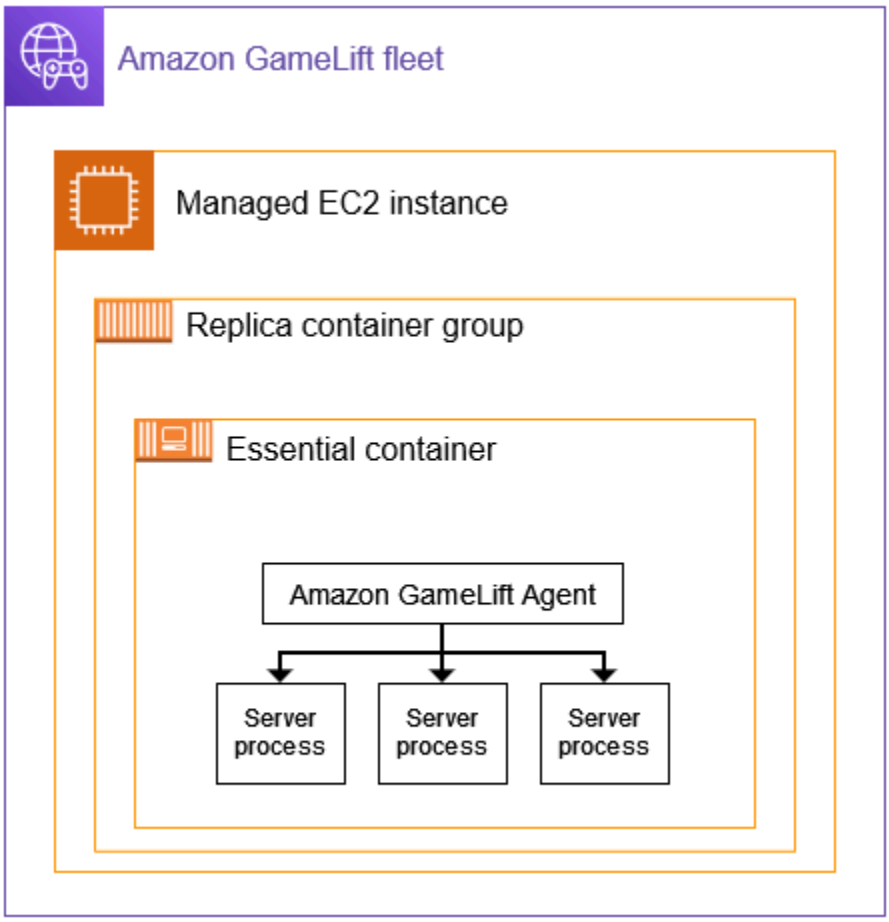
- 重要なレプリカコンテナには、ゲームサーバープロセスを実行し、プレイヤーのゲームセッションをホストするために必要なすべてが含まれています。これには、Amazon サーバー SDK と統合されたゲーム GameLift サーバービルドと依存ソフトウェアが含まれます。また、ゲームサーバープロセスのライフサイクルを管理する Amazon GameLift Agent も含まれています。フリートのレプリカコンテナグループには、必須のレプリカコンテナが 1 つだけあります。
- 「サイドカー」コンテナとも呼ばれる非必須レプリカコンテナ は、ゲームサーバーアプリケーションをサポートするソフトウェアを実行します。サイドカーコンテナを使用すると、ゲームサーバーと一緒にサポートソフトウェアを実行およびスケールできますが、個別のコンテナとして を管理できます。このタイプのコンテナが失敗した場合、コンテナ自体のみが再起動し、コンテナグループは影響を受けません。
- デーモンコンテナは、デーモンサービスを実行してバックグラウンドプロセスを管理します。デーモンコンテナの一般的な用途は、Amazon CloudWatch (CloudWatch) エージェントを実行してコンテナのメトリクス、ログ、トレースを収集することです。デーモンコンテナは、コンテナの障害によってコンテナグループが再起動する必要がある時期に応じて、必須または重要でない場合があります。

コンピューティング

コンピューティングは、Amazon GameLift サービスに登録され、サービスと通信できるフリートホスティングリソースです。コンテナフリートでは、コンピューティングはコンピューティング登録プロセスを管理するプロセスを持つコンテナです。コンテナフリートの必須レプリカコンテナでは、Amazon GameLift エージェントはこのコンテナをコンピューティングとして自動的に登録します。

一般的なアーキテクチャ

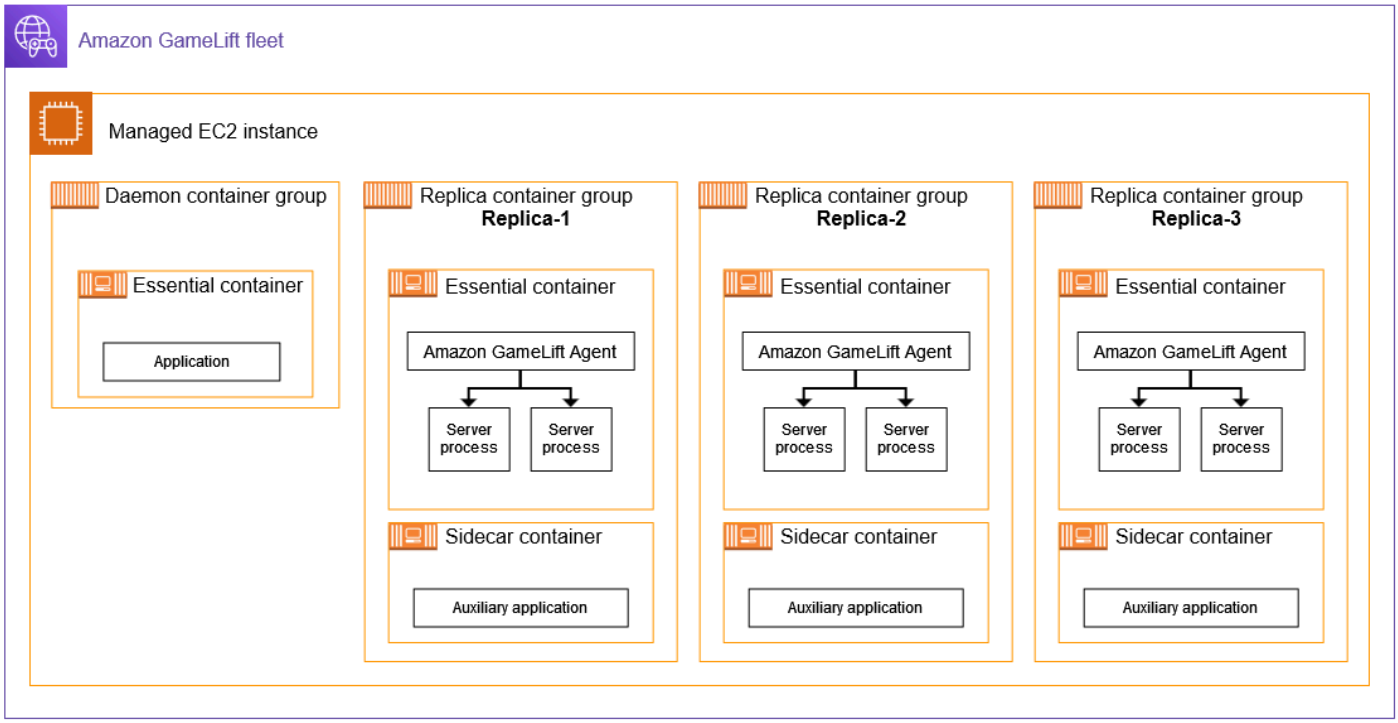
次の図は、最も単純なコンテナフリート構造を示しています。この構造では、フリート内の各インスタンスはレプリカコンテナグループの 1 つのコピーを保持します。コンテナグループには、Amazon GameLift エージェント、ゲームサーバーアプリケーション、およびゲームセッションをホストするためのすべてのサポートソフトウェアを実行する 1 つの必須コンテナがあります。エージェントは、フリート固有の指示を実装して、3 つのサーバープロセスを同時に実行します。インスタンスごとに 1 つのレプリカコンテナグループがあるため、各フリートインスタンスは 3 つのサーバープロセスを同時に実行します。



この 2 番目の例は、より複雑なコンテナフリート設計を示しています。この例では、フリートには複数のコンテナを持つレプリカコンテナグループと、1 つのコンテナを持つデーモンコンテナグループがあります。フリート設定では、各フリートインスタンスにレプリカコンテナグループの 3 つのコピーが配置されます。デーモンコンテナグループはレプリケートされません。

内のレプリカグループコンテナの各セットには、各インスタンスに 3 つのコピーがあります。必須の各レプリカコンテナでは、エージェントは 2 つのサーバープロセスを同時に実行するように指示

されます。その結果、各フリートインスタンスは 6 つのサーバープロセスを同時に実行します (3 つの必須レプリカコンテナのそれぞれに 2 つのプロセス)。



重要な概念

このセクションでは、Amazon がいくつかの基本的なコンテナ概念 GameLift を実装する方法をまとめています。コンテナフリートの操作方法については、このガイドの関連トピックを参照してください。

コンテナグループのパッキング

コンテナフリートにデプロイするためのコンテナ構造を開発する場合、一般的な目標は利用可能なコンピューティング能力の使用を最適化することです。この目標を達成するために、ゲームサーバーのパフォーマンスに影響を与えることなく、フリートインスタンスに配置できるレプリカコンテナグループの最大数を見つけます。

Amazon GameLift はこれを支援します。以下の情報に基づいて、インスタンスあたりのレプリカグループの最大数を計算します。

- フリートのインスタンスタイプ、使用可能な CPU およびメモリリソース。
- レプリカグループ内のすべてのコンテナに設定する CPU とメモリの要件。

  デーモングループ内のすべてのコンテナに設定した CPU とメモリの要件があれば、その要件。

- 各インスタンスのコンテナやその他の重要なアプリケーションを管理するために予約されているリソース。

コンテナフリートを作成するときは、計算された最大数を使用するか、必要な数を指定して計算された数を上書きするかを選択できます。ベストプラクティスとして、コンテナ化されたゲームサーバーソフトウェアを試して、正確なリソース要件を判断します。このデータを使用して、ゲームサーバーのパフォーマンスに最適なパッキング戦略を見つけます。

ゲームサーバーと Amazon GameLift エージェント

必須レプリカコンテナを構築するときは、ゲームサーバーソフトウェアと Amazon GameLift Agent を同じコンテナイメージにまとめてパッケージ化します。このコンピューティング上のエージェントは、コンテナ内のゲームサーバーのライフサイクルを制御します。各レプリカコンテナグループでは、必須のレプリカコンテナは エージェントとすべてのゲームサーバープロセスを実行します。

Amazon GameLift エージェントは、コンテナフリートのランタイム設定 で指示を実行します。ランタイム設定では、(1) 実行を開始する実行可能ファイル、(2) オプションの起動パラメータセット、および (3) 同時に実行するプロセスの数を識別します。ランタイム設定には、複数の異なる実行可能ファイルに関する指示を含めることができます。少なくとも 1 つの命令がゲームサーバー実行可能ファイル用である必要があります。例えば、ランタイム設定では、エージェントに、実稼働用にゲームサーバー実行可能ファイルの 10 個のプロセス、テスト用の特別な起動パラメータを持つ同じ実行可能ファイルの 1 つのプロセス、およびログ記録ユーティリティ用の 1 つのプロセスを維持するように指示できます。

フリートのランタイム設定はいつでも変更できます。Amazon GameLift エージェントは、サービスの更新を定期的にリクエストします。更新されたランタイム設定が使用可能になると、エージェントはその設定を受け取り、指示の実装を開始します。アクションには、サーバープロセスの追加やシャットダウンが含まれる場合があります。

Amazon GameLift Agent は、Amazon がマネージド EC2 フリート GameLift に使用するコンピューティング上のエージェントのオープンソースバージョンです。このガイドでは、ソースから エージェントを構築し、コンテナイメージにビルドする方法について説明します。エージェントは次のタスクを処理します。

サーバープロセス管理：

- ランタイム設定に基づいて、サーバープロセスを開始、シャットダウン、および置き換えます。
- サーバープロセスが時間内にアクティブ化されない場合は、シャットダウンします。
- サーバープロセスが終了した GameLift ら Amazon に報告します。

- サーバープロセスのフリートイベントを発行します。

コンテナ管理：

- Amazon からのプロンプトに応じてサーバープロセスをシャットダウンします GameLift。
- コンテナのヘルスをレポートします。

ログのアップロードタスク：

- 指定された Amazon S3 バケットにゲームセッションログをアップロードします。
- 指定された Amazon S3 バケットにコンピューティング上のエージェントログをアップロードします。

フリートの容量のスケーリング

フリート容量は、フリートが一度にホストできるゲームセッションの数を測定します。また、フリートが同時にサポートできるプレイヤーの数に基づいて容量を測定することもできます。

フリートのホスティング容量を増減するには、フリートインスタンスを追加または削除します。コンテナフリートのパッキング戦略は、各フリートインスタンスで同時に実行されるゲームセッションの数を決定します。この数は、フリートの容量を増減するときに追加または減算するゲームセッション (およびプレイヤースロット) の数を示します。

コンテナフリートでは、Amazon が提供するスケーリング方法のいずれかを使用できます GameLift。具体的には次のとおりです。

- 特定の希望するフリートインスタンス数を設定して、フリート容量を手動で設定します。
- 使用可能なインスタンスの希望するバッファ (ターゲット追跡) をターゲットにして、自動スケーリングを設定します。この方法では、入ってくるプレイヤーがいつでも迅速にゲームに参加できるように、一連のアイドルホスティングリソースが自動的に維持されます。プレイヤーの需要が増減すると、このバッファのサイズは継続的に調整されます。
- カスタムスケーリングルール (高度な機能) を使用して自動スケーリングを設定します。

ゲームクライアント/サーバー接続

マネージド EC2 フリートとコンテナフリートは、ゲームクライアントとクラウドホスト型ゲームサーバー間の接続を同様の方法で処理します。Amazon が新しいゲームセッション GameLift を作成

すると、サービスはゲームセッションの接続情報 を通信します。ゲームクライアントは情報を使用して、ゲームセッションをホストしているゲームサーバーに直接接続します。すべてのタイプのフリートについて、接続情報は IP アドレスとポートの割り当てで構成されます。

コンテナフリートを作成するときは、2 セットのポート範囲を定義します。まず、ゲームクライアントがゲームに接続できるようにする外部向け接続ポートの範囲を定義します。次に、コンテナで実行されている各ゲームサーバープロセスに割り当てられる内部専用コンテナポートのセットを定義します。Amazon は、内部コンテナポートを外部接続ポートに GameLift 動的にマッピングして、プレイヤーにゲームへのアクセスを許可します。このアプローチでは、ゲームサーバーがコンテナポートに直接アクセスできないように保護することで、セキュリティをさらに強化します。

コンテナフリートのポート範囲を定義するときは、インスタンス上のコンテナ間で同時に実行されるすべてのサーバープロセスに対応するのに十分なポートを持つ範囲を指定する必要があります。

追加のコントロールのために、フリートのインバウンドアクセス許可も設定します。インバウンドアクセス許可は、受信トラフィックに対して開いている接続ポートを決定します。フリートのインバウンドアクセス許可はいつでも変更できます。インバウンドアクセス許可を使用すると、必要に応じてすべての接続ポートをすばやくシャットダウンしたり、一部を開いたり、すべてを開くことができます。

Amazon GameLift コンテナの開発ロードマップ

このドキュメントは、パブリックプレビューリリースの機能に関するものです。このドキュメントは変更される可能性があります。

次のワークフローは、ゲームサーバーを Amazon GameLift コンテナフリートで実行するためのステップをまとめたものです。

ステップ 1: ゲームを Amazon と統合する GameLift

ゲームサーバーに機能を追加して、コンテナフリートにデプロイされたときに Amazon GameLift サービスと通信できるようにします。 FlexMatch マッチメーキングを使用している場合は、この機能をゲームサーバーとクライアントに追加します。詳細については、「ゲームを Amazon と統合する GameLift」を参照してください。

- Amazon GameLift サーバー SDK (バージョン 5 以降) を取得し、ゲームプロジェクトでセットアップします。サーバー SDK は C++、C#、Go で使用できます。
- ゲームサーバーコードを変更して、必要なサーバー SDK 機能を追加します。
- Linux 用のゲームサーバービルドをパッケージ化します。Windows で開発している場合、このステップでは Linux 環境をセットアップするための追加の作業が必要になる場合があります。

- （オプション) Amazon GameLift Anywhere フリートを使用してゲームサーバーの統合をテストします。コンテナイメージを準備する前にテストして、統合作業の問題を特定します。ゲームクライアント/サーバー接続をテストするには、ゲームクライアントも統合します。

> **Note**
>
> Windows で開発している場合は、別の Linux ワークスペースを設定するか、Windows サブシステム for Linux (WSL) などのツールを使用します。ゲームサーバービルドをテストしたり、コンテナイメージを構築してテストしたりするには Linux 環境が必要です。

ステップ 2: ゲームサーバーのコンテナイメージを準備する

ゲームサーバープロセスを実行するコンテナイメージを作成し、Amazon で使用する Amazon Elastic Container Registry (Amazon ECR) リポジトリに保存します GameLift。詳細な手順については、「ゲームサーバーソフトウェアでコンテナイメージを準備する」を参照してください。

- Linux ゲームビルド、インストールスクリプト、およびすべてのサポートソフトウェアと依存関係を使用して、コンテナイメージの作業ディレクトリを設定します。
- Amazon GameLift Agent ソースコードを取得し、構築して、`jar`ファイルを作業ディレクトリに追加します。
- デフォルトの Dockerfile を取得し、変更してゲームサーバーソフトウェアでコンテナイメージを設定します。
- コンテナイメージを構築します。Linux 環境でこのステップを実行します。
- Amazon ECR プライベートリポジトリを作成し、コンテナイメージをそのリポジトリにプッシュします。コンテナフリートをデプロイする予定の同じ AWS アカウント と AWS リージョン にリポジトリを作成します。
- （オプション) Anywhereフリートを使用してコンテナイメージをテストします。Amazon GameLift エージェントに指示を渡すようにランタイム設定を設定できます。

ステップ 3: コンテナとコンテナグループを作成する

Amazon でのゲームホスティング用のコンテナアーキテクチャを設計します GameLift。「Amazon GameLift コンテナフリートの設計」および「Amazon コンテナフリートの GameLift コンテナグループ定義を作成する」を参照してください。

- コンテナ設定を定義します。コンテナごとに、ランタイムプロセス、メモリ割り当て、ヘルスチェック、ネットワークポートなどの問題を定義します。

- Amazon GameLift コンソールまたは AWS CLI を使用して、コンテナ設定でコンテナグループ定義を作成します。コンテナグループ定義を作成すると、Amazon はその時点で各コンテナイメージのスナップショット GameLift を作成します。

ステップ 4: コンテナ化されたゲームサーバーをコンテナフリートにデプロイする

前のステップで作成したコンテナグループ定義を使用して、コンテナフリートを作成し、コンテナ化されたゲームサーバーソフトウェアをデプロイします。Amazon GameLift コンテナフリートを作成する を参照してください。

- Amazon GameLift コンソールまたは AWS CLI を使用してコンテナフリートを作成します。
- フリートインスタンスがデプロイおよびアクティブ化されるたびにフリートのステータスを追跡します。フリート作成イベントをチェックして、フリートがすべてのロケーションに正常にデプロイされていることを確認します。
- ゲームクライアントがゲームセッションをリクエストして参加し、ゲームをプレイできることを確認します。マッチメーキングを設定している場合は、それらのシナリオをテストします。

ステップ 5: フリートを管理する

本番稼働レベルの使用に備えながら、ゲームホスティングソリューションを構築し、ホスティングライフサイクルを管理します。

- プレイヤーベースをサポートするために AWS リージョン 、他の にマルチロケーションフリートとフリートを作成します。
- キューまたは FlexMatch マッチメーキングを使用してゲームホスティングプレイスメントを設定します。次のリソースを参照してください。
  - ゲームセッションプレイスメント用の Amazon GameLiftt キューのセットアップ
  - FlexMatch 開発者ガイド
- ゲームセッションのプレイヤーの需要に基づいてフリート容量を管理するようにオートスケーリングを設定します。
- コンテナフリートのモニタリングを設定します。Amazon GameLift メトリクスの操作、ゲームセッションログとコンテナログの取得、個々のコンテナへのリモートアクセスの設定を行います。
- コンテナフリートの長期管理を設定します。フリートエイリアスを使用して、コンテナフリートを更新するプロセスを効率化します。フリートのライフサイクルを管理するための AWS CloudFormation テンプレートを作成します。次のリソースを参照してください。
  - Amazon GameLift フリートにエイリアスを追加する
  - AWS CloudFormation を使用したリソースの管理

ゲームを Amazon と統合する GameLift

このドキュメントは、パブリックプレビューリリースの機能に関するものです。このドキュメントは変更される可能性があります。

ゲームサーバーソフトウェアでコンテナイメージを作成し、クラウドホスティング GameLift 用に Amazon にデプロイする前に、まずゲームプロジェクトを Amazon GameLift Server SDK と統合し、Linux で実行するゲームサーバーを構築します。このトピックでは、Amazon GameLift が提供するさまざまな統合ツールを紹介します。

ホストされたゲームサーバーは、Amazon GameLift サービスと通信できる必要があります。Amazon GameLift サーバー SDK (バージョン 5 以降) をゲームプロジェクトに追加し、ゲームのサーバーコードを変更して、通信を設定します。Amazon GameLift は、複数の言語とゲームエンジンをサポートするサーバー SDK リソースとドキュメントを提供しています。

コンテナ化されたゲームサーバーの統合プロセスは、マネージド EC2 または Amazon GameLift Anywhere フリートでホストするためのゲームサーバーの統合とほぼ同じです。

統合ツール

Amazon GameLift では、統合のために以下のツールと言語がサポートされています。

Unreal Engine デベロッパー向け

Unreal 用の軽量プラグインを使用します。このプラグインには、必要な Amazon GameLift 機能を備えた C++ サーバー SDK ライブラリが含まれています。ドキュメントを使用してプラグインの Unreal ゲームプロジェクトを設定し、提供されたコードブロックでゲームコードを更新して、サーバーとクライアントビルドに必要な機能を追加します。

- SDK プラグインのダウンロード
- ガイド: Unreal プロジェクトを Amazon と統合する GameLift
- リファレンスガイド: Unreal 用 C++ Server SDK 5

注： Unreal Engine 用 Amazon GameLift スタンドアロンプラグインは、コンテナフリートの使用をサポートしていません。

Unity デベロッパー向け

Unity 用の軽量プラグインを使用します。このプラグインには、必要な Amazon GameLift 機能を持つ C# サーバー SDK ライブラリが含まれています。ドキュメントを使用してプラグインの Unreal

ゲームプロジェクトを設定し、提供されたコードブロックでゲームコードを更新して、サーバーとクライアントビルドに必要な機能を追加します。

- SDK プラグインのダウンロード
- ガイド: Unity プロジェクトを Amazon と統合する GameLift
- リファレンスガイド: Unity 用 C# Server SDK 5

注: Unity 用 Amazon GameLift スタンドアロンプラグインは、コンテナフリートの使用をサポートしていません。

他のゲームエンジンを使用するデベロッパー向け

この一般的なサーバーとクライアントの統合ガイダンスに従ってください。

- ゲームサーバーの統合
- ゲームクライアントを統合する

Amazon GameLift は、以下の言語用のサーバー SDK 5 ライブラリを提供しています。

- Server SDK 5 for C++ [SDK download] [Reference guide]
- Server SDK 5 for C# [SDK download] [Reference guide]
- Server SDK 5 for Go [SDK download] [Reference guide]

Linux 用のゲームサーバーを構築する

Amazon GameLift コンテナフリートは、Linux プラットフォームで実行されるゲームサーバーをサポートします。Linux ターゲット用のゲームサーバーを構築するためのヒントをいくつか紹介します。

- Unity ゲームエンジンでゲームを開発している場合、ゲームエディタは組み込みサポートを提供し、Linux 用に構築するための特別な要件はありません。
- C++ でゲームを開発している場合は、Amazon GameLift Server SDK for C++ を構築するとき、およびゲームサーバーを構築するときに、Linux 用の OpenSSL ライブラリを含める必要があります。ゲームサーバーコンテナイメージに同じライブラリも含めます。
- Windows で Unreal Engine を使用してゲームを開発している場合は、次のオプションを検討してください。

- Unreal Engine を使用して、クロスコンパイルツールチェーン を設定します。
- 別の Linux ワークスペースを設定するか、Windows サブシステム for Linux (WSL) などのツールを使用します。この環境を使用して Linux で Unreal Editor を実行し、ゲームサーバーを構築できます。

統合をローカルでテストする

Amazon GameLift Anywhere フリートを使用して、ゲーム統合をローカルでテストできます。このアプローチは、統合に直接関連する問題を特定するのに役立つベストプラクティスです。Anywhere フリートは、ゲームセッションの開始/停止やプレイヤーの接続の追跡など、テストアプリケーションやゲームシナリオを実行するのに便利なツールです。Anywhere フリートを使用すると、反復的に構築およびテストを高速化できるため、ホスティングアクティビティの可視性が向上します。

統合テストに Amazon GameLift Anywhere フリートを使用する方法については、Amazon GameLift Anywhere フリートを使用して統合をテストする「」を参照してください。テスト環境を設定するためのワークフローは次のようになります。

1. Linux を実行しているローカルデバイスをセットアップします。
2. Anywhereフリートをセットアップします。ローカルデバイスのカスタムロケーションを作成し、Anywhereフリートを作成してから、ローカルデバイスをフリートのコンピューティングとして登録します。
3. ゲームサーバーの認証トークンを取得します。統合サーバープロセスでは、Amazon GameLift サービスで認証するためにトークンが必要です。同時に実行される複数のサーバープロセスに同じトークンを再利用できます。このステップは、統合テストに Anywhereフリートを使用する場合にのみ必要です。

   > **Note**
   >
   > 認証トークンは一時的なものであり、定期的に更新する必要があります。新しいトークンをリクエストするには、サーバービルドパッケージにスクリプトを追加することを検討してください。

4. のゲームサーバーコードを更新しますAnywhere。Anywhere フリートで実行する場合、ゲームサーバーは次のサーバーパラメータを使用してサーバー SDK アクション `InitSdk()` (C++) (C#) (Unreal) を呼び出す必要があります。このステップは、統合テストに Anywhereフリートを使用する場合にのみ必要です。Amazon GameLift Agent をコンテナイメージに追加すると、これらのパラメータが自動的に処理されます。

ベストプラクティスとして、環境変数または起動時に指定したコンソール引数からこれらの値を取得するようにサーバーコードを設定します。

- webSocketUrl – の値を使用します。これは GameLiftServiceSdkEndpointへの呼び出しから返されますregister-compute。
- processId – サーバープロセスに一意の識別子を割り当てます。
- fleetId – Anywhere フリート識別子。 への呼び出しから返されますcreate-fleet。
- authToken – への呼び出しから返される有効な認証トークンget-compute-auth-token。

5. ローカルマシンで、ゲームサーバービルドソフトウェアをセットアップし、サーバープロセスを起動します。

   サーバー統合が成功すると、サーバープロセスはサーバー SDK アクションを呼び出しInitSDK()て Amazon GameLift サービスとの接続を確立し、次に を呼び出しProcessReady()て、ゲームセッションをホストする準備ができたことをサービスに通知します。

6. ゲームセッションを開始します。ゲームクライアントを統合してゲームセッションをリクエストしている場合は、それを使用して新しいゲームセッションをリクエストできます。そうでない場合は、 AWS CLI コマンド を使用します create-game-session。Amazon GameLift は GameSession オブジェクトを作成し、新しいゲームセッションを開始するプロセスを開始します。

   統合が機能している場合、Amazon はローカルワークステーションでサーバープロセスを GameLift 呼び出して、新しいゲームセッションを開始します (onStartGameSession()コールバックを使用) 。ゲームセッションでプレイヤーが対応できるようになると、サーバープロセスは を呼び出しますActivateGameSession()。それに応じて、Amazon はGameSessionステータスと接続情報 GameLift を更新して、ゲームクライアントがゲームセッションに接続してゲームをプレイできるようにします。

ゲームサーバーソフトウェアでコンテナイメージを準備する

このドキュメントは、パブリックプレビューリリースの機能に関するものです。このドキュメントは変更される可能性があります。

コンテナは、Amazon GameLift コンテナフリートの最も基本的な要素です。コンテナには、ゲームサーバーと SDKs、ソフトウェア、ディレクトリ、ファイルなどの依存関係が含まれます。

コンテナフリートで機能するには、ゲームサーバーが Linux で実行され、サーバー SDK 5.x と統合されている必要があります。

トピック

- 作業ディレクトリをセットアップする
- コンテナイメージを構築する
- コンテナイメージを Amazon ECR にプッシュする

作業ディレクトリをセットアップする

作業ディレクトリは、コンテナイメージの構築に必要なすべてのファイルを配置し、Amazon GameLift の実行方法を定義する場所です。

コンテナ作業ディレクトリを設定するには

1. Amazon GameLift コンテナイメージを使用するディレクトリを作成します。

   Example

   例:

   ```
   [~/]$ mkdir -p work/glc/gamebuild && cd work && find .
   .
   ./glc
   ./glc/gamebuild
   ```

2. Amazon GameLift Agent のクローンを作成します。

   Example

   例:

   ```
   [~/work]$ git clone https://github.com/aws/amazon-gamelift-agent.git
   Cloning into 'amazon-gamelift-agent'...
   ```

3. Maven GameLiftAgent を使用して を構築します。

   Example

   例:

```
[~/work]$ cd amazon-gamelift-agent
```

Example

```
[~/work/amazon-gamelift-agent]$ mvn clean compile assembly:single && \
mv target ../glc && cd .. && find glc
```

4. サーバー SDK 5.x と統合され、構築され、.ZIPファイルにパッケージ化されたゲームサーバーを追加します。
5. .ZIP ファイルを にコピーします~/work/glc/gamebuild/。

SDK 5.x ゲームサーバーがない場合は、サンプルSimpleServerゲームをダウンロードして使用して、コンテナフリートを試すことができます。

Example

```
[~/work]$ curl -o glc/gamebuild/SimpleServer.zip \
'https://ws-assets-prod-iad-r-iad-ed304a55c2ca1aee.s3.us-
east-1.amazonaws.com/086bb355-4fdc-4e63-8ca7-af7cfc45d4f2/
AmazonGameLiftSampleServerBinary.zip' &&
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
Dload  Upload   Total   Spent    Left  Speed
100 5140k  100 5140k    0     0  12.3M      0 --:--:-- --:--:-- --:--:-- 12.3M
glc
glc/target
glc/target/GameLiftAgent-1.0.jar
glc/gamebuild
glc/gamebuild/SimpleServer.zip
```

コンテナイメージを構築する

Dockerfile は、コンテナを構築するための環境、ソフトウェア、および手順を指定します。

Dockerfile を作成するには

1. glc サブディレクトリに移動します。

Example

```
[~/work]$ cd glc && find

.
./target
./target/GameLiftAgent-1.0.jar
./gamebuild
```

2. 新しい Dockerfile を作成して開きます。

   Example

   例:

   ```
   [~/work/glc]$ nano Dockerfile
   ```

3. 次のいずれかのテンプレートからコピーし、コンテンツを Dockerfile に貼り付けます。

ゲームサーバーの Dockerfile テンプレート

このテンプレートには、コンテナが Amazon GameLift フリートで使用できるようにするために必要な最小限の手順が含まれています。ゲームサーバーの必要に応じてコンテンツを変更します。

```
# Base image
# ----------
  # Add the base image that you want to use over here,
  # Make sure to use an image with the same architecture as the
  # Instance type you are planning to use on your fleets.
  # We require JDK to be installed in the base image, so that
  # it can be used to run the &AGS; Agent
FROM public.ecr.aws/amazoncorretto/amazoncorretto:17-amd64
  #
# Game build directory
# --------------------
  # Add your game build to gamebuild directory and add the zip file name in the
 'GAME_BUILD_ZIP' env variable below.
  # The game build provided over here needs to be integrated with gamelift server sdk.
  # This template assumes that the game build is in a zip format.
ENV GAME_BUILD_ZIP="<ADD_GAME_BUILD_ZIP_FILE_NAME>" \
  #
# Default directory
```

```
# -----------------
  # Default directory, the value provided here should be where the game executable
 exists.
  # Provide this same value as your launch path in RuntimeConfiguration when creating a
 fleet.
  # Ref: https://docs.aws.amazon.com/gamelift/latest/apireference/
API_ServerProcess.html
GAME_EXECUTABLE="<ADD NAME OF EXECUTABLE WITHIN THE GAME BUILD>" \
HOME_DIR="/local/game" \
  #
  # Registered compute in anywhere fleet (not used in container fleets)
  # ------------------------------------------------------------------
  # Add the name for the registered compute in an anywhere fleet.
  # This environment variable is required only for anywhere fleets, but not for
 container fleets.
  # If it is set for container fleets, it will be overridden by Gamelift.
  GAMELIFT_COMPUTE_NAME="<ADD_COMPUTE_NAME>" \
  #
# Default Gamelift Agent jar
# --------------------------
GAMELIFT_AGENT_EXEC="GameLiftAgent-1.0.jar" \
  #
  # This env variable defines the name of the S3 bucket that stores the GameLift Agent
 logs.
  # This S3 bucket should exist in the customer AWS account.
  # In order to allow GameLift agent to upload logs to this s3 bucket, customers would
 need to
  # include s3:PutObject permission in the IAM role provided as instanceRoleArn during
 CreateFleet operation.
GAMELIFT_AGENT_LOGS_BUCKET_NAME="<ADD NAME OF GAMELIFT AGENT LOGS S3 BUCKET>" \
  #
# -----------
  # This env variable defines the name of the S3 bucket that stores the game session
 logs.
  # This S3 bucket should exist in the customer AWS account.
  # In order to allow GameLift agent to upload logs to this s3 bucket, customers would
 need to
  # include s3:PutObject permission in the IAM role provided as instanceRoleArn during
 CreateFleet operation.
# -----------
GAME_SESSION_LOGS_BUCKET_NAME="<ADD NAME OF GAME SESSION LOGS S3 BUCKET>" \
#
# -----------
GAMELIFT_AGENT_LOGS_PATH="/local/game/agentlogs/" \
```

```
  #
# NOT USED in container fleets - USED in Anywhere fleets
# ------------------------------------------------------
  # Specifiy the type of compute resource used to host the game servers.
  # This env variable is required only for anywhere fleets, but not for container
 fleets.
  # If it is set for container fleets, it will be overridden by Gamelift.
#
# -----------
COMPUTE_TYPE="ANYWHERE" \
  #
  # Specify the credential to be used for creating the client.
  # This env variable is required only for anywhere fleets, but not for container
 fleets.
  # If it is set for container fleets, it will be overridden by Gamelift.
#
# -----------
CREDENTIAL_PROVIDER="environment-variable"

USER root

    # intall dependencies as necessary
    RUN yum install -y sudo \
                        unzip \
                        git \
                        shadow-utils \
                        iputils \
                        tar \
                        gcc \
                        make \
                        openssl-devel \
                        zlib-devel \
                        vim \
                        net-tools \
                        nc \
                        procps

    # Set up the ground for 'gamescale' user
    RUN groupadd -r gamescale -g 500 && \
      useradd -u 500 -r -g gamescale -m -s /sbin/nologin -c "Gamescale user" gamescale
 && \
      echo "gamescale ALL=(ALL) NOPASSWD: ALL" | (EDITOR="tee -a" visudo) && \
      mkdir -p $HOME_DIR && \
      mkdir $HOME_DIR/mono && \
```

```
      chown -R gamescale:gamescale $HOME_DIR

   WORKDIR $HOME_DIR

   # extract game build as necessary
   COPY ./gamebuild/$GAME_BUILD_ZIP .
   RUN unzip ./$GAME_BUILD_ZIP -d ./

   # copy Gamelift Agent jar
   COPY ./gameliftAgent/$GAMELIFT_AGENT_EXEC ./

   # Add permissions to game build and gamelift agent jar
   RUN chmod +x ./$GAME_EXECUTABLE
   RUN chmod +x ./$GAMELIFT_AGENT_EXEC

   # Check if java is installed on the image, if not then the Agent will not be able
 to run
   RUN java --version

   USER gamescale

   ENV PATH="$PATH:$HOME_DIR/bin:$JAVA_HOME"

   # Change directory to bin
   WORKDIR $HOME_DIR

   # check path before starting the container
   RUN echo $PATH

   # Create logs directory for GameLift Agent & server processes
   RUN mkdir logs
   RUN mkdir agentlogs

   # Start the GameLift Agent
   ENTRYPOINT sleep 90 && java -jar $GAMELIFT_AGENT_EXEC -ip "192.168.1.1" -gslb
 "$GAME_SESSION_LOGS_BUCKET_NAME" -galb "$GAMELIFT_AGENT_LOGS_BUCKET_NAME" -galp
 "$GAMELIFT_AGENT_LOGS_PATH" -glc environment-variable
```

SimpleServer サンプルの Dockerfile

```
# Base image
# ----------
```

```
  # Add the base image that you want to use over here,
  # Make sure to use an image with the same architecture as the
  # Instance type you are planning to use on your fleets.
  # We require JDK to be installed in the base image, so that
  # it can be used to run the &AGS; Agent
FROM public.ecr.aws/amazoncorretto/amazoncorretto:17-amd64
  #
# Game build directory
# --------------------
  # Add your game build to gamebuild directory and add the zip file name in the
 'GAME_BUILD_ZIP' env variable below.
  # The game build provided over here needs to be integrated with gamelift server sdk.
  # This template assumes that the game build is in a zip format.
ENV GAME_BUILD_ZIP="SimpleServer.zip" \
  #
# Default directory
# -----------------
  # Default directory, the value provided here should be where the game executable
 exists.
  # Provide this same value as your launch path in RuntimeConfiguration when creating a
 fleet.
  # Ref: https://docs.aws.amazon.com/gamelift/latest/apireference/
API_ServerProcess.html
GAME_EXECUTABLE="GameLiftSampleServer" \
HOME_DIR="/local/game" \
  #
  # Registered compute in anywhere fleet (not used in container fleets)
  # ---------------------------------------------------------------------
  # Add the name for the registered compute in an anywhere fleet.
  # This environment variable is required only for anywhere fleets, but not for
 container fleets.
  # If it is set for container fleets, it will be overridden by Gamelift.
  GAMELIFT_COMPUTE_NAME="<ADD_COMPUTE_NAME>" \
  #
# Default Gamelift Agent jar
# --------------------------
GAMELIFT_AGENT_EXEC="GameLiftAgent-1.0.jar" \
  #
  # This env variable defines the name of the S3 bucket that stores the GameLift Agent
 logs.
  # This S3 bucket should exist in the customer AWS account.
  # In order to allow GameLift agent to upload logs to this s3 bucket, customers would
 need to
```

```
  # include s3:PutObject permission in the IAM role provided as instanceRoleArn during
 CreateFleet operation.
GAMELIFT_AGENT_LOGS_BUCKET_NAME="<ADD NAME OF GAMELIFT AGENT LOGS S3 BUCKET>" \
  #
# -----------
  # This env variable defines the name of the S3 bucket that stores the game session
 logs.
  # This S3 bucket should exist in the customer AWS account.
  # In order to allow GameLift agent to upload logs to this s3 bucket, customers would
 need to
  # include s3:PutObject permission in the IAM role provided as instanceRoleArn during
 CreateFleet operation.
# -----------
GAME_SESSION_LOGS_BUCKET_NAME="<ADD NAME OF GAME SESSION LOGS S3 BUCKET>" \
#
# -----------
GAMELIFT_AGENT_LOGS_PATH="/local/game/agentlogs/" \
  #
# NOT USED in container fleets - USED in Anywhere fleets
# ------------------------------------------------------------
  # Specifiy the type of compute resource used to host the game servers.
  # This env variable is required only for anywhere fleets, but not for container
 fleets.
  # If it is set for container fleets, it will be overridden by Gamelift.
#
# -----------
COMPUTE_TYPE="ANYWHERE" \
  #
  # Specify the credential to be used for creating the client.
  # This env variable is required only for anywhere fleets, but not for container
 fleets.
  # If it is set for container fleets, it will be overridden by Gamelift.
#
# -----------
CREDENTIAL_PROVIDER="environment-variable"

USER root

    # intall dependencies as necessary
    RUN yum install -y sudo \
                        unzip \
                        git \
                        shadow-utils \
                        iputils \
```

```
                        tar \
                        gcc \
                        make \
                        openssl-devel \
                        zlib-devel \
                        vim \
                        net-tools \
                        nc \
                        procps

   # Set up the ground for 'gamescale' user
   RUN groupadd -r gamescale -g 500 && \
     useradd -u 500 -r -g gamescale -m -s /sbin/nologin -c "Gamescale user" gamescale
&& \
     echo "gamescale ALL=(ALL) NOPASSWD: ALL" | (EDITOR="tee -a" visudo) && \
     mkdir -p $HOME_DIR && \
     mkdir $HOME_DIR/mono && \
     chown -R gamescale:gamescale $HOME_DIR

   WORKDIR $HOME_DIR

   # extract game build as necessary
   COPY ./gamebuild/$GAME_BUILD_ZIP .
   RUN unzip ./$GAME_BUILD_ZIP -d ./

   # copy Gamelift Agent jar
   COPY ./target/$GAMELIFT_AGENT_EXEC ./

   # Add permissions to game build and gamelift agent jar
   RUN chmod +x ./$GAME_EXECUTABLE
   RUN chmod +x ./$GAMELIFT_AGENT_EXEC

   # Check if java is installed on the image, if not then the Agent will not be able
to run
   RUN java --version

   USER gamescale

   ENV PATH "$PATH:$HOME_DIR/bin:$JAVA_HOME"

   # Change directory to bin
   WORKDIR $HOME_DIR

   # check path before starting the container
```

```
    RUN echo $PATH

    # Create logs directory for GameLift Agent & server processes
    RUN mkdir logs
    RUN mkdir agentlogs

    # Start the GameLift Agent
    ENTRYPOINT sleep 90 && java -jar $GAMELIFT_AGENT_EXEC -ip "192.168.1.1" -gslb
"$GAME_SESSION_LOGS_BUCKET_NAME" -galb "$GAMELIFT_AGENT_LOGS_BUCKET_NAME" -galp
"$GAMELIFT_AGENT_LOGS_PATH" -glc environment-variable
```

> **Note**
>
> 注： Dockerfile の一部の環境変数は、 で上書きできますContainerDefinition。

コンテナイメージを構築するには

1. コンテナイメージを構築します。

   独自の SDK 5.x サーバーを使用している場合

   任意のローカルリポジトリ名を指定できます。

   Example

   ```
   [~/work/glc]$ docker build -t <local repository name>:<optional tag> .
   ```

   SimpleServer サンプルを使用している場合

   Example

   ```
   [~/work/glc]$ docker build -t simple-server:version-1 .
   Successfully built 0123456789012
   Successfully tagged simple-server:version-1
   ```

> **Note**
>
> 次の例では、初期REPOSITORY値として *simpler-server* を使用し、TAG値version-1として を使用します。

2. イメージのリストを表示し、 REPOSITORYと のIMAGE ID値を書き留めます。これらは、以下の手順で必要になります。

   Example

```
[~/work/glc]$ docker images
REPOSITORY       TAG          IMAGE ID        CREATED           SIZE
simple-server    version-1    0123456789012    14 minutes ago    1.24GB
```

コンテナイメージを Amazon ECR にプッシュする

Amazon ECR のプライベートリポジトリにコンテナイメージをアップロードします。コンテナグループ定義を作成するときは、このリポジトリの場所を参照して GameLift 、Amazon がコンテナイメージのスナップショットを作成し、コンテナフリートをデプロイするときに使用できるようにします。

> **Note**
>
> Amazon ECR プライベートリポジトリがまだない場合は、作成します。

Amazon ECR 認証情報を取得するには

- コンテナイメージを Amazon ECR にプッシュする前に、一時的な形式で認証情報を取得し AWS 、Docker に提供する必要があります。Docker がログインできるように、Amazon ECR 認証情報を取得します。

  Example

```
[~/work/glc]$ aws ecr get-login-password --region us-west-2 | docker login --
username AWS --password-stdin aws_account_id.dkr.ecr.us-west-2.amazonaws.com
WARNING! Your password will be stored unencrypted in
```

```
/home/user-name/.docker/config.json.
Configure a credential helper to remove this warning.
See https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

コンテナイメージを Amazon ECR にプッシュするには

1. 使用する Amazon ECR プライベートリポジトリの URI をコピーします。
2. Amazon ECR タグをコンテナイメージに適用します。

   Example

   ```
   [~/work/glc]$ docker tag <IMAGE ID from above> <Amazon ECR private repository
    URI>:<optional tag>
   ```
3. コンテナイメージを Amazon ECR にプッシュする

   Example

   ```
   [~/work/glc]$ docker image push  <Amazon ECR private repository URI>
   ```

Amazon GameLift コンテナフリートの設計

このドキュメントは、パブリックプレビューリリースの機能に関するものです。このドキュメントは変更される可能性があります。

これらのトピックでは、Amazon GameLift コンテナフリートを設定するときに行う重要な決定事項について説明します。決定は、コンテナ、コンテナグループ、フリートの設定方法に影響します。

トピック

- フリートコンテナ構造を設計する
- リソース制限を設定する
- 必須コンテナの指定
- ネットワーク接続を設定する
- コンテナのヘルスチェックを設定する
- コンテナの依存関係を設定する

- コンテナフリートを設定する

フリートコンテナ構造を設計する

最初のステップとして、ゲームサーバーのホストに必要なソフトウェアとリソースを特定します。これには次のようなものがあります。

- ゲームサーバーアプリケーション。アプリケーションは、サーバー SDK バージョン 5 以降を含め、ホスティング用の Amazon GameLift 機能と統合する必要があります。ゲームを Amazon と統合する GameLift を参照してください。
- Amazon GameLift エージェント。このコンピューティング上のエージェントは、Amazon GameLift サービスとの通信を維持し、すべてのゲームサーバープロセスのライフサイクルを管理します。詳細については、「ゲームサーバーと Amazon GameLift エージェント」を参照してください。
- 必要に応じて追加のソフトウェアとリソース。これには、ゲームサーバーアプリケーションの実行に必要なソフトウェアが含まれる場合があります。一般的なサポートソフトウェアは、ログ記録とモニタリング、セキュリティ、コンテンツ配信、データ同期に使用されます。

次に、Amazon GameLift コンテナフリートのソフトウェアとリソースを構築する方法を決定します。Amazon GameLift はコンテナグループを使用してコンテナを整理します。フリートには常に 1 つのレプリカコンテナグループがあり、オプションでデーモンコンテナフリートを持つことができます。詳細については、「コンテナフリートのコンポーネント」を参照してください。

- まず、レプリカコンテナグループを設計します。以下のガイドラインを検討します。
  - ゲームサーバーアプリケーションと Amazon GameLift エージェントを同じコンテナにバンドルします。このコンテナをレプリカグループの唯一の必須コンテナにします。
  - ゲームサーバーの他のすべてのソフトウェアをコンテナに整理します。すべてをレプリカグループ内の 1 つのコンテナに入れることもできます。または、1 つ以上のサイドカーコンテナを作成することもできます。サイドカーを使用する理由には次のようなものがあります。
    - 個々のソフトウェアの起動/シャットダウンシーケンスを設定するには。ソフトウェアを別々のコンテナに配置し、それらの間の依存関係を設定することで、これを実現できます。
    - メモリと CPU 使用率にコンテナ固有の制限を設定するには。
    - 起動コマンド、エントリポイント、作業ディレクトリ、環境変数、ヘルスチェックなど、コンテナごとに異なるコンテナ設定を指定する。
- フリートにデーモンコンテナグループが必要かどうかを判断します。以下の点を考慮します。

- デーモンコンテナは通常、バックグラウンドまたはモニタリングプロセスを実行するために使用されます。
- デーモングループのコンテナはフリートインスタンスにレプリケートされません。つまり、デーモングループのコンテナはレプリカコンテナグループと一緒にスケールされません。
- デーモングループは複数のコンテナを持つことができます。デーモングループ内の任意のコンテナを必須として指定できます。

リソース制限を設定する

コンテナグループごとに、ソフトウェアを実行するためにグループが必要とするメモリと CPU の量を決定します。Amazon GameLift はこの情報を使用して、コンテナグループのリソースを管理します。また、この情報を使用して、フリートイメージが保持できるレプリカコンテナグループの数を計算します。個々のコンテナの制限を設定することもできます。

コンテナのオプションの制限を設定する

コンテナ固有のリソース制限を設定すると、個々のコンテナがグループのリソースをどのように使用できるかを詳細に制御できます。コンテナ固有の制限を設定しない場合、グループ内のすべてのコンテナがグループリソースを共有します。共有することで、必要な場所でリソースをより柔軟に使用できます。また、プロセスが相互に競争し、コンテナに障害が発生する可能性も高くなります。

コンテナに次のいずれかの`ContainerDefinition`プロパティを設定します。

- `SoftLimit` (メモリ) - コンテナの排他的使用のために最小量のメモリを予約します。コンテナには常に予約済みの金額があります。追加のリソースが利用可能な場合、いつでもこの最小値を超える可能性があります。
- `HardLimit` (メモリ) - コンテナの最大メモリ制限を設定します。コンテナがこの制限を超えると、再起動されます。
- `Cpu` 制限 — コンテナの排他的使用のために、最小量の CPU リソースを予約します。コンテナには常に予約済みの金額があります。追加のリソースが利用可能な場合、いつでもこの最小値を超える可能性があります。(1024 CPU ユニットは 1 vCPU に相当します) 。

コンテナグループの合計リソース制限を設定する

各コンテナグループが必要とするメモリと CPU GameLift リソースの量を Amazon に伝えます。目標は、ゲームサーバーのパフォーマンスを最適化するために十分なリソースを割り当てることです。Amazon GameLift はこれらの制限を使用して、フリートインスタンスでレプリカコンテナ

グループをパックする方法を計算します。また、コンテナフリートのインスタンスタイプを選択するときにも使用します。

グループ内の各コンテナ内のすべてのプロセスに必要なメモリと CPU の合計を計算します。以下の点を考慮します。

- コンテナグループ内のすべてのコンテナで実行されるプロセス これらのプロセスに必要なリソースを合計します。
- 各コンテナグループで実行する予定の同時ゲームサーバープロセスはいくつありますか？ この値はフリートのランタイム設定の一部として設定しますが、ここで十分なメモリを計画する必要があります (「ランタイム設定の最適化」を参照)。

コンテナグループ要件の見積もりに基づいて、次のContainerGroupDefinitionプロパティを設定します。

- TotalMemoryLimit – コンテナグループの最大メモリ制限を設定します。グループ内のすべてのコンテナは割り当てられたメモリを共有します。個々のコンテナの制限を設定する場合、合計メモリ制限は次の条件を満たす必要があります。
  - すべてのコンテナのソフトメモリ制限の合計以上
  - グループ内のコンテナの最大ハードメモリ制限以上
- TotalCpuLimit — コンテナグループの CPU 制限を設定します。グループ内のすべてのコンテナは、割り当てられた CPU リソースを共有します。個々のコンテナの制限を設定する場合、合計 CPU 制限は次の条件を満たす必要があります。
  - すべてのコンテナ CPU 制限の合計以上。ベストプラクティスとして、この値をコンテナの CPU 制限の合計の 2 倍に設定することを検討してください。

シナリオの例

次の 3 つのコンテナを持つレプリカコンテナグループを定義するとします。

- コンテナ A は必須のレプリカコンテナです。ゲームサーバープロセスと Amazon GameLift エージェントを実行します。1 つのゲームサーバーのリソース要件は、512 MiB と 1024 CPU と推定されます。コンテナで 10 個のサーバープロセスを実行する予定です。このコンテナは最も重要なソフトウェアを実行するため、6144 MiB のソフトメモリ予約を設定し、ハードメモリ制限や CPU 予約制限はありません。
- コンテナ B は、1024 MiB と 1536 CPU の推定リソース要件を備えたサポートソフトウェアを実行します。ソフトメモリ予約制限を 1024 MiB、ハードメモリ制限を 2048 MiB、CPU 予約制限を 1024 CPU に設定します。
- コンテナ C は、重要でないログ記録やその他のモニタリングユーティリティを実行します。ハードメモリ制限を 512 MiB、CPU 予約制限を 512 CPU に設定します。

この情報を使用して、コンテナグループに次の合計制限を設定します。

- 合計メモリ制限: 7680 MiB。この値は、(1) ソフトメモリ制限 (6144+1024 MiB) の合計、および (2) 最大のハードメモリ制限 (1024 MiB) を超えています。
- 合計 CPU 制限: 13312 CPU。この値は CPU 制限 (1024+512 CPU) の合計を超えています。

必須コンテナの指定

コンテナごとに、コンテナを必須または非必須として指定します。すべてのコンテナグループには、少なくとも 1 つの必須コンテナが必要です。必須コンテナは、ゲームサーバーのホストなど、コンテナグループの重要な作業を行います。必須コンテナは常に実行されていると予想されます。失敗すると、コンテナグループ全体が再起動します。

- フリートのレプリカコンテナグループは、必須コンテナを 1 つだけ持つことができます。このコンテナは Amazon GameLift エージェントを実行し、管理しているゲームサーバープロセスを実行します。
- フリートにデーモンコンテナグループがある場合は、複数の必須コンテナを指定できます。コンテナグループの再起動を求めるコンテナ障害が発生した場合は、デーモンコンテナを必須にします。

各コンテナの `ContainerDefinition`プロパティを true または false `Essential`に設定します。

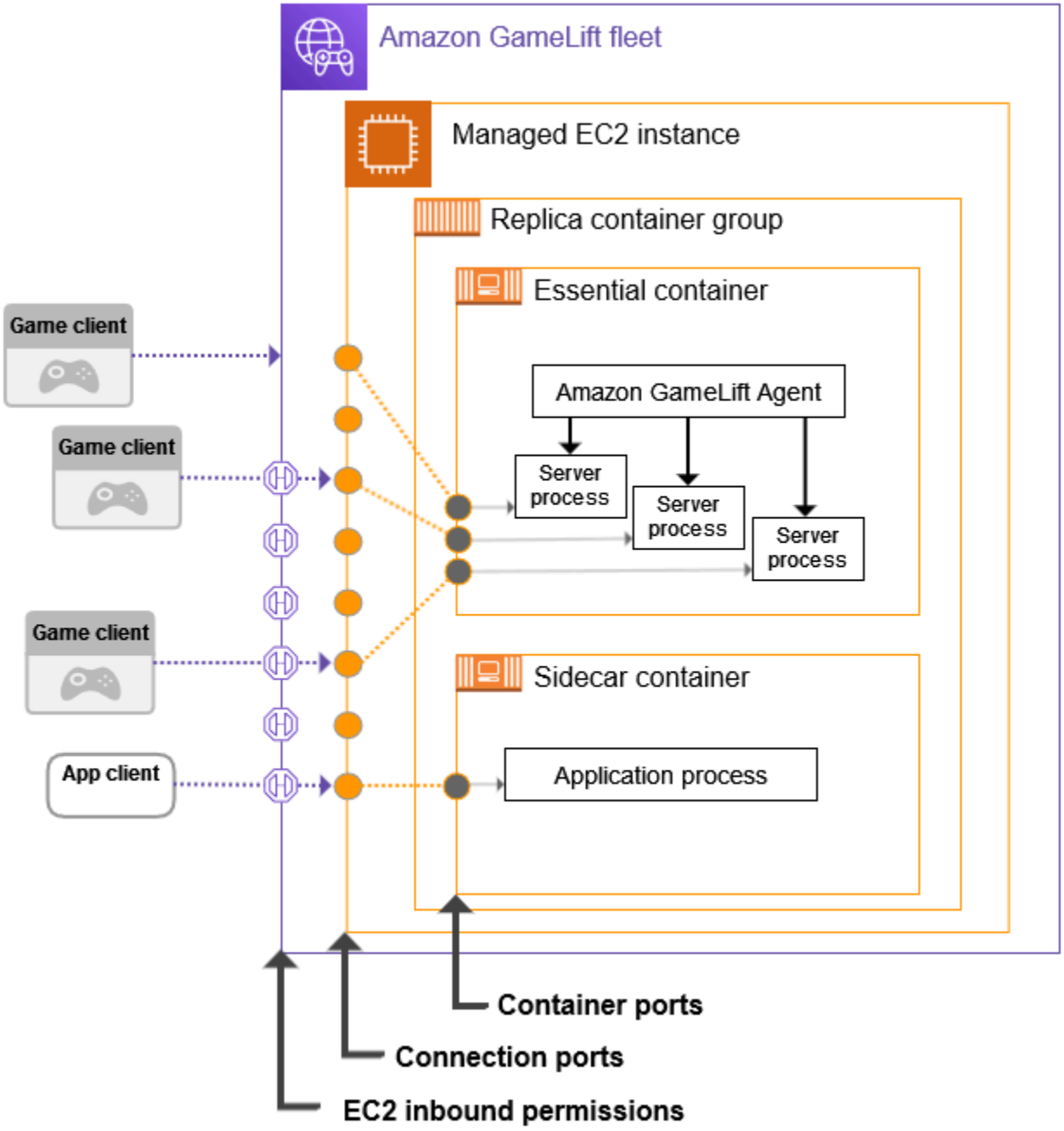
ネットワーク接続を設定する

ネットワークアクセスを確立して、外部トラフィックをコンテナフリート内の任意のコンテナに接続できます。例えば、ゲームクライアントがゲームに参加してプレイできるように、ゲームサーバープロセスを実行するコンテナへのネットワーク接続を確立する必要があります。ゲームクライアントは、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。

コンテナフリートでは、クライアントとサーバー間の接続は直接ではありません。内部的には、コンテナ内のプロセスはコンテナポート をリッスンします。外部的には、着信トラフィックは接続ポート を使用してフリートインスタンスに接続します。Amazon は、内部コンテナポートと外部向け接続ポート間のマッピング GameLift を維持し、着信トラフィックがインスタンスの正しいプロセスにルーティングされるようにします。

Amazon GameLift は、ネットワーク接続をさらに細かく制御します。各コンテナフリートには、各外部向け接続ポートへのアクセスを制御できるインバウンドアクセス許可設定があります。既存のフリートのポート設定を変更することはできませんが、インバウンドアクセス許可を調整することで、

必要に応じてアクセスを許可または制限できます。例えば、すべての接続ポートのアクセス許可を削除して、フリートのコンテナへのすべてのアクセスをシャットダウンできます。



コンテナポート範囲を設定する

外部アクセスを必要とするプロセスに十分なコンテナポートを使用してコンテナ定義を設定します。一部のコンテナはポートを必要としません。その他のプロセスには、必要なすべてのプロセスに 1 つのポートを割り当てるのに十分なポートが必要です。

ゲームサーバーを実行する必須のレプリカコンテナグループには、同時に実行されるゲームサーバープロセスごとにポートが必要です (フリートの で設定) RuntimeConfiguration。ゲームサーバープロセスは、割り当てられたポートをリッスンし、Amazon に報告します GameLift。

コンテナグループ定義を作成するときは、ネットワークアクセスを必要とする各コンテナのコンテナポート範囲を定義します (ContainerDefinitionInput：PortConfiguration を参照)。範囲が、必要な各プロセスにポートを割り当てるのに十分な大きさであることを確認します。プロセスには、コンテナのポート設定でポート番号を割り当てる必要があります。

接続ポート範囲を設定する

接続ポートのセットを使用してコンテナフリートを設定します。接続ポートは、コンテナを実行しているフリートインスタンスへの外部アクセスを提供します。Amazon は接続ポートを GameLift 割り当て、必要に応じてコンテナポートにマッピングします。

コンテナフリートを作成するときは、接続ポート範囲を定義します (ContainerGroupsConfiguration「：ConnectionPortRange」を参照)。範囲に、フリートインスタンスのすべてのコンテナポートにマップするのに十分なポートがあることを確認します。必要な最小接続ポートを計算するには、次の式を使用します。

```
[Total number of container ports defined for containers in the replica container group] * [Number of replica container groups per instance] + [Total number of container ports defined for containers in the daemon container group]
```

ベストプラクティスとして、接続ポートの最小数を 2 倍にします。

> **Note**
>
> 接続ポートの数によって、インスタンスあたりのレプリカコンテナグループの数が制限される可能性があります。フリートにインスタンスごとに 1 つのレプリカコンテナグループに対して十分な接続ポートしかない場合、インスタンスに複数のレプリカコンテナグループに対して十分なコンピューティング能力がある場合でも、Amazon は 1 つのレプリカコンテナグループのみをデプロイ GameLift します。

インバウンドアクセス許可を設定する

インバウンドアクセス許可は、受信トラフィックに対して開く接続ポートを指定することで、コンテナフリートへの外部アクセスを制御します。この設定を使用して、必要に応じてフリートのネットワークアクセスをオンまたはオフにすることができます。

コンテナフリートを作成するときは、一連のインバウンドアクセス許可を定義します (CreateFleet「:EC2InboundPermissions」を参照)。インバウンドアクセス許可ポートのプロパティを設定して、フリートの接続ポート設定の一部またはすべての値を含めます。既存のコンテナフリートのインバウンドアクセス許可を変更するには、 UpdateFleetPortSettingsを呼び出します。

シナリオの例

この例では、3 つのネットワーク接続プロパティをすべて設定する方法を示します。

- フリートのレプリカコンテナグループには、ゲームサーバープロセスを実行するコンテナが 1 つあります。ランタイム設定は、10 個のゲームサーバープロセスを同時に実行するようにコンテナに指示します。

  レプリカコンテナグループ定義では、このコンテナの PortConfigurationパラメータを次のように設定します。

  ```
  "PortConfiguration": {
    "ContainerPortRanges": [ { "FromPort": 10, "ToPort": 20, "Protocol": "TCP"} ]
    }
  ```

- フリートには、1 つのコンテナを持つデーモンコンテナグループもあります。ネットワークアクセスを必要とするプロセスが 1 つあります。デーモンコンテナグループ定義では、このコンテナの PortConfigurationパラメータを次のように設定します。

  ```
  "PortConfiguration": {
    "ContainerPortRanges": [ { "FromPort": 25, "ToPort": 25, "Protocol": "TCP"} ]  }
  ```

- フリートは、フリートインスタンスごとに 3 つのレプリカコンテナグループで構成されます。この情報を考慮すると、次の式を使用して必要な接続ポートの数を計算できます。
  - 最小: 31 ポート [10 個のレプリカコンテナポート * インスタンスあたり 3 個のレプリカコンテナグループ + 1 個のデーモンコンテナポート〕
  - ベストプラクティス: 62 ポート [最小ポート * 2]

  コンテナフリートを作成するときに、 で ConnectionPortRangeパラメータをContainerGroupsConfiguration次のように設定します。

  ```
  "ConnectionPortRange": { "FromPort": 1010, "ToPort": 1071 }
  ```

- 使用可能なすべての接続ポートへのアクセスを許可したいと考えています。コンテナフリートを作成するときに、 EC2InboundPermissionsパラメータを次のように設定します。

```
"EC2InboundPermissions": [
  {"FromPort": 1010, "ToPort": 1071, "IpRange": "10.24.34.0/23", "Protocol":
 "TCP"} ]
```

コンテナのヘルスチェックを設定する

ターミナルに障害が発生した場合、コンテナは自動的に再起動し、実行を停止します。コンテナが必須の場合、コンテナグループ全体が再起動します。

追加のカスタム基準を定義してコンテナのヘルスを測定し、ヘルスチェックを使用してその基準をテストできます。コンテナのヘルスチェックを設定するには、Docker コンテナイメージまたはコンテナ定義で定義できます。コンテナ定義でヘルスチェックを設定すると、コンテナイメージの設定が上書きされます。

コンテナタイプに基づいてオプションのヘルスチェックを次のように設定します。

- 必須のレプリカコンテナの場合は、ヘルスチェックを設定しないでください。Amazon GameLift エージェントは、このコンテナのヘルスレポートを自動的に処理します。
- 重要でないレプリカコンテナやデーモンコンテナの場合は、オプションでヘルスチェックパラメータを設定できます。

コンテナのヘルスチェックに次のContainerDefinitionプロパティを設定します。

- Command — コンテナのヘルス状態の一部を確認するコマンドを指定します。ヘルスの測定に使用する基準を決定します。このコマンドでは、終了値が 1 (異常) または 0 (正常) になる必要があります。
- StartPeriod — ヘルスチェックの失敗がカウントを開始する前に、最初の遅延を指定します。この遅延により、コンテナはプロセスをブートストラップする時間を確保できます。
- Interval — ヘルスチェックコマンドを実行する頻度を決定します。コンテナの障害をどのくらい早く検出して解決しますか？
- Timeout — ヘルスチェックコマンドを再試行する前に、成功または失敗を待機する時間を決定します。ヘルスチェックコマンドの完了にはどのくらいの時間がかかりますか？
- Retries — 障害を登録する前に、ヘルスチェックコマンドを何回再試行すべきですか？

コンテナの依存関係を設定する

各コンテナグループ内では、コンテナのステータスに基づいてコンテナ間の依存関係を設定できます。依存関係は、別のコンテナのステータスに基づいて依存コンテナを開始またはシャットダウンできるタイミングに影響します。

依存関係の主なユースケースは、コンテナグループの起動シーケンスとシャットダウンシーケンスを作成することです。

例えば、コンテナ B と C を開始する前に、コンテナ A を最初に開始して正常に完了させることができます。これを実現するには、まずコンテナ A が正常に完了する必要があるという条件で、コンテナ A のコンテナ B の依存関係を作成します。次に、同じ条件でコンテナ A にコンテナ C の依存関係を作成します。起動シーケンスは、シャットダウンの逆の順序で行われます。

コンテナフリートを設定する

コンテナフリートを作成するときは、次の決定点を考慮してください。これらのポイントのほとんどは、コンテナのアーキテクチャと設定によって異なります。

フリートをデプロイする場所を決定する

一般に、レイテンシーを最小限に抑えるために、プレイヤーの近くに地理的にフリートをデプロイします。コンテナフリートは、Amazon GameLift がサポート AWS リージョン する任意の各にデプロイできます。同じゲームサーバーを別の地理的場所にデプロイする場合は、 AWS リージョン や Local Zones を含むリモートロケーションをフリートに追加できます。マルチロケーションフリートの場合、フリートロケーションごとに個別に容量を調整できます。サポートされているフリートロケーションの詳細については、「」を参照してくださいAmazon GameLift ホスティングロケーション。

フリートのインスタンスタイプとサイズを選択する

Amazon は、さまざまな Amazon EC2 インスタンスタイプ GameLift をサポートしており、それらはすべてコンテナフリートで使用できます。インスタンスタイプの可用性と料金はロケーションによって異なります。サポートされているインスタンスタイプのリストは、Amazon GameLift コンソール (リソース、インスタンス、およびサービスクォータ) で場所別にフィルタリングして表示できます。

インスタンスタイプを選択するときは、まずインスタンスファミリーを検討してください。インスタンスファミリーは、CPU、メモリ、ストレージ、ネットワーク機能のさまざまな組み合わせを提供します。EC2 インスタンスファミリーの詳細をご覧ください。各ファミリーには、選択す

るインスタンスサイズの範囲があります。インスタンスサイズを選択するときは、次の点を考慮してください。

- ワークロードをサポートできる最小インスタンスサイズはどのくらいですか？ この情報を使用して、小さすぎるインスタンスタイプをすべて排除します。
- コンテナアーキテクチャに適したインスタンスタイプのサイズは何ですか？ 理想的には、無駄な領域を最小限に抑えながら、レプリカコンテナグループの複数のコピーに対応できるサイズを選択します。
- ゲームにはどのようなスケーリング粒度が適していますか？ フリート容量のスケーリングにはインスタンスの追加または削除が必要で、各インスタンスは特定の数のゲームセッションをホストする能力を表します。各インスタンスで追加または削除する容量を考慮します。プレイヤーの需要が 1 分から 1 分の間で変化する場合は、数百または数千のゲームセッションをホストできる非常に大きなインスタンスを使用するのが理にかなっているかもしれません。対照的に、より小さなインスタンスタイプで、よりきめ細かなスケーリング制御が必要になる場合があります。
- サイズに基づいてコスト削減は利用できますか？ 特定のインスタンスタイプのコストは、可用性によって場所によって異なる場合があります。

ランタイム設定の最適化

フリートのランタイム設定は、ゲームセッションホスティングのサーバープロセスを実行する方法に関する一連の手順です。これらの手順は、フリート内の各レプリカコンテナグループに Amazon GameLift エージェントによって実装されます。

フリートのランタイム設定は、各レプリカコンテナグループで同時に実行されるサーバープロセスの数を決定します。この設定は、コンテナグループのリソース制限の計算方法と、フリートのインスタンスタイプの選択方法に影響します。フリートを設計するときは、これら 3 つの要素のバランスを取る必要があります。

ランタイム設定の使用方法の詳細については、「」を参照してくださいAmazon GameLift がゲームサーバーを起動する方法を管理する。

その他のオプションフリート設定を設定する

コンテナフリートを設定するときは、次のオプション機能を使用できます。

- 他の AWS リソースにアクセスするようにゲームサーバーを設定します。フリートの他の AWS リソースと通信する を参照してください。
- アクティブなプレイヤーがスケールダウンイベント中に途中で終了しないように、ゲームセッションを保護します。
- 1 人の個人がフリートで作成できるゲームセッションの数を制限します。

Amazon コンテナフリートの GameLift コンテナグループ定義を作成する

このドキュメントは、パブリックプレビューリリースの機能を対象としています。このドキュメントは変更される可能性があります。

コンテナグループ定義は、コンテナ化されたゲームサーバーアプリケーションをコンテナフリートにデプロイする方法を説明します。これは、フリートで実行するコンテナのセットとその実行方法を識別する設計図です。コンテナフリートを作成するときは、フリートにデプロイするコンテナグループ定義を指定します。コンテナグループの詳細については、「」を参照してくださいコンテナフリートのコンポーネント。

開始する前に

以下のタスクを実行します。

- ゲームサーバーをホストするためのコンテナアーキテクチャを設計します。Amazon GameLift コンテナフリートの設計 を参照してください。
- コンテナグループに含めるコンテナ定義を計画します。 AWS CLI を使用している場合は、JSON ファイルにコンテナ定義を作成します。
- 最終的なコンテナイメージを、コンテナグループを作成する予定と同じ AWS リージョン の Amazon Elastic Container Registry (Amazon ECR) レジストリにプッシュします。Amazon は、コンテナグループ定義の作成時に各イメージのスナップショット GameLift を保存し、コンテナフリートにデプロイするときにコピーを使用します。ゲームサーバーソフトウェアでコンテナイメージを準備する を参照してください。
- AWS ユーザーが Amazon ECR リポジトリにアクセスするための IAM アクセス許可を持っていることを確認します。Amazon のユーザーアクセス許可を管理する GameLift を参照してください。少なくとも、次のアクションに対するアクセス許可が必要です。
  - `ecr:DescribeImages`
  - `ecr:BatchGetImage`
  - `ecr:GetDownloadUrlForLayer`

コンテナグループ定義のクローンを作成する

Amazon GameLift コンソールを使用して、既存のコンテナグループ定義のクローンを作成できます。

コンテナグループのクローンを作成するには

1. Amazon GameLift コンソール で、左側のナビゲーションペインに移動し、コンテナグループ を選択します。
2. コンテナグループリストページで、クローンを作成する既存のコンテナグループを選択します。コンテナグループを選択すると、クローンボタンがアクティブになります。
3. [クローンを作成] を選択します。このアクションにより、コンテナグループ作成ウィザードが開き、設定が事前に入力されています。
4. クローンされたコンテナグループの新しい名前を入力します。同じリージョンのコンテナグループには一意の名前が必要です。
5. コンテナグループとコンテナ定義ページをステップスルーし、新しいコンテナグループを確認して作成します。

レプリカコンテナグループ定義を作成する

レプリカコンテナグループは、ゲームサーバーソフトウェアを管理します。レプリカコンテナグループには、Amazon GameLift Agent とゲームサーバープロセスを実行するコンテナが少なくとも 1 つあります。グループには、サポートソフトウェアを実行するための追加の「サイドカー」コンテナがある場合があります。

このトピックでは、Amazon GameLift コンソールまたは AWS CLI ツールを使用してコンテナグループ定義を作成する方法について説明します。コンテナグループ設定の詳細については、「」を参照してくださいAmazon GameLift コンテナフリートの設計。

Console

Amazon GameLift コンソール で、コンテナグループ AWS リージョン を作成する を選択します。

コンソールの左側のナビゲーションバーを開き、コンテナグループ を選択します。「コンテナグループ」ページで、「コンテナグループの作成」を選択します。

ステップ 1: グループの詳細を定義する。

1. コンテナグループ定義名を入力します。この名前は、 AWS アカウント および リージョンに一意である必要があります。コンソールでは、グループ定義が名前でリストされるため、意味のあるラベルを割り当てると便利です。
2. レプリカのスケジューリング戦略を選択します。

3. 合計メモリ制限 には、コンテナグループで使用できる最大メモリを入力します。この値の計算については、「」を参照してくださいリソース制限を設定する。
4. 合計 CPU 制限 には、コンテナグループで使用できる最大コンピューティング能力を入力します。この値の計算については、「」を参照してくださいリソース制限を設定する。

ステップ 2: コンテナ定義を追加する。

ゲームサーバーアプリケーションと Amazon GameLift エージェントでコンテナを定義します。これは必須のレプリカコンテナです。

1. コンテナ定義名 を指定します。グループに定義された各コンテナには、一意の名前の値が必要です。
2. コンテナイメージの Amazon ECR イメージ URI を特定します。次のいずれかの形式を入力します。

   - イメージ URI のみ: `[AWS #####].dkr.ecr.[AWS #####].amazonaws.com/[repository ID]`
   - イメージ URI + ダイジェスト: `[AWS #####].dkr.ecr.[AWS #### #].amazonaws.com/[repository ID]@[digest]`
   - イメージ URI + タグ: `[AWS #####].dkr.ecr.[AWS #####].amazonaws.com/[repository ID]:[tag]`
3. 必須コンテナ の場合、最初のコンテナ定義には Yes が自動的に選択されます。別のコンテナ定義を追加する場合は、定義ごとにこの設定のオンとオフを切り替えることができます。詳細については、「必須コンテナの指定」を参照してください。
4. 1 つ以上の内部コンテナポート範囲 を設定します。このコンテナはゲームサーバーをホストするため、コンテナグループで実行するサーバープロセスごとに十分なポートを持つ範囲を定義します。詳細については、「ネットワーク接続を設定する」を参照してください。
5. オプションの設定 オーバーライド変数と環境変数を使用すると、起動時にコンテナに渡す値を指定できます。ここで設定した値は、コンテナイメージに既に存在する設定を上書きします。
6. オプションのコンテナ制限を設定して、このコンテナのリソース割り当てを管理します。詳細については、「リソース制限を設定する」を参照してください。
7. 必要に応じて、必須ではないコンテナをさらに定義します。

   - コンテナ定義名と ECR イメージ URI を指定します。必須ではないコンテナは Amazon GameLift エージェントを実行しないでください。

- 内部コンテナのポート範囲は、コンテナにネットワークアクセスを必要とするプロセスがある場合にのみ設定します。
- 必要に応じて、コンテナのヘルスチェックを設定します。重要でないコンテナがヘルスチェックに失敗すると、失敗したコンテナの再起動のみを求められます。
- 必要に応じて、オーバーライド、環境変数、リソース割り当ての制限をオプションで設定します。

ステップ 3: 依存関係を設定する。

コンテナグループ定義に複数のコンテナがある場合は、それらの間で依存関係を定義できます。依存関係を使用して、コンテナ条件に基づいて起動シーケンスとシャットダウンシーケンスを設定します。詳細については、「コンテナの依存関係を設定する」を参照してください。

1. 依存関係を追加するコンテナ名を特定します。このコンテナは、依存関係条件が満たされるまで開始されません。
2. 依存関係コンテナ名と条件 を特定します。このコンテナは、依存コンテナを起動する前に条件を満たす必要があります。
3. 必要に応じて追加の依存関係を設定します。どのコンテナにも複数の依存関係を作成できます。循環依存関係の作成は避けてください。

ステップ 4: 確認して作成する。

1. すべてのコンテナグループ定義設定を確認します。コンテナグループ定義の作成後に設定を変更することはできません。編集を使用して、グループの各コンテナ定義を含む任意のセクションを変更します。
2. 確認が完了したら、 の作成 を選択します。

   リクエストが成功すると、コンソールに新しいコンテナグループ定義リソースの詳細ページが表示されます。Amazon がグループのすべてのコンテナイメージのスナップショットの作成 GameLift を開始するためCOPYING、初期状態ではステータスは です。このフェーズが完了すると、コンテナグループ定義のステータスは に変わりますREADY。コンテナグループ定義でコンテナフリートを作成する前に、コンテナグループ定義が READYステータスになっている必要があります。

AWS CLI

AWS CLI を使用してコンテナグループ定義を作成する場合は、コンテナ定義の設定を別のJSONファイルに保持します。CLI コマンドで ファイルを参照できます。スキーマの例コンテナ定義JSONファイルを作成するについては、「」を参照してください。

コンテナグループ定義を作成する

新しいコンテナグループ定義を作成するには、 `create-container-group-definition` CLI コマンドを使用します。このコマンドの詳細については、 CLI コマンドリファレンスcreate-container-group-definitionの「」を参照してください。 AWS

Example : レプリカコンテナグループ

この例は、レプリカコンテナグループ定義のリクエストを示しています。レプリカとデーモンのグループ定義を作成するためのコマンド構造は基本的に同じです。各タイプのグループの具体的な詳細は、個々のコンテナ定義で説明されています。

この例では、このグループのコンテナ定義を使用して JSON ファイルを作成していることを前提としています。

```
aws gamelift create-container-group-definition \
    --name MyAdventureGameContainerGroup \
    --operating-system AMAZON_LINUX_2023 \
    --scheduling-strategy REPLICA \
    --total-memory-limit 4096 \
    --total-cpu-limit 1024 \
    --container-definitions file://SimpleServer.json
```

コンテナ定義JSONファイルを作成する

コンテナグループ定義を作成するときは、グループのコンテナも定義します。コンテナ定義は、コンテナイメージが保存されている Amazon ECR リポジトリ、ネットワークポートのオプション設定、CPU とメモリの使用量の制限、およびその他の設定を指定します。コンテナグループ内のすべてのコンテナの設定を含む単一のJSONファイルを作成することをお勧めします。ファイルのメンテナンスは、これらの重要な設定の保存、共有、バージョン追跡に役立ちます。 AWS CLI を使用してコンテナグループ定義を作成する場合は、 コマンドで ファイルを参照できます。

コンテナ定義を作成するには

1. 新しい.JSONファイルを作成して開きます。例:

```
[~/work/glc]$ vim SimpleServer.json
```

2. グループのコンテナごとに個別のコンテナ定義を作成します。次のサンプルコンテンツをコピーし、コンテナの必要に応じて変更します。コンテナ定義の構文の詳細については、「Amazon GameLift API リファレンスContainerDefinitionInput」の「」を参照してください。
3. AWS CLI コマンドで参照できるように、ファイルをローカルに保存します。

例: 必須レプリカコンテナの定義

Example

この例では、レプリカコンテナグループの必須コンテナについて説明します。必須のレプリカコンテナには、ゲームサーバーアプリケーション、Amazon GameLift エージェントが含まれ、ゲームホスティング用の他のサポートソフトウェアを含めることができます。定義には、名前、イメージ URI、およびポート設定を含める必要があります。この例では、コンテナ固有のリソース制限も設定しています。

```
[
  {
    "ContainerName": "SimpleServer",
    "ImageUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/gl-containers:complex-
server",
    "Essential": true,
    "Cpu": 256,
    "MemoryLimits": {
      "HardLimit": 128
  },
     "PortConfiguration": {
           "ContainerPortRanges": [
              {
                 "FromPort": 2000,
                 "Protocol": "TCP",
                 "ToPort": 2100
              }
           ]
      }
    }
```

```
]
```

Amazon GameLift コンテナフリートを作成する

このドキュメントは、パブリックプレビューリリースの機能に関するものです。このドキュメントは変更される可能性があります。

コンテナグループ定義を作成したら、Amazon GameLift コンソールまたは AWS Command Line Interface（ AWS CLI) を使用してコンテナフリートを作成します。

新しいフリートを作成すると、Amazon GameLift がコンテナグループを各フリートインスタンスにデプロイし、ゲームサーバーを起動するときに、フリートのステータスがいくつかのステージを通過します。フリートのステータスが になるとACTIVE、ゲームセッションをホストする準備が整います。フリート作成の問題については、「Amazon GameLift フリートの問題をデバッグする」を参照してください。

Console

Amazon GameLift コンソール で、フリートを作成する AWS リージョン を選択します。コンテナグループ定義は、フリートを作成するリージョンと同じリージョンにある必要があります。

コンソールの左側のナビゲーションバーを開き、フリート を選択します。フリートページで、フリートの作成を選択します。

ステップ 1: コンピューティングタイプを選択する

- Containers のコンピューティングタイプを選択します。

ステップ 2: フリートの詳細を定義する

1. フリートの詳細セクションに、フリートの名前と説明を入力します。
2. コンテナグループの詳細セクションで、フリートにデプロイするコンテナグループを特定します。レプリカコンテナグループを追加する必要があります。オプションでデーモンコンテナグループを追加できます。各グループはステータス である必要がありますREADY。
3. フリートの接続ポート範囲を設定します。詳細については、「ネットワーク接続を設定する」を参照してください。
4. 必要に応じて、デプロイするインスタンスごとに必要なレプリカを指定します。必要な数を指定するか、Amazon に最大数を GameLift 計算させることができます。計算された最大値

よりも大きい必要数を指定すると、フリートの作成は失敗します。フリートの作成後にこの設定を変更することはできません。レプリカコンテナグループのパッキングの詳細については、「」を参照してください重要な概念。

5. （オプション）「追加の詳細」の下：

   a. インスタンスロール で、ゲームビルド内のアプリケーションがアカウントの他の AWS リソースにアクセスすることを許可する IAM ロールを指定します。詳細については、「フリートの他の AWS リソースと通信する」を参照してください。インスタンスロールを使用してフリートを作成するには、アカウントに IAM PassRole アクセス許可が必要です。詳細については、「Amazon GameLift の IAM アクセス許可の例」を参照してください。

   b. [メトリクスグループ] には、新規または既存のフリートメトリクスグループの名前を入力します。複数のフリートのメトリクスを同じメトリクスグループに追加して、メトリクスを集約できます。

ステップ 3: インスタンスの詳細を定義する

1. インスタンスデプロイ で、インスタンスをデプロイする 1 つ以上のリモートロケーションを選択します。ホームリージョンが自動的に選択されます (これはフリートを作成しているリージョンです)。追加のロケーションを選択すると、フリートインスタンスもこれらのロケーションにデプロイされます。

> **Important**
>
> デフォルトで有効になっていないリージョンを使用するには、 でリージョンを有効にします AWS アカウント。
>
> - 2022 年 2 月 28 日より前に作成した、有効化されていないリージョンのあるフリートは影響を受けません。
> - 新しいマルチロケーションフリートを作成したり、既存のマルチロケーションフリートを更新するには、まず、使用することを選択したリージョンをすべて有効にします。
>
> デフォルトで有効になっていないリージョンとそれを有効にする方法についての詳細は、「AWS 全般のリファレンス」の「AWS リージョンの管理」を参照してください。

2. フリートのインスタンス設定を選択します。コンソールは、必要な最小 vCPU とメモリを自動的に計算します (各コンテナグループに設定した合計制限に基づく)。リソース要件と入力した場所に基づいて、使用可能なインスタンスタイプの完全なリストがフィルタリングされます。必要に応じてフィルターを追加できます。

   インスタンスタイプの選択の詳細については、「コンテナフリートを設定する」を参照してください。選択したインスタンスタイプのサイズは、レプリカコンテナグループが各フリートインスタンスにパックされる方法に影響します。選択に応じて、インスタンスごとに目的のレプリカの設定を確認することを検討してください。

ステップ 4: ランタイムを設定する

ランタイム設定は、ゲームサーバープロセスの開始方法と実行方法を決定します。これらの命令は Amazon GameLift エージェントに渡され、各レプリカコンテナグループと同じ方法で実装されます。を呼び出すことで、フリートのランタイム設定を更新できますUpdateRuntimeConfiguration。

1. 起動パス に、ゲーム実行可能ファイルへのパスを入力します。
2. (オプション) [起動パラメータ] には、ゲーム実行ファイルにコマンドラインパラメータのセットとして渡す情報を入力します。
3. 各レプリカコンテナグループで実行を維持する同時プロセスの数を指定します。インスタンスあたりのサーバープロセス数の Amazon GameLift クォータを確認します。インスタンスあたりの同時サーバープロセスの制限は、すべての設定の同時プロセスの合計に適用されます。フリートの設定が制限を超えると、フリートはアクティブ化されません。
4. ゲームセッションの同時アクティベーションにオプションの制限を設定します。これらの設定により、新しいゲームセッションの開始時に消費されるリソースの量を制限できます。ゲームセッションのアクティベーションは、既存のゲームセッションのパフォーマンスに影響を与える可能性があります。
5. EC2 ポート設定を設定して、外部トラフィックがフリートで実行されているプロセスにアクセスできるようにします。フリートに定義されている接続ポート番号の一部またはすべてを指定します。フリートの作成時にこれらのポートを設定する必要はありませんが、これらのポートがないと、ゲームサーバーに接続できるトラフィックはありません。後でフリートのポート設定を更新するには、 を呼び出します。 UpdateFleetPortSettings
6. ゲームセッションリソース設定で、次のオプション機能を設定します。

a. ゲームスケーリング保護ポリシーをオンまたはオフにします。保護をオンに GameLift すると、インスタンスがアクティブなゲームセッションをホストしている場合、Amazon はスケールダウンイベント中にインスタンスをシャットダウンしません。

b. リソース作成の上限を設定して、指定した期間に 1 人のプレイヤーが作成できるゲームセッションの数を制限します。

ステップ 5: タグを設定する

- (オプション) [キー] と [値] のペアを入力して、ビルドにタグを追加します。[次へ] を選択して、フリート作成のレビューを続行します。

ステップ 6: 確認して作成します。

- フリート設定を確認します。

  フリートのステータスにかかわらず、フリートのメタデータと設定をいつでも更新できます。詳細については、「Amazon GameLift フリートを管理する」を参照してください。フリートキャパシティを更新できるのは、フリートが [アクティブ] ステータスになった後です。詳細については、「Amazon GameLift ホスティングキャパシティのスケーリング」を参照してください。リモートロケーションを追加または削除することもできます。

  確認が完了したら、 の作成 を選択します。

  リクエストが成功すると、コンソールに新しいフリートリソースの詳細ページが表示されます。Amazon がフリート作成プロセス GameLift を開始するとNEW、最初はステータスは になります。新しいフリートのステータスを [フリート] ページで追跡できます。フリートは、ステータス に達すると、ゲームセッションをホストする準備が整いますACTIVE。

AWS CLI

を使用してコンテナフリートを作成するには AWS CLI、コマンドラインウィンドウを開き、`create-fleet` コマンドを使用します。このコマンドの詳細については、「 コマンドリファレンス`create-fleet`」の「」を参照してください。 AWS CLI

以下に示す`create-fleet`リクエスト例では、次の特性を持つ新しいコンテナフリートを作成します。

- ContainerGroupsConfiguration は、単一のレプリカコンテナグループ定義を指定します: `MegaFrogRaceServer.NA.v2`。レプリカグループの 3 つのコピーが各フリートインスタンスにデプロイされます。各インスタンスには、インスタンス上のプロセスへのアクセスに使用できる 30 個の接続ポートがあります。
- フリートは c5.large オンデマンドインスタンスを使用します。
- コンテナグループを次の場所にデプロイします。
  - us-west-2 (ホームリージョン)
  - ca-central-1 (リモートロケーション)
- インスタンスの各レプリカコンテナグループは、5 つのゲームサーバープロセスを同時に実行し、各インスタンスが一度に最大 15 のゲームセッションをホストできるようにします。
- 各レプリカコンテナグループでは、Amazon GameLift は 2 つの新しいゲームセッションを同時にアクティブ化できます。また、アクティブ化されたゲームセッションのうち 300 秒以内にプレイヤーをホストする準備が完了しないものは終了されます。
- このフリートのインスタンスでホストされているすべてのゲームセッションでは、ゲームセッションの保護が有効になっています。
- 各プレイヤーは 15 分以内に 3 つの新しいゲームセッションを作成できます。

```
aws gamelift create-fleet \
    --name SampleFleet123 \
    --description "The sample test fleet" \
    --compute-type "CONTAINER" \
    --container-groups-configuration
 "ContainerGroupDefinitionNames=['MegaFrogRaceServer.NA.v2'],
 DesiredReplicaContainerGroupPerInstance=3,
 ConnectionPortRange={FromPort=1010,ToPort=1040}" \
    --ec2-instance-type c5.large \
    --region us-west-2 \
    --locations "Location=ca-central-1" \
    --fleet-type ON_DEMAND \
    --runtime-configuration "GameSessionActivationTimeoutSeconds=300,
 MaxConcurrentGameSessionActivations=2, ServerProcesses=[{LaunchPath=/local/game/
MegaFrogRace/server.exe,ConcurrentExecutions=5}]" \
    --new-game-session-protection-policy "FullProtection" \
    --resource-creation-limit-policy "NewGameSessionsPerCreator=3,
 PolicyPeriodInMinutes=15" \
    --ec2-inbound-permissions
 "FromPort=1010,ToPort=1040,IpRange=0.0.0.0/0,Protocol=UDP" \
```

フリート作成リクエストが成功すると、Amazon はリクエストした構成設定と新しいフリート ID を含むフリート属性のセット GameLift を返します。 GameLift 次に、Amazon はフリートのステータスとロケーションのステータスを新規に設定し、フリートのアクティベーションプロセスを開始します。フリートのステータスをトラッキングし、他のフリート情報を表示するには、次の CLI コマンドを使用します。

- describe-fleet-events
- describe-fleet-attributes
- describe-fleet-capacity
- describe-fleet-port-settings
- describe-fleet-utilization
- describe-runtime-configuration
- describe-fleet-location-attributes
- describe-fleet-location-capacity
- describe-fleet-location-utilization

以下のコマンドを使用して、必要に応じてフリートの容量と他の設定を変更できます。

- update-fleet-attributes
- update-fleet-capacity
- update-fleet-port-settings
- update-runtime-configuration
- create-fleet-locations
- delete-fleet-locations

Amazon GameLift コンテナフリートの管理

このドキュメントは、パブリックプレビューリリースの機能に関するものです。このドキュメントは変更される可能性があります。

コンテナフリートに関する情報を取得したり、変更を加えたりする場合は、次のアクションを使用してコンテナフリートを管理できます。

リソースを表示する

コンテナフリートのリソースに関する情報を取得する方法はいくつかあります。

- DescribeCompute - コンピューティングとして登録されたコンテナに関する詳細を返します。
- DescribeContainerGroupDefinition - コンテナグループ定義に関する詳細を返します。このリソースでは、グループとそのコンテナの設定方法について説明します。
- DescribeFleetAttributes - 接続ポート範囲およびその他の属性を含むフリート属性を取得します。
- DescribeFleetCapacity - フリート内のレプリカコンテナグループの数とそのステータスを取得します。
- DescribeRuntimeConfiguration - 各レプリカコンテナグループで実行されるサーバープロセスを記述します。
- GetComputeAccess - コンテナグループをホストするインスタンスへのリモートアクセスを提供します。
- GetComputeAuthToken - コンテナフリート内のコンピューティングリソース GameLift の認証トークンを Amazon にリクエストします。
- ListCompute - コンピューティングとして登録されたコンテナグループを一覧表示します。
- ListContainerGroupDefinitions - コンテナグループ定義を一覧表示します。
- ListFleets - 特定のコンテナグループを使用しているフリートを一覧表示します。

リソースの更新

コンテナフリートリソースを作成および変更する方法を次に示します。

- CreateContainerGroupDefinition - コンテナグループ定義を作成します。
- CreateFleet - `ComputeType`が `CONTAINER`に設定されている場合にコンテナフリートを作成します。
- RegisterCompute - コンピューティングをコンテナフリートに登録します。
- UpdateFleetAttributes - Anywhere フリート設定オプションなど、フリートの変更可能な属性を更新します。
- UpdateFleetCapacity - マネージド EC2 フリートまたはコンテナフリートの容量設定を更新します。
- UpdateRuntimeConfiguration - コンピューティングとして登録された各レプリカコンテナグループで実行するサーバープロセスを記述するランタイム設定を更新します。

リソースの削除

コンテナフリートリソースを削除する方法をいくつか次に示します。

- DeleteContainerGroupDefinition - コンテナグループ定義を削除します。
- DeleteFleet - フリートを削除します。
- DeregisterCompute - コンテナフリートからコンピューティングリソースを削除します。

Amazon GameLift コンテナフリートのスケーリング

このドキュメントは、パブリックプレビューリリースの機能を対象としています。このドキュメントは変更される可能性があります。

ゲームホスティングで最も困難なタスクの 1 つが、不要なリソースに無駄なコストを浪費することなく、プレイヤーの需要に合わせてキャパシティをスケーリングすることです。コンテナフリートでは、フリートインスタンスを追加または削除することで、フリート容量をスケーリングします。

新しいフリートを作成すると、Amazon はフリートの希望する容量を 1 つのインスタンス GameLift に設定し、フリートのホームリージョンに 1 つのインスタンスをデプロイします。マルチロケーションフリートの場合、Amazon はホームリージョンと各リモートロケーションに 1 つのインスタンスを GameLift デプロイします。フリートのステータスが に達したらACTIVE、希望する容量を増やしてスケールアップしたり、希望する容量を減らしてスケールダウンしたりできます。

Amazon GameLift スケーリング機能を使用して、キャパシティを手動で変更したり、プレイヤーの需要に基づいて自動スケーリングを設定したりできます。

- ターゲット追跡を使用して自動スケーリングを設定します。ターゲットベースの自動スケーリングを参照してください。
- フリートの容量を手動で変更します。Amazon GameLift フリートのキャパシティを手動で設定する を参照してください。

コンテナフリートをスケーリングするときは、インスタンスを追加または削除すると、ゲームセッションとプレイヤーをホストするフリートの容量にどのように影響するかを検討してください。

- [インスタンスあたりのゲームセッション]
  - インスタンスで実行されている各ゲームサーバープロセスは、1 つのゲームセッションをホストする容量を表します。

- コンテナフリートインスタンスで同時に実行されるゲームセッションの数を計算するには、次の式を使用します。

```
[Game sessions per instance] = [# of processes per replica container group] * [# of
 replica container groups per instance]
```

  - レプリカコンテナグループあたりのプロセスについては、 を呼び出し DescribeRuntimeConfiguration、ゲームサーバープロセスの同時実行数をカウントします。
  - インスタンスあたりのレプリカコンテナグループの場合は、 DescribeFleetAttributesを呼び出して DesiredReplicaContainerGroupPerInstance値を取得します。この値が設定されていない場合は、 MaxReplicaContainerGroupsPerInstance値を使用します。
- インスタンスあたりのプレイヤー
  - 各ゲームセッションで許可するプレイヤースロットの数を決定します。ホスティングソリューションがゲームセッションプレイスメントを処理する方法に応じて、マッチメーキング設定または呼び出しでゲームセッションごとにプレイヤーを定義して、ゲームセッションプレイスメントを開始できます。
  - この式を使用して、コンテナフリートインスタンスでゲームを同時にプレイできるプレイヤーの数を計算します。

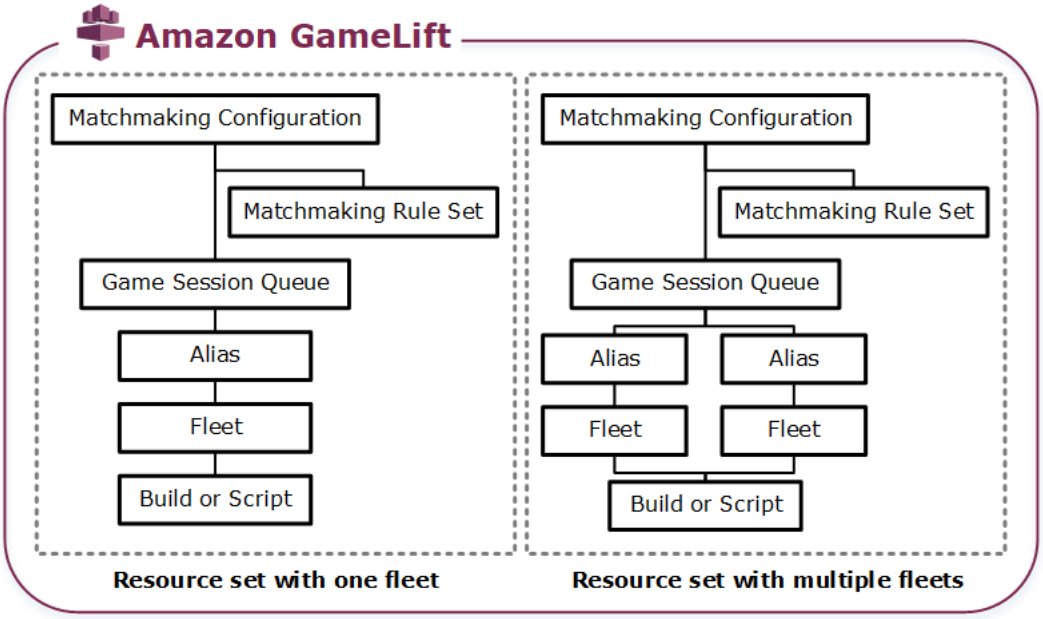
```
[Players per instance] = [# of game sessions per instance] * [# of player slots per
 game session]
```

コンテナフリートの現在の合計容量を取得するには、 DescribeFleetCapacityまたは DescribeFleetLocation 容量を呼び出して、フリート内のレプリカコンテナグループの数を取得します。アクティブグループは、現在ゲームセッションをホストしているグループです。アイドルグループは、新しいゲームセッションをホストする準備が整いました。これらの値を、レプリカコンテナグループあたりのサーバープロセス数で乗算します。

Amazon GameLift ホスティングリソースの管理

このセクションでは、ゲームサーバーを実行し、プレイヤーのゲームセッションをホストするための Amazon GameLift マネージドリソースの設定に関する詳細情報を提供します。リソースを設定してデプロイし、プレイヤーの需要を満たすためにキャパシティをスケーリングし、ゲームセッションをホストするのに利用可能なリソースを見つける必要があります。

次の図は、Amazon GameLift リソースオブジェクトの相互関係を示しています。ビルドまたはスクリプトを使用してフリートを作成し、フリートにエイリアスを与え、エイリアスを使用してフリートをゲームセッションキューに追加します。FlexMatch マッチメイキングを使用するゲームの場合、ゲームセッションキューとマッチメイキングルールセットを使用してマッチメイキング設定を作成します。



ゲームサーバーコード

- ビルド – Amazon GameLift で実行され、プレイヤーのゲームセッションをホストするカスタム構築されたゲームサーバーソフトウェア。ゲームビルドは、特定のオペレーティングシステムでゲームサーバーを実行し、Amazon GameLift に統合する必要のある一連のファイルを表します。フリートをセットアップする予定の AWS リージョン の Amazon GameLiftt サービスにゲームビルドファイルをアップロードします。詳細については、「カスタムゲームサーバー構築を Amazon GameLift にアップロードする」を参照してください。
- スクリプト — リアルタイムサーバーで使用するための設定とカスタムゲームロジック。スクリプトを JavaScript で作成して、ゲームクライアント用にリアルタイムサーバーを設定するために使用します。また、プレイヤーのゲームセッションをホストするカスタムゲームロジッ

クを追加するためにも使用します。詳細については、「リアルタイムサーバースクリプトを Amazon GameLift にアップロードします」を参照してください。

フリート

コンピューティングリソースの集合であり、ゲームサーバーを実行し、プレイヤーのゲームセッションをホストします。フリートをデプロイできるについては、「Amazon GameLift ホスティングロケーション」を参照してください。フリートの作成に関する詳細については、「Amazon GameLift のフリートのセットアップ」を参照してください。

エイリアス

プレイヤーが接続しているフリートをいつでも変更できるフリートの抽象化識別子。詳細については、「Amazon GameLift フリートにエイリアスを追加する」を参照してください。

ゲームセッションキュー

新しいゲームセッションのリクエストを受け取り、新しいセッションをホストするための利用可能なゲームサーバーを検索するゲームセッション配置メカニズムです。ゲームセッションキューの詳細については、「ゲームセッションプレイスメント用の Amazon GameLiftt キューのセットアップ」を参照してください。

Amazon GameLift のビルドとスクリプトのアップロード

Amazon GameLift サービスでホストするためのマルチプレイヤーゲームサーバーをデプロイする前に、ゲームサーバーファイルをアップロードする必要があります。このセクションのトピックでは、カスタムゲーム サーバー 構築ファイルまたはリアルタイムサーバーのサーバースクリプトファイルの準備とアップロードに関するガイダンスを提供します。

トピック

- カスタムゲームサーバー構築を Amazon GameLift にアップロードする
- リアルタイムサーバースクリプトを Amazon GameLift にアップロードします

カスタムゲームサーバー構築を Amazon GameLift にアップロードする

ゲームサーバーを Amazon GameLift と統合したら、ビルドファイルを Amazon GameLift にアップロードします。このトピックでは、ゲームのビルドファイルをパッケージ化し、オプションのびるインストールスクリプトを作成した後、AWS Command Line Interface (AWS CLI) または AWS SDK を使用してそれらのファイルをアップロードする方法について説明します。

トピック

ゲームビルドファイルをパッケージ化する

設定したゲームサーバーを Amazon GameLift にアップロードする前に、ゲームビルドファイルをビルドディレクトリにパッケージ化してください。このディレクトリには、ゲームサーバーの実行と、ゲームセッションのホストを行うために必要なコンポーネントがすべて含まれている必要があります。以下に例を示します。

- [Game server binaries] (ゲームサーバーバイナリ) - ゲームサーバーを実行するために必要なバイナリファイル。ビルドには、同じプラットフォームで実行するように構築された複数のゲームサーバーのバイナリを含めることができます。サポートされているプラットフォームのリストについては、「Amazon での開発サポート GameLift」を参照してください。
- [依存関係] - ゲームサーバー実行ファイルを実行するためのすべての依存関係ファイル。例として、アセット、設定ファイル、依存ライブラリがあります。

> **Note**
>
> C++ 用 Amazon GameLift サーバー SDK で作成されたゲームビルド (Unreal プラグインで作成されたものを含む) には、サーバー SDK を構築したのと同じバージョンの OpenSSL 用の OpenSSL DLL を含めてください。詳細については、サーバー SDK README ファイルを参照してください。

- [インストールスクリプト] (オプション) – Amazon GameLift ホスティングサーバーにゲームビルドをインストールするタスクを処理するスクリプトファイル。このファイルをビルドディレクトリのルートに配置します。Amazon GameLift はフリート作成の一部としてインストールスクリプトを実行します。

インストールスクリプトを含む、ビルド内の任意のアプリケーションについて、AWS の他のサービスのリソースに安全にアクセスするようにセットアップできます。これを行う方法については、「フリートの他の AWS リソースと通信する」を参照してください。

ビルドファイルをパッケージ化したら、ターゲット OS のクリーンインストールでゲームサーバーを実行できることを確認します。これにより、必要な依存関係がすべてパッケージに含められ、インストールスクリプトが正しいことを検証できます。

Amazon GameLift ビルドを作成する

ビルドを作成してファイルをアップロードするときは、いくつかのオプションがあります。

- ファイルディレクトリから構築を作成する。これは、最もシンプルで一般的に使用されるオプションです。
- Amazon Simple Storage Service (Amazon S3) 内のファイルを使用してビルドを作成します。このオプションを使用すると、Amazon S3 でビルドバージョンを管理できます。

いずれの方法でも、Amazon GameLift は、一意のビルド ID と他のメタデータを使用して新しいビルドリソースを作成します。ビルドは [初期化済み] ステータスで開始されます。Amazon GameLift によって正常にゲームサーバーファイルが取得されると、ビルドは [準備完了] ステータスに移行されます。

ビルドの準備ができたら、新しい Amazon GameLift フリートにデプロイできます。詳細については、「Amazon GameLift マネージドフリートを作成する」を参照してください。Amazon GameLift によって新しいフリートが設定されると、ビルドファイルが各フリートインスタンスにダウンロードされ、ビルドファイルがインストールされます。

ファイルディレクトリから構築を作成する

ローカルディレクトリなどの任意のロケーションに保存済みのパッケージゲームビルドを作成するには、`upload-build` AWS CLI コマンドを使用します。このコマンドで Amazon GameLift に新しいビルドレコードを作成し、指定した場所からファイルをアップロードします。

アップロードリクエストを送信します。コマンドラインウィンドウで、upload-build コマンドとパラメータを入力します。

```
aws gamelift upload-build \
    --name user-defined name of build \
    --operating-system supported OS \
    --server-sdk-version Amazon GameLift server SDK version \
    --build-root build path \
    --build-version user-defined build number \
    --region region name
```

- operating-system – ゲームサーバービルドのランタイム環境。OS を指定する必要があります。これを後で更新することはできません。
- server-sdk-version – ゲームサーバーが統合されている Amazon GameLift サーバー SDK のバージョン。値を指定しない場合、Amazon GameLift はデフォルト値の `4.0.2` が使用されます。間違ったサーバー SDK バージョンを指定すると、Amazon GameLift サービスへの接続を確立するために `InitSdk` を呼び出すとき、ゲームサーバービルドが失敗する可能性があります。
- build-root – ビルドファイルのディレクトリパス。
- name – 新しいビルドのわかりやすい名前。
- build-version – ビルドファイルのバージョンの詳細。
- region – ビルドを作成する AWS リージョン。フリートをデプロイする予定のリージョンにビルドを作成します。ゲームを複数のリージョンにデプロイする場合、各リージョンにビルドを作成します。

> **Note**
>
> aws configure get region を使用して現在のデフォルトのリージョンを表示します。デフォルトのリージョンを変更するには、aws configure set region ***region name*** コマンドを使用します。

例

```
aws gamelift upload-build \
    --operating-system AMAZON_LINUX_2023 \

    --server-sdk-version "5.0.0" \
    --build-root "~/mygame" \
    --name "My Game Nightly Build" \
    --build-version "build 255" \
    --region us-west-2
```

```
aws gamelift upload-build \
    --operating-system WINDOWS_2016 \
    --server-sdk-version "5.0.0" \
    --build-root "C:\mygame" \
    --name "My Game Nightly Build" \
    --build-version "build 255" \
    --region us-west-2
```

アップロードリクエストに応じて、Amazon GameLift はアップロードの進行状況を表示します。アップロードが成功すると、Amazon GameLift は新しいビルドレコード ID を返します。アップロードの時間はゲームファイルのサイズおよび接続速度によって異なります。

Amazon S3 内のファイルを使用して構築を作成する

Amazon S3 にビルドファイルを保管し、そしてそこから Amazon GameLift にアップロードすることができます。ビルドを作成するときに S3 バケットのロケーションを指定すると、Amazon GameLift は Amazon S3 から直接ビルドファイルを取得します。

ビルドリソースを作成するには

1. 構築ファイルをAmazon S3 に保存する。パッケージ化されたビルドファイルを含む .zip ファイルを作成し、それを AWS アカウント の S3 バケットにアップロードします。バケットラベル付けとファイル名をメモしておきます。Amazon GameLift ビルドを作成するときに必要になります。
2. Amazon GameLift にビルドファイルへのアクセス権を付与します。「Amazon S3 でゲームビルドファイルにアクセスする」の指示に従って IAM ロールを作成します。ロールを作成したら、新しいロールの Amazon リソースネーム (ARN) をメモしておきます。ビルドを作成するときにこれが必要になります。
3. ビルドを作成します。Amazon GameLift コンソールまたは AWS CLI を使用して新しいビルドレコードを作成します。「Amazon GameLift の IAM アクセス許可の例」の説明の通り、`PassRole` アクセス許可が必要です。

Console

1. [Amazon GameLift コンソール] のナビゲーションペインで、[ホスティング]、[ビルド] を選択します。
2. [ビルド] ページで [ビルドを作成] を選択します。
3. [ビルドを作成] ページの [ビルド設定] で、次の操作を行います。
   a. [名前] にスクリプト名を入力します。
   b. [バージョン] に、バージョンを入力します。ビルドのコンテンツは更新できるので、バージョンデータは更新の追跡に役立ちます。
   c. [オペレーティングシステム (OS)] では、ゲームサーバービルドの OS を選択します。この値を後で更新することはできません。

d. [ゲームサーバービルド] では、Amazon S3 にアップロードしたビルドオブジェクトの S3 URI を入力し、[オブジェクトのバージョン] を選択します。Amazon S3 URI とオブジェクトのバージョンを覚えていない場合は、[S3 の参照] を選択し、ビルドオブジェクトを検索します。

e. [IAM ロール] では、Amazon GameLift に S3 バケットとビルドオブジェクトへのアクセスを許可するために作成したロールを選択します。

4. (オプション) [タグ] に [キー] と [値] のペアを入力して、ビルドにタグを追加します。

5. [Create] (作成) を選択します。

Amazon GameLift は、ID を新しいビルドに割り当て、指定した zip ファイルをアップロードします。[ビルド] ページでは、ステータスを含めて新しいビルドを確認できます。

AWS CLI

新しいビルドを定義し、サーバービルドファイルをアップロードするには、`create-build` コマンドを使用します。

1. コマンドラインウィンドウを開き、AWS CLI を使用できるディレクトリに切り替えます。

2. 次の create-build コマンドを入力します。

```
aws gamelift create-build \
    --name user-defined name of build \
    --server-sdk-version Amazon GameLift server SDK version \
    --operating-system supported OS \
    --build-version user-defined build number \
    --storage-location "Bucket"=S3 bucket label,"Key"=Build .zip file
 name,"RoleArn"=Access role ARN} \
    --region region name
```

- name – 新しいビルドのわかりやすい名前。
- server-sdk-version – ゲームサーバーを Amazon GameLift と統合するために使用した Amazon GameLift サーバー SDK のバージョン。値を指定しない場合、Amazon GameLift はデフォルト値の `4.0.2` が使用されます。
- operating-system – ゲームサーバービルドのランタイム環境。OS を指定する必要があります。これを後で更新することはできません。
- build-version – ビルドファイルのバージョンの詳細。ゲームサーバーの新しいバージョンごとに新しいビルドリソースが必要になるため、この情報は役に立ちます。

- storage-location
  - Bucket – ビルドを含む S3 バケットの名前。例:「my_build_files」。
  - Key – ビルドファイルを含む .zip ファイルの名前。例:「my_game_build_7.0.1, 7.0.2」。
  - RoleARN – 作成した IAM ロールに割り当てられた ARN。例:「arn:aws:iam::111122223333:role/GameLiftAccess」 ポリシーの例については、Amazon S3 でゲームビルドファイルにアクセスするを参照してください。
- region – フリートをデプロイする予定の AWS リージョンにビルドを作成します。ゲームを複数のリージョンにデプロイする場合、各リージョンにビルドを作成します。

> **Note**
>
> configure get コマンド を使用して現在のデフォルトリージョンを確認することをおすすめします。デフォルトのリージョンを変更するには、configure set コマンドを使用します。

例

```
aws gamelift create-build \
    --operating-system WINDOWS_2016 \
    --storage-location
 "Bucket"="my_game_build_files","Key"="mygame_build_101.zip","RoleArn"="arn:aws:iam::111
gamelift" \
    --name "My Game Nightly Build" \
    --build-version "build 101" \
    --region us-west-2
```

3. 新しいビルドを表示するには、describe-build コマンドを使用します。

構築ファイルを更新する

Amazon GameLift コンソールまたは update-buildAWS CLI コマンドを使用して、ビルドリソースのメタデータを更新できます。

Amazon GameLift ビルドを作成した後は、それに関連するビルドファイルを更新することはできません。新しいファイルセットごとに、新しい Amazon GameLift ビルドを作成します。upload-build コマンドを使用して、Amazon GameLift はリクエストごとに新しいビルドレコードを自動的

に作成します。`create-build` コマンドを使用してビルドファイルを指定する場合は、新しいビルドの .zip ファイルを別の名前で Amazon S3 にアップロードし、その新しいファイル名を参照してビルドドを作成します。

更新したビルドをデプロイする場合は、次のヒントを試してみてください。

- 必要に応じて、キューを使用し、フリートを交換します。Amazon GameLift でゲーム クライアントを設定するときは、フリートの代わりにキューを指定します。キューを使用する場合は、新しいビルドの新しいフリートをキューに追加し、古いフリートを削除します。詳細については、「ゲームセッションプレイスメント用の Amazon GameLiftt キューのセットアップ」を参照してください。
- エイリアスを使用して、新しいゲームのビルドにプレイヤーを移行します。ゲーム クライアントを Amazon GameLift に統合するときは、フリート ID の代わりにフリートエイリアスを指定します。詳細については、「Amazon GameLift フリートにエイリアスを追加する」を参照してください。
- ビルドの自動更新を設定します。Amazon GameLift デプロイをビルドシステムに組み込むためのサンプルスクリプトと情報については、AWS ゲームテックブログの「Amazon GameLift へのデプロイの自動化」を参照してください。

ビルドインストールスクリプトを追加する

ゲームビルドのオペレーティングシステム (OS) 用のインストールスクリプトを作成します。

- Windows: 「`install.bat`」という名前のバッチファイルを作成します。
- Linux: 「`install.sh`」という名前のシェルスクリプトファイルを作成します。

インストールスクリプトを作成するときは、次の点に留意してください。

- このスクリプトはユーザー入力を一切受け付けません。
- Amazon GameLift はビルドをインストールし、以下のロケーションのホスティングサーバー上のビルドパッケージにファイルディレクトリを再作成します。
  - Windows フリート: `C:\game`
  - Linux フリート: `/local/game`
- Linux フリートのインストールプロセスで、run-as ユーザーは、インスタンスのファイル構造へのアクセスが制限されます。このユーザーは、ビルドファイルがインストールされているディレクトリに対しては完全な権限があります。インストールスクリプトが管理者アクセス許可を必要とする

アクションを実行する場合は、sudo を使用して管理者アクセスを指定します。Windows フリートの run-as ユーザーには、デフォルトで管理者アクセス許可があります。インストールスクリプトに関連するアクセス許可の失敗により、スクリプトに問題があることを示すイベントメッセージが生成されます。

- Amazon GameLift では、bash などの一般的なシェルインタープリタ言語がサポートされています。shebang (`#!/bin/bash` など) を、インストールスクリプトの先頭に追加します。目的のシェルコマンドのサポートについて確認するには、アクティブな Linux インスタンスにリモートにアクセスし、シェルコマンドを開きます。詳細については、「Amazon GameLift フリートインスタンスにリモート接続する」を参照してください。
- インストールスクリプトは VPC ピアリング接続に依存できません。VPC ピアリング接続は、Amazon GameLift がフリートインスタンスにビルドをインストールするまで使用できません。

Example Windows インストール bash ファイル

この `install.bat` ファイルの例では、ゲームサーバーに必要な Visual C++ ランタイムコンポーネントをインストールし、結果をログファイルに書き込みます。スクリプトはルートのビルドパッケージにコンポーネントファイルを含めます。

```
vcredist_x64.exe /install /quiet /norestart /log c:\game\vcredist_2013_x64.log
```

Example Linux インストールシェルスクリプト

この `install.sh` ファイルの例では、インストールスクリプトで bash を使用し、結果をログファイルに書き込みます。

```
#!/bin/bash
echo 'Hello World' > install.log
```

この `install.sh` ファイルの例では、Amazon CloudWatch エージェントを使用してシステムレベルおよびカスタムのメトリクスを収集し、ログローテーションを処理する方法を示します。Amazon GameLift はサービス VPC で実行されているため、Amazon GameLift がユーザーに代わって AWS Identity and Access Management (IAM) ロールを引き受けるためのアクセス許可を与える必要があります。Amazon GameLift がロールを引き受けることができるようにするには、AWS マネージドポリシー `CloudWatchAgentAdminPolicy` を含むロールを作成し、フリートを作成するときにそのロールを使用します。

```
sudo yum install -y amazon-cloudwatch-agent
```

```
sudo yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-
latest-7.noarch.rpm
sudo yum install -y collectd
cat <<'EOF' > /tmp/config.json
{
    "agent": {
        "metrics_collection_interval": 60,
        "run_as_user": "root",
        "credentials": {
            "role_arn": "arn:aws:iam::account#:role/rolename"
        }
    },
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                    {
                        "file_path": "/tmp/log",
                        "log_group_name": "gllog",
                        "log_stream_name": "{instance_id}"
                    }
                ]
            }
        }
    },
    "metrics": {
       "namespace": "GL_Metric",
        "append_dimensions": {
            "ImageId": "${aws:ImageId}",
            "InstanceId": "${aws:InstanceId}",
            "InstanceType": "${aws:InstanceType}"
        },
        "metrics_collected": {
            // Configure metrics you want to collect.
            // For more information, see Manually create or edit the CloudWatch agent
 configuration file.
        }
    }
}
EOF
sudo mv /tmp/config.json /opt/aws/amazon-cloudwatch-agent/bin/config.json
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -
m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json
```

```
sudo systemctl enable amazon-cloudwatch-agent.service
```

リアルタイムサーバースクリプトを Amazon GameLift にアップロードします

ゲームにリアルタイムサーバーをデプロイする準備ができたら、完了したリアルタイムサーバースクリプトファイルを Amazon GameLift にアップロードします。これは、Amazon GameLift スクリプトリソースを作成し、スクリプトファイルの場所を指定して行います。既存のスクリプトリソースの新しいファイルをアップロードして、すでにデプロイされているサーバースクリプトファイルを更新することもできます。

新しいスクリプトリソースを作成すると、Amazon GameLift は一意のスクリプト ID (`script-1111aaaa-22bb-33cc-44dd-5555eeee66ff` など) を割り当て、スクリプトファイルのコピーをアップロードします。アップロードの時間はスクリプトファイルのサイズおよび接続速度によって異なります。

スクリプトリソースを作成すると、Amazon GameLift は新しいリアルタイムサーバーフリートでスクリプトをデプロイします。Amazon GameLift は、サーバースクリプトをフリート内の各インスタンスにインストールし、スクリプトファイルを `/local/game` に配置します。

サーバースクリプトに関するフリートのアクティブ化の問題をトラブルシューティングする方法については、「Amazon GameLift フリートの問題をデバッグする」を参照してください。

スクリプトファイルをパッケージ化する

サーバースkクリプトでは、1 つ以上のファイルを組み合わせて単一 .zip のファイルにまとめ、アップロードすることができます。.zip ファイルには、スクリプトが実行に必要なすべてのファイルが含まれている必要があります。

zip スクリプトファイルは、ローカルファイルディレクトリまたは Amazon Simple Storage Service (Amazon S3) バケットのいずれかに保存することができます。

ローカルディレクトリからスクリプトファイルをアップロードする

スクリプトファイルをローカルに保存している場合は、そこからスクリプトファイルを Amazon GameLift にアップロードできます。スクリプトリソースを作成するには、Amazon GameLift コンソールまたは AWS Command Line Interface (AWS CLI) のいずれかを使用します。

Amazon GameLift console

スクリプトリソースを作成するには

1. [Amazon GameLift コンソール] を開きます。
2. ナビゲーションペインで [ホスティング]、[スクリプト] を選択します。
3. [スクリプト] ページで [スクリプトを作成] を選択します。
4. [スクリプトを作成] ページの [スクリプト設定] で、次の操作を行います。
   a. [名前] にスクリプト名を入力します。
   b. (オプション) [バージョン] にバージョン情報を入力します。スクリプトのコンテンツは更新できるため、バージョンデータは、更新を追跡する場合に便利です。
   c. [スクリプトソース] で [.zip ファイルをアップロード] を選択します。
   d. [スクリプトファイル] では、[ファイルを選択] を選択し、スクリプトを含む.zip ファイルを参照して、そのファイルを選択します。
5. (オプション) [タグ] に [キー] と [値] のペアを入力して、スクリプトにタグを追加します。
6. [Create] (作成) を選択します。

   Amazon GameLift は、ID を新しいスクリプトに割り当て、指定した zip ファイルをアップロードします。新しいスクリプト (例: ステータス) は、コンソールの [スクリプト] ページで確認できます。

AWS CLI

`create-script` AWS CLI コマンドを使用して、新しいスクリプトを定義し、サーバースクリプトファイルをアップロードします。

スクリプトリソースを作成するには

1. この zip ファイルを、AWS CLI を使用できるディレクトリに配置します。
2. コマンドラインウィンドウを開き、zip ファイルを配置したディレクトリに切り替えます。
3. 次の create-script コマンドとパラメータを入力します。--zip-file パラメータで、文字列 `fileb://` を zip ファイルの名前に必ず追加します。Amazon GameLift が圧縮されたコンテンツを処理できるように、ファイルはバイナリとして識別されます。

```
aws gamelift create-script \
    --name user-defined name of script \
```

```
    --script-version user-defined version info \
    --zip-file fileb://name of zip file \
    --region region name
```

例

```
aws gamelift create-script \
    --name "My_Realtime_Server_Script_1" \
    --script-version "1.0.0" \
    --zip-file fileb://myrealtime_script_1.0.0.zip \
    --region us-west-2
```

リクエストに応じて、Amazon GameLift より、新しいスクリプトオブジェクトが返ります。

4. 新しいスクリプトを表示するには、describe-script を呼び出します。

Amazon S3 からスクリプトファイルをアップロードする

スクリプトファイルは Amazon S3 バケットに保存する、そしてそこから Amazon GameLift にアップロードできます。スクリプトを作成する場合、S3 バケットのロケーションを指定すると、Amazon GameLift によって、スクリプトファイルは Amazon S3 から取得されます。

スクリプトリソースを作成するには

1. スクリプトファイルを S3 バケットに保存します。サーバースクリプトファイルを含む.zip ファイルを作成し、それを管理しているAWS アカウント 内の S3 バケットにアップロードします。オブジェクト URI をメモします。Amazon GameLift スクリプトを作成するときに必要になります。

   > **Note**
   >
   > Amazon GameLift は、ピリオド (.) を含む名前の S3 バケットからのアップロードをサポートしていません。

2. Amazon GameLift にスクリプトファイルへのアクセス権を付与します。サーバースクリプトを含む S3 バケットへのアクセスを Amazon GameLift に許可する AWS Identity and Access Management IAM ロールを作成するには、Amazon の IAM サービスロールを設定する GameLift の手順に従ってください。新しいロールを作成したら、名前を書き留めておきます。これはスクリプトの作成時に必要になります。

3. [Create a script].(スクリプトを作成します)。Amazon GameLift コンソールまたは AWS CLI を使用して新しいスクリプトレコードを作成します。このリクエストを実行するには、「Amazon GameLift の IAM アクセス許可の例」に記載のとおり、IAM `PassRole` アクセス許可が必要です。

Amazon GameLift console

1. [Amazon GameLift コンソール] のナビゲーションペインで、[ホスティング]、[スクリプト] を選択します。
2. [スクリプト] ページで [スクリプトを作成] を選択します。
3. [スクリプトを作成] ページの [スクリプト設定] で、次の操作を行います。
   a. [名前] にスクリプト名を入力します。
   b. (オプション) [バージョン] にバージョン情報を入力します。スクリプトのコンテンツは更新できるため、バージョンデータは、更新を追跡する場合に便利です。
   c. [スクリプトソース] では [Amazon S3 URI] を選択します。
   d. Amazon S3 にアップロードしたスクリプトオブジェクトの S3 URI を入力し、[オブジェクトバージョン] を選択します。Amazon S3 URI とオブジェクトのバージョンを覚えていない場合は、[S3 の参照] を選択し、スクリプトオブジェクトを検索します。
4. (オプション) [タグ] に [キー] と [値] のペアを入力して、スクリプトにタグを追加します。
5. [Create] (作成) を選択します。

   Amazon GameLift は、ID を新しいスクリプトに割り当て、指定した zip ファイルをアップロードします。新しいスクリプト (例: ステータス) は、コンソールの [スクリプト] ページで確認できます。

AWS CLI

`create-script` AWS CLI コマンドを使用して、新しいスクリプトを定義し、サーバースクリプトファイルをアップロードします。

1. コマンドラインウィンドウを開き、AWS CLI を使用できるディレクトリに切り替えます。
2. 次の create-script コマンドとパラメータを入力します。--storage-location パラメータでは、スクリプトファイルの Amazon S3 バケットの場所を指定します。

```
aws gamelift create-script \
```

```
    --name [user-defined name of script] \
    --script-version [user-defined version info] \
    --storage-location "Bucket"=S3 bucket name,"Key"=name of zip file in S3
 bucket,"RoleArn"=Access role ARN \
    --region region name
```

例

```
aws gamelift create-script \
    --name "My_Realtime_Server_Script_1" \
    --script-version "1.0.0" \
    --storage-location "Bucket"="gamelift-
script","Key"="myrealtime_script_1.0.0.zip","RoleArn"="arn:aws:iam::123456789012:role/
S3Access" \
    --region us-west-2
```

リクエストに応じて、Amazon GameLift より、新しいスクリプトオブジェクトが返ります。

3. 新しいスクリプトを表示するには、describe-script を呼び出します。

スクリプトファイルを更新する

Amazon GameLift コンソールまたは update-scriptAWS CLI コマンドを使用して、スクリプトリソースのメタデータを更新できます。

スクリプトリソースのスクリプトコンテンツを更新することもできます。Amazon GameLift は、更新されたスクリプトリソースを使用するすべてのフリートインスクリプトコンテンツをスクリプトデプロイします。更新されたスクリプトがデプロイされると、新しいゲームセッションの開始時に、インスタンスはそれを使用します。更新時にすでに実行されているゲームセッションは更新済みスクリプトを使用しません。

スクリプトファイルを更新するには

- ローカルに保存されたスクリプトファイルの場合、更新されたスクリプト .zip ファイルをアップロードするには、Amazon GameLift コンソールまたは update-script コマンドを使用します。
- Amazon S3 バケットに保存されたスクリプトファイルの場合、更新されたスクリプトファイルを S3 バケットにアップロードします。Amazon GameLift は、更新されたスクリプトファイルを定期的にチェックし、S3 バケットから直接取得します。

Amazon GameLift のフリートのセットアップ

このセクションでは、Amazon GameLift で使用するフリートの設計、構築、保守に関する詳細情報を提供します。Amazon GameLift フリートを使用して、カスタムゲームサーバーとリアルタイムサーバーをデプロイすることができます。

フリートは、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスの集合または物理ハードウェアとしてのホスティングリソースを表します。フリートの口ケーションによって、プレイヤーのゲームセッションをホストするために、インスタンスまたはハードウェアがデプロイされる場所が決まります。フリートのサイズ、およびサポートできるゲームセッションとプレイヤーの数は、提供するインスタンスの数またはハードウェアの量によって決まります。仮想インスタンスの調整は、手動または自動スケーリングを使用して行うことができます。

本番稼働の多くのゲームは、複数のフリートを使用します。例えば、複数のバージョンのゲームサーバーを同時に実行したり、スポットフリートのバックアップキャパシティを提供したり、または冗長性を備えて構築するため、複数のフリートを使用できます。

ゲームのニーズを満たす設計のフリートを作成する方法を学ぶには、Amazon GameLift フリート設計ガイド から開始します。フリートが実行されたら、Amazon GameLift ホスティングキャパシティのスケーリング、Amazon GameLift フリートにエイリアスを追加する、および ゲームセッションプレイスメント用の Amazon GameLiftt キューのセットアップ を参照してください。

トピック

- Amazon GameLift フリート設計ガイド
- 新しい Amazon GameLift フリートを作成する
- Amazon GameLift フリートを管理する
- Amazon GameLift フリートにエイリアスを追加する
- Amazon GameLift フリートの問題をデバッグする
- Amazon GameLift フリートインスタンスにリモート接続する

Amazon GameLift フリート設計ガイド

この設計ガイドでは、Amazon GameLift で使用するホスティングリソースのフリートを作成する際のベストプラクティスについて説明します。ホスティングリソースの組み合わせを選択し、それらの組み合わせをゲームに適した設定にする方法について説明します。

トピック

- Amazon GameLift コンピューティングリソースの選択
- Amazon GameLift がゲームサーバーを起動する方法を管理する
- Amazon でのスポットインスタンスの使用 GameLift

Amazon GameLift コンピューティングリソースの選択

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するために、Amazon GameLift はインスタンス または物理ハードウェアと呼ばれる Amazon Elastic Compute Cloud (Amazon EC2) リソースを使用します。 インスタンスを使って新しいフリートを設定する際に、必要なインスタンスのタイプとそれらでゲームサーバープロセスを実行する方法を決定します。マネージド EC2 フリートがアクティブでゲームセッションをホストする準備ができたら、プレイヤーの需要に対応するために必要に応じてインスタンスを追加または削除できます。

Amazon GameLift ゲームサーバーは、次の 2 つのコンピューティングタイプの組み合わせでデプロイできます。

- [マネージド EC2] – マネージド EC2 フリートは 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。Amazon はインスタンス GameLift を管理し、ハードウェアとソフトウェアの管理の負担をゲームのホストから取り除きます。
- Amazon GameLift Anywhere – Amazon GameLift Anywhere フリートは既存のインフラストラクチャを使用してゲームサーバーをホストし、Amazon はマッチメーキングとキュー GameLift を管理します。

フリートにコンピューティングリソースを選択するときは、次の要素を考慮してください。

- 使用可能なハードウェア
- フリートのロケーション
- オンデマンドインスタンスとスポットインスタンスの比較
- オペレーティングシステム
- インスタンスのタイプ
- Service Quotas

使用可能なハードウェア

導入環境における既存のインフラストラクチャを検討してください。ゲームを Amazon に移行する間も GameLift、インフラストラクチャを引き続き使用できます。Amazon では GameLift

Anywhere、Amazon GameLift マネージド EC2 インスタンスとともに独自のインフラストラクチャを使用できます。また、既存のインフラストラクチャを使用して、サポートされている Amazon GameLift ロケーションで許可されているよりもプレイヤーの近くでゲームをホストすることもできます。Amazon GameLift Anywhere フリートの設定の詳細については、「」を参照してくださいAmazon GameLift Anywhere フリートを作成する。

フリートのロケーション

ゲームサーバーをデプロイする予定の地理的な場所を検討してください。インスタンスタイプの可用性は、 AWS リージョン とローカルゾーンによって異なります。

複数のロケーションで使用しているフリートの場合、インスタンスの可用性とクォータに関する情報は、フリートのホームリージョンと選択したリモートロケーションの組み合わせによって異なります。フリートのロケーションの詳細については、「Amazon GameLift ホスティングロケーション」を参照してください。

Amazon GameLift Anywhere フリートの場合、物理ハードウェアの場所を決定します。カスタムロケーションの詳細については、「Amazon GameLift Anywhere」を参照してください。

オンデマンドインスタンスとスポットインスタンスの比較

Amazon EC2 オンデマンドインスタンスとスポットインスタンスは同じハードウェアとパフォーマンスを提供しますが、可用性とコストは異なります。

オンデマンドインスタンス

オンデマンドインスタンスは必要なときに取得し、必要な期間維持しておくことができます。オンデマンドインスタンスのコストは固定であり、使用時間に応じて支払いが発生します。長期契約はありません。

スポットインスタンス

スポットインスタンスは、未使用の AWS コンピューティング容量を利用することで、オンデマンドインスタンスに代わるコスト効率の高い代替手段を提供できます。スポットインスタンスの料金は、各ロケーションの各インスタンスタイプの需要と供給に応じて変動します。 は、容量を戻す必要があるたびにスポットインスタンスを中断 AWS できます。Amazon GameLift はキューと FleetIQ アルゴリズムを使用して、 AWS がスポットインスタンスを中断すると判断し、インスタンスをリサイクル状態にします。次に、インスタンスにアクティブなゲームセッションがない場合、Amazon はそれを置き換え GameLift ようとします。

スポットインスタンスを使用する方法の詳細については、「Amazon でのスポットインスタンスの使用 GameLift」を参照してください。

オペレーティングシステム

Amazon GameLift インスタンスは、Microsoft Windows または Amazon Linux で実行されるゲームサーバービルドをサポートします。ゲームビルドを Amazon にアップロードするときは GameLift、ゲームのオペレーティングシステムを指定します。Amazon EC2 フリートを作成してゲームビルドをデプロイすると、Amazon はビルドのオペレーティングシステムを使用してインスタンス GameLift を自動的にセットアップします。サポートされるゲームサーバーのオペレーティングシステムの詳細については、「Amazon での開発サポート GameLift」を参照してください。

Amazon GameLift Anywhere フリートを使用する場合、ハードウェアがサポートする任意のオペレーティングシステムを使用できます。Amazon GameLift Anywhere フリートでは、Amazon を使用してリソースを 1 か所で管理しながら、ゲームビルドをハードウェア GameLift にデプロイする必要があります。

インスタンスのタイプ

Amazon EC2 Fleet のインスタンスのタイプによって、各インスタンスで使用されるハードウェアの種類が決まります。インスタンスタイプによって、コンピューティング能力、メモリ、ストレージ、ネットワーキング機能など、提供される組み合わせが異なります。

ゲームに対して使用可能なインスタンスタイプから選択するときは、次の要素を考慮してください。

- ゲームサーバーのコンピューティングアーキテクチャ: x64 または Arm (AWS Graviton)。

  > **Note**
  >
  > Graviton Arm インスタンスには、Linux OS 上に構築された Amazon GameLift サーバーが必要です。C++ と C# には、サーバー SDK 5.1.1 以降が必要です。Go にはサーバー SDK 5.0 以降が必要です。これらのインスタンスは、Amazon Linux 2023 (AL2023) または Amazon Linux 2 (AL2) でのモノラルインストール out-of-the-box をサポートしていません。

- ゲームサーバービルドのコンピューティング、メモリ、およびストレージ要件。
- インスタンスごとに実行する予定のサーバープロセスの数。

より大きいインスタンスタイプを使用すると、各インスタンスで複数のサーバープロセスを実行できます。これにより、プレイヤーの需要を満たすのに必要なインスタンス数を減らすことができます。

詳細については:

- インスタンスタイプについては、「Amazon EC2 インスタンスタイプ」を参照してください。
- インスタンスごとに複数のプロセスを実行する方法については、「Amazon GameLift がゲームサーバーを起動する方法を管理する」を参照してください。

Service Quotas

Amazon のデフォルトのサービスクォータと GameLiftの現在のクォータを確認するには AWS アカウント、次の手順を実行します。

- Amazon の一般的なサービスクォータ情報については GameLift、「」の「Amazon GameLift エンドポイントとクォータ」を参照してくださいAWS 全般のリファレンス。
- アカウントのロケーションごとに使用可能なインスタンスタイプのリストについては、Amazon コンソールの Service Quotas ページを開きます。 GameLift このページには、各ロケーションの各インスタンスタイプに関するアカウントの現在の使用状況も表示されます。
- リージョンごとのインスタンスタイプのアカウントの現在のクォータのリストについては、AWS Command Line Interface （ AWS CLI) コマンド を実行しますdescribe-ec2-instance-limits。このコマンドは、デフォルトリージョン (または指定した別のリージョン) にあるアクティブなインスタンスの数を返します。

ゲームを起動する準備ができたら、Amazon GameLift コンソール で起動アンケートに入力します。Amazon GameLift チームは起動アンケートを使用して、ゲームの正しいクォータと制限を決定します。

Amazon GameLift がゲームサーバーを起動する方法を管理する

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するようにマネージド EC2 フリートのランタイム設定を定義できます。これにより、ホスティングリソースがより効率的に使用されます。

フリートが複数のプロセスを管理する方法

Amazon GameLift はフリートのランタイム設定を使用して、各インスタンスで実行するプロセスのタイプと数を決定します。1 つのランタイム設定には、少なくとも 1 個のゲームサーバー実行可能ファイルを表す 1 つのサーバープロセス設定が含まれます。追加のサーバープロセス設定を定義して、ゲームに関連する他のタイプのプロセスを実行することができます。各サーバープロセス設定には、以下の情報が含まれています。

- ゲームビルドの実行可能ファイルのファイル名とパス。

- (オプション) 起動時にサーバープロセスに渡すパラメータ。
- 同時に実行するプロセスの数。

フリートのインスタンスがアクティベートされると、ランタイム設定で定義された一連のサーバープロセスが起動します。複数のプロセスで、Amazon GameLift は各プロセスの起動をずらします。サーバープロセスの存続時間は限られています。プロセスが終了すると、Amazon GameLift は新しいプロセスを起動して、ランタイム設定で定義されているサーバープロセスの数とタイプを維持します。

ランタイム設定は、サーバープロセス設定を追加、変更、削除することでいつでも変更できます。各インスタンスは定期的にフリートのランタイム設定の更新をチェックし、変更を実装します。Amazon GameLift がランタイム設定の変更を適用する方法は以下のとおりです。

1. インスタンスは Amazon GameLift に最新バージョンのランタイム設定についてリクエストを送信します。
2. インスタンスはアクティブなプロセスを最新のランタイム設定と比較し、次の操作を行います。
   - 更新されたランタイム設定によりサーバープロセスタイプが削除された場合、このタイプのアクティブなサーバープロセスは、終了するまで実行され続けます。インスタンスはこれらのサーバープロセスを置き換えるものではありません。
   - 更新されたランタイム設定によりサーバープロセスタイプの同時プロセス数が減る場合、このタイプの過剰なサーバープロセスは、終了するまで実行され続けます。インスタンスがこれらのサーバープロセスを置き換えることはありません。
   - 更新されたランタイム設定により新しいタイプのサーバープロセスが追加されるか、既存のタイプの同時プロセスが増える場合、インスタンスは新しいサーバープロセスを Amazon GameLift の最大数まで開始します この場合、インスタンスは、既存のプロセスが終了したときに新しいサーバープロセスを起動します。

フリートを複数のプロセスに合わせて最適化する

フリートで複数のプロセスを使用するには、次の操作を実行します。

- フリートにデプロイしようとするゲームサーバー実行可能ファイルを含む ビルドを作成 し、Amazon GameLift にそのビルドをアップロードします。ビルド内のすべてのゲームサーバーは同じプラットフォームで実行され、Amazon GameLift Server SDK を使用する必要があります。
- ひとつ以上のサーバープロセス設定と複数の同時プロセスのあるランタイム設定を作成します。
- ゲームクライアントを AWS SDK バージョン 2016-08-04 以降に統合します。

フリートのパフォーマンスを最適化するには、次の操作を実行することをお勧めします。

- Amazon GameLift がプロセスを効率よくリサイクルするように、サーバープロセスシャットダウンシナリオを処理します。例:
  - サーバー API `ProcessEnding()` を呼び出すシャットダウン手順をゲームサーバーコードに追加する。
  - Amazon GameLift からの終了リクエストを処理するために、コールバック関数 `OnProcessTerminate()` をゲームサーバーコードに実装します。
- Amazon GameLift が異常なサーバープロセスをシャットダウンし、再起動することを確認します。ゲームサーバーコードに `OnHealthCheck()` コールバック関数を実装することで、Amazon GameLift にこのヘルスステータスの報告を返します。Amazon GameLift は、異常報告が 3 回連続なされたサーバープロセスを自動的にシャットダウンします。`OnHealthCheck()` を実装しない場合、Amazon GameLift は、プロセスが通信への応答に失敗しない限り、サーバープロセスが正常であるとみなします。

インスタンスごとのプロセスの数を選択する

インスタンスで実行する同時プロセスの数を決定するときは、以下に注意してください。

- Amazon GameLift は各インスタンスの 同時プロセスの最大数 を制限します。フリートのサーバープロセス設定のすべての同時プロセスの合計数がこのクォータを超えることはできません。
- 許容可能なパフォーマンスレベルを維持するため、Amazon EC2 インスタンスタイプによっては、同時に実行できるプロセスの数が制限される場合があります。ゲームでさまざまな設定を試して、選択したインスタンスタイプにとって適切なプロセス数を見つけます。
- Amazon GameLift は、設定した合計数を超える同時プロセスを実行しません。つまり、以前のランタイム設定から新しい設定への移行は段階的に行われる可能性があります。

Amazon でのスポットインスタンスの使用 GameLift

yourAmazon GameLift マネージド EC2 フリートを設定するときは、スポットインスタンス、オンデマンドインスタンス、または組み合わせを使用できます。Amazon が でスポットインスタンス GameLift を使用する方法の詳細をご覧ください[オンデマンドインスタンスとスポットインスタンスの比較]。スポットフリートを使用するには、ゲーム統合でこのページに記載されている調整を行う必要があります。

マッチメーキング FlexMatch に を使用していますか？ マッチメイキングを配置する既存のゲームセッションキューにスポットフリートを追加できます。

1. スポットインスタンスのゲームセッションキューを設計します。

   キューを使用したゲームセッションプレイスメントの管理はベストプラクティスであり、スポットインスタンスを使用する場合は必須です。キューを設計するには、以下の点を考慮します。

   - ロケーション – 最高のプレイヤーエクスペリエンスを実現するには、プレイヤーに地理的に近いロケーションを選びます。
   - インスタンスタイプ – ゲームサーバーのハードウェア要件と、選択した場所でのインスタンスの可用性を考慮します。

   スポットの可用性と復元性を最適化するキューの設計については、「チュートリアル:スポットインスタンスのゲームセッションキューを設定する」を参照してください。

2. スポット最適化キューのフリートを作成します。

   キューの設計に基づいて、ゲームサーバーを希望の場所とインスタンスタイプにデプロイするフリートを作成します。新しいフリートの作成および設定のヘルプについては、「Amazon GameLift マネージドフリートを作成する」を参照してください。

3. ゲームセッションキューを作成します。

   フリートの送信先を追加し、ゲームセッションの配置プロセスを設定し、配置の優先順位を定義します。新しいキューの作成および設定のヘルプについては、「ゲームセッションキューを作成する」を参照してください。

4. キューを使用するようにゲームクライアントサービスを更新します。

   ゲームクライアントがキューを使用してリソースをリクエストすると、キューは中断される可能性が高いリソースを避け、定義した優先順位に合ったロケーションを選択します。ゲームクライアントにゲームセッション配置を実装する方法については、「ゲームセッションを作成する」を参照してください。

5. ゲームサーバーでスポットの中断を処理するように更新します。

   AWS は、容量を戻す必要があるときに、2 分間の通知でスポットインスタンスを中断できます。プレイヤーへの影響を最小限に抑えるため、中断を処理するようにゲームサーバーをセットアップします。

   がスポットインスタンスを AWS 再利用する前に、終了通知を送信します。Amazon GameLift は、Amazon GameLift Server SDK コールバック関数 を呼び出して、影響を受けるすべてのサーバープロセスに通知を渡しますonProcessTerminate()。このコールバックを実装して、

ゲームセッションを終了するか、ゲームセッションとプレイヤーを新しいインスタンスに移動します。onProcessTerminate() の実装のヘルプについては、「サーバープロセスのシャットダウン通知に応答する」トラブルシューティングを参照してください。

> **Note**
>
> AWS は、インスタンスを再利用する前に通知を提供するようあらゆる努力をしますが、警告が到着する前に がスポットインスタンスを AWS 再利用する可能性があります。ゲームサーバーで予期しない中断を処理できるように準備します。

6. スポットフリートとキューのパフォーマンスを評価します。

   Amazon GameLift コンソールまたは Amazon で Amazon GameLift メトリクスを表示 CloudWatch して、パフォーマンスを確認します。Amazon GameLift メトリクスの詳細については、「」を参照してくださいAmazon CloudWatch で Amazon GameLift をモニタリングする。主なメトリクスには次のものがあります。

   - 中断率 – スポットフリートのインスタンスとゲームセッションについてスポットに関連した中断の回数と頻度を追跡する InstanceInterruptions および GameSessionInterruptions メトリクスを使用します。によって再利用されるゲームセッション AWS のステータスは TERMINATED、ステータスの理由は ですINTERRUPTED。
   - キューの有効性 – プレイスメントの成功率、平均待機時間、キューの深度などのキューメトリクスを追跡し、スポットフリートによってキューのパフォーマンスに影響が出ないことを確認します。
   - フリート使用率 – インスタンス、ゲームセッションおよびプレイヤーセッションをモニタリングします。オンデマンドフリートの使用率は、中断を避けるためにキューがスポットフリートへのプレイスメントを避けている指標になります。

新しい Amazon GameLift フリートを作成する

新しいフリートを作成し、カスタムゲームサーバービルドまたはリアルタイムサーバーをホスティング用にデプロイします。Amazon GameLift にアップロードしたゲームビルドまたはスクリプトリソースならどれでもデプロイできます。

トピック

- Amazon GameLift のフリート作成の仕組み
- Amazon GameLift マネージドフリートを作成する

- Amazon GameLift Anywhere フリートを作成する

Amazon GameLift のフリート作成の仕組み

新しいフリートを作成すると、Amazon GameLift が、フリートの口ケーションごとに 1 つの Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを持つフリートを作成するワークフローを開始します。Amazon GameLift がワークフローの各ステップを完了すると、フリートはイベントを発行し、Amazon GameLift はフリートのステータスを更新します。Amazon GameLift コンソールを使用するか、Amazon GameLift API オペレーション DescribeFleetEvents を呼び出すことで、すべてのイベントを追跡できます。個々の口ケーションのステータスを追跡するには、DescribeFleetLocationAttributes も使用します。

EC2 フリート作成のワークフロー:

- Amazon GameLift は、フリートのホームリージョンと、フリートで定義された各リモート口ケーションにフリートリソースを作成します。
- Amazon GameLift は、必要なキャパシティを 1 つのインスタンスに設定します。
- Amazon GameLift はフリートと口ケーションのステータスを [新規] に設定します。
- Amazon GameLift はフリートイベントログへのイベントの書き込みを開始します。
- Amazon GameLift は、要求されたコンピューティングリソースを各フリート場所の 1 つの新しいインスタンスに割り当てます。
- Amazon GameLift が、ゲームサーバーファイルを各インスタンスにダウンロードし、フリートステータスを [ダウンロード中] に設定します。
- Amazon GameLift は、各インスタンスでダウンロードされたゲームサーバーファイルを評価し、ダウンロード中にエラーが発生していないこと検証します。Amazon GameLift はフリートステータスを [検証中] に設定しています。
- Amazon GameLift は各インスタンスにゲームサーバーを構築し、フリートステータスを [構築中] に設定します。
- Amazon GameLift は、フリートのランタイム設定の指示に従って、各インスタンスでサーバープロセスの起動を開始します。フリートがインスタンスごとに複数の同時サーバープロセスを実行するように設定されている場合、Amazon GameLift はプロセスを数秒だけずらします。各プロセスがオンラインになると、準備状況が Amazon GameLift に報告されます。Amazon GameLift はフリートのステータスを [アクティベート中] に設定します。
- Amazon GameLift は、サーバーがレポートの準備状況を報告するときに、フリートのステータスと口ケーションのステータスを [アクティブ] に設定します。

Amazon GameLift Anywhere fleet creation

- Amazon GameLift がフリートリソースを作成します。フリートのホームリージョンおよびフリートで定義されている各カスタムロケーションでは、Amazon GameLift が、フリートのホームリージョンおよびロケーションステータスを [新規] に設定します。
- Amazon GameLift はフリートイベントログへのイベントの書き込みを開始します。
- フリート内の 1 つのサーバープロセスが Amazon GameLift に準備完了を通知すると、Amazon GameLift はフリートのステータスとロケーションのステータスを [アクティブ] に設定します。他のフリートのロケーションのサーバープロセスが準備状況を報告すると、Amazon GameLiftは、各フリートのロケーションのステータスを [アクティブ] に設定します。

フリート作成に関する問題のトラブルシューティングについては、「Amazon GameLift フリートの問題をデバッグする」を参照してください。

Amazon GameLift マネージドフリートを作成する

マネージドフリートを作成するには、[Amazon GameLift コンソール] または AWS Command Line Interface (AWS CLI) を使用します。

新しいマネージド EC2 フリートを作成した後は、Amazon GameLift がフリートをデプロイし、ゲームサーバーをインストールして起動するまでに、フリートのステータスがいくつかの段階を経ます。フリートが `ACTIVE` ステータスに達すると、ゲームセッションをホストする準備ができています。フリート作成の問題については、「Amazon GameLift フリートの問題をデバッグする」を参照してください。

Console

マネージド EC2 フリートを作成するには

1. [Amazon GameLift コンソール] のナビゲーションペインで、[フリート] を選択します。
2. [フリート] ページで、[フリートの作成] を選択します。
3. [マネージド EC2] を選択します。
4. [フリートの詳細] ページで、以下の操作を実行します。

    a. [名前] にフリート名を入力します。フリート名にフリートタイプ (スポットまたはオンデマンド) を含めることをお勧めします。これにより、フリートを一覧表示するときにフリートタイプを簡単に識別できます。

    b. [説明] には、フリートの簡単な説明を入力します。

c. バイナリタイプ] では、[ビルド または [スクリプト] を選択して、Amazon GameLift がこのフリートにデプロイするゲームサーバータイプを定義します。

d. アップロードされたスクリプトまたはビルドのドロップダウンリストから、[スクリプト] または [ビルド] を選択します。

5. (オプション) 以下の [その他の詳細] の下にあります。

a. [インスタンスロール] には、ゲームビルドのアプリケーションがアカウント内の他の AWS リソースにアクセスすることを許可する IAM ロールを指定します。詳細については、「フリートの他の AWS リソースと通信する」を参照してください。インスタンスロールを使用してフリートを作成するには、アカウントに IAM `PassRole` アクセス許可が必要です。詳細については、「Amazon GameLift の IAM アクセス許可の例」を参照してください。

CloudWatch エージェントなど、サーバー実行ファイルではないアプリケーションを認証する場合は、共有の認証情報オプションを有効にします。

これらの設定は、フリート作成後に更新できません。

b. [証明書の生成] には、Amazon GameLift にフリートの TLS 証明書を生成させることを選択します。フリートの TLS 証明書を使用して、ゲームクライアントが接続時にゲームサーバーに認証するようにでき、すべてのクライアント/サーバー通信を暗号化できます。TLS 対応のフリートのインスタンスごとに、Amazon GameLift によってこの証明書を使用して新しい DNS エントリも作成されます。これらのリソースを使用して、ゲームの認証と暗号化を設定します。

c. [メトリクスグループ] には、新規または既存のフリートメトリクスグループの名前を入力します。複数のフリートのメトリクスを同じメトリクスグループに追加して、メトリクスを集約できます。

フリートの作成後にメトリクスグループを更新することはできません。

6. [Next] (次へ) をクリックします。

7. [ロケーションを選択] ページで、インスタンスをデプロイする追加のリモートロケーションを 1 つ以上選択します。ホームリージョンは、コンソールにアクセスしているリージョンに基づいて自動的に選択されます。追加のロケーションを選択すると、フリートインスタンスもこれらのロケーションにデプロイされます。

> **Important**
>
> デフォルトで有効になっていないリージョンを使用するには、AWS アカウント でそのリージョンを有効にします。
>
> - 2022 年 2 月 28 日より前に作成した、有効化されていないリージョンのあるフリートは影響を受けません。
> - 新しいマルチロケーションフリートを作成したり、既存のマルチロケーションフリートを更新するには、まず、使用することを選択したリージョンをすべて有効にします。
>
> デフォルトで有効になっていないリージョンとそれを有効にする方法についての詳細は、AWS 全般のリファレンス の「AWS リージョン の管理」を参照してください。

8. [Next] (次へ) をクリックします。
9. [インスタンスの詳細を定義する] ページで、

   a. このフリートの [オンデマンド] インスタンスまたは [スポットインスタンス] を選択します。フリートタイプの詳細については、「オンデマンドインスタンスとスポットインスタンスの比較」を参照してください。

   b. [フィルターアーキテクチャ] メニューから [x64] または [Arm] を選択します。

   > **Note**
   >
   > Graviton Arm インスタンスには Linux OS 上に構築された Amazon GameLift サーバーが必要です。C++ と C# には、サーバー SDK 5.1.1 以降が必要です。Go にはサーバー SDK 5.0 以降が必要です。これらのインスタンスでは、Amazon Linux 2023 (AL2023) または Amazon Linux 2 (AL2) へのモノラルインストールに対する追加設定なしでのサポートは提供していません。

   Amazon EC2 Arm アーキテクチャについては、「AWS Graviton プロセッサ」と「Amazon EC2 インスタンスタイプ」を参照してください。

Amazon GameLift でサポートされているインスタンスタイプについては、[CreateFleet() リクエストパラメータ] の `EC2InstanceType` 値を参照してください。

10. リストから Amazon EC2 インスタンスタイプを選択します。インスタンスタイプの選択の詳細については、「インスタンスのタイプ」を参照してください。フリートを作成した後に、そのインスタンスタイプを変更することはできません。
11. [Next] (次へ) をクリックします。
12. [ランタイムを設定] ページの [ランタイム設定] で、次の操作を行います。

    a. [起動パス]には、ビルドまたはスクリプトのゲーム実行可能ファイルへのパスを入力します。Windows インスタンスでは、ゲームサーバーはパス`C:\game` に構築されます。Linux インスタンスでは、ゲームサーバーは `/local/game` に構築されます。例: **`C:\game\MyGame\server.exe`**、**`/local/game/MyGame/server.exe`**、または **`MyRealtimeLaunchScript.js`**。

    b. (オプション) [起動パラメータ] には、ゲーム実行ファイルにコマンドラインパラメータのセットとして渡す情報を入力します。例えば、**`+sv_port 33435 +start_lobby`** などです。

    c. [同時プロセス] では、フリートの各インスタンスで同時に実行するサーバープロセスの数を選択します。同時サーバープロセス数については、Amazon GameLift の [制限] をチェックしてください。

    インスタンスあたりの同時サーバープロセスの制限は、すべての設定の同時プロセスの合計に適用されます。フリートの設定が制限を超えると、フリートはアクティブ化されません。
13. [ゲームセッションアクティベーション] で、このフリート内のインスタンスで新しいゲームセッションをアクティベートする際の制限を指定します。

    a. [最大の同時ゲームセッションアクティベーション] では、1 つのインスタンスで同時にアクティブ化できるゲームセッションの数を入力します。この制限は、複数の新しいゲームセッションを起動することでインスタンスで実行中の他のゲームセッションのパフォーマンスに影響が出る可能性がある場合に役立ちます。

    b. [新しいアクティベーションタイムアウト] には、セッションがアクティブ化されるまでの待ち時間を入力します。ゲームセッションがタイムアウト前に `ACTIVE` ステータスに移行しない場合、Amazon GameLift はゲームセッションのアクティベーションを終了します。
14. (オプション) [EC2 ポート設定] で、次の操作を行います。

    a. [ポート設定の追加] を選択して、フリートにデプロイされているサーバープロセスに接続するインバウンドトラフィックに対するアクセス許可を定義します。

b. [タイプ] には、[カスタム TCP] または [カスタム UDP] を選択します。

c. [ポート範囲] には、インバウンド接続を許可するポート番号の範囲を入力します。ポートの範囲は nnnnn[-nnnnn] 形式で、1026〜60000 の値を使用する必要があります。例: **1500** または**1500-20000**。

d. [IP アドレス範囲] には、IP アドレスの範囲を入力します。CIDR 表記を使用します。例: **0.0.0.0/0** (この例は、接続を試行するすべてのユーザーにアクセスを許可します)。

15. (オプション) [ゲームセッションリソース設定] では、次の操作を行います。

a. [ゲームスケーリング保護ポリシー] では、スケーリング保護をオンまたはオフにします。Amazon GameLift は、保護のあるインスタンスがアクティブなゲームセッションをホストしている場合、スケールダウンイベントの間に終了することはありません。

b. [リソース作成制限] には、ポリシー期間中にプレイヤーが作成できるゲームセッションの最大数を入力します。

16. [Next] (次へ) をクリックします。

17. (オプション) [キー] と [値] のペアを入力して、ビルドにタグを追加します。[次へ] を選択して、フリート作成のレビューを続行します。

18. [Create] (作成) を選択します。Amazon GameLift は、ID を新しいフリートに割り当て、フリートのアクティベーションプロセスを開始します。新しいフリートのステータスを [フリート] ページで追跡できます。

フリートのステータスにかかわらず、フリートのメタデータと設定をいつでも更新できます。詳細については、「Amazon GameLift フリートを管理する」を参照してください。フリートキャパシティを更新できるのは、フリートが [アクティブ] ステータスになった後です。詳細については、「Amazon GameLift ホスティングキャパシティのスケーリング」を参照してください。リモートロケーションを追加または削除することもできます。

AWS CLI

AWS CLI でフリートを作成するには、コマンドラインウィンドウを開き、`create-fleet` コマンド使用します。`create-fleet` コマンドの詳細については、AWS CLI コマンドリファレンスの「`create-fleet`」を参照してください。

次の `create-fleet` リクエスト例では、以下のような特徴を持つ新しいフリートを作成します。

- フリートは、選択したゲームビルドに適したオペレーティングシステムで c5.large オンデマンドインスタンスを使用します。

- 指定されたゲームサーバービルドをデプロイします。このためには、次のロケーションで [準備完了] ステータスである必要があります。
  - us-west-2 (ホームリージョン)
  - sa-east-1 (リモートロケーション)
- TLS 証明書の生成が有効になっています。
- フリートの各インスタンスはゲームサーバーの同一プロセスを十個同時に処理するため、各インスタンスで最大十個のゲームセッションを同時にホストできます。
- 各インスタンスで、Amazon GameLift が同時アクティブ化を許可する新しいゲームセッションは 2 つです。また、アクティブ化されたゲームセッションのうち 300 秒以内にプレイヤーをホストする準備が完了しないものは終了されます。
- このフリートのインスタンスでホストされているすべてのゲームセッションでは、ゲームセッションの保護が有効になっています。
- 各プレイヤーは 15 分以内に 3 つの新しいゲームセッションを作成できます。
- このフリートでホストされている各ゲームセッションには、指定された IP アドレスおよびポート範囲内にある接続ポイントがあります。
- Amazon GameLift は、このフリートのメトリクスを EMEAfleets メトリクスグループに追加します。これは、(この例では) EMEA リージョンのすべてのフリートのメトリクスを結合します。

```
aws gamelift create-fleet \
    --name SampleFleet123 \
    --description "The sample test fleet" \
    --ec2-instance-type c5.large \
    --region us-west-2 \
    --locations "Location=sa-east-1" \
    --fleet-type ON_DEMAND \
    --build-id build-92f061ed-27c9-4a02-b1f4-6f85b2385620 \
    --certificate-configuration "CertificateType=GENERATED" \
    --runtime-configuration "GameSessionActivationTimeoutSeconds=300,
 MaxConcurrentGameSessionActivations=2, ServerProcesses=[{LaunchPath=C:\game
\Bin64.dedicated\MultiplayerSampleProjectLauncher_Server.exe, Parameters=+sv_port
 33435 +start_lobby, ConcurrentExecutions=10}]" \
    --new-game-session-protection-policy "FullProtection" \
    --resource-creation-limit-policy "NewGameSessionsPerCreator=3,
 PolicyPeriodInMinutes=15" \
```

```
    --ec2-inbound-permissions
"FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"
"FromPort=33235,ToPort=33235,IpRange=0.0.0.0/0,Protocol=UDP" \
    --metric-groups  "EMEAfleets"
```

フリート作成のリクエストが成功すると、Amazon GameLift はリクエストした構成設定と新しいフリート ID を含む一連のフリート属性を返します。次に Amazon GameLiftはフリートアクティベーションのプロセスを開始し、フリートのステータスを [新規] に設定します。フリートのステータスをトラッキングし、他のフリート情報を表示するには、次の CLI コマンドを使用します。

- describe-fleet-events
- describe-fleet-attributes
- describe-fleet-capacity
- describe-fleet-port-settings
- describe-fleet-utilization
- describe-runtime-configuration
- describe-fleet-location-attributes
- describe-fleet-location-capacity
- describe-fleet-location-utilization

以下のコマンドを使用して、必要に応じてフリートの容量と他の設定を変更できます。

- update-fleet-attributes
- update-fleet-capacity
- update-fleet-port-settings
- update-runtime-configuration
- create-fleet-locations
- delete-fleet-locations

Amazon GameLift Anywhere フリートを作成する

Amazon を使用して GameLift 、環境のハードウェアを Amazon GameLift ゲームホスティングに統合します。Amazon GameLift Anywhere は、ハードウェアを Anywhereフリート GameLift の

Amazon に登録します。Anywhere およびマネージド EC2 フリートをマッチメーカーキューとゲームセッションキューに統合し、マッチメーキングとゲームプレイスメントを管理できます。

Amazon でのゲームサーバーのテストの詳細については GameLift Anywhere、「」を参照してくださいAmazon GameLift Anywhere フリートを使用して統合をテストする。

開始するには、Amazon での開発サポート GameLiftバージョン 5 以降を使用し、Amazon GameLift Anywhere フリートを使用するための以下の概念を確認してください。

カスタムロケーション

Amazon GameLift Anywhere フリートは、カスタムロケーションを使用してインフラストラクチャの物理的なロケーションを表します。

デバイス登録

Amazon GameLift Anywhere フリートがコンピューティングリソースと通信するには、まずデバイスを登録します。`RegisterCompute` オペレーションを使用して、Amazon GameLift AWS SDK からデバイス登録を完了できます。このオペレーションでは、デバイスの IP アドレスを使用してフリートロケーションに関連付け、Amazon と通信します GameLift。

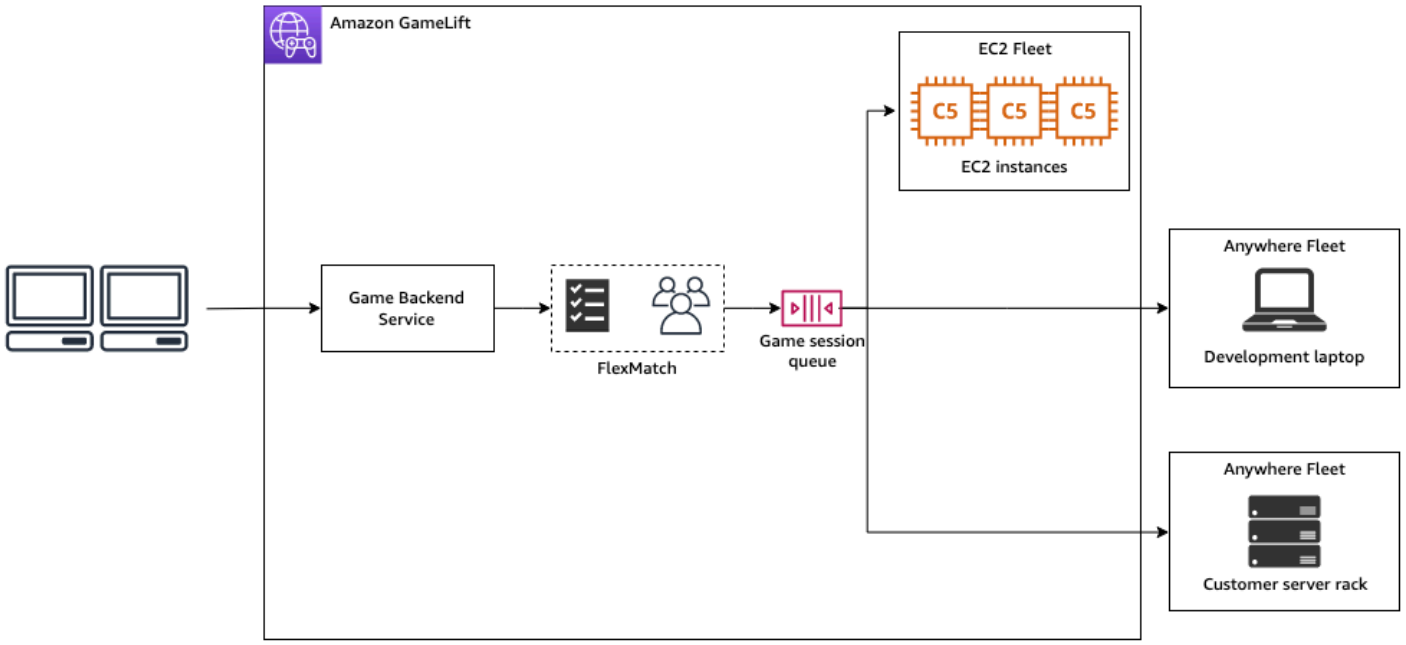
認証トークン

コンピューティングでゲームサーバーを初期化すると、Amazon GameLift Server SDK は認証トークンを使用してゲームサーバーを Amazon に認証します GameLift。認証トークンの有効期限までは、同じコンピューティング上のすべてのゲームサーバーに同じ認証トークンを再利用できます。認証トークンを取得するには、 `get-compute-auth-token` AWS Command Line Interface （ AWS CLI) コマンドを呼び出します。必要に応じてトークンを各ゲームサーバーに渡します。

ゲームセッション

コンピューティング上の各ゲームセッションは、コンピューティングをフリートロケーションに登録する際に作成された同じ認証トークンを使用します。

次の図は、 FlexMatch マッチメーキングと複数のフリートを使用するゲームセッションキューを示しています。フリートには、C5 インスタンスを搭載した EC2 フリート、開発用ラップトップを備えた Anywhereフリート、およびカスタマーがホストするサーバーラックを備えた Anywhere フリートが含まれます。



トピック

- カスタムロケーションを作成する
- フリートを作成する
- コンピューティングを登録する
- サーバープロセスを実行する
- ゲームセッションを作成する
- マネージド EC2 に移行する

カスタムロケーションを作成する

コンピューティングリソースでゲームのホスティングを開始するには、コンピューティングが置かれている場所を説明するカスタムロケーションを作成します。

Console

カスタムロケーションを作成するには

1. Amazon GameLift コンソール を開きます。
2. ナビゲーションペインの [ホスティング] で [ロケーション] を選択します。
3. [Locations (場所)] ページで、[Create location (場所の作成)] を選択します。
4. [ロケーションを作成する] ダイアログボックスで、次の操作を行います。

a. [ロケーション名] を入力します。これにより、Amazon がAnywhereフリートでゲームを実行するために GameLift 使用するハードウェアの場所がラベル付けされます。Amazon はカスタムロケーションの名前に custom- GameLift を追加します。

b. (オプション) キー値のペアとしてタグを追加します。追加するタグごとに [タグを追加] を選択します。

c. [作成] を選択します。

AWS CLI

`create-location` コマンドを使用してカスタムロケーションを作成します。は、Amazon GameLift がAnywhereフリートでゲームを実行するために使用するハードウェアの場所をlocation-nameラベル付けします。カスタムロケーションを作成する場合、ロケーション名は custom- で始まる必要があります。

```
aws gamelift create-location \
    --location-name custom-location-1
```

出力

```
{
    "Location": {
        "LocationName": "custom-location-1",
        "LocationArn": "arn:aws:gamelift:us-east-1:111122223333:location/custom-
location-1"
    }
}
```

フリートを作成する

Amazon GameLift コンソールまたは を使用して AWS CLI フリーAnywhereトを作成します。

新しい Anywhere フリートを作成すると、フリートのステータスは NEW から ACTIVE に移行します。フリートが ACTIVE ステータスに達すると、ゲームセッションをホストする準備ができています。フリート作成の問題については、「Amazon GameLift フリートの問題をデバッグする」を参照してください。

Console

Anywhere フリートを作成するには

1. Amazon GameLift コンソール を開きます。
2. ナビゲーションペインの [ホスティング] で [フリート] を選択します。
3. [フリート] ページで、[フリートの作成] を選択します。
4. [コンピューティングタイプ] ステップで[Anywhere] を選択し、[次へ] を選択します。
5. [フリートの詳細] ステップで、詳細を定義し、[次へ] を選択します。
6. [カスタムロケーション] ステップで、作成したカスタムロケーションを選択し、[次へ] を選択します。Amazon は、フリートを作成するリージョン AWS リージョン としてホーム GameLift を自動的に選択します。ホームリージョンを使用してリソースにアクセスし、使用することができます。
7. 残りのフリート作成手順を完了し、[送信] を選択して Anywhere フリートを作成します。

AWS CLI

create-fleet コマンドを使用して Anywhere フリートを作成します。カスタムロケーションを locations に含めます。Amazon GameLift は、ホームリージョンと指定したカスタムロケーションにフリートを作成します。以下の例では、*FleetName* と *custom-location-1* を独自の情報に置き換えます。変数 custom-location-1 は カスタムロケーションを作成する ステップで作成されたロケーションの名前です。

```
aws gamelift create-fleet \
--name FleetName \
--compute-type ANYWHERE \
--locations "Location=custom-location-1"
```

出力例

```
{
    "FleetAttributes": {
        "FleetId": "fleet-cebb4da2-52a8-4c27-9b85-587f945c6445",
        "FleetArn": "arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445",
        "Name": "HardwareAnywhere",
        "CreationTime": "2023-02-23T17:57:42.293000+00:00",
        "Status": "ACTIVE",
```

```
        "MetricGroups": [
            "default"
        ],
        "CertificateConfiguration": {
            "CertificateType": "DISABLED"
        },
        "ComputeType": "ANYWHERE"
    }
}
```

コンピューティングを登録する

作成したフリートにコンピューティングリソースを登録するには、register-compute コマンドを使用します。*fleet-id* を前のステップで返された fleet-id、またはコンソールのフリートの詳細ページにあるフリート ARN に置き換えます。*compute-name*、と *ip-address* をコンピューティングリソースの IP アドレスに置き換えます。

> **Note**
>
> ゲームサーバーとは別に、スクリプトまたはプロセスマネージャーから register-computeおよび get-compute-auth-token コマンドの両方を呼び出すことをお勧めします。

```
aws gamelift register-compute \
    --compute-name HardwareAnywhere \
    --fleet-id arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445 \
    --ip-address 10.1.2.3 \
    --location custom-location-1
```

出力例

```
{
    "Compute": {
        "FleetId": "fleet-cebb4da2-52a8-4c27-9b85-587f945c6445",
        "FleetArn": "arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445",
        "ComputeName": "HardwareAnywhere",
```

```
        "ComputeArn": "arn:aws:gamelift:us-east-1:111122223333:compute/
HardwareAnywhere",
        "IpAddress": "10.1.2.3",
        "ComputeStatus": "Active",
        "Location": "custom-location-1",
        "CreationTime": "2023-02-23T18:09:26.727000+00:00",
        "GameLiftServiceSdkEndpoint": "wss://us-east-1.api.amazongamelift.com"
    }
}
```

サーバープロセスを実行する

1. 作成したフリートからコンピューティングリソースの認証トークンを取得します。

   ゲームサーバーは認証トークンを使用して Amazon で認証します GameLift。各認証トークンには有効期限があります。引き続きコンピューティングリソースを使用してゲームサーバーをホストするには、有効期限が切れる前に新しい認証トークンを取得してください。

   > **Note**
   >
   > Amazon GameLift では、 `register-compute`と `get-compute-auth-token` コマンドの両方を、ゲームサーバーとは別のスクリプトまたはプロセスマネージャーから呼び出すことをお勧めします。

   次の例では、`fleet-id` を前のステップで作成したフリートの ARN またはフリート ID に置き換えます。`compute-name` を前のステップで `register-compute` コマンドを使用して作成したコンピューティングの名前に置き換えます。

   ```
   aws gamelift get-compute-auth-token \
       --fleet-id arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
   cebb4da2-52a8-4c27-9b85-587f945c6445 \
       --compute-name HardwareAnywhere
   ```

   出力例:

   ```
   {
       "FleetId": "fleet-cebb4da2-52a8-4c27-9b85-587f945c6445",
       "FleetArn": "arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
   cebb4da2-52a8-4c27-9b85-587f945c6445",
   ```

```
    "ComputeName": "HardwareAnywhere",
    "ComputeArn": "arn:aws:gamelift:us-east-1:111122223333:compute/
HardwareAnywhere",
    "AuthToken": "0c728041-3e84-4aaa-b927-a0fb202684c0",
    "ExpirationTimestamp": "2023-02-23T18:47:54+00:00"
}
```

2. ゲームサーバーの実行ファイルのインスタンスを実行します。

   ゲームサーバーを実行するには、InitSDK() を呼び出して、サーバーパラメータに渡すことでゲームサーバーを初期化します。詳細については、「ServerParameters」を参照してください。

   サーバー SDK 入力:

```
//Define the server parameters
ServerParameters serverParameters = new ServerParameters(
    webSocketUrl=wss://us-east-1.api.amazongamelift.com,
    processId=PID1234,
    hostId=HardwareAnywhere,
    fleetId=arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445,
    authToken=0c728041-3e84-4aaa-b927-a0fb202684c0);

//InitSDK establishes a connection with GameLift's websocket server for
 communication.
var initSDKOutcome = GameLiftServerAPI.InitSDK(serverParameters);
```

3. サーバープロセスがゲームセッションをホストする準備ができたら、ゲームサーバーProcessReady()から Amazon に を呼び出します GameLift。プロセスパラメータについての詳細は、「ProcessParameters」を参照してください。

```
// Set parameters and call ProcessReady
var processParams = new ProcessParameters(
   this.OnStartGameSession,
   this.OnProcessTerminate,
   this.OnHealthCheck,
   this.OnUpdateGameSession,
   port=1024,
   new LogParameters(new List<string>()          // Examples of log and error files
 written by the game server
   {
      "C:\\game\\logs",
```

```
      "C:\\game\\error"
   })
);

var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);
```

ゲームセッションを作成する

1. ゲームサーバーにロジックを追加して、サーバープロセスが onStartGameSession() メッセージに ActivateGameSession() で応答するようにします。このオペレーションにはパラメータはありませんが、サーバーがゲームセッション作成メッセージを受信して承諾 GameLift したことの確認を Amazon に送信します。

```
void OnStartGameSession(GameSession gameSession)
{
    // game-specific tasks when starting a new game session, such as loading map

    // When ready to receive players
    var activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}
```

2. ゲームクライアントのバックエンドサービスから、start-matchmaking、start-game-session-placement または create-game-session コマンドを使用してゲームセッションを開始します。

```
aws gamelift create-game-session \
    --fleet-id arn:aws:gamelift:us-east-1:682428703967:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445 \
    --name GameSession1 \
    --maximum-player-session-count 2 \
    --location custom-location-1
```

出力例:

```
GameSession {
  FleetId = arn:aws:gamelift:us-east-1:682428703967:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445,
  GameSessionId = 4444-4444,
  Name = GameSession1,
  Location = custom-location-1,
```

```
  IpAddress = 10.2.3.4,
  Port = 1024,
  ...
}
```

Amazon GameLift は、登録されたサーバープロセスにonStartGameSession()メッセージを送信します。メッセージには、ゲームプロパティ、ゲームセッションデータ、マッチメーカーデータなど、ゲームセッションに関する前のステップの GameSession オブジェクトが含まれます。

3. ゲームセッションが完了したら、ゲームサーバープロセスを終了します。

   サーバー SDK 入力:

```
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();
if (processReadyOutcome.Success)
   Environment.Exit(0);
// otherwise, exit with error code
Environment.Exit(errorCode);
```

4. ProcessReady(processParams) を呼び出して、別のゲームサーバープロセスを開始します。

マネージド EC2 に移行する

ゲームサーバーを開発し、本番環境の準備が整ったら、Amazon にハードウェア GameLift を管理させることができます。マネージド EC2 フリートに移行するには、ビルドを Amazon にアップロード GameLift し、マネージド EC2 フリートを作成します。ビルドのアップロードとフリートのセットアップの詳細については、「カスタムゲームサーバー構築を Amazon GameLift にアップロードする」と「Amazon GameLift マネージドフリートを作成する」を参照してください。

Amazon GameLift フリートを管理する

Amazon GameLift コンソールか AWS CLI を使用して、フリート設定を更新するか、リモートロケーションを変更するか、またはフリートを削除します。

フリート設定を更新する

変更可能なフリート属性、ポート設定、およびランタイム設定を更新するには、Amazon GameLift コンソールまたは AWS CLIを使用します。スケーリング制限を変更するには、Amazon GameLift を使ったフリートキャパシティの自動スケーリング を参照してください。

Amazon GameLift console

1. [Amazon GameLift コンソール] のナビゲーションペインで、[フリート] を選択します。
2. 更新するフリートを選択します。フリートを編集するには、フリートのステータスが ACTIVE になっている必要があります。
3. [フリートの詳細] ページの以下のいずれかのセクションで、[編集] を選択します。
   - フリート設定
     - 名前 や 説明 などのフリート属性を変更します。
     - Amazon CloudWatch が複数のフリートの Amazon GameLift メトリクスの集計を追跡するのに使用する [メトリクスグループ] を追加または削除します。
     - [リソース作成制限] の設定を更新します。
     - ゲームセッション保護をオンまたはオフにします。
   - [ランタイム設定] – ランタイム設定の以下の設定を変更したり、ランタイム設定を追加または削除できます。
     - ゲームサーバーの[起動パス] を変更します。
     - オプションの [起動パラメータ] を追加、削除、または変更します。
     - ゲームサーバーが実行する [同時実行プロセス] の数を変更します。
   - [ゲームセッションアクティベーション] – [最大同時ゲームセッションアクティベーション数] と [新しいアクティベーションタイムアウト] を更新して、サーバープロセスを実行する方法とゲームセッションをホストする方法を変更します。
   - [EC2 ポート設定] – フリートへのインバウンドアクセスを許可する IP アドレスとポート範囲を更新します。
4. [確定] を選択し、変更を保存します。

AWS CLI

以下のAWS CLI コマンドを使用してフリートを更新します。

- update-fleet-attributes
- update-fleet-port-settings
- update-runtime-configuration

フリートのロケーションを更新する

フリートのリモートロケーションを追加または削除するには、Amazon GameLift コンソールまたは AWS CLIを使用します。フリートのホームリージョンは変更できません。

Amazon GameLift console

1. [Amazon GameLift コンソール] のナビゲーションペインで、[フリート] を選択します。
2. 更新するフリートを選択します。フリートを編集するには、フリートのステータスが ACTIVE になっている必要があります。
3. フリートの詳細ページでフリートのロケーションを表示するには、[ロケーション] タブをクリックします。
4. 新しいリモートロケーションを追加するには、[ロケーションを追加] をクリックし、インスタンスのデプロイ先となるロケーションを選択します。このリストには、フリートのインスタンスタイプを使用できないインスタンスは含まれません。
5. 新しいロケーションを選択したら、[追加] を選択します。Amazon GameLift は、新しいロケーションをリストに追加し、ステータスを NEW に設定します。次に、Amazon GameLift は、追加された各ロケーションでのインスタンスのプロビジョニング、およびゲームセッションをホストする準備を開始します。
6. フリートから既存のリモートロケーションを削除するには、チェックボックスを使用して、リストされたロケーションを 1 つ以上選択します。
7. 1 つまたは複数のフリートを選択した状態で、[削除] を選択します。削除された場所はリストに残り、ステータスは DELETING に設定されます。次に、Amazon GameLift は、削除されたロケーションでアクティビティを終了するプロセスを開始します。ゲームセッションをホストしているアクティブなインスタンスがある場合、Amazon GameLift はゲームサーバーの終了プロセスを使用して、ゲームセッションを正常に終了し、ゲームサーバーを終了し、インスタンスをシャットダウンします。

AWS CLI

以下の AWS CLI コマンドを使用してフリートロケーションを更新します。

- create-fleet-locations
- delete-fleet-locations

フリートを削除する

不要になったフリートは削除できます。フリートを削除すると、ゲームセッションとプレイヤーセッションに関連付けられているすべてのデータおよび収集されたメトリクスデータが完全に削除されます。別の方法として、フリートを保持し、Auto Scaling を無効にして、手動でフリートをゼロ (0) インスタンスまでスケーリングできます。

> **Note**
>
> フリートに VPC ピアリング接続がある場合は、最初に、CreateVpcPeeringAuthorization を呼び出して承認をリクエストします。Amazon GameLift は、フリートの削除中に VPC ピアリング接続を削除します。

フリートを削除するには、Amazon GameLift コンソールまたはAWS CLI ツールを使用できます。

Amazon GameLift console

1. [Amazon GameLift コンソール] のナビゲーションペインで、[フリート] を選択します。
2. 削除するフリートを選択します。削除できるのは、`ACTIVE` または`ERROR` ステータスのフリートのみです。
3. [Delete] (削除) をクリックします。
4. [フリートを削除] ダイアログボックスに、**`delete`** を入力して削除を確認します。
5. [Delete] (削除) をクリックします。

AWS CLI

以下のAWS CLI コマンドを使用して、フリートを削除します。

- delete-fleet

Amazon GameLift フリートにエイリアスを追加する

Amazon GameLift エイリアスは、フリートの指定を抽象化するために使用します。フリートの指定は、プレイヤーの新しいゲームセッションを作成するときに利用可能なリソースを検索する GameLift 場所を Amazon に指示します。特定のフリート ID の代わりにエイリアスを使用すると、

エイリアスの送信先を変更することで、あるフリートから別のフリートへのプレイヤートラフィックをシームレスに切り替えます。

エイリアスのルーティング戦略には、二種類あります。

- [シンプル] – プレイヤートラフィックを指定されたフリート ID にルーティングします。エイリアスのフリート ID はいつでも更新できます。
- [ターミナル] – クライアントにメッセージを返します。例えば、 out-of-date クライアントを使用しているプレイヤーを、アップグレードを取得できる場所に誘導できます。

フリートには有効期限があり、ゲームの存続中にフリートを切り替える理由は複数あります。フリートのゲームサーバービルドを更新したり、既存のフリートの特定のコンピューティングリソース属性を変更するといったことはできません。その代わりに、変更された内容で新しいフリートを作成し、プレイヤーを新しいフリートに切り替えます。エイリアスを使用すると、フリートの切り替えがゲームに与える影響が最小限になり、プレイヤーからは見えなくなります。

エイリアスはキューを使用しないゲームで役に立ちます。キュー内のフリートの切り替えは、単純に新しいフリートを作成し、それをキューに追加して古いフリートを削除するだけであり、どれもプレイヤーからは見えません。対照的に、キューを使用しないゲームクライアントは、Amazon GameLift サービスと通信するときに使用するフリートを指定する必要があります。エイリアスを使用しない場合、フリートの切り替えにはゲームコードの更新と、場合によっては更新されたゲームクライアントのプレイヤーへの配布も必要となります。

エイリアスが指定するフリート ID を更新する時には、エイリアスのゲームセッションが古いフリートで終了する可能性があるため、最長 2 分の移行時間が掛かります。 x

新しいエイリアスの作成

エイリアスは、ここで説明するように Amazon GameLift コンソールを使用するか、 AWS CLI コマンド create-alias を使用して作成できます。

1. Amazon GameLift コンソール のナビゲーションペインで、エイリアス を選択します。
2. [エイリアス] ページで、[エイリアスの作成] を選択します。エイリアス名にフリートタイプを含めることをお勧めします。これにより、エイリアスのリストを表示するときにフリートタイプを簡単に識別できます。
3. [エイリアスを作成] ページの [エイリアスの詳細] で、次の操作を行います。

   a. [名前] には、エイリアス名を入力します。

b. [説明] には、識別用の簡単な説明を入力します。

c. ルーティングタイプは [シンプル] または [ターミナル] を選択します。

4. (オプション) [タグ] に [キー] と [値] のペアを入力して、エイリアスにタグを追加します。

5. [作成] を選択します。

エイリアスの編集

Amazon GameLift コンソールまたは AWS CLI コマンド update-alias を使用してエイリアスを編集できます。

1. Amazon GameLift コンソール のナビゲーションペインで、エイリアス を選択します。

2. [エイリアス] ページで、編集するエイリアスを選択します。

3. エイリアスページで [編集] を選択します。

4. [エイリアスの編集] ページで、以下の編集ができます。

   - [Alias name] (エイリアス名) – わかりやすいエイリアスの名前。
   - [Description] (説明) – エイリアスの簡単な説明。
   - [Type] (タイプ) – プレイヤーのトラフィックのルーティング戦略。[シンプル] を選択して関連するフリートを変更するか、[ターミナル] を選択して終了メッセージを編集します。

5. [変更を保存] を選択します。

Amazon GameLift フリートの問題をデバッグする

このトピックでは、Amazon GameLift マネージドソリューションのフリート設定の問題に関するガイダンスを提供します。追加のトラブルシューティングを行うには、フリートがアクティブになってから、フリートインスタンスにリモートでアクセスします。「Amazon GameLift フリートインスタンスにリモート接続する」を参照してください。

フリート作成の問題

フリートが作成されると、Amazon GameLift サービスがフリートの各ロケーションに新しいインスタンスをデプロイするワークフローを開始し、ゲームサーバーを実行する準備をします。詳細な説明については、Amazon GameLift のフリート作成の仕組み を参照してください。フリートは、[Active] (アクティブ) ステータスに到達するまでは、ゲームセッションとプレイヤーをホストすることができません。このセクションでは、フリートがアクティブにならないという、最も一般的な問題について説明します。

ダウンロードと検証

このフェーズでは、抽出したビルドファイルに問題がある場合、インストールスクリプトが実行しない場合、またはランタイム設定で指定した実行可能ファイルがビルドファイルに含まれていない場合に、フリート作成が失敗することがあります。Amazon GameLift は、これらの各問題に関するログを提供しています。

ログから問題を確認できない場合、問題の原因は内部のサービスエラーである可能性があります。この場合は、再度フリートを作成してみます。問題が解決しない場合は、ゲームビルドの再アップロードを検討します (ファイルが破損している場合があるため)。Amazon GameLift サポートにお問い合わせいただくか、フォーラムで質問を投稿することもできます。

構築

構築フェーズ中に失敗した場合は、ほぼ確実に、ゲームビルドファイルやインストールスクリプトに問題があります。Amazon GameLift にアップロードしたゲームビルドファイルが、適切なオペレーティングシステムで動作するマシンにインストールできているか確認します。既存の開発環境ではなく、新規の OS インストールを使用してください。

アクティブ化

フリート作成に伴う最も一般的な問題は、アクティブ化 フェーズの最中に発生します。このフェーズでは、例えば、ゲームサーバーの実行可能性、ランタイム構成の設定がテストされています。また、サーバーSDK を使用して、Amazon GameLift サービスとやり取りするための、ゲームサーバーの機能もテストされています。フリートのアクティブ化中に発生する一般的な問題は以下のとおりです。

サーバープロセスが開始されない。

最初に、起動パスとオプションの起動パラメータがフリートのランタイム設定内に正しく設定されているかチェックします。フリートの現在のランタイム設定を表示するには、[フリート] 詳細ページの [詳細] セクションを使用するか、AWS CLI コマンド describe-runtime-configuration を呼び出します。ランタイム設定が正しいと思われる場合は、ゲームビルドファイルやインストールスクリプトの問題をチェックします。

サーバープロセスは起動するが、フリートがアクティブにならない。

サーバープロセスは正常に開始して実行されるが、フリートが [アクティブ] ステータスに移行しない場合は、サーバープロセスがゲームセッションをホストする準備ができているということを Amazon GameLift に通知していない可能性があります。ゲームサーバーが適切にサーバー API

アクション `ProcessReady()` (「サーバープロセスを初期化する」を参照) を呼び出しているかチェックします。

VPC ピアリングリクエストが失敗する。

作成されたフリートで VPC ピアリングが使用される場合 (「新しいフリートとの VPC ピア接続を設定するには」を参照)、VPC ピアリングはこの アクティブ化 フェーズ中に行われます。VPC ピアリングが何らかの理由で失敗した場合、新しいフリートは ACTIVE (アクティブ) ステータスに移行しません。describe-vpc-peering-connections を呼び出して、ピアリングリクエストの成否を追跡できます。認証は 24 時間のみ有効であるため、有効な VPC ピアリング承認 (describe-vpc-peering-authorizations) が存在することを確認してください。

サーバープロセスの問題

サーバープロセスは開始するが、すぐに失敗するか障害がレポートされる。

ゲームビルドの問題以外にも、インスタンスで同時に多くのサーバープロセスを実行しようとすると、このような問題が発生することがあります。同時処理の適正な数は、インスタンスタイプとゲームサーバーのリソース要件の両方に依存します。フリートのランタイム設定で定義している同時処理の数を減らして、パフォーマンスが向上するかどうか確認してください。フリートのランタイム設定を変更するには、Amazon GameLift のコンソールでフリートのキャパシティ割り当て設定を編集するか、または AWS CLI コマンド update-runtime-configuration を呼び出します。

フリート削除の問題

最大インスタンス数のため、フリートを終了できない。

このエラーメッセージは、削除されるフリートにまだアクティブインスタンスがあることを示しています。この状態は許可されていません。最初に、フリートをアクティブインスタンスゼロまでスケールダウンする必要があります。そのためには、フリートの必要インスタンス数を手動で「0」に設定してから、スケールダウンが有効になるのを待ちます。手動設定の妨げにならないように、Auto Scaling をオフにしてください。

VPC アクションが承認されていない。

この問題は、VPC ピアリング専用に作成したフリートにのみ適用されます (「Amazon GameLift の VPC ピアリング」参照)。このシナリオは、フリートの削除プロセスで、フリートの VPC と VPC ピアリングもすべて削除されるため起こるものです。まず Amazon GameLift サービス API CreateVpcPeeringAuthorization() を呼び出すか、AWS CLI コマンド`create-vpc-peering-`

authorization を使用して、承認を取得する必要があります。承認を取得したら、フリートを削除できます。

リアルタイムサーバーフリートの問題

ゾンビゲームセッション: ゲームセッションを開始し、実行するが、終了することがない。

この問題は、以下のシナリオで見られる場合があります。

- スクリプトの更新をフリートのリアルタイムサーバーが受け取らない。
- フリートがすぐに最大容量に達し、プレイヤーのアクティビティ (新しいゲームセッションリクエストなど) が減少してもスケールダウンされない。

この問題の原因はほとんど確実に、リアルタイムスクリプトでprocessEnding を正常に呼び出すことができなかったことです。フリートがアクティブになり、ゲームセッションが開始されますが、それらを停止する方法はありません。その結果、ゲームセッションを実行しているリアルタイムサーバーは、解放されて、新しいセッションを開始することができなくなり、新しいゲームセッションは、新しいリアルタイムサーバーがスピンアップされて初めて開始できるようになります。さらに、リアルタイムスクリプトの更新は、すでに実行中のゲームセッションには影響しないのです。

この問題を回避するには、スクリプトにprocessEnding の呼び出しをトリガーするメカニズムを提供する必要があります。「リアルタイムサーバースクリプト例」に示しているように、ひとつの方法はアイドルセッションタイムアウトをプログラムすることで、一定時間プレイヤーからの接続がない場合に現在のゲームセッションが終了するようにすることです。

ただし、このシナリオが生じた場合には、リアルタイムサーバーがスタックから抜け出すための回避策がいくつかあります。コツは、リアルタイムサーバープロセス (または基盤となるフリートインスタンス) をトリガーして再起動することです。この場合、ゲームセッションはGameLiftによって自動的に終了されます。リアルタイムサーバーが解放されると、最新バージョンのリアルタイムスクリプトを使用して新しいゲームセッションを開始できます。

そのための方法は、この問題の波及する範囲に応じていくつかあります。

- フリート全体をスケールダウンする。この方法は最も簡単に実行できますが、広範な効果があります。フリートをインスタンスゼロまでスケールダウンし、フリートが完全にスケールダウンするのを待ってから、再びスケールアップします。これにより、既存のゲームセッションがすべて消去されてから、直近に更新されたリアルタイムスクリプトが開始されます。
- インスタンスにリモートアクセスしてプロセスを再開する。この方法は、修正するプロセスがわずかしかない場合にお勧めします。ログ記録やデバッグなどのために、すでにインスタンス

にログオンしている場合は、これが最も迅速な方法です。「Amazon GameLift フリートインスタンスにリモート接続する」を参照してください。

リアルタイムスクリプトにprocessEnding を呼び出す方法を含めないことを選択した場合、フリートがアクティブになってゲームセッションが開始されたとしても、いくつかの予想外の状況が生じることがあります。ひとつは、実行中のゲームセッションが終了しない状況です。この結果、そのゲームセッションを実行しているサーバープロセスが解放されず、新しいゲームセッションを開始できなくなります。もう ひとつは、リアルタイムサーバーがスクリプトの更新を受け取らない状況です。

Amazon GameLift フリートインスタンスにリモート接続する

アクティブな Amazon GameLift マネージド EC2 フリート内の任意のインスタンスに接続できます。インスタンスにアクセスする一般的な理由は次のとおりです。

- ゲームサーバー統合に関する問題のトラブルシューティング
- ランタイム設定やその他のフリート固有の設定を微調整する
- ログ追跡などのゲームサーバーアクティビティをリアルタイムで取得します。
- 実際のプレイヤートラフィックを使用してベンチマークツールを実行します。
- ゲームセッションまたはサーバープロセスの特定の問題を調査します。

インスタンスに接続するときは、次の潜在的な問題を考慮してください。

- アクティブなフリートのインスタンスに接続できます。アクティブでないフリート、またはエラー状態にあるフリートには、短期間アクセスできる場合があります。フリートのアクティブ化に関する問題のヘルプは、「Amazon GameLift フリートの問題をデバッグする」を参照してください。
- アクティブなインスタンスに接続しても、インスタンスのホスティングアクティビティには影響しません。インスタンスは、ランタイム設定に基づいてサーバープロセスを開始および停止します。ゲームセッションをアクティブ化してホストします。スケールダウンイベントやその他のイベントに応じてシャットダウンする場合があります。
- インスタンスのファイルまたは設定を変更すると、インスタンスのアクティブなゲームセッションと接続されたプレイヤーに影響する可能性があります。

次の手順では、AWSコマンドラインインターフェイス (CLI) を使用してインスタンスにリモート接続する方法について説明します。「Amazon GameLift サービス API リファレンス」に記載されているように、 AWS SDK を使用してプログラムによる呼び出しを行うこともできます。

インスタンスデータの収集

次の情報を収集します。

- 接続するインスタンスの ID。インスタンス ID または ARN のいずれかを使用できます。
- インスタンスで使用されている Amazon GameLift サーバー SDK のバージョン。サーバー SDK は、インスタンスで実行されているゲームビルドと統合されています。

インスタンスデータを取得するには

次の手順では、接続するインスタンスのマネージド EC2 フリート ID があることを前提としています。

1. コンピューティング名を取得します。

   マネージド EC2 フリートの list-compute を呼び出して、フリート内のすべてのアクティブなコンピューティングのリストを取得します。単一ロケーションフリートの場合は、フリート ID または ARN を指定します。マルチロケーションフリートの場合は、フリート ID または ARN とロケーションを指定します。マネージド EC2 フリートの場合、コンピューティングは EC2 インスタンスであり、返されるプロパティ`ComputeName`はインスタンス ID です。例:

   リクエスト

   ```
   aws gamelift list-compute \
     --fleet-id  "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa" \
     --location ""sa-east-1"
   ```

   レスポンス

   ```
   {
     "ComputeList": [
       {
         "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
         "FleetArn": "arn:aws:gamelift:us-west-2::fleet/
   fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
         "ComputeName": "i-0abc12d3e45fa6b78",
         "IpAddress": "00.00.000.00",
         "DnsName":
    "b08444ki909kvqu6zpw3is24x5pyz4b6m05i3jbxvpk9craztu0lqrbbrbnbkks.uwp57060n1k6dnlnw49b78hg1
   west-2.amazongamelift.com",
   ```

```
      "ComputeStatus": "Active",
      "Location": "sa-east-1",
      "CreationTime": "2023-07-09T22:51:45.931000-07:00",
      "OperatingSystem": "AMAZON_LINUX",
      "Type": "c4.large"
    }
  ]
}
```

2. サーバー SDK のバージョンを見つけます。

   サーバー SDK バージョンは、ビルドリソースの属性です。

   a. フリート ID describe-fleet-attributesで を呼び出して、フリートのビルド ID と ARN を取得します。

   b. ビルド ID または ARN を使用して describe-build を呼び出して、ビルドのサーバー SDK バージョンを取得します。

      例:

      リクエスト

```
aws gamelift describe-fleet-attributes /
  --fleet-ids  "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
```

レスポンス

```
{
  "FleetAttributes": [
    {
      "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
      "ComputeType": "EC2",
      "BuildId": "build-3333cccc-44dd-55ee-66ff-00001111aa22",
      . . .
    }
  ]
}
```

リクエスト

```
aws gamelift describe-build /
  --build-id "build-3333cccc-44dd-55ee-66ff-00001111aa22"
```

レスポンス

```
"Build": {
  "BuildId": "build-1111aaaa-22bb-33cc-44dd-5555eeee66ff",
  "Name": "My_Game_Server_Build_One",
  "OperatingSystem": "AMAZON_LINUX_2",
  "ServerSdkVersion": "5.1.1",
  . . .
}
```

インスタンスに接続する (サーバー SDK 5)

接続先のインスタンスがサーバー SDK バージョン 5.x でゲームビルドを実行している場合は、以下の手順で Amazon EC2 Systems Manager (SSM) を使用してインスタンスに接続します。Windows または Linux を実行しているリモートインスタンスにアクセスできます。

1. インスタンスのアクセス認証情報をリクエストします。接続するインスタンスのコンピューティング名とフリート ID がある場合は、 を呼び出しますget-compute-access。成功すると、Amazon はインスタンスにアクセスするための一時的な認証情報のセット GameLift を返します。例:

   リクエスト

   ```
   aws gamelift get-compute-access \
   --compute-name i-11111111a222b333c \
   --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa
   --region us-west-2
   ```

   レスポンス

   ```
   {
     "ComputeName": " i-11111111a222b333c ",
     "Credentials": {
       "AccessKeyId": " ASIAIOSFODNN7EXAMPLE ",
       "SecretAccessKey": " wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY ",
       "SessionToken": " AQoDYXdzEJr...<remainder of session token>"
   ```

```
  },
  "FleetArn": " arn:aws:gamelift:us-west-2::fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa ",
  "FleetId": " fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa "
}
```

2. アクセス認証情報をエクスポートします。オプションで、認証情報を環境変数にエクスポートし、それを使用してデフォルトユーザーの AWS CLI を設定できます。詳細については、「 ユーザーガイド」の「CLI AWS を設定するための環境変数AWS Command Line Interface」を参照してください。

```
export AWS_ACCESS_KEY_ID=ASIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of session token>
```

3. フリートインスタンスに接続します。接続先のインスタンスで SSM セッションを開始します。インスタンスのAWSリージョンまたは場所を含めます。詳細については、「Amazon EC2 Systems Manager ユーザーガイド」の「セッションの開始 (AWS CLI)」を参照してください。Amazon EC2 Systems Manager ステップ 1 で取得した認証情報を使用します。例:

```
aws ssm start-session \
--target i-11111111a222b333c \
--region us-west-2
```

インスタンスに接続する (サーバー SDK 4.x 以前)

接続先のインスタンスがサーバー SDK バージョン 4 以前でゲームビルドを実行している場合は、以下の手順に従います。Windows または Linux を実行しているインスタンスに接続できます。リモートデスクトッププロトコル (RDP) クライアントを使用して Windows インスタンスに接続します。SSH クライアントを使用して Linux インスタンスに接続します。

1. インスタンスのアクセス認証情報をリクエストします。インスタンス ID がある場合は、 コマンドを使用してアクセス認証情報をget-instance-accessリクエストします。成功すると、Amazon はインスタンスのオペレーティングシステム、IP アドレス、および認証情報のセット (ユーザー名とシークレットキー) GameLift を返します。認証情報の形式は、インスタンスのオペレーティングシステムによって決まります。次の手順を使用して、RDP または SSH の認証情報を取得します。

- Windowsインスタンスの場合 – Windows インスタンスに接続する場合、RDP ではユーザー名とパスワードを必要とします。get-instance-access リクエストはこれらの値を簡単な文字列として返すため、返された値をそのまま使用できます。認証情報の例:

```
"Credentials": {
    "Secret": "aA1bBB2cCCd3EEE",
    "UserName": "gl-user-remote"
}
```

- Linux インスタンスの場合 - Linux インスタンスに接続する場合、SSH ではユーザー名とプライベートキーを必要とします。Amazon は RSA プライベートキー GameLift を発行し、改行文字 (\n) を使用して 1 つの文字列として返します。プライベートキーを使用できるようにするには、(1) 文字列を.pemファイルに変換し、(2) 新しいファイルのアクセス許可を設定します。返される認証情報の例:

```
"Credentials": {
    "Secret": "-----BEGIN RSA PRIVATE KEY-----
nEXAMPLEKEYKCAQEAy7WZhaDsrA1W3mRlQtvhwyORRX8gnxgDAfRt/gx42kWXsT4rXE/b5CpSgie/
\nvBoU7jLxx92pNHoFnByP+Dc21eyyz6CvjTmWA0JwfWiW5/akH7iO5dSrvC7dQkW2duV5QuUdE0QW
\nZ/aNxMniGQE6XAgfwlnXVBwrerrQo+ZWQeqiUwwMkuEbLeJFLhMCvYURpUMSC1oehm449ilx9X1F
\nG50TCFeOzfl8dqqCP6GzbPaIjiU19xX/azOR9V+tpUOzEL+wmXnZt3/nHPQ5xvD2OJH67km6SuPW
\noPzev/D8V+x4+bHthfSjR9Y7DvQFjfBVwHXigBdtZcU2/wei8D/HYwIDAQABAoIBAGZ1kaEvnrqu
\n/uler7vgIn5m7lN5LKw4hJLAIW6tUT/fzvtcHK0SkbQCQXuriHmQ2MQyJX/0kn2NfjLV/
ufGxbL1\nmb5qwMGUnEpJaZD6QSSs3kICLwWUYUiGfc0uiSbmJoap/
GTLU0W5Mfcv36PaBUNy5p53V6G7hXb2\nbahyWyJNfjLe4M86yd2YK3V2CmK+X/
BOsShnJ36+hjrXPPWmV3N9zEmCdJjA+K15DYmhm/
tJWSD9\n81oGk9TopEp7CkIfatEATyyZiVqoRq6k64iuM9JkA3OzdXzMQexXVJ1TLZVEH0E7bhlY9d8O1ozR
\noQs/FiZNAx2iijCWyv0lpjE73+kCgYEA9mZtyhkHkFDpwrSM1APaL8oNAbbjwEy7Z5Mqfql
+lIp1\nYkriL0DbLXlvRAH+yHPRit2hHOjtUNZh4Axv+cpg09qbUI3+43eEy24B7G/Uh
+GTfbjsXsOxQx/x\np9otyVwc7hsQ5TA5PZb
+mvkJ5OBEKzet9XcKwONBYELGhnEPe7cCgYEA06Vgov6YHleHui9kHuws
\nayav0elc5zkxjF9nfHFJRry21R1trw2Vdpn+9g481URrpzWVOEihvm+xTtmaZlSp//lkq75XDwnU
\nWA8gkn6O3QE3fq2yN98BURsAKdJfJ5RL1HvGQvTe10HLYYXpJnEkHv+Unl2ajLivWUt5pbBrKbUC
\ngYBjbO+OZk0sCcpZ29sbzjYjpIddErySIyRX5gV2uNQwAjLdp9PfN295yQ+BxMBXiIycWVQiw0bH
\noMo7yykABY7Ozd5wQewBQ4AdSlWSX4nGDtsiFxWiI5sKuAAeOCbTosy1s8w8fxoJ5Tz1sdoxNeGs
\nArq6Wv/G16zQuAE9zK9vvwKBgF+09VI/1wJBirsDGz9whVWfFPrTkJNvJZzYt69qezxlsjgFKshy
\nWBhd4xHZtmCqpBPlAymEjr/TOlbxyARmXMnIOWIAnNXMGB4KGSyl1mzSVAoQ+fqR+cJ3d0dyPl1j
\njjb0Ed/NY8frlNDxAVHE8BSkdsx2f6ELEyBKJSRr9snRAoGAMrTwYneXzvTskF/S5Fyu0iOegLDa
\nNWUH38v/nDCgEpIXD5Hn3qAEcju1IjmbwlvtW+nY2jVhv7UGd8MjwUTNGItdb6nsYqM2asrnF3qS
\nVRkAKKKYeGjkpUfVTrW0YFjXkfcrR/V+QFL5OndHAKJXjW7a4ejJLncTzmZSpYzwApc=\n-----END
 RSA PRIVATE KEY-----",
```

```
    "UserName": "gl-user-remote"
}
```

AWS CLI を使用する場合、--query パラメータと --output パラメータをget-instance-accessリクエストに含めることで、.pemファイルを自動的に生成できます。

.pem ファイルでアクセス許可を設定するには、次のコマンドを実行します。

```
$ chmod 400 MyPrivateKey.pem
```

2. リモート接続でポートを開きます。Amazon GameLift フリートのインスタンスには、フリート設定で許可されている任意のポートからアクセスできます。フリートのポート設定は、コマンド describe-fleet-port-settings を使用して表示できます。

   ベストプラクティスとして、必要な場合にのみ、リモートアクセス用のポートを開き、完了したらそれを閉じるようにお勧めします。フリートの作成後、アクティブになる前にポート設定を更新することはできません。スタックした場合は、ポート設定を開いた状態でフリートを再作成します。

   コマンド update-fleet-port-settings を使用してリモート接続 (SSH 用は22 または RDP 用は3389) にポート設定を追加します IP 範囲値には、接続に使用する予定のデバイスの IP アドレスを指定します (CIDR 形式に変換)。例：

   ```
   $ AWS gamelift update-fleet-port-settings
       --fleet-id  "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
       --inbound-permission-authorizations
    "FromPort=22,ToPort=22,IpRange=54.186.139.221/32,Protocol=TCP"
   ```

   次の例では、Windows フリートでポート 3389 を開きます。

   ```
   $ AWS gamelift update-fleet-port-settings
   --fleet-id  "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
       --inbound-permission-authorizations
    "FromPort=3389,ToPort=3389,IpRange=54.186.139.221/32,Protocol=TCP"
   ```

3. リモート接続クライアントを開きます。Windows インスタンスにはリモートデスクトップ、Linux インスタンスには SSH を使用します。IP アドレス、ポート設定、アクセス認証情報を使用してインスタンスに接続します。

   SSH の例:

```
ssh -i MyPrivateKey.pem gl-user-remote@192.0.2.0
```

リモートインスタンスでファイルを表示する

インスタンスにリモートで接続すると、完全なユーザーアクセスや管理アクセスを得ることになります。これは、ゲームホスティングでエラーや障害を引き起こす可能性があることも意味します。インスタンスがアクティブなプレイヤーとゲームをホストしている場合、ゲームセッションがクラッシュしてプレイヤーがドロップされたり、ゲームのシャットダウンプロセスが中断され、保存されたゲームデータとログにエラーが発生するリスクがあります。

ホスティングインスタンスで次のリソースを探します。

- ゲーム用のビルドファイル。これらのファイルは、Amazon にアップロードしたゲームビルドです GameLift。これには、1 つ以上のゲームサーバー実行可能ファイル、アセット、依存関係が含まれます。ゲームビルドファイルは、 というルートディレクトリにありますgame。
  - Windows の場合: `c:\game`
  - Linux の場合: `/local/game`
- ゲームログファイル。ゲームサーバーが生成するログファイルは、指定した任意のディレクトリパスのgameルートディレクトリで検索します。
- Amazon GameLift ホスティングリソース。ルートディレクトリには、ゲームホスティングアクティビティを管理するために Amazon GameLift サービスが使用するファイルWhitewaterが含まれています。何らかの理由でこれらのファイルを変更しないでください。
- ランタイム設定。個々のインスタンスのランタイム設定にはアクセスしないでください。ランタイム設定プロパティを変更するには、フリートのランタイム設定を更新します (AWS「 SDK オペレーションUpdateRuntimeConfiguration」または「」を参照AWS CLIupdate-runtime-configuration ）。
- フリートのデータ。JSON ファイルには、インスタンスで実行されているサーバープロセスで使用する、インスタンスが属するフリートに関する情報が含まれています。JSON ファイルは次の場所にあります。
  - Windows の場合: `C:\GameMetadata\gamelift-metadata.json`
  - Linux の場合: `/local/gamemetadata/gamelift-metadata.json`
- TLS 証明書。インスタンスが TLS 証明書の生成が有効になっているフリートにある場合は、証明書、証明書チェーン、プライベートキー、ルート証明書などの証明書ファイルを次の場所に探します。

- Windows の場合: `c:\\GameMetadata\Certificates`
- Linux の場合: `/local/gamemetadata/certificates/`

Amazon GameLift ホスティングキャパシティのスケーリング

インスタンスで測定されるホスティングキャパシティは、Amazon GameLift が同時にホストできるゲームセッションの数とそれらのゲームセッションが適応できる同時プレイヤーの数を表します。ゲームホスティングで最も困難なタスクの 1 つが、不要なリソースに無駄なコストを浪費することなく、プレイヤーの需要に合わせてキャパシティをスケーリングすることです。詳細については、「フリートの容量のスケーリング」を参照してください。

容量は、フリートの口ケーションレベルで調整されます。すべてのフリートには少なくとも 1 つの場所、つまりフリートのホーム AWS リージョンがあります。容量を表示またはスケーリングすると、フリートのホーム リージョンおよび追加のリモートロケーションなど、ロケーションごとに情報がリストされます。

維持するインスタンスの数を手動で設定することも、プレイヤーの需要の変化に合わせてキャパシティを動的に調整するように自動スケーリングを設定することもできます。ターゲットベースの自動スケーリングオプションをオンにして開始することをお勧めします。ターゲットベースの自動スケーリングの目標は、現在のプレイヤーに対応するのに十分なホスティングリソースと、プレイヤーの需要の予期しない急増に対処する追加のリソースを維持することです。ほとんどのゲームに対して、ターゲットベースの自動スケーリングは非常に効果的なスケーリングソリューションです。

このセクションのトピックでは、次のタスクの詳細を提供します。

- [Set minimum and maximum limits for capacity scaling]（容量のスケーリングの上限と下限を設定する）
- [Manually set capacity levels]（容量レベルを手動で設定する）
- ターゲットベースの自動スケーリングを使用する
- ルールベースの自動スケーリングを管理する (高度な機能)
- 自動スケーリングを一時的に無効にする

ほとんどのフリートスケーリングアクティビティは、Amazon GameLift コンソールを使用して行うことができます。また、Amazon GameLift サービス APIで、AWS SDK または AWS Command Line Interface (AWS CLI) を使用することもできます。

コンソールでフリートの容量を管理するには

1. [Amazon GameLift コンソール] を開きます。
2. ナビゲーションペインで、[ホスティング]、[フリート] を選択します。
3. [フリート] ページで、アクティブなフリートの名前を選択して、フリートの詳細ページを開きます。
4. [スケーリング] タブを選択します。このタブでは、次の操作を実行できます。
   - フリート全体の履歴スケーリングメトリクスを表示します。
   - スケーリング制限や現在の容量設定など、各フリートロケーションのキャパシティ設定を表示および更新します。
   - ターゲットベースの自動スケーリングを更新し、フリート全体に適用されるルールベースの自動スケーリングポリシーを表示し、各ロケーションの自動スケーリングアクティビティを停止します。

トピック

- Amazon GameLift のキャパシティ制限を設定する
- Amazon GameLift フリートのキャパシティを手動で設定する
- Amazon GameLift を使ったフリートキャパシティの自動スケーリング

Amazon GameLift のキャパシティ制限を設定する

Amazon GameLift フリートロケーションのホスティングキャパシティを手動で、または自動スケーリングでスケーリングする場合は、ロケーションのスケーリング制限を考慮します。すべてのフリートロケーションには、ロケーションの容量の許容範囲を定義する最小および最大制限があります。デフォルトでは、フリートロケーションの制限の最小値は 0 インスタンス、最大値が 1 インスタンスです。フリートロケーションの場所をスケールする前に、制限を調整します。

自動スケーリングを使用している場合、最大値の制限によって Amazon GameLift はプレイヤーの需要に合わせてフリートのロケーションをスケールアップしつつ、DDOS 攻撃中など、ホスティング費用のランナウェイも防止します。キャパシティが最大限に到達すると通知するように [Amazon CloudWatch アラーム] をセットアップすることで、状況を評価して必要に応じて手動で調整できます。(また、請求アラームを作成して、AWS コストをモニタリングすることもできます。) 最小値の制限は、プレイヤーの需要が低い場合でも、ホスティングの可用性を維持するのに役立ちます。

フリートのロケーションのキャパシティ制限は、[Amazon GameLift コンソール] で、または AWS Command Line Interface (AWS CLI) を使用して設定できます。

容量制限を設定するには

Console

1. [Amazon GameLift コンソール] を開きます。
2. ナビゲーションペインで [ホスティング]、[フリート] を選択します。
3. [フリート] ページで、アクティブなフリートの名前を選択して、フリートの詳細ページを開きます。
4. [スケーリング] タブの [キャパシティのスケーリング] で、フリートのロケーションを選択し、[編集] を選択します。
5. [キャパシティのスケーリングの編集] ダイアログボックスで、[最小サイズ]、[目的のインスタンス]、および [最大サイズ] にインスタンス数を設定します。
6. [確認] を選択します。

AWS CLI

1. 現在のキャパシティ設定を確認します。コマンドライン ウィンドウで、容量を変更するフリートの ID を使用してdescribe-fleet-location-capacityコマンドを実行します。このコマンドは、ロケーションの現在の容量設定を含む [FleetCapacity] (フリート容量) オブジェクトを返します。新しいインスタンスの制限が現在機能するインスタンス設定に合うかどうかを決定します。

```
aws gamelift describe-fleet-location-capacity \
    --fleet-id <fleet identifier> \
    --location <location name>
```

2. 制限設定を更新します。コマンドラインウィンドウで、以下のパラメータを使用して update-fleet-capacity コマンドを実行します。インスタンス制限および希望するインスタンス数の両方を同じコマンドで調整できます。

```
--fleet-id <fleet identifier>
--location <location name>
--max-size <maximum capacity for scaling>
--min-size <minimum capacity for scaling>
--desired-instances <fleet capacity goal>
```

例:

```
aws gamelift update-fleet-capacity \
    --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
    --location us-west-2 \
    --max-size 10 \
    --min-size 1 \
    --desired-instances 10
```

リクエストが成功すると、Amazon GameLift からフリート ID が返されます。新しい `max-size` 値または `min-size` 値が現在の `desired-instances` 設定と矛盾する場合、Amazon GameLift はエラーを返します。

Amazon GameLift フリートのキャパシティを手動で設定する

新しいフリートを作成すると、Amazon GameLift は、各フリートロケーションで、希望するインスタンスを 1 つのインスタンスに自動的に設定します。次に、Amazon GameLift は各ロケーションに 1 つの新しいインスタンスをデプロイします。フリートキャパシティを変更するには、ターゲットベースの自動スケーリングを追加するか、または手動でロケーションに必要なインスタンスの数を設定します。詳細については、「フリートの容量のスケーリング」を参照してください。

フリートのキャパシティを手動で設定することは、自動スケーリングが必要でない場合、または任意のレベルでキャパシティを確保することが必要な場合に便利です。キャパシティの手動設定は、ターゲットベースの自動スケーリングポリシーを使用していない場合にのみ機能します。ターゲットベースの自動スケーリングポリシーがある場合、独自のスケーリングルールに基づく希望のキャパシティが即座にリセットされます。

キャパシティは、Amazon GameLift コンソールで、または AWS Command Line Interface (AWS CLI) を使用して手動で設定できます。フリートのステータスは、アクティブである必要があります。

自動スケーリングを停止する

フリートロケーションごとにすべての自動スケーリングアクティビティを一時停止できます。自動スケーリングを停止にすると、フリートロケーション内の必要なインスタンス数は、手動で変更しない限り、同じままになります。ロケーションに対して自動スケーリングを停止すると、フリートの現在のポリシーや今後定義されるすべてのポリシーに影響します。

フリートの容量を手動で設定するには

Console

1. [Amazon GameLift コンソール] を開きます。
2. ナビゲーションペインで [ホスティング]、[フリート] を選択します。
3. [フリート] ページで、アクティブなフリートの名前を選択して、フリートの詳細ページを開きます。
4. [スケーリング] タブの [一時停止中の自動スケーリングロケーション] で、自動スケーリングを一時停止する各ロケーションを選択し、[停止] を選択します。
5. [キャパシティーのスケーリング] で、手動で設定するロケーションを選択し、[編集] を選択します。
6. [キャパシティーのスケーリングの編集] ダイアログボックスで、目的のインスタンス] に希望の値を設定し、[確認] を選択します。この値は、ゲームセッションをホストできるアクティブな状態で維持するインスタンスの数を Amazon GameLift に伝えます。

Amazon GameLift は、追加のインスタンスをデプロイするか、不要なインスタンスをシャットダウンして、この変更に対応します。Amazon GameLift がこのプロセスが完了すると、ロケーション内のアクティブなインスタンスの数が新しく更新された目的のインスタンス値に一致するように変わります。このプロセスには多少時間がかかることがあります。

AWS CLI

1. 現在のキャパシティ設定を確認します。コマンドライン ウィンドウで、容量を変更するフリートの ID を使用してdescribe-fleet-location-capacityコマンドを実行します。このコマンドは、ロケーションの現在の容量設定を含む [FleetCapacity]（フリート容量）オブジェクトを返します。インスタンスの制限が現在機能する新しい目的のインスタンス設定に合うかどうかを決定します。

```
aws gamelift describe-fleet-location-capacity \
    --fleet-id <fleet identifier> \
    --location <location name>
```

2. 希望する容量を更新します。update-fleet-capacity コマンドを目的のインスタンスのフリート ID、ロケーション、新しい値で使用します。この値が現在の制限範囲外にある場合、同じコマンドに調整した制限値を調節できます。

```
--fleet-id <fleet identifier>
```

```
--location <location name>
--desired-instances <fleet capacity as an integer>
--max-size <maximum capacity>    [Optional]
--min-size <minimum capacity>    [Optional]
```

例:

```
aws gamelift update-fleet-capacity \
    --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
    --location us-west-2 \
    --desired-instances 5 \
    --max-size 10 \
    --min-size 1
```

リクエストが成功すると、Amazon GameLift からフリート ID が返されます。新しい目的のインスタンス設定が上限と下限の制限外にある場合、Amazon GameLift はエラーを返します。

Amazon GameLift を使ったフリートキャパシティの自動スケーリング

自動スケーリングを使用して、Amazon GameLift のゲームサーバーアクティビティに応じてフリートキャパシティを動的にスケールします。プレイヤーが参加してゲームセッションが開始すると、自動スケーリングはさらにインスタンスを追加できます。プレイヤーの需要が低くなると、自動スケーリングは不必要なインスタンスを終了できます。自動スケーリングはホスティングリソースとコストを最小化するための効果的な方法である一方で、スムーズで高速なプレイヤーエクスペリエンスを提供します。

自動スケーリングを使用するには、Amazon GameLift にスケールアップまたはスケールダウンのタイミングを指示するスケーリングポリシーを作成します。スケーリングポリシーには、ターゲットベースとルールベースの 2 タイプがあります。ターゲットベースのアプローチであるターゲット追跡は、完全なソリューションです。これは、最もシンプルで効果的な方法のためお勧めします。ルールベースのスケーリングポリシーは、自動スケーリングの意思決定プロセスにおける各要素を定義するために必要となり、特定の問題の対処に役立ちます。このソリューションは、ターゲットベースの自動スケーリングを補足するものとして最適です。

Amazon GameLift コンソール、AWS Command Line Interface (AWS CLI)、または AWS SDK を使用して、ターゲットベースの自動スケーリングを管理できます。ルールベースのスケーリングポリシーはコンソールで表示できますが、ルールベースの自動スケーリングは、AWS CLI またはAWS SDK のみを使用して管理できます。

トピック

ターゲットベースの自動スケーリング

Amazon GameLift のターゲットベースの自動スケーリングは、フリートメトリクス `PercentAvailableGameSessions` に基づいてキャパシティレベルを調整します。このメトリクスはプレイヤーの需要の急激な増加に対してフリートで利用できるバッファを示します。

容量バッファを維持する主な利用は、プレイヤーの待機時間です。ゲームセッションスロットの準備ができ、待機中の場合、ゲームセッションに新規プレイヤーを入れるのにかかる時間は数秒です。使用可能なリソースがない場合、プレイヤーは既存のゲームセッションが終了するまで、あるいは新しいリソースが使用可能になるまで待機する必要があります。新しいインスタンスとサーバープロセスを起動するには数分かかる場合があります。

ターゲットベースの自動スケーリングをセットアップするには、フリートが維持するバッファのサイズを指定します。`PercentAvailableGameSessions` は使用可能なリソースの割合を測定するため、実際のバッファサイズは全フリートキャパシティの割合となります。Amazon GameLift は、ターゲットバッファサイズを維持するためにインスタンスを追加あるいは削除します。より大きなバッファでは待機時間が最小になりますが、使用しないかもしれない余分なリソースに対してもコストが発生します。待機時間に対してより耐性のあるプレイヤーの場合、バッファを小さく設定することでコストをより低く抑えることができます。

ターゲットベースの自動スケーリングを設定するには

Console

1. [Amazon GameLift コンソール] を開きます。
2. ナビゲーションペインで [ホスティング]、[フリート] を選択します。
3. [フリート] ページで、アクティブなフリートの名前を選択して、フリートの詳細ページを開きます。
4. [スケーリング] タブを選択します。このタブには、フリートのスケーリングメトリクスの履歴が表示され、現在のスケーリング設定を調整するコントロールを含んでいます。
5. [キャパシティのスケーリング] で、[最小サイズ] と [最大サイズ] の制限がフリートに適していることを確認します。自動スケーリングを有効にすると、これらの 2 つの制限間でキャパシティが調整されます。

6. [ターゲットベースの自動スケーリングポリシー] で、[編集] を選択します。
7. [ターゲットベースの自動スケーリングポリシーの編集] ダイアログボックスの [使用可能なゲームセッションの割合] で、維持したい割合 (パーセント) を設定し、[確認] を選択します。設定を確認すると、Amazon GameLift は [ターゲットベースの自動スケーリングポリシー] の下に新しいターゲットベースのポリシーを追加します。

AWS CLI

1. 容量制限を設定します。update-fleet-capacity コマンドを使用して制限値を設定します。詳細については、「Amazon GameLift のキャパシティ制限を設定する」を参照してください。
2. 新規ポリシーを作成します。コマンドラインウィンドウを開き、ポリシーのパラメータ設定で put-scaling-policy コマンドを使用します。既存のポリシーを更新するには、ポリシー名を指定して、更新されたポリシーの完全バージョンを提供します。

```
--fleet-id <unique fleet identifier>
--name "<unique policy name>"
--policy-type <target- or rule-based policy>
--metric-name <name of metric>
--target-configuration <buffer size>
```

例:

```
aws gamelift put-scaling-policy \
    --fleet-id "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa" \
    --name "My_Target_Policy_1" \
    --policy-type "TargetBased" \
    --metric-name "PercentAvailableGameSessions" \
    --target-configuration "TargetValue=5"
```

ルールベースのポリシーによる自動スケーリング

Amazon GameLift のルールベースのスケーリングポリシーは、プレイヤーのアクティビティに対応してフリートの容量を自動スケーリングするための詳細な制御を提供します。各ポリシーでは、複数のフリートメトリクスの 1 つにスケーリングをリンクし、トリガーポイントを識別して、スケールアップレスポンスをカスタマイズするか、あるいはイベントをスケールダウンすることができます。ルールベースのポリシーは、特別な状況を処理するためにターゲットベースのスケーリングの補助として便利です。

ルールベースのポリシーでは、「フリートメトリクスが一定期間しきい値以上になる場合、フリートの容量を指定された量に変更する」と述べています。このトピックでは、ポリシーステートメントを構成するために使用する構文を説明し、ルールベースのポリシーを作成して管理するためのヘルプを提供します。

ルールベースのポリシーを管理する

AWS SDK または AWS Command Line Interface (AWS CLI) を Amazon GameLift サービス API で使用して、ルールベースのポリシーを作成、更新、または削除します。すべてのアクティブなポリシーは Amazon GameLift コンソールで表示できます。

フリートのすべてのスケーリングポリシーを一時的に停止にするには、AWS CLI コマンド stop-fleet-actions を使用します。

ルールベースのスケーリングポリシー (AWS CLI) を作成または更新するには

1. 容量制限を設定します。update-fleet-capacity コマンドを使用して両方またはどちらかの制限値を設定します。詳細については、「Amazon GameLift のキャパシティ制限を設定する」を参照してください。
2. 新規ポリシーを作成します。コマンドラインウィンドウを開き、ポリシーのパラメータ設定で put-scaling-policy コマンドを使用します。既存のポリシーを更新するには、ポリシー名を指定して、更新されたポリシーの完全バージョンを提供します。

```
--fleet-id <unique fleet identifier>
--name "<unique policy name>"
--policy-type <target- or rule-based policy>
--metric-name <name of metric>
--comparison-operator <comparison operator>
--threshold <threshold integer value>
--evaluation-periods <number of minutes>
--scaling-adjustment-type <adjustment type>
--scaling-adjustment <adjustment amount>
```

例:

```
aws gamelift put-scaling-policy \
    --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
    --name "Scale up when AGS<50" \
    --policy-type RuleBased \
    --metric-name AvailableGameSessions \
    --comparison-operator LessThanThreshold \
```

```
    --threshold 50 \
    --evaluation-periods 10 \
    --scaling-adjustment-type ChangeInCapacity \
    --scaling-adjustment 1
```

AWS CLI を使用して、ルールベースのスケーリング ポリシーを削除するには

- コマンドラインウィンドウを開き、フリート ID とポリシー名で delete-scaling-policy コマンドを使用します。

  例:

```
aws gamelift delete-scaling-policy \
    --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
    --name "Scale up when AGS<50"
```

自動スケーリングルールの構文

ルールベースのスケーリングポリシーステートメントを作成するには、6 つの変数を指定します。

<######> が *<####>* にわたって *<####>* *<#####>* である場合は、*<#####>* を使用してフリートの容量を *<###>* に変更します。

例えば、このポリシーステートメントは、フリートの余分なキャパシティが 50 の新規ゲームセッションを処理するために必要な数より少ない場合に、スケールアップイベントを開始します。

`AvailableGameSessions` が `less than 50` で `10 minutes` の場合、`ChangeInCapacity` を使用してフリート容量を `1 instances` で調整します。

メトリクス名

スケーリングイベントを開始するには、自動スケーリングポリシーを次のフリート指定のメトリクスの 1 つにリンクします。メトリクスの完全な説明については、「フリートの Amazon GameLift メトリクス」を参照してください。

- ゲームセッションのアクティブ化
- アクティブなゲームセッション
- 使用可能なゲームセッション
- 使用可能なゲームセッションの割合

- アクティブなインスタンス
- 使用可能なプレイヤーセッション
- 現在のプレイヤーセッション
- アイドル状態のインスタンス
- アイドル状態のインスタンスの割合

フリートがゲームセッションキューに含まれている場合、次のメトリクスを使用できます。

- キューの深さ – このフリートを最適なホスティング場所とする保留中のゲームセッションリクエストの数。
- 待機時間 – フリート固有の待機時間。保留中の最も古いゲームセッションリクエストが受理されるまでに待機する時間です。フリートの待機時間は、現在のキュー内で最も古いリクエストの時間と同じです。

Comparison operator (比較演算子)

Amazon GameLift に、メトリクスデータとしきい値を比較する方法を伝えます。有効な比較演算子は、より大きい (>)、より小さい (<)、以上 (>=)、および以下 (<=) です。

しきい値

指定したメトリック値がそのしきい値に到達したか、または超えた場合、スケーリングイベントが開始されます。この値は常に正の整数です。

評価期間

スケーリングイベントを開始する前に、メトリクスが全評価期間にわたってしきい値に到達するか、超える必要があります。評価期間は連続しています。メトリクスがしきい値を下回ると、評価期間が再度開始します。

調整値と調整タイプ

この一連の変数は連動して、スケーリング イベントがトリガーされるときに Amazon GameLift がフリートの容量を調整する方法を指定します。3 つの可能な調整タイプから選択します。

- [Change in capacity]（容量の変更） – 指定したインスタンス数だけ現在の容量を増減させます。調整値は、フリートに追加または削除するインスタンス数に設定します。正の値はインスタンスを追加し、負の値はインスタンスを削除します。例えば、「-10」の値は、フリートの合計サイズに関係なく、フリートを 10 インスタンス分スケールダウンします。
- [Percen change in capacity]（容量の割合変更） – 指定したパーセントだけ現在の容量を増減させます。調整値は、フリートのキャパシティを増減させるパーセントに設定します。正の値はインスタンスを追加し、負の値はインスタンスを削除します。例えば、50 個のインスタンスが

あるフリートでは、「20」パーセントの変更によって 10 個のインスタンスがフリートに追加されます。

- [正確な必要キャパシティー] – 指定した値まで現在のキャパシティを増減させます。調整値では、フリート内で維持する正確なインスタンス数を設定します。

ルールベースの自動スケーリングのヒント

次の推奨事項は、ルールベースのポリシーの自動スケーリングを最大限に活用するのに役立ちます。

複数のポリシーの使用

フリートには、複数の自動スケーリングポリシーを同時に適用できます。最も一般的なシナリオでは、ターゲットベースのポリシーでほとんどのスケーリングニーズを管理し、特別なケースを処理するためにルールベースのポリシーを使用します。複数のポリシーを使用することに制限はありません。

複数のポリシーでは、各ポリシーが独立して動作します。スケーリングイベントの順序を制御する方法はありません。例えば、スケールアップを扱う複数のポリシーがある場合、プレイヤーアクティビティによって複数のスケーリングポリシーが同時に開始される可能性があります。互いに開始するポリシーは避けてください。例えば、互いのしきい値を超えたキャパシティを設定するスケールアップポリシーとスケールダウンポリシーを作成した場合、無限ループが作成される可能性があります。

最大容量と最小容量の設定

各フリートには、最大容量と最小容量の制限があります。この機能は自動スケーリングを使用するときに重要です。自動スケーリングが、キャパシティをこの範囲外の値に設定することはありません。デフォルトでは、新しく作成されたフリートの最小値は 0、最大値は 1 です。自動スケーリングポリシーが意図どおりにキャパシティに影響を与えるには、最大値を大きくします。

フリートのキャパシティは、フリートのインスタンスタイプの制限および AWS アカウント のサービスクォータによっても制約されます。これらの制限とアカウントのクォータを超えて最小値と最大値を設定することはできません。

容量の変更後のメトリクスの追跡

自動スケーリングポリシーに応じてキャパシティを変更した後、Amazon GameLift は同じポリシーからのトリガーに応答するまで 10 分間待ちます。この待機により、Amazon GameLift は新しいインスタンスの追加、ゲームサーバーの起動、プレイヤーの接続、新しいインスタンスからのデータ収集の開始に必要な時間を確保できます。この間、Amazon GameLift はメトリクスに対してポリシーを

評価し、スケーリングイベントの発生後に再開されたポリシーの評価期間を追跡します。これは、待機時間が終わるとすぐに、スケーリングポリシーにより別のスケーリングイベントが開始される可能性があることを意味します。

異なる自動スケーリングポリシーが開始するスケーリングイベント間に待機時間はありません。

ゲームセッションプレイスメント用の Amazon GameLiftt キューのセットアップ

ゲームセッションキューは、新規のゲームセッションリクエストを処理し、それらをホストするために利用可能なゲームサーバーを探す主要なメカニズムです。キューはゲーム開発者やプレイヤーに大きなメリットをもたらします。具体的には次のとおりです。

- キューは可能な限り最適なプレイスメントを提供します。ゲームセッションプレイスメントリクエストを処理する場合、キューは Amazon GameLift アルゴリズムを使用して、定義された一連のプリファレンスに基づいてキューのロケーションに優先順位を付けます。
- 低料金のスポットフリートでゲームをホストします。キューを使用したAWS スポットフリートの使用の最適化は、ホスティングコストを大幅に削減します。デフォルトでは、キューは常に新規のゲームセッションをスポットフリートに配置しようとします。
- 需要の多いときに新規のゲームをより速く配置します。キューは、プレイスメントが可能な複数のロケーションを使用します。つまり、優先的に配置されるロケーションが使用できない場合は、常に予備の容量が存在します。
- ゲームの稼働率をより回復性のあるものにします。停電は起こり得ます。マルチロケーションキューを使用すれば、速度の低下または停止によって、プレイヤーのゲームへのアクセスが影響を受けることはありません。
- 追加のフリート容量をより効率的に使用します。予期しないプレイヤーの需要の急増を処理するため、キューは追加のホスティングキャパシティへの迅速なアクセスを提供します。キュー内のフリートロケーションは互いにバックアップ容量を提供します。ロケーションはプレイヤーの需要に基づいてスケールアップまたはスケールダウンします。
- ゲームセッションの配置とキューパフォーマンスのメトリクス (測定法) を取得します。Amazon GameLift はキューのメトリクスを提供します。これには、プレイスメントの成功と失敗、キューのリクエスト数、キューでリクエストにかかる平均時間の統計が含まれます。Amazon GameLift コンソールまたは CloudWatch でこれらのメトリクスを表示することもできます。

キューの使用を開始するには、「ゲームセッションキューの設計」を参照してください。

ゲームセッションキューの設計

このトピックでは、レイテンシーを最小限に抑えてプレイヤーエクスペリエンスを提供し、ホスティングリソースを効率的に使用するキューを設計する方法について説明します。ゲームセッションキューとその仕組みの詳細については、「ゲームセッションプレイスメント用の Amazon GameLiftt キューのセットアップ」を参照してください。

Amazon GameLift の以下の機能にはキューが必要です。

- FlexMatch でのマッチメーキング
- Amazon でのスポットインスタンスの使用 GameLift

キューのスコープを定義します。

ゲームのプレイヤー集団には、一緒にプレイしてはいけないプレイヤーグループが存在する可能性があります。例えば、ゲームを 2 つの言語でパブリッシュする場合、各言語には独自のゲームサーバーが必要です。

プレイヤー集団のゲームセッション配置を設定するには、プレイヤーセグメントごとに個別のキューを作成します。各キューをスコープ設定して、プレイヤーを正しいゲームサーバーに配置します。キューのスコープ設定の一般的な方法には、次のようなものがあります。

- 地理的なロケーション別。複数の地理的エリアにゲームサーバーをデプロイする場合、プレイヤーのレイテンシーを削減するために、ロケーション別にプレイヤー用のキューを構築できます。
- ビルド/スクリプトのバリエーション別。ゲームサーバーのバリエーションが複数ある場合は、同じゲームセッションでプレイできないプレイヤーグループをサポートしている可能性があります。例えば、ゲームサーバーのビルドまたはスクリプトは、異なる言語やデバイスタイプなどをサポートすることがあります。
- イベントタイプ別。特別なキューを作成して、トーナメントやその他の特別なイベント参加者のゲームを管理できます。

プレイヤーレイテンシーポリシーを作成する

プレイスメントリクエストにプレイヤーレイテンシーデータが含まれている場合、アルゴリズムは、すべてのプレイヤーに対して、最小のレイテンシーロケーションのゲームセッションを検出します。平均プレイヤーレイテンシーに基づいてゲームセッションを配置することで、Amazon GameLift は多くのプレイヤーを高いレイテンシーのゲームに配置することを防ぎます。ただし、Amazon

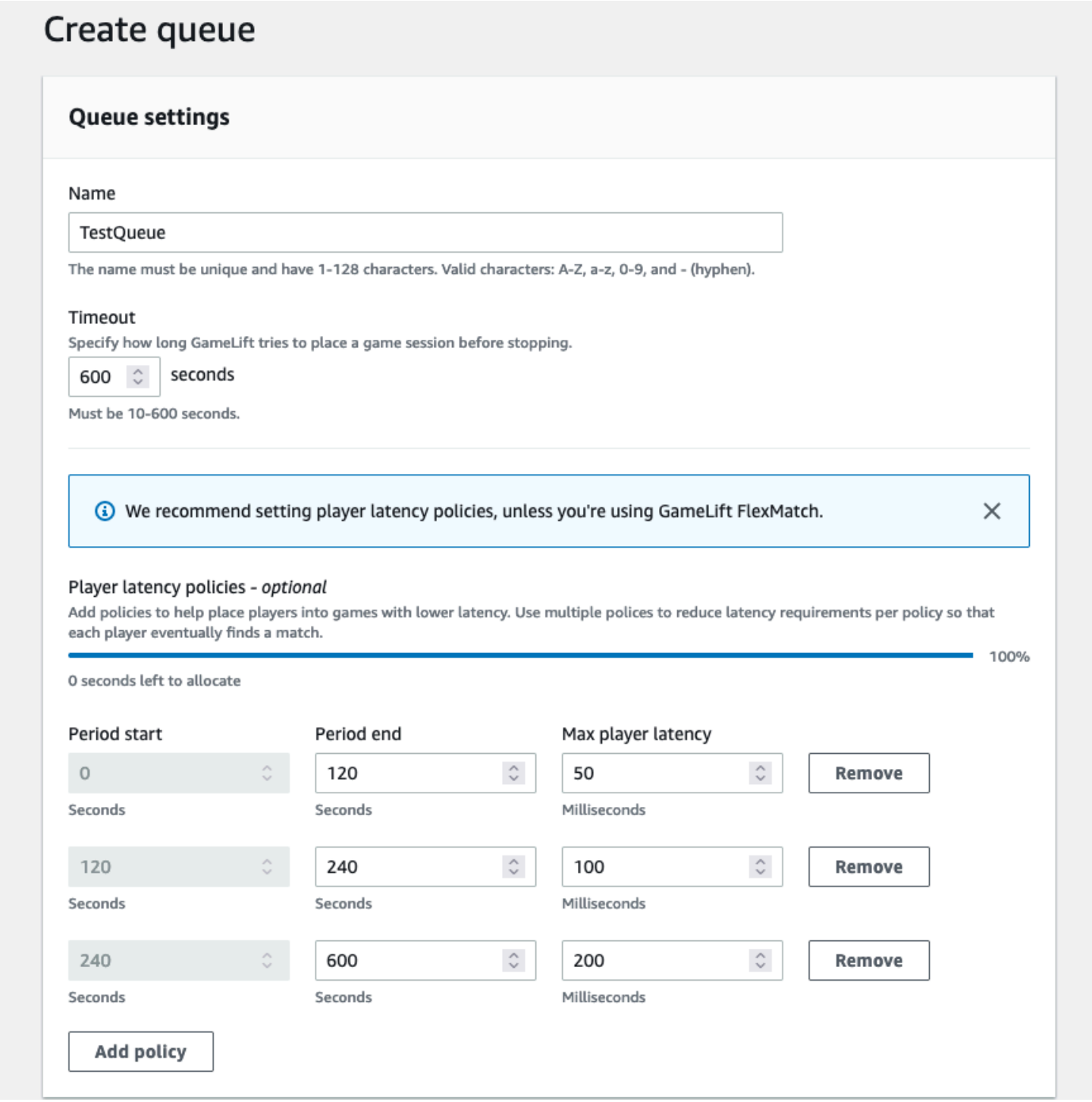
GameLift はそれでもプレイヤーを非常に高いレイテンシーで配置します。このようなプレイヤーに対応するため、プレイヤーレイテンシーポリシーを作成します。

プレイヤーレイテンシーポリシーは、Amazon GameLift がリクエストされたゲームセッションを、リクエスト内のプレイヤーが最大値を超えるレイテンシーを経験するロケーションに配置するのを防止します。また、プレイヤーレイテンシーポリシーは、Amazon GameLift がゲームセッションリクエストをレイテンシーの高いプレイヤーとマッチングしないようにすることもできます。

> **Tip**
>
> グループ内のすべてのプレイヤーで同様のレイテンシーを必要とするなど、プレイヤーのレイテンシーのルールを具体的に管理したい場合は、Amazon GameLift FlexMatch を使用して、レイテンシーベースのマッチメイキングルールを作成します。

例えば、タイムアウトが 5 分間で、次のプレイヤーレイテンシーポリシーのあるキューについて検討してみましょう。

1. 120 秒かけて、すべてのプレイヤーのレイテンシーが 50 ミリ秒未満のロケーションを検索します。
2. 120 秒かけて、すべてのプレイヤーのレイテンシーが 100 ミリ秒未満のロケーションを検索します。
3. 次に、キューのタイムアウトまでの残り時間を使って、すべてのプレイヤーレイテンシーが 200 ミリ秒未満のロケーションを検索します。



マルチロケーションキューを構築する

すべてのキューにマルチロケーション設計を推奨します。この設計により、プレイスメント速度とホスティングの耐障害性が向上します。プレイヤーレイテンシーデータを使用して、最小のレイテンシーでプレイヤーをゲームに参加させるには、マルチロケーション設計が必要です。スポットインス

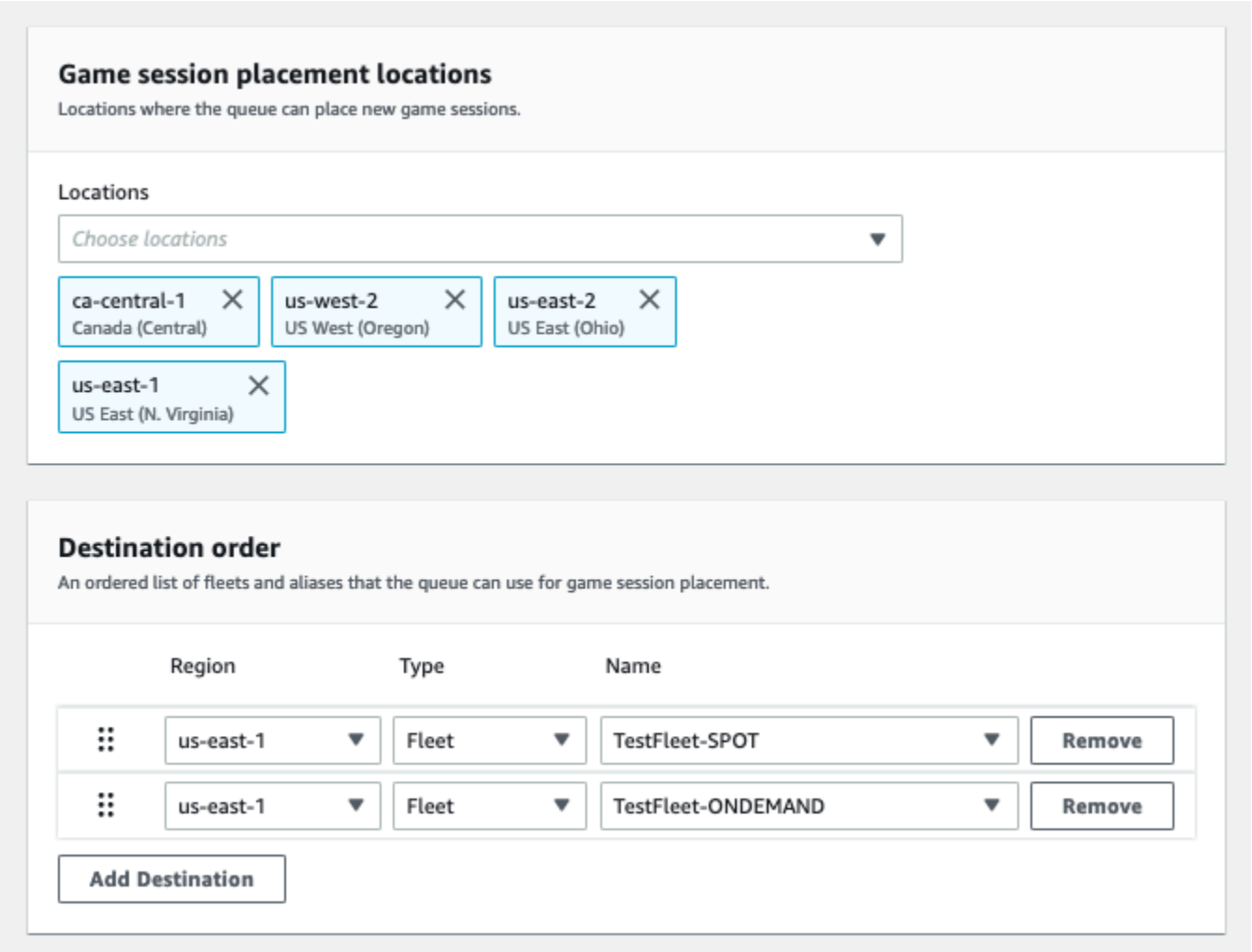
タンスフリートを使用するマルチロケーションキューを構築する場合は、「チュートリアル:スポットインスタンスのゲームセッションキューを設定する」の手順に従ってください。

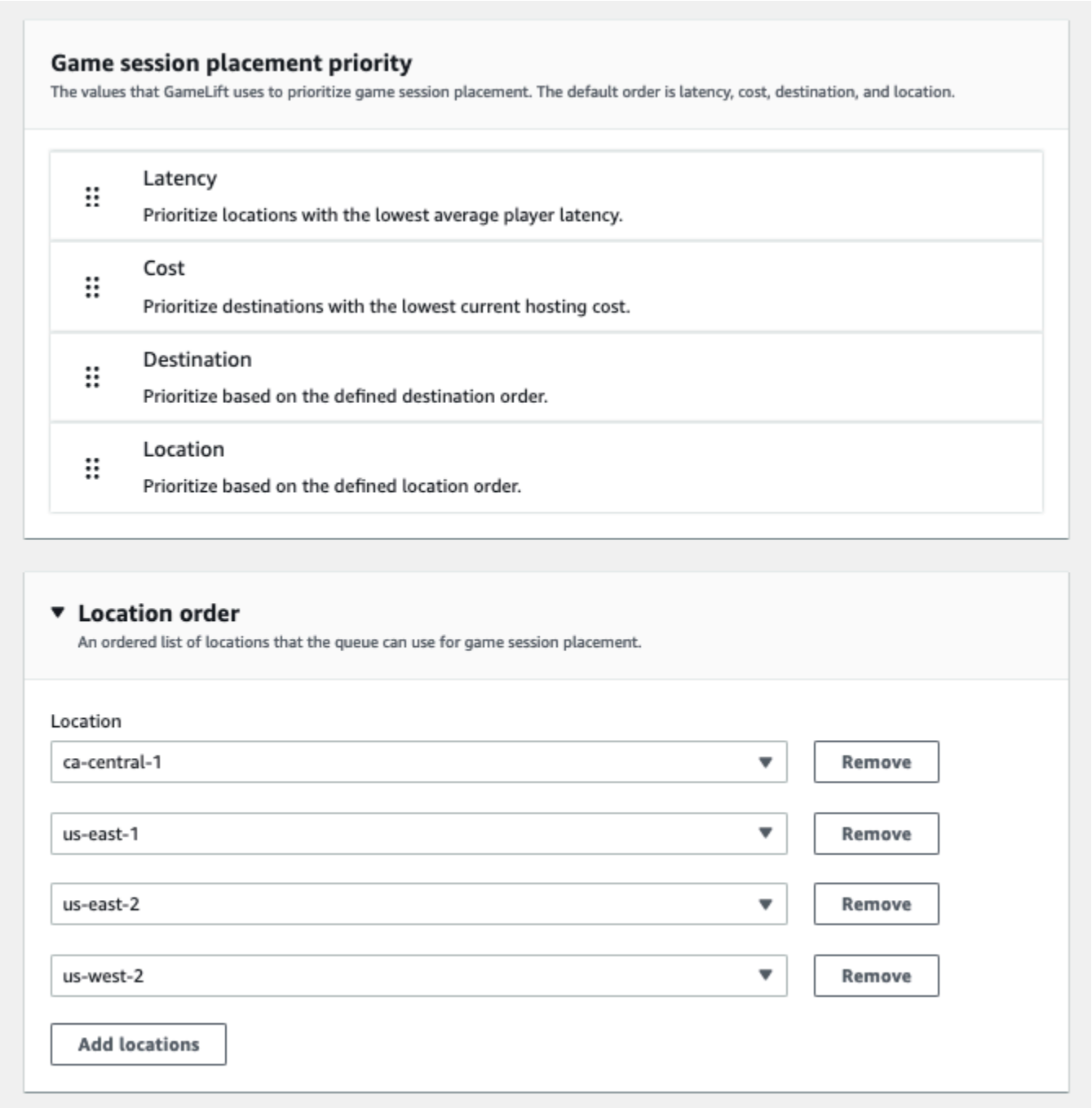
マルチロケーションキューを作成する 1 つの方法は、マルチロケーションフリートをキューに追加することです。こうすることで、キューは、任意のフリートのロケーションにゲームセッションを配置できます。冗長性を保つため、設定やホームロケーションが異なる他のフリートを追加することもできます。マルチロケーションのスポットインスタンスのフリートを使用している場合は、ベストプラクティスに従って、同じロケーションを含むオンデマンドインスタンスのフリートを含めてください。

次の例では、基本的なマルチロケーションキューの設計プロセスについて説明します。この例では、2 つのフリートを使用します。1 つはスポットインスタンスフリートで、もう 1 つはオンデマンドインスタンスフリートです。各フリートにはプレイスメントロケーション `us-east-1`、`us-east-2` `ca-central-1`、および `us-west-2` に次の AWS リージョン があります。

マルチロケーションフリートを含む基本的なマルチロケーションキューを作成するには

1. キューを作成するロケーションを選択します。クライアントサービスをデプロイした場所の近くのロケーションにキューを配置することで、リクエストの待ち時間を最小限に抑えることができます。この例では、キューを `us-east-1` に構築します。
2. 新しいキューを作成し、キューの送信先としてマルチロケーションフリートを追加します。送信先の順序によって、Amazon GameLift がゲームセッションを配置する方法が決まります。この例では、最初にスポットインスタンスフリートをリストし、オンデマンドインスタンスフリートを 2 番目にリストしています。
3. キューのゲームセッションのプレイスメント優先順位を定義します。この順序により、キューが利用可能なゲームサーバーを最初に検索するロケーションが決まります。この例では、デフォルトの優先順序を使用します。
4. ロケーションの順序を定義します。ロケーションの順序を定義しない場合、Amazon GameLift はロケーションをアルファベット順に使用します。


Game session placement locations
Locations where the queue can place new game sessions.
Locations
Choose locations
ca-central-1
Canada (Central)
us-west-2
US West (Oregon)
us-east-2
US East (Ohio)
us-east-1
US East (N. Virginia)
Destination order
An ordered list of fleets and aliases that the queue can use for game session placement.
Region
Type
Name
us-east-1
Fleet
TestFleet-SPOT
Remove
us-east-1
Fleet
TestFleet-ONDEMAND
Remove
Add Destination



ゲームセッションプレイスメントに優先順位を付ける

Amazon GameLift は FleetIQ アルゴリズムを使用して、順序付けられた一連の基準に基づいて新しいゲームセッションをどこに配置するかを決定します。デフォルトの優先順位を使用することも、順序をカスタマイズすることもできます。

デフォルトの優先順序

プレイヤーレイテンシーデータを含むプレイスメントリクエストの場合、FleetIQ はゲームセッションのプレイスメント条件を次のデフォルト順序で優先順位付けします。

1. [レイテンシー] – リクエスト内のすべてのプレイヤーの最小平均レイテンシー。
2. [コスト] – レイテンシーが複数のロケーションで等しい場合、最小のホスティングコスト。ホスティングコストは、主にインスタンスタイプとロケーションの組み合わせに基づいています。
3. [送信先] – レイテンシーとコストがマルチロケーションで等しい場合は、送信先の順序。FleetIQ は、キュー構成にリストされている順番に基づいて送信先の優先順位を付けます。
4. [ロケーション] – レイテンシー、コスト、および送信先が複数のロケーションにおいて等しい場合はロケーションの順序。FleetIQ は、キュー構成にリストされている順番に基づいてロケーションの優先順位を付けます。

カスタム優先順序

[Amazon GameLift コンソール] でキューの優先順位をカスタマイズするには、優先度の値を希望の位置にドラッグします。AWS Command Line Interface (AWS CLI) を使用してキューの優先順位をカスタマイズするには、create-game-session-queue コマンドと --priority-configuration オプションを使用します。このコマンドを使用して新しいキューを作成したり、既存のキューを更新できます。

FleetIQ アルゴリズムは、デフォルトの順序に基づいて、明示的に言及されていない条件をリストの最後に追加します。優先度設定にロケーション基準を含める場合は、ロケーションの順序付きリストも指定する必要があります。

必要に応じて複数のキューを設計する

ゲームとプレイヤーによっては、複数のゲームセッションキューを作成する必要がある場合があります。ゲームクライアントサービスが新規のゲームセッションをリクエストする際に、どのゲームセッションキューを使用するか指定されます。複数のキューを使用するかどうかを判断するには、以下を検討してください。

- ゲームサーバーのバリエーション ゲームサーバーのバリエーションごとに個別のキューを作成することができます。キュー内のすべてのフリートは、互換性のあるゲームサーバーをデプロイする必要があります。これは、キューを使用してゲームに参加するプレイヤーが、キュー内の任意のゲームサーバーでプレイできる必要があるためです。
- 異なるプレイヤーグループ。Amazon GameLift がプレイヤーグループに基づいてゲームセッションを配置する方法をカスタマイズできます。たとえば、特別なインスタンスタイプまたはランタイム設定を必要とする特定のゲームモードに合わせてキューをカスタマイズする必要がある場合

があります。または、トーナメントやその他のイベントのプレースメントを管理するために特別なキューが必要な場合があります。

- ゲームセッションのキューメトリクス。ゲームセッションプレイスメントメトリクスの収集方法に基づいて、キューを設定できます。詳細については、「キューの Amazon GameLift メトリクス」を参照してください。

キューメトリクスの評価

キューのパフォーマンスを評価するには、メトリクスを使用します。キューに関係するメトリクスは、Amazon GameLift コンソール または Amazon CloudWatch で表示できます。キューメトリクスのリストと説明については、「キューの Amazon GameLift メトリクス」を参照してください。

キューメトリクスは、以下に関する分析情報を提供します。

- [キュー全体のパフォーマンス] – キューメトリックは、キューがプレイスメントリクエストにどの程度正常に応答したかを示します。これらのメトリクスは、プレイスメントが失敗するタイミングと理由を特定するのにも役立ちます。フリートを手動でスケールしたキューの場合、`AverageWaitTime` および `QueueDepth` メトリクスはキューのキャパシティをいつ調整する必要があるか示す場合があります。
- [FleetIQ アルゴリズムのパフォーマンス — FleetIQ アルゴリズムを使用するプレイスメントリクエストの場合、メトリクスはアルゴリズムが理想的なゲームセッションプレイスメントを見つける頻度を示します。プレイスメントでは、プレイヤーのレイテンシーが最も低いリソースや、コストが最も低いリソースを優先的に使用することができます。Amazon GameLift が理想的なプレイスメントを発見できない一般的な理由を特定するエラーメトリクスもあります。メトリクスの詳細については、「Amazon CloudWatch で Amazon GameLift をモニタリングする」を参照してください。
- [ロケーション固有のプレイスメント] – マルチロケーションキューの場合、メトリクスはロケーション別の正常なプレイスメントを示します。FleetIQ アルゴリズムを使用するキューの場合、このデータはプレイヤーのアクティビティの発生箇所に関する有益な分析情報を提供します。

FleetIQ アルゴリズムのパフォーマンスメトリクスを評価する場合は、以下のヒントを参考にしてください。

- キューの理想的なプレイスメントを発見する比率を追跡するには、`PlacementsSucceeded` メトリクスと最小レイテンシーおよび最低料金に関する FleetIQ のメトリクスを併用します。

- キューが理想的なプレイスメントを見つける比率を上げるには、以下のエラーメトリクスを確認してください。
  - FirstChoiceOutOfCapacity が高い場合は、キューのフリートに合わせてキャパシティスケーリングを調整してください。
  - FirstChoiceNotViable エラーメトリクスが高い場合は、スポットインスタンスフリートを確認してください。特定のスポットインスタンスタイプの中断率が高すぎる場合、スポットフリートは「有効でない」とみなされます。この問題を解決するには、キューを変更して異なるインスタンスタイプのスポットインスタンスのフリートを使用します。ロケーションごとに異なるインスタンスタイプのスポットインスタンスフリートを含めることをお勧めします。

Amazon GameLift ゲームセッションキューのベストプラクティス

ここでは、ゲームセッション配置のために効果的なゲームセッションキューを構築するのに役立つベストプラクティスをいくつか紹介します。

任意のフリートタイプのキューのベストプラクティス

キューには、新規のゲームセッションを配置することができるフリートの送信先リストが含まれます。各フリートは、複数の地理的な場所にデプロイされたインスタンスを持つことができます。配置を選択するとき、キューはフリートとフリートロケーションの組み合わせを選択します。配置を選択するときに使用するキューの優先度のセットを指定します。

次のガイドラインおよびベストプラクティスを考慮します。

- プレイヤーが含まれるロケーションにフリートを追加します。フリートとエイリアスは任意のロケーションに追加できます。ロケーションは、報告されたプレイヤーのレイテンシーに基づいてプレイスメントを行う場合に重要です。
- すべてのフリートにエイリアスを使用します。キューの各フリートにエイリアスを割り当て、キューに送信先を設定するときにエイリアス名を使用します。
- すべてのフリートに同じ、または類似のゲームビルドまたはスクリプトを使用します。キューは、プレイヤーをキューにある任意のフリートのゲームセッションに配置できます。プレイヤーは、どのフリートのゲームセッションでもプレイできる必要があります。
- 2 つ以上のリージョンにフリートを作成します。ゲームサーバーを少なくとも他のひとつのロケーションでホストすることで、地域的な停止がプレイヤーに与える影響を軽減できます。バックアップフリートをスケールダウンしたままにして、使用量が増えた場合は自動スケーリングを使用してキャパシティを増やすことができます。

- ゲームセッションプレイスメントに優先順位を付けます。キューは、送信先リストの順序を含むいくつかの要素に基づいて配置の選択肢を優先的に選びます。
- クライアントサービスと同じロケーションにキューを作成します。クライアントサービスの近くのロケーションにキューを配置することで、通信レイテンシーを最小限に抑えることができます。
- 複数のロケーションでフリートを使用します。キューフィルター設定を使用して、指定したロケーションには、キューがゲームセッションを配置しないようにします。異なるホームロケーションで少なくとも 2 つのマルチロケーションフリートを使用して、地域的な停止時のゲームプレースメントの影響を軽減することができます。
- すべてのフリートに同じ TLS 証明書設定を使用します。フリートのゲームセッションに接続するゲームクライアントには、互換性のある通信プロトコルが必要です。

スポットフリートを使用したキューのベストプラクティス

キューにスポットフリートが含まれている場合は、耐障害性のあるキューを設定してください。スポットフリートによるコスト削減を生かしつつ、ゲームセッションの中断の影響を最小限に抑えます。スポットフリートで使用するためのフリートとゲームセッションキューを正しく構築する方法については、「チュートリアル:スポットインスタンスのゲームセッションキューを設定する」を参照してください。スポットインスタンスの詳細については、「Amazon でのスポットインスタンスの使用 GameLift」を参照してください。

前のセクションの一般的なベストプラクティスに加えて、次のスポット固有のベストプラクティスも考慮してください。

- 各ロケーションに少なくとも ひとつのオンデマンドフリートを作成します。オンデマンドフリートはプレイヤーにバックアップゲームサーバーを提供します。バックアップフリートが必要になるまでスケールダウンし、スポットフリートが利用できないときにはオートスケーリングを使用してオンデマンドの容量を増やすことができます。
- ロケーション内の複数のスポットフリートで異なるインスタンスタイプを選択します。1 つのスポットインスタンスタイプが一時的に使用できなくなった場合でも、中断はロケーション内のひとつのスポットフリートにのみ影響します。ベストプラクティスとしては、広く利用可能なインスタンスタイプを選択し、同じファミリのインスタンスタイプを使用します (たとえば、m5.large、m5.xlarge、m5.2xlarge)。Amazon GameLift コンソール を使用して、インスタンスタイプのコスト履歴データを確認します。

ゲームセッションキューを作成する

キューを使用して、複数のフリートおよびリージョン間で最適なホスティングリソースに新規のゲームセッションを配置します。ゲームのキューを構築する方法の詳細については、「ゲームセッションキューの設計」を参照してください。

ゲームクライアントで、配置リクエストを使用して、新規のゲームセッションがキューで開始されます。ゲームセッションの配置の詳細については、ゲームセッションを作成する を参照してください。

キュー内のキューの送信先を更新する場合には、短い移行期間 (最大 30 秒) があり、その間にキューの送信先に配置されたゲームセッションが古いフリートに残ってしまう可能性があります。

Console

1. [Amazon GameLift コンソール] のナビゲーションペインで、[キュー] を選択します。
2. [Queues] (キュー) ページで、[Create queue] (新しいキューの作成) を選択します。
3. [キューの作成] ページの [キュー設定] で、次の操作を行います。

   a. [名前] にキューの名前を入力します。

   b. [タイムアウト] には、Amazon GameLift がゲームセッションを停止するまで待機させたい時間を入力します。Amazon GameLift は、リクエストがタイムアウトするまで、あらゆるフリートの使用可能なリソースの検索します。

   c. (オプション) [プレイヤーレイテンシーポリシー] には、Amazon GameLift が定義された最大レイテンシー内でリソースを検索する時間を入力します。ポリシーを追加して、最大レイテンシーを徐々に緩和します。ポリシーを追加するには、[ポリシーを追加] を選択します。
4. [ゲームセッションの配置場所] で、キューに追加するロケーションを選択します。デフォルトでは [すべてのロケーション] が含まれます。キュー内のすべてのフリートに対して同じ証明書設定が必要です。すべてのフリートは、キューを使用するゲームクライアントと互換性があるゲームビルドを実行している必要があります。
5. [送信先] で、キューに 1 つ以上の送信先を追加します。

   a. 送信先の追加を選択します。

   b. 送信先がある [ロケーション] を選択します。

   c. 送信先のタイプを選択します。

d. 表示されたフリートまたはエイリアス名のリストから、追加するフリートまたはエイリアスを選択します。

e. 送信先が複数ある場合、送信先の左側の 6 つの点アイコンをドラッグしてデフォルトの順序を設定します。Amazon GameLift は、新規のゲームセッションを配置するのに使用可能なリソースの送信先を検索する際にこの順序を使用します

6. [ゲームセッションの配置優先順位] では、[レイテンシー]、[コスト]、[送信先]、[ロケーション] の値を追加してドラッグし、Amazon GameLift がキュー内のフリートに優先順位を付ける方法を定義します。フリートの優先順位の詳細については、「ゲームセッションプレイスメントに優先順位を付ける」を参照してください。

7. ロケーションを [ロケーションの順序] に追加し、キューが使用する優先度までドラッグします。[ロケーション] がゲームセッション配置の最優先順位である場合、Amazon GameLift はそれをタイブレーカーとして使用します。

8. (オプション) [イベント通知の設定] で、次の操作を行います。

   a. プレースメント関連のイベント通知を受信する SNS トピックを選択または作成します。 イベント通知の詳細については、「ゲームセッション配置のイベン通知を設定」を参照してください。

   b. このキューで作成されたイベントに追加する [カスタムイベントデータ] を追加します。

9. (オプション) [タグ] を追加します。タグ付けの詳細については、「AWS リソースのタグ付け」を参照してください。

10. [Create] (作成) を選択します。

AWS CLI

Example キューを作成する

次の例では、これらの設定を使用してゲームセッションキューを作成します。

- 5 分のタイムアウト
- 2 つのフリートの送信先
- `us-east-1`、`us-east-2`、`us-west-2` および `ca-central-1` 内のロケーションのみを許可するようにフィルタリングします。
- コストに基づいて送信先に優先順位を付け、次に、定義した順序でロケーションに優先順位を付けます。

```
aws gamelift create-game-session-queue \
    --name "sample-test-queue" \
    --timeout-in-seconds 300 \
    --destinations DestinationArn="arn:aws:gamelift:us-east-1:111122223333:fleet/
fleet-772266ba-8c82-4a6e-b620-a74a62a93ff8" DestinationArn="arn:aws:gamelift:us-
east-1:111122223333:fleet/fleet-33f28fb6-aa8b-4867-85b4-ceb217bf5994" \
    --filter-configuration "AllowedLocations=us-east-1, ca-central-1, us-east-2, us-
west-2" \
    --priority-configuration
 PriorityOrder="LOCATION","DESTINATION",LocationOrder="us-east-1","us-east-2","ca-
central-1","us-west-2" \
    --notification-target "arn:aws:sns:us-east-1:111122223333:gamelift-test.fifo"
```

> **Note**
>
> フリートおよびエイリアス ARN の値を取得するには、フリートまたはエイリアス ID を指定して describe-fleet-attributes または describe-alias を呼び出します。

`create-game-session-queue` リクエストが成功すると、Amazon GameLift は新しいキュー設定を含む GameSessionQueue オブジェクトを返します。これで、StartGameSessionPlacement を使用してリクエストをキューに送信できます。

Example プレイヤーレイテンシーポリシーを使用してキューを作成する

次の例では、これらの設定を使用してゲームセッションキューを作成します。

- 10 分のタイムアウト
- 3 つのフリートの送信先
- プレイヤーレイテンシーポリシーのセット

```
aws gamelift create-game-session-queue \
    --name "matchmaker-queue" \
    --timeout-in-seconds 600 \
    --destinations DestinationArn=arn:aws:gamelift:us-east-1::alias/alias-a1234567-
b8c9-0d1e-2fa3-b45c6d7e8910 \
                DestinationArn=arn:aws:gamelift:us-west-2::alias/alias-b0234567-
c8d9-0e1f-2ab3-c45d6e7f8901 \
                DestinationArn=arn:aws:gamelift:us-west-2::fleet/fleet-f1234567-
b8c9-0d1e-2fa3-b45c6d7e8912 \
```

```
   --player-latency-policies
"MaximumIndividualPlayerLatencyMilliseconds=50,PolicyDurationSeconds=120" \

"MaximumIndividualPlayerLatencyMilliseconds=100,PolicyDurationSeconds=120" \
              "MaximumIndividualPlayerLatencyMilliseconds=150" \
```

`create-game-session-queue` リクエストが成功すると、Amazon GameLift は新しいキュー設定を含む GameSessionQueue オブジェクトを返します。

ゲームセッション配置のイベン通知を設定

イベント通知を使用して、個々のプレイスメントリクエストのステータスをモニタリングできます。プレイスメントアクティビティが多いすべてのゲームにイベント通知を設定することをおすすめします。

イベント通知を設定するためには二つのオプションがあります。

- キューを使用して Amazon Simple Notification Service (Amazon SNS) トピックに対して Amazon GameLift にイベント通知をパブリッシュさせます。
- 自動的にパブリッシュされる Amazon EventBridge イベントとイベント管理ツール一式を使用します。

Amazon GameLift から生成されるゲームセッションプレイスメントイベントのリストについては、「ゲームセッションプレイスメントイベント」を参照してください。

SNS トピックの設定

Amazon GameLift を使用して、ゲームセッションキューで生成されたすべてのイベントを 1 つのトピックにパブリッシュします。

Amazon GameLift イベント通知用 SNS トピックのセットアップ

1. AWS Management Console にサインインし、Amazon SNS コンソール (https://console.aws.amazon.com/sns/v3/home) を開きます。
2. SNS [トピック] ページから [トピックを作成] を選択し、トピックを作成する手順に従います。
3. [アクセスポリシー] で、以下の作業を行います。

   a. [詳細設定] 方法を選択します。
   b. JSON オブジェクトの次の太字のセクションを既存のポリシーに追加します。

```
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__default_statement_ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SNS:GetTopicAttributes",
        "SNS:SetTopicAttributes",
        "SNS:AddPermission",
        "SNS:RemovePermission",
        "SNS:DeleteTopic",
        "SNS:Subscribe",
        "SNS:ListSubscriptionsByTopic",
        "SNS:Publish"
      ],
      "Resource": "arn:aws:sns:your_region:your_account:your_topic_name",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "your_account"
        }
      }
    },
    {
      "Sid": "__console_pub_0",
      "Effect": "Allow",
      "Principal": {
        "Service": "gamelift.amazonaws.com"
      },
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:your_region:your_account:your_topic_name",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn":
 "arn:aws:gamelift:your_region:your_account:gamesessionqueue/your_queue_name"
        }
      }
    }
  ]
```

```
}
```

c. (オプション) リソースポリシーに条件を追加して、トピックにアクセス制御を追加します。

4. [Create topic] (トピックの作成) を選択します。
5. SNS トピックを作成したら、キューの作成時にそのトピックをキューに追加するか、既存のキューを編集して追加します。

サーバー側の暗号化を使用して SNS トピックをセットアップする

サーバー側の暗号化 (SSE) では、機密データを暗号化されたトピックに保存できます。SSE は、AWS Key Management Service (AWS KMS) のマネージドキーを使用して、Amazon SNS トピック内のメッセージの内容を保護します。Amazon SNS によるサーバー側の暗号化の詳細については、Amazon Simple Storage Service 開発者ガイドの「保管時の暗号化」を参照してください。

サーバー側の暗号化を使用して SNS トピックを設定する方法については、以下のトピックを確認してください。

- AWS Key Management Service デベロッパーガイドの「キーの作成」。
- Amazon Simple Notification Service 開発者ガイドの「トピックの SSE を有効にする」

KMS キーを作成するときは、次の KMS キーポリシーを使用します。

```
{
    "Effect": "Allow",
    "Principal": {
        "Service": "gamelift.amazonaws.com"
     },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*",
    "Condition": {
        "ArnLike": {
            "aws:SourceArn":
 "arn:aws:gamelift:your_region:your_account:gamesessionqueue/your_queue_name"
        },
        "StringEquals": {
            "kms:EncryptionContext:aws:sns:topicArn":
 "arn:aws:sns:your_region:your_account:your_sns_topic_name"
```

```
        }
    }
}
```

EventBridge をセットアップする

Amazon GameLift は、すべてのゲームセッションプレイスメントイベントを EventBridge に自動的に投稿します。EventBridge を使用すると、処理するためにイベントをターゲットにルーティングするルールを設定できます。例えば、イベント `PlacementFulfilled` を、ゲームセッションに接続する前のタスクを処理する AWS Lambda 関数にルーティングするルールを設定できます。Eventbridge の詳細については、Amazon EventBridge ユーザーガイドの Amazon EventBridge とは を参照してください。

Amazon GameLift キューで使用する EventBridge ルールの例を以下に示します。

Amazon GameLift のすべてのキューからのイベントにマッチします

```
{
    "source": [
        "aws.gamelift"
    ],
    "detail-type": [
        "GameLift Queue Placement Event"
    ]
}
```

特定のキューからのイベントに一致します

```
{
    "source": [
        "aws.gamelift"
    ],
    "detail-type": [
        "GameLift Queue Placement Event"
    ],
    "resources": [
        "arn:aws:gamelift:your_region:your_account:gamesessionqueue/your_queue_name"
    ]
}
```

チュートリアル:スポットインスタンスのゲームセッションキューを設定する

序章

このチュートリアルでは、低コストのスポットフリートにデプロイされるゲームのゲームセッションプレイスメントを設定する方法について説明します。スポットフリートでは、プレイヤーのゲームサーバーの継続的な可用性を維持するために、追加の手順が必要です。

対象者

このチュートリアルは、スポットフリートを使用してカスタムゲームサーバーまたはリアルタイムサーバーをホストするゲームデベロッパー向けです。

学習する内容

- ゲームセッションキューによって提供されるプレイヤーのグループを定義します。
- ゲームセッションキューのスコープをサポートするフリートインフラストラクチャを構築します。
- 各フリートにエイリアスを割り当てて、フリート ID を抽象化します。
- キューを作成し、フリートを追加し、Amazon GameLift がゲームセッションを配置する場所に優先順位を付けます。
- プレイヤーレイテンシーポリシーを追加して、レイテンシー問題を最小限に抑えます。

前提条件

ゲームセッションプレイスメント用のフリートとキューを作成する前に、次のタスクを完了する必要があります。

- 確認Amazon GameLift の仕組み。
- ゲームサーバーを Amazon GameLift と統合する。
- ゲームサーバービルドまたはリアルタイムスクリプトを Amazon GameLift にアップロードする。
- フリート設定を計画する。

ステップ 1: キューのスコープを定義する

このチュートリアルでは、ゲームサーバービルドのバリエーションが 1 つのゲームのキューを設計します。開始時には、アジアパシフィック (ソウル) とアジアパシフィック (シンガポール)) の 2 か所

でリリースされます。これらのロケーションは互いに近いので、レイテンシーはプレイヤーにとって問題ではありません。

この例では、プレイヤーセグメントが 1 つしかありません。つまり、キューを 1 つ作成します。将来、北米でゲームがリリースされるときには、北米のプレイヤーを対象とした 2 つ目のキューを作成できます。

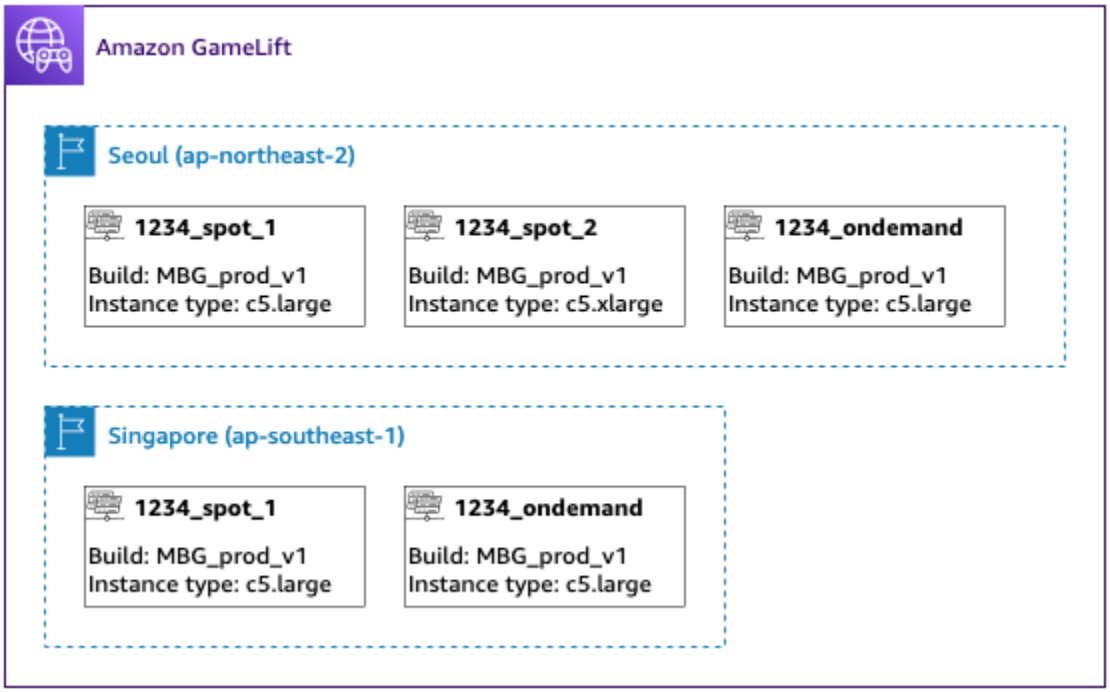
詳細については、「キューのスコープを定義します。」を参照してください。

ステップ 2: スポットフリートインフラストラクチャを作成する

ロケーションにフリートを作成し、ステップ 1: キューのスコープを定義する で定義したスコープに適合するゲームサーバービルドまたはスクリプトを使用します。

このチュートリアルでは、各ロケーションに、少なくとも 1 つのスポットフリートと 1 つのオンデマンドフリートを含む 2 つのロケーションインフラストラクチャを作成します。各フリート毎に同じゲームサーバービルドをデプロイします。さらに、ソウルのロケーションではプレイヤーのトラフィックが重くなることが予想されるため、そこにスポットフリートを追加します。

次の図は、ap-northeast-2 (ソウル) ロケーションに 3 つのフリートと ap-southeast-1 (シンガポール) ロケーションに 2 つのフリートがあるスポットフリートインフラストラクチャの例を示しています。両方のフリートのすべてのインスタンスは MBG_prod_v1 というビルドを使用しています。ap-northeast-2 のフリートにはインスタンスタイプが c5.large のフリート 1234_spot_1、インスタンスタイプが c5.xlarge のフリート 1234_spot_2、インスタンスタイプが c5.large のフリート 1234_ondemand のフリート設定が含まれています。ap-southeast-1 のフリートにはインスタンスタイプが c5.large のフリート 1234_spot_1 と、インスタンスタイプが c5.large のフリート 1234_ondemand のフリート設定が含まれています。

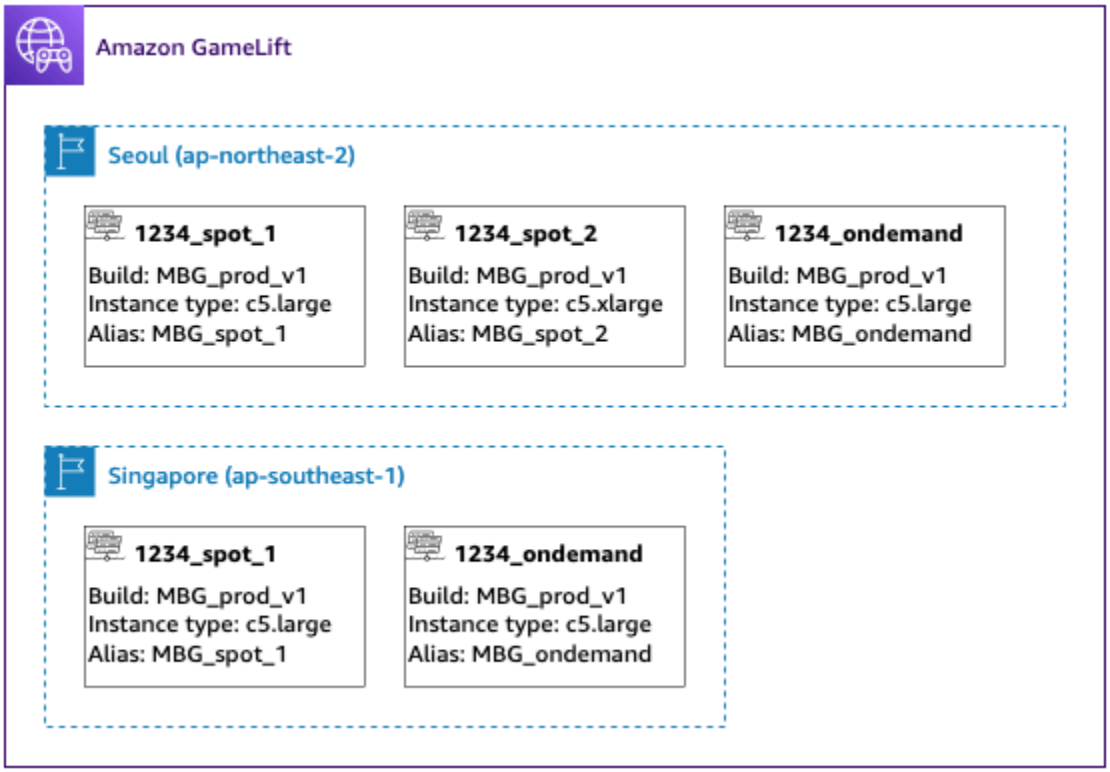


ステップ 3: フリートごとにエイリアスを割り当てる

インフラストラクチャの各フリートに新しいエイリアスを作成する。エイリアスはフリート ID を抽象化し、定期的なフリート交換を効率的にします。アラーム作成の詳細については、「Amazon GameLift フリートにエイリアスを追加する」を参照してください。

フリートインフラストラクチャにはフリートが 5 つあるため、ルーティング戦略を使用して 5 つのエイリアスを作成します。アジアパシフィック (ソウル) のロケーションには 3 つのエイリアスが必要で、アジアパシフィック (シンガポール) のロケーションには 2 つのエイリアスが必要です。

次の図は、ステップ 2 で説明したスポットフリートのインフラストラクチャと、各フリートにエイリアスを追加したものです。フリート 1234_spot_1 のエイリアスには MBG_spot_1、フリート 1234_spot_2 のエイリアスには MBG_spot_2、フリート 1234_ondemand のエイリアスには MBG_ondemand があります。



詳細については、「マルチロケーションキューを構築する」を参照してください。

ステップ 4: 送信先のキューを作成する

ゲームセッションキューを作成し、フリートの送信先を追加します。キューの作成方法の詳細については、「ゲームセッションキューを作成する」を参照してください。

キューを作成するとき:

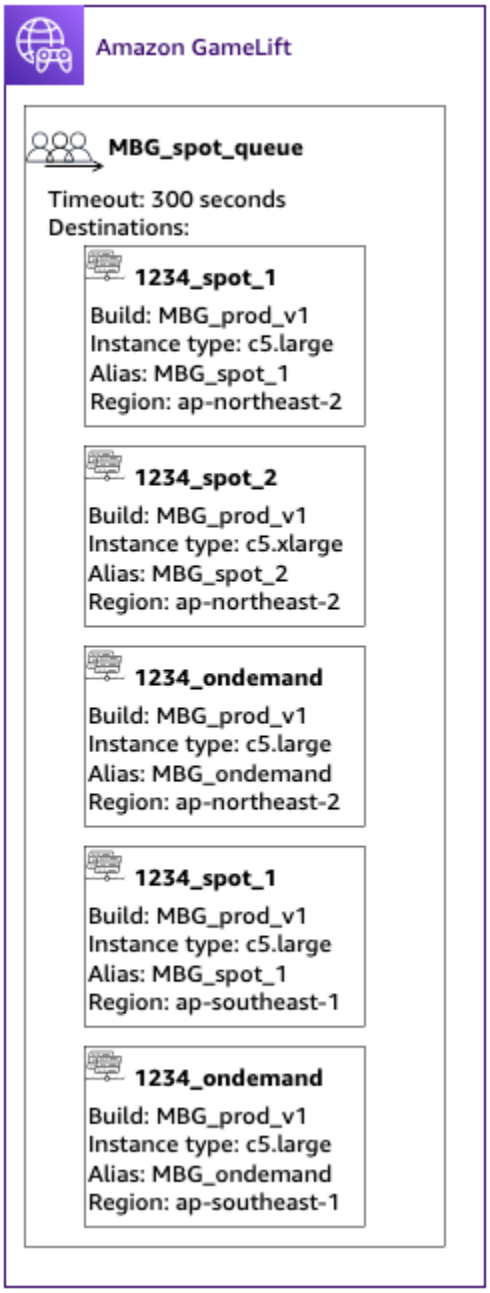
- デフォルトのタイムアウト値を 10 分に設定します。後に、キューのタイムアウトがゲームに参加するためのプレイヤーの待ち時間にどのように影響するかをテストできます。
- 今のところ、プレイヤーレイテンシーポリシーのセクションはスキップしてください。これについては、次のステップで説明します。
- キュー内のフリートに優先順位を付けます。スポットフリートを操作する場合は、次のいずれかの方法をお勧めします。
  - インフラストラクチャで、バックアップ用にセカンダリロケーションにフリートがあるプライマリロケーションを使用している場合は、最初にロケーション別、次にフリートタイプ別に優先順位を付けます。

- インフラストラクチャで複数の口ケーションを均等に使用する場合は、フリートタイプでフリートに優先順位を付け、スポットフリートをキューの先頭に配置します。

この例では、**MBG_spot_queue** という名前の新しいキューを作成し、5 つのフリートすべてに工イリアスを追加します。次に、最初に口ケーションでプレイスメントを優先順位付けし、次にフリートタイプで優先順位付けします。

この設定に基づいて、このキューでは常に新規のゲームセッションをソウルのスポットフリートに配置しようとします。これらのフリートがいっぱいになると、キューはソウルオンデマンドフリートの利用可能なキャパシティをバックアップとして使用します。3 つのソウルのフリートすべてが利用できなくなった場合、Amazon GameLift はシンガポールのフリートにゲームセッションを配置します。

次の図は、タイムアウトが 300 秒のキューと優先順位付けされた送信先を示しています。目的地は、ap-northeast-2 の 1234_spot_1、ap-northeast-2 の 1234_spot_2、ap-northeast-2 の 1234_ondemand、ap-southeast-1 の 1234_spot_1、ap-southeast-1 の 1234_ondemand の順になっています。



ステップ 5: レイテンシー制限をキューに追加する

このゲームでは、ゲームセッションのプレイスメントリクエストにレイテンシー情報が含まれています。プレイヤーグループのためのゲームセッションを作成するプレイヤーパーティー機能もあります。私たちは、ゲーム参加までプレイヤーに少し長く待ってもらうことで、可能な限り理想的なゲームプレイエクスペリエンスが得られるようにできます。弊社のゲームテストでは次のような結果が出ています。

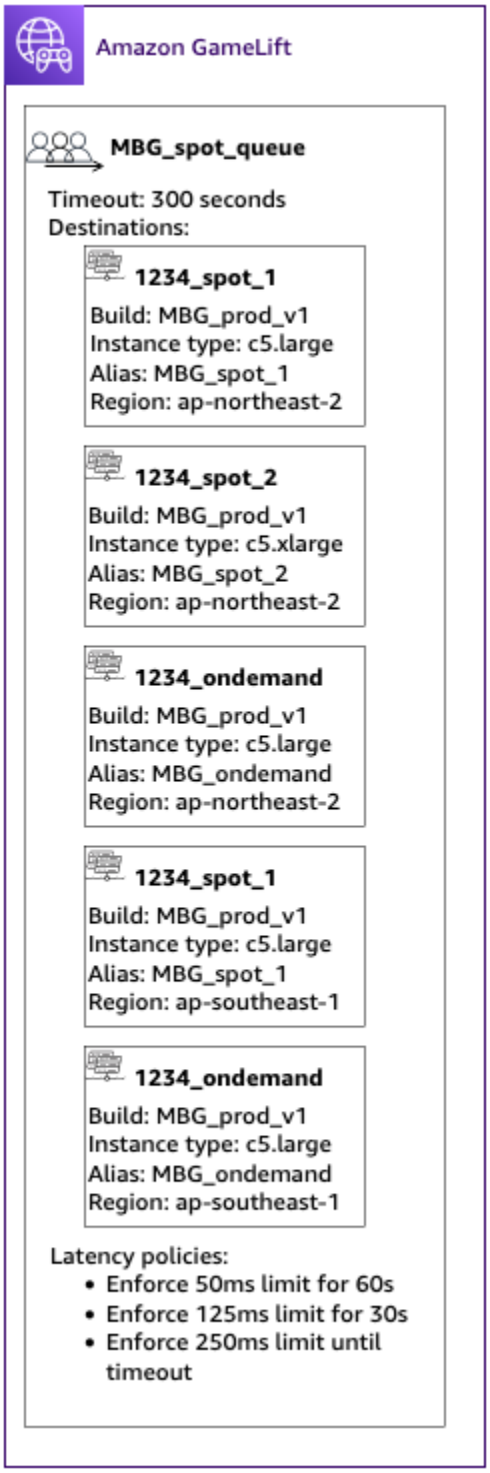
- 理想的なレイテンシーは 50 ミリ秒未満。

- 250 ミリ秒を超えるレイテンシーではゲームはプレイできない。
- プレイヤーは約 1 分以内に我慢の限界に達する。

300 秒のタイムアウトがあるキューには、許容レイテンシーを制限するポリシーステートメントを追加します。ポリシーステートメントでは、レイテンシー値を徐々に 250 ミリ秒まで増加させることができます。

このポリシーでは、キューは最初の 1 分間、最適なレイテンシー（50 ミリ秒未満）のプレイスメントを検索し、それ以降は制限を緩和します。キューは、プレイヤーのレイテンシーが 250 ミリ秒以上のプレイスメントは行いません。

次の図は、プレイヤーのレイテンシーポリシーが追加されたステップ 4 のキューを示しています。プレイヤーレイテンシーポリシーには、60 秒間は 50 ミリ秒の制限を適用、30 秒の間は 125 ミリ秒の制限を適用、タイムアウトまで 250 ミリ秒の制限を強制すると規定されています。



まとめ

お疲れ様でした。以下はあなたが達成した内容です。

- プレイヤー集団の一部を対象とするゲームセッションキューができました。

- キューはスポットフリートを効果的に使用し、スポットの中断が発生したときにも回復力があります。
- キューでは、最高のプレイヤーエクスペリエンスが得られるフリートが優先されます。
- キューには、劣悪なゲームプレイエクスペリエンスからプレイヤーを保護するためにレイテンシー制限が設定されています。

これで、キューを使用して、提供しているプレイヤーゲームセッションを配置できます。これらのプレイヤーに対してゲームセッションプレイスメントリクエストを行う場合は、リクエストでこのゲームセッションキュー名を参照します。ゲームセッションのプレイスメントリクエストの詳細については、ゲームセッションを作成する または リアルタイムサーバー用のゲームクライアントの統合 を参照してください。

次のステップ:

- 独自のキューを設計する。
- キューを作成する。
- ゲームクライアントでキューを使用する。

AWS CloudFormation を使用したリソースの管理

AWS CloudFormation を使用して、Amazon GameLift リソースへのアクセスを管理できます。AWS CloudFormation で、各リソースをモデル化するテンプレートを作成し、そのテンプレートを使用してリソースを作成します。リソースを更新するには、テンプレートに変更を加え、AWS CloudFormation を使用して更新を実装します。リソースは、スタックおよびスタックセットと呼ばれる論理グループに編成できます。

AWS CloudFormation を使用して Amazon GameLift ホスティングリソースを維持すると、AWS リソースセットをより効率的に管理できます。バージョン管理を使用して、テンプレートの経時的な変更を追跡し、複数のチームメンバーによる更新を調整できます。テンプレートを再利用することもできます。たとえば、複数のリージョンにゲームをデプロイする場合、同じテンプレートを使用して各リージョンに同一のリソースを作成できます。これらのテンプレートを使用して、同じリソースセットを別のパーティションにデプロイすることもできます。

AWS CloudFormation の詳細については、「AWS CloudFormation ユーザーガイド」を参照してください。Amazon GameLift リソースのテンプレート情報を表示するには、「Amazon GameLift リソースタイプリファレンス」を参照してください。

ベストプラクティス

AWS CloudFormation の使用に関する詳細なガイダンスについてはAWS CloudFormation ユーザーガイドの「AWS CloudFormation のベストプラクティス」を参照してください。さらに、これらのベストプラクティスは Amazon GameLift と特別な関連性があります。

- AWS CloudFormation によりリソースを一貫して管理する。AWS CloudFormation リソース以外のリソースを変更すると、リソースはリソーステンプレートと同期しなくなります。
- AWS CloudFormation スタックおよびスタックセットを使用して、複数のリソースを効率的に管理する。
  - スタックを使用して、接続されたリソースのグループを管理します。例えば、ビルド、ビルドを参照するフリート、フリートを参照するエイリアスを含むスタックがあるとします。テンプレートを更新してビルドを置き換えた場合、AWS CloudFormation はビルドに接続されているフリートを置き換えます。その後、AWS CloudFormation によって、新しいフリートを参照するように既存のエイリアスが更新されます。詳細については、AWS CloudFormation ユーザーガイドの「スタックの操作」を参照してください。
  - 複数のリージョンまたは AWS アカウントに同一のスタックをデプロイする場合は、AWS CloudFormation スタックセットを使用します。詳細については、AWS CloudFormationユーザーガイドの「スタックセットの操作」を参照してください。
- スポットインスタンスを使用する場合は、オンデマンドフリートをバックアップとして含める。リージョンごとに 2 つのフリート (スポットインスタンスを使用するフリートおよびオンデマンドインスタンスを使用するフリート) でテンプレートを設定することをお勧めします。
- 複数のリージョンでリソースを管理している場合は、ロケーション固有のリソースとグローバルリソースを別々のスタックにグループ化します。
- グローバルリソースは、そのリソースを使用するサービスの近くに配置します。キューやマッチメーキング設定などのリソースは、特定のソースから大量のリクエストを受信する傾向があります。リソースをこれらのリクエストのソースの近くに配置することで、リクエストの移動時間を最短に抑え、全体的なパフォーマンスを向上させることができます。
- マッチメーキング設定は、その設定を使用するゲームセッションキューと同じリージョンに配置する。
- スタック内のフリートごとに個別のエイリアスを作成する。

AWS CloudFormation スタックの使用

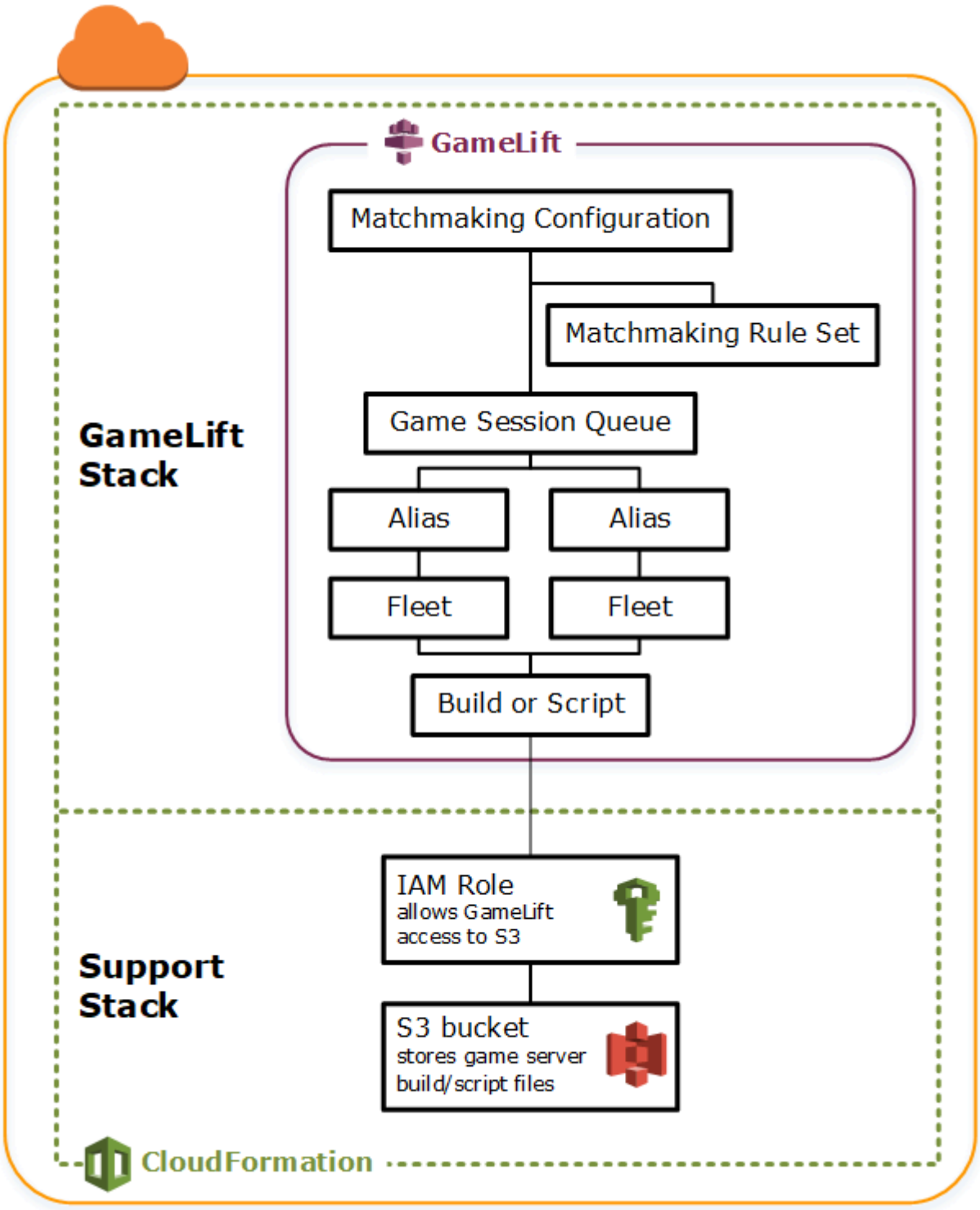
Amazon GameLift 関連リソースの AWS CloudFormation スタックを設定するときに以下の構造を使用することをお勧めします。最適なスタック構造は、ゲームを 1 つの口ケーションにデプロイするか、複数の口ケーションにデプロイするかによって異なります。

1 つの口ケーションのスタック

1 つの口ケーションで Amazon GameLift リソースを管理するには、2 スタック構造をお勧めします。

- サポートスタック – このスタックには、Amazon GameLift リソースが依存するリソースが含まれます。少なくとも、このスタックには、カスタムゲームサーバーまたは Realtime スクリプトファイルを保存する S3 バケットが含まれる必要があります。また、このスタックには、Amazon GameLift ビルドまたはスクリプトリソースを作成するときに S3 バケットからファイルを取得するためのアクセス許可を Amazon GameLift に付与する、IAM 口ールも含まれる必要があります。このスタックには、DynamoDB テーブル、Amazon Redshift クラスター、Lambda 関数など、ゲームで使用される他の AWS リソースも含まれる場合があります。
- Amazon GameLift スタック - このスタックには、すべての Amazon GameLift リソース (ビルドまたはスクリプト、フリートのセット、エイリアス、ゲームセッションキューなど) が含まれます。AWS CloudFormation は、S3 バケットの場所に保存されているファイルを使用してビルドまたはスクリプトリソースを作成し、1 つまたは複数のフリートリソースにデプロイします。フリートごとに対応するエイリアスが必要です。ゲームセッションキューはフリートエイリアスの一部またはすべてを参照します。マッチメーキングに FlexMatch を使用する場合、このスタックにはマッチメーキング設定とルールセットも含まれます。

以下の図では、1 つの AWS リージョンにリソースをデプロイする 2 スタック構造を示しています。



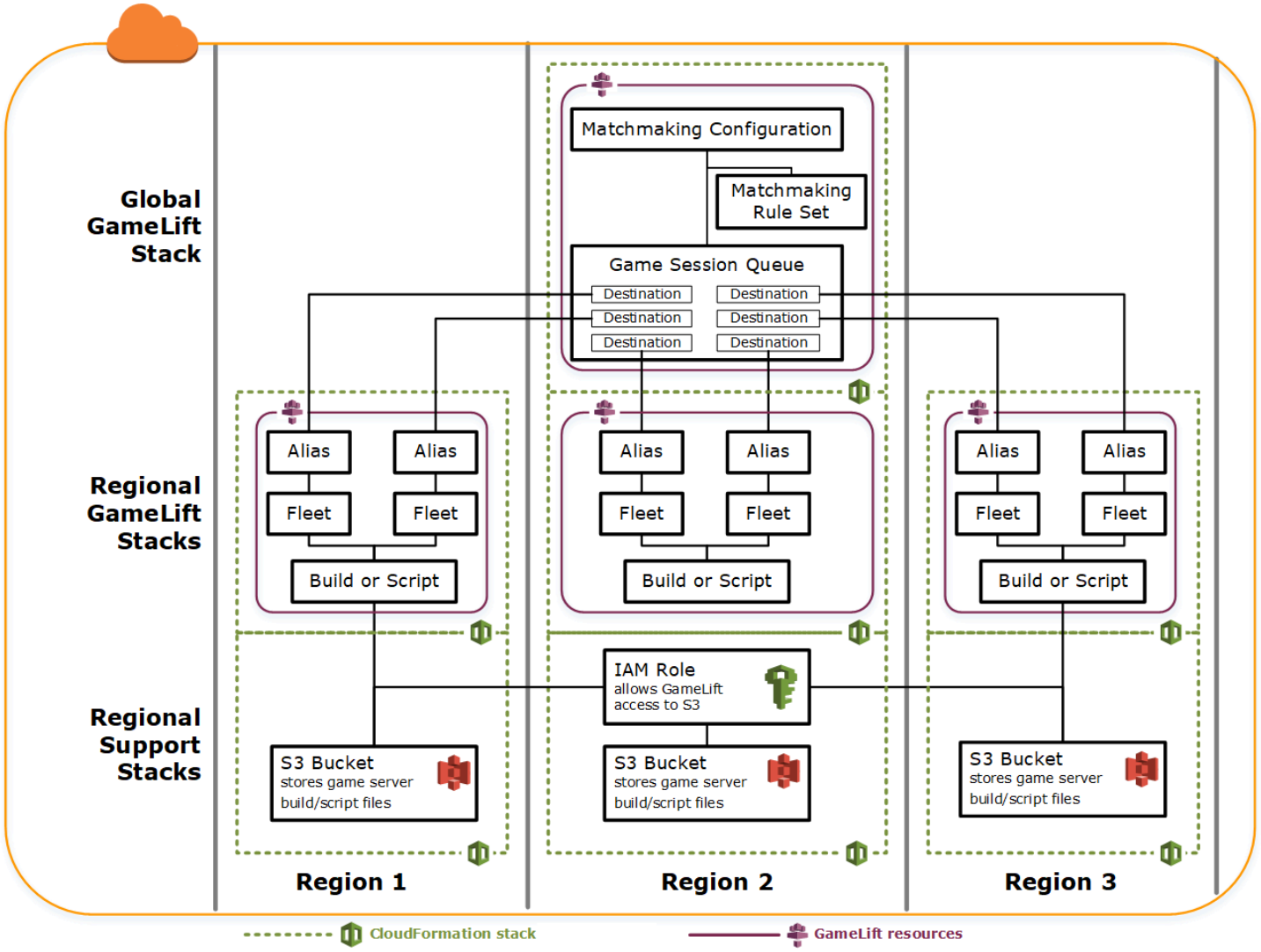
リージョンが複数の場合のスタック

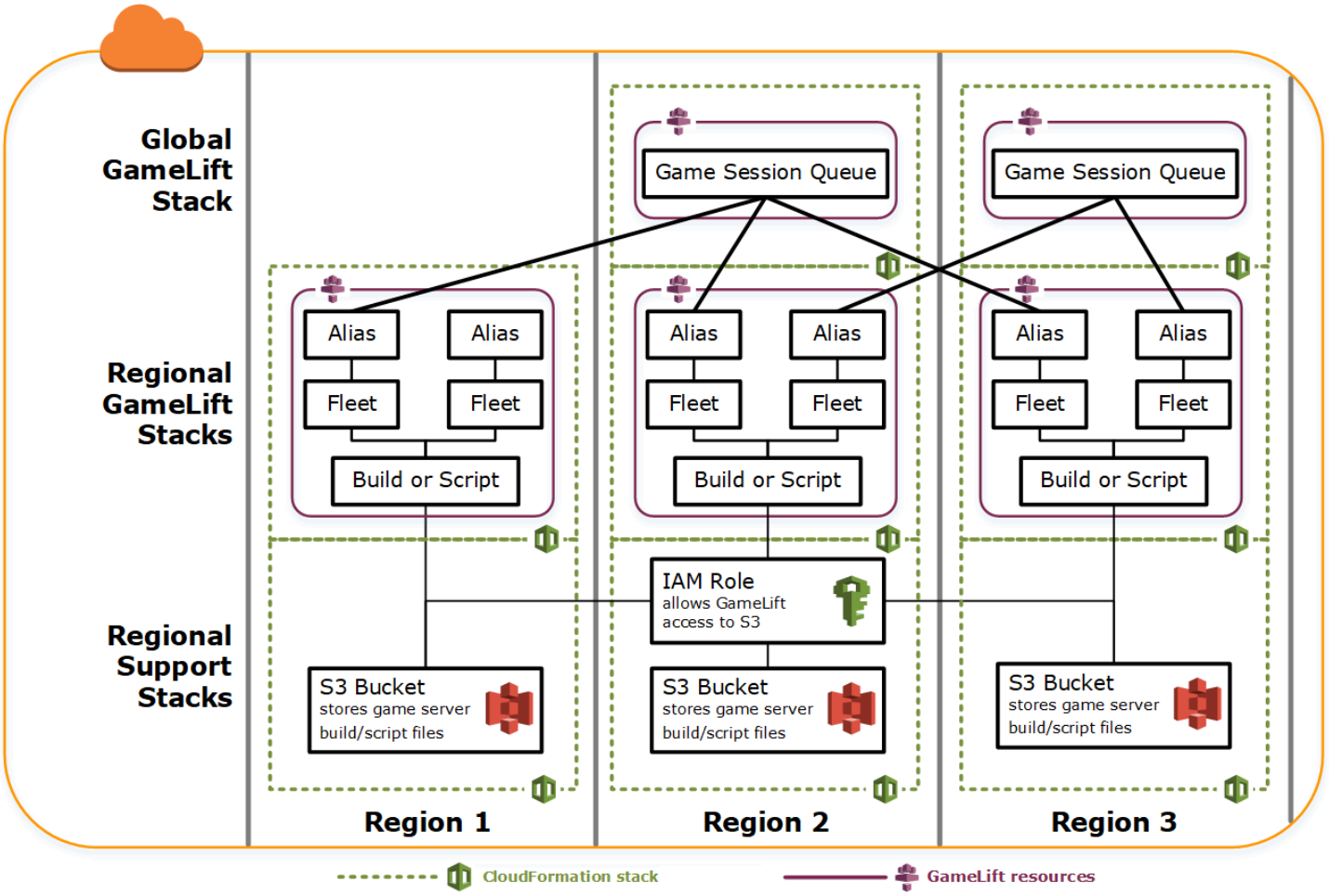
ゲームを複数のリージョンにデプロイする場合、リソースがリージョン間でどのようにやり取りするかに留意してください。Amazon GameLift フリートなどの一部のリソースは同じリージョン内の他のリソースのみをリファレンスできます。Amazon GameLift キューなどの他のリソースはリージョ

ンに依存しません。複数のリージョンで Amazon GameLift リソースを管理するには、以下の構造をお勧めします。

- リージョン別サポートスタック - これらのスタックには、Amazon GameLift リソースが依存するリソースが含まれます。このスタックには、カスタムゲームサーバーまたは Realtime スクリプトファイルを保存する S3 バケットが含まれる必要があります。DynamoDB テーブル、Amazon Redshift クラスター、Lambda 関数など、ゲームで使用される他の AWS リソースも含まれる場合があります。これらのリソースの多くはリージョン固有であるため、リージョンごとに作成する必要があります。Amazon GameLift には、これらのサポートリソースへのアクセスを許可する IAM ロールも必要です。IAM ロールはリージョンに依存しないため、必要なロールリソースは 1 つのみであり、任意のリージョンに配置され、他のすべてのサポートスタックで参照されます。
- リージョン別 Amazon GameLift スタック - このスタックには、ゲームがデプロイされる各リージョンに存在する必要のある Amazon GameLift リソース (ビルドまたはスクリプト、フリートのセット、エイリアスなど) が含まれます。AWS CloudFormation は、S3 バケットの場所に保存されているファイルを使用してビルドまたはスクリプトリソースを作成し、1 つまたは複数のフリートリソースにデプロイします。フリートごとに対応するエイリアスが必要です。ゲームセッションキューはフリートエイリアスの一部またはすべてを参照します。このタイプのスタックを記述する 1 つのテンプレートを維持し、そのテンプレートを使用してリージョンごとに同一のリソースセットを作成できます。
- グローバル Amazon GameLift スタック – このスタックには、ゲームセッションキューとマッチメイキングリソースが含まれます。これらのリソースは任意のリージョンに配置でき、通常は同じリージョンに配置されます。キューは、任意のリージョンにあるフリートまたはエイリアスを参照できます。別のリージョンに追加のキューを配置するには、追加のグローバルスタックを作成します。

以下の図では、複数の AWS リージョンにリソースをデプロイするマルチスタック構造を示しています。最初の図では、1 つのゲームセッションキューを使用した構造を示しています。2 番目の図では、複数のゲームセッションキューを使用した構造を示しています。


Global GameLift Stack
Regional GameLift Stacks
Regional Support Stacks
Matchmaking Configuration
Matchmaking Rule Set
Game Session Queue
Destination
Alias
Fleet
Build or Script
IAM Role
allows GameLift access to S3
S3 Bucket
stores game server build/script files
Region 1
Region 2
Region 3
CloudFormation stack
GameLift resources



ビルドの更新

Amazon GameLift ビルドはイミュータブルです。ビルドとフリートの関係も同様です。その結果、ゲームビルドファイルの新しいセットを使用するようにホスティングリソースを更新する場合、以下の手順を実行する必要があります。

- 新しいファイルのセットを使用して新しいビルドを作成する (置き換え)。
- 新しいゲームビルドをデプロイするための新しいフリートのセットを作成する (置き換え)。
- 新しいフリートを参照するようにエイリアスをリダイレクトする (中断することなく更新)。

詳細については、AWS CloudFormation ユーザーガイドの「スタックリソースの更新動作」を参照してください。

ビルドの更新を自動的にデプロイする

関連するビルド、フリート、エイリアスリソースを含むスタックを更新すると、デフォルトの AWS CloudFormation の動作では、以下のステップが順番に自動的に実行されます。この更新をトリガーするには、まず新しいビルドファイルを新しい S3 の場所にアップロードします。次に、新しい S3 の場所を参照するように AWS CloudFormation ビルドテンプレートを変更します。新しい S3 の場所でスタックを更新すると、以下の AWS CloudFormation シーケンスがトリガーされます。

1. S3 から新しいファイルを取得し、それらのファイルを検証して、新しい Amazon GameLift ビルドを作成します。
2. フリートテンプレートでビルドの参照を更新すると、新しいフリートの作成がトリガーされます。
3. 新しいフリートがアクティブになったら、エイリアス内のフリートの参照を更新します。これにより、エイリアスの更新がトリガーされて、新しいフリートがターゲットになります。
4. 古いフリートを削除します。
5. 古いビルドを削除します。

ゲームセッションキューがフリートエイリアスを使用している場合、エイリアスが更新されるとすぐに、プレイヤーのトラフィックは新しいフリートに自動的に切り替えられます。ゲームセッションが終了すると、古いフリートは段階的にプレイヤーからドレインされます。Auto Scaling は、プレイヤーのトラフィックの変動に応じてフリートの各セットに対してインスタンスを追加および削除するタスクを処理します。または、切り替え用にすばやく増やせることを前提に、最初に必要になるインスタンスの数を指定し、後で Auto Scaling を有効にすることもできます。

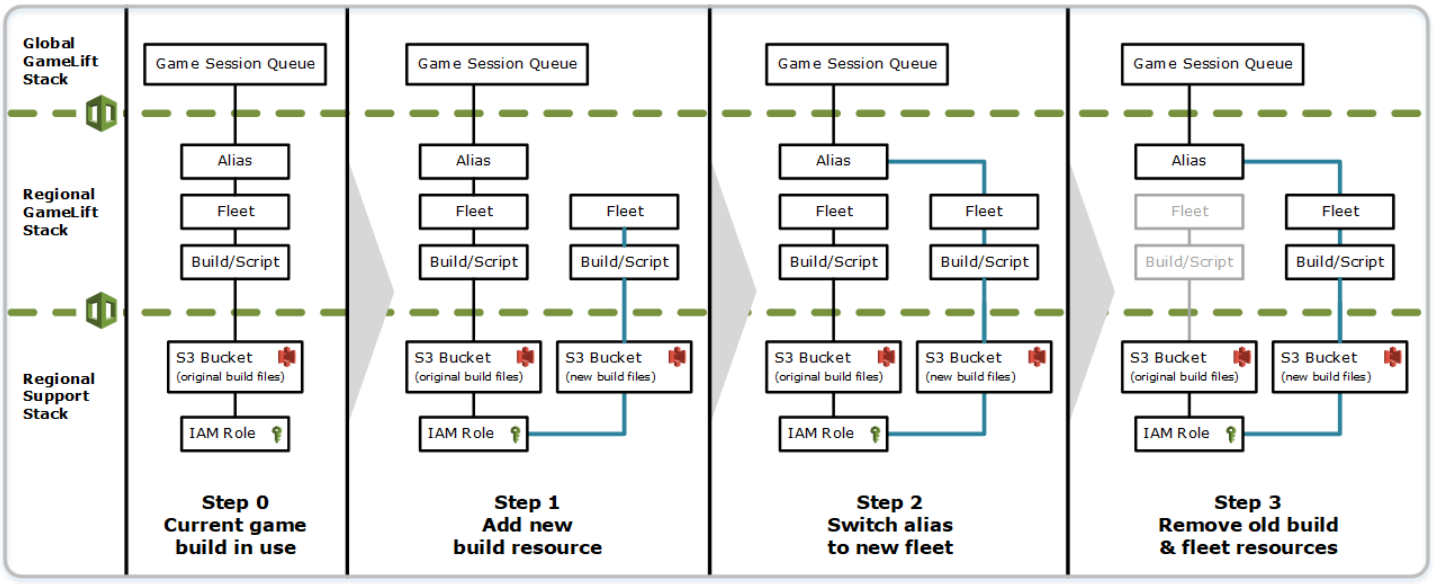
AWS CloudFormation で、リソースを削除する代わりに保持することもできます。詳細については、AWS CloudFormationAPI リファレンスの「RetainResources」を参照してください。

ビルドの更新を手動でデプロイする

プレイヤーが新しいフリートをいつ稼働させるかをさらに細かく制御する場合は、いくつかのオプションがあります。Amazon GameLift コンソールまたは CLI を使用して、エイリアスを手動で管理することを選択できます。または、ビルドテンプレートを更新してビルドとフリートを置き換える代わりに、2 番目のビルドとフリートのセットの定義をテンプレートに追加することを選択できます。テンプレートを更新すると、AWS CloudFormation によって 2 番目のビルドリソースと対応するフリートが作成されます。既存のリソースは置き換えられないため、それらのリソースは削除されず、エイリアスは元のフリートを参照したままになります。

このアプローチの主な利点は、柔軟性が得られることです。ビルドの新しいバージョン用に個別のリソースを作成し、新しいリソースをテストして、新しいフリートをプレイヤーに公開するタイミングを制御できます。考えられる欠点は、短期間にリージョンごとに 2 倍のリソースが必要になることです。

次の図は、このプロセスを示したものです。



ロールバックの仕組み

リソースの更新を実行するときに、いずれかのステップが正常に完了しない場合、AWS CloudFormation によってロールバックが自動的に開始されます。このプロセスでは、各ステップを順番に反転させて、新しく作成されたリソースを削除します。

ロールバックを手動でトリガーする必要がある場合は、ビルドテンプレートの S3 の場所キーを元の場所に戻し、スタックを更新します。Amazon GameLift の新しいビルドとフリートが作成され、フリートがアクティブになると、エイリアスが新しいフリートに切り替わります。エイリアスを個別に管理している場合は、新しいフリートを参照するようにエイリアスを切り替える必要があります。

失敗またはスタックしたロールバックの処理方法の詳細については、AWS CloudFormation ユーザーガイドの「更新のロールバックを続ける」を参照してください。

Amazon GameLift の VPC ピアリング

このトピックでは、Amazon GameLift がホストするゲームサーバーと他の Amazon GameLift 以外のリソースとの VPC ピアリング接続をセットアップする方法に関するガイダンスを提供しま

す。Amazon Virtual Private Cloud(VPC) ピアリングを使用して、ゲームサーバーがウェブサービスやリポジトリなどの他の AWS リソースと直接かつプライベートに通信を有効にします。AWS で実行されてお客様がアクセス可能な AWS アカウントでマネージドされているすべてのリソースと、VPCピアリングを確立できます。

> **Note**
>
> VPC ピア接続は高度な機能です。ゲームサーバーが他の AWS リソースと直接かつプライベートに通信できるようにするための推奨オプションについては、「フリートの他の AWS リソースと通信する 」を参照してください。

Amazon VPC と VPC ピアリングにすでに精通している場合は、Amazon GameLift ゲームサーバーとのピア接続の設定は多少異なっているのがわかります。ゲームサーバーを含む VPC にアクセスすることはできません (これは Amazon GameLift サービスによってコントロールされます)。そのため、VPC ピアリングを直接リクエストすることはできません。代わりに、Amazon GameLift サービスへのピア接続リクエストを受け入れるために、最初に Amazon GameLift 以外のリソースで VPC を事前承認します。次に、Amazon GameLift をトリガーし、先ほど許可した VPC ピアリングをリクエストします。Amazon GameLift はピア接続の作成、ルートテーブルの設定、接続の設定のタスクを処理します。

既存のフリート用に VPC ピア接続を設定するには

1. AWS アカウント ID と認証情報を取得します。

   以下のAWS アカウントには、ID とサインインの認証情報が必要です。AWS Management Consoleにサインインしてアカウント設定を表示することで、AWS アカウント ID を確認できます。認証情報を取得するには、IAM コンソールに移動します。

   - Amazon GameLift ゲームサーバーの管理に使用する AWS アカウント。
   - Amazon GameLift 以外のリソースの管理に使用する AWS アカウント。

   Amazon GameLift とAmazon GameLift 以外のリソースで同じアカウントを使用している場合は、そのアカウントのみの ID と認証情報が必要です。

2. 各 VPC の識別子を取得します。

   ピア接続される 2 つの VPC について以下の情報を取得します。

- Amazon GameLift ゲームサーバー用の VPC – Amazon GameLift の フリート ID。ゲームサーバーは、EC2 インスタンスのフリート上の Amazon GameLift にデプロイされます。フリートは、Amazon GameLift サービスで管理される専用の VPC に自動的に配置されます。VPC への直接アクセスは許可されないため、VPC はフリート ID によって識別されます。
- Amazon GameLift 以外の AWS リソースの VPC – AWS で実行され、アクセス可能な AWS アカウントで管理されているすべてのリソースと VPC ピアリングを確立できます。これらのリソース用の VPC をまだ作成していない場合は、「Amazon VPC の開始方法」を参照してください。VPC を作成したら、Amazon VPC の AWS Management Console にサインインして VPC を一覧表示することで VPC ID を確認できます。

> **Note**
>
> ピア接続を設定するときは、両方の VPC が同じリージョンに存在する必要があります。Amazon GameLift フリート ゲームサーバー用の VPC はフリートと同じリージョンにあります。

3. VPC ピア接続を認可します。

   このステップでは、Amazon GameLift への今後のリクエストを事前認可して、ゲームサーバーを含む VPC を、GameLift 以外のリソース用の VPC にピア接続します。このアクションにより、VPC のセキュリティグループが更新されます。

   VPC ピアリングを承認するには、Amazon GameLift サービス API CreateVpcPeeringAuthorization() を呼び出すか、AWS CLI コマンド `create-vpc-peering-authorization` を使用します。Amazon GameLift 以外のリソースを管理するアカウントを使用してこの呼び出しを行います。以下の情報を識別します。

   - ピア VPC ID – Amazon GameLift 以外のリソースのある VPC の ID。
   - Amazon GameLift AWS アカウント ID – これは Amazon GameLift フリートの管理に使用するアカウントです。

   VPC ピア接続を承認した後は、取り消されない限り、24 時間有効です。以下の操作を使用して、VPC ピア接続承認を管理できます。

   - [DescribeVpcPeeringAuthorizations](DescribeVpcピアリング認可)(AWS CLI `describe-vpc-peering-authorizations`)

- [DescribeVpcPeeringAuthorizations](DescribeVpcピアリング認可)(AWS CLI `delete-vpc-peering-authorization`)

4. ピア接続をリクエストします。

   有効な認可があれば、Amazon GameLift にピア接続の確立をリクエストできます。

   VPCピアリングをリクエストするには、Amazon GameLift サービス API CreateVpcPeeringConnection() を呼び出すか、AWS CLI コマンド `create-vpc-peering-connection` を使用します。Amazon GameLift ゲームサーバーを管理するアカウントを使用してこの呼び出しを行います。以下の情報を使用して、ピア接続する 2 つの VPC を指定します。

   - ピア VPC ID と AWS アカウント ID – これは Amazon GameLift 以外のリソースの VPC およびそれらの管理に使用するアカウントです。VPC ID は有効なピア接続認可の ID と一致する必要があります。
   - フリート ID – これは Amazon GameLift ゲームサーバーの VPC を識別します。

5. ピア接続のステータスを追跡します。

   VPC ピア接続のリクエストは、非同期操作です。ピア接続リクエストのステータスを追跡し、成功または失敗のケースを処理するには、次のいずれかのオプションを使用します。

   - `DescribeVpcPeeringConnections()` を使用して継続的にポーリングします。この操作では、リクエストのステータスを含む VPC ピア接続レコードが取得されます。ピア接続が正常に作成された場合、接続レコードには、VPC に割り当てられたプライベート IP アドレスの CIDR ブロックも含まれます。
   - 成功イベントと失敗イベントを含む DescribeFleetEvents() の VPC ピア接続に関連付けられたフリートイベントを処理します。

ピア接続が確立されると、以下のオペレーションを使用してその接続を管理できます。

- [DescribeVpcPeeringAuthorizations](DescribeVpcピアリング認可) (AWS CLI `describe-vpc-peering-connections`)。
- [DeleteVpcPeeringAuthorizations] (DeleteVpcピアリング認可)(AWS CLI `delete-vpc-peering-connection`)。

新しいフリートとの VPC ピア接続を設定するには

新しい Amazon GameLift フリートを作成し、VPC ピアリング接続を同時にリクエストできます。

1. AWS アカウント ID と認証情報を取得します。

   次の 2 つの AWS アカウントには、ID とサインインの認証情報が必要です。AWS Management Consoleにサインインしてアカウント設定を表示することで、AWS アカウント ID を確認できます。認証情報を取得するには、IAM コンソールに移動します。

   - Amazon GameLift ゲームサーバーの管理に使用する AWS アカウント。
   - Amazon GameLift 以外のリソースの管理に使用する AWS アカウント。

   Amazon GameLift とAmazon GameLift 以外のリソースで同じアカウントを使用している場合は、そのアカウントのみの ID と認証情報が必要です。

2. Amazon GameLift 以外の AWS リソースの VPC ID を取得します。

   これらのリソースの VPC をまだ作成していない場合は、ここで作成してください (「Amazon VPC の開始方法」を参照)。新しいフリートを作成する予定の同じリージョンに新しい VPC を作成してください。Amazon GameLift 以外のリソースが、Amazon GameLift で使用するものとは異なる AWS アカウントまたはユーザー/グループで管理されている場合、次のステップで承認をリクエストするときに、これらのアカウント認証情報を使用する必要があります。

   VPC を作成したら、VPC を表示して Amazon VPC コンソールで VPC ID を見つけることができます。

3. Amazon GameLift 以外のリソースとの VPC ピアリングを承認します。

   Amazon GameLift によって新しいフリートおよび対応する VPC が作成されるとき、Amazon GameLift 以外のリソースの VPC にピア接続するためのリクエストも送信されます。そのリクエストを事前承認する必要があります。このステップにより、VPC のセキュリティグループが更新されます。

   Amazon GameLift 以外のリソースを管理するアカウント認証情報を使用して、Amazon GameLift サービス API CreateVpcPeeringAuthorization() を呼び出すか、AWS CLI コマンド `create-vpc-peering-authorization` を使用します。以下の情報を識別します。

   - ピア VPC ID – Amazon GameLift 以外のリソース用の VPC の ID。
   - Amazon GameLift AWS アカウント ID – Amazon GameLift フリートを管理するために使用するアカウントの ID。

VPC ピア接続を承認した後は、取り消されない限り、24 時間有効です。以下の操作を使用して、VPC ピア接続承認を管理できます。

- [DescribeVpcPeeringAuthorizations](DescribeVpcピアリング認可)(AWS CLI `describe-vpc-peering-authorizations`)
- [DeleteVpcPeeringAuthorizations] (DeleteVpcピアリング認可)(AWS CLI `delete-vpc-peering-authorization`)。

4. [creating a new fleet using the AWS CLI](CLIを使用して新しいフリートを作成する)手順に従います。以下の追加のパラメータを含めます。

   - peer-vpc-aws-account-id – Amazon GameLift 以外のリソースで VPC を管理するために使用するアカウントの ID。
   - peer-vpc-id – GameLift 以外のアカウント用の VPC のID。

VPC ピア接続パラメータを設定した create-fleet の呼び出しが成功すると、新しいフリートと新しい VPC ピア接続リクエストの両方が生成されます。フリートのステータスは New に設定され、フリートのアクティブ化プロセスが開始されます。ピア接続リクエストのステータスは initiating-request に設定されます。describe-vpc-peering-connections を呼び出して、ピアリングリクエストの成否を追跡できます。

新しいフリートと VPC ピア接続の両方をリクエストすると、両方のアクションが成功するか失敗します。フリートの作成プロセス中にエラーが発生すると、VPC ピア接続は確立されません。同様に、VPC ピア接続が何らかの理由で失敗した場合も、新しいフリートのステータスは [アクティブ化中] から [アクティブ] に移行しません。

> **Note**
>
> 新しい VPC ピア接続は、フリートがアクティブになる準備が整うまで完了しません。これは、その接続が利用可能になっておらず、ゲームサーバービルドのインストールプロセス中に使用できないことを意味します。

以下の例では、新しいフリートと、事前に確立された VPC と新しいフリートの VPC 間のピア接続の両方を作成します。事前設定された VPC は、Amazon GameLift 以外の AWS アカウント ID と VPC ID の組み合わせによって一意に識別されます。

```
$ AWS gamelift create-fleet
    --name "My_Fleet_1"
    --description "The sample test fleet"
    --ec2-instance-type "c5.large"
    --fleet-type "ON_DEMAND"
    --build-id "build-1111aaaa-22bb-33cc-44dd-5555eeee66ff"
    --runtime-configuration "GameSessionActivationTimeoutSeconds=300,
                             MaxConcurrentGameSessionActivations=2,
                             ServerProcesses=[{LaunchPath=C:\game\Bin64.dedicated
\MultiplayerSampleProjectLauncher_Server.exe,
                                               Parameters=+sv_port 33435 +start_lobby,
                                               ConcurrentExecutions=10}]"
    --new-game-session-protection-policy "FullProtection"
    --resource-creation-limit-policy "NewGameSessionsPerCreator=3,
                                      PolicyPeriodInMinutes=15"
    --ec2-inbound-permissions
 "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"

 "FromPort=33235,ToPort=33235,IpRange=0.0.0.0/0,Protocol=UDP"
    --metric-groups  "EMEAfleets"
    --peer-vpc-aws-account-id "111122223333"
    --peer-vpc-id "vpc-a11a11a"
```

コピー可能バージョン:

```
AWS gamelift create-fleet --name "My_Fleet_1" --description "The
 sample test fleet" --fleet-type "ON_DEMAND" --metric-groups
 "EMEAfleets" --build-id "build-1111aaaa-22bb-33cc-44dd-5555eeee66ff"
 --ec2-instance-type "c5.large" --runtime-configuration
 "GameSessionActivationTimeoutSeconds=300,MaxConcurrentGameSessionActivations=2,ServerProcesses
\game\Bin64.dedicated\MultiplayerSampleProjectLauncher_Server.exe,Parameters=
+sv_port 33435 +start_lobby,ConcurrentExecutions=10}]" --new-game-session-
protection-policy "FullProtection" --resource-creation-limit-policy
 "NewGameSessionsPerCreator=3,PolicyPeriodInMinutes=15" --ec2-inbound-
permissions "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"
 "FromPort=33235,ToPort=33235,IpRange=0.0.0.0/0,Protocol=UDP" --peer-vpc-aws-account-id
 "111122223333" --peer-vpc-id "vpc-a11a11a"
```

VPC ピア接続に関する問題のトラブルシューティング

Amazon GameLift ゲームサーバーの VPC ピアリング接続の確立に問題がある場合は、以下の一般的な根本原因を検討してください。

- リクエストされた接続の認可が見つからなかった。
  - Amazon GameLift 以外の VPC の VPC 承認のステータスをチェックします。存在しないか、有効期限が切れている可能性があります。
  - ピア接続しようとしている 2 つの VPC のリージョンを確認します。それらの VPC が同じリージョンにない場合は、ピア接続できません。
- 2 つの VPC の CIDR ブロック (「無効な VPC ピアリング接続設定」を参照) が重複している。ピア接続された VPC に割り当てられた IPv4 CIDR ブロックは、重複することはできません。Amazon GameLift フリートの VPC の CIDR ブロックは自動的に割り当てられ、変更できません。そのため、Amazon GameLift 以外のリソース用の VPC の CIDR ブロックを変更する必要があります。この問題を解決するには。
  - `DescribeVpcPeeringConnections()` を呼び出して、Amazon GameLift フリートのこの CIDR ブロックを検索します。
  - [Amazon VPC コンソール] に移動し、Amazon GameLift 以外のリソース用の VPC を見つけて、それらが重複しないように CIDR ブロックを変更します。
- 新しいフリートがアクティブにならなかった (新しいフリートとの VPC ピア接続のリクエスト時)。新しいフリートが Active (アクティブ) ステータスに移行しなかった場合、ピア接続する VPC がないため、ピア接続は成功しません。

コンソールでのゲームデータの表示

マネージド Amazon GameLift サービスは、プレイヤーの行動とパフォーマンスを把握できるように、アクティブなゲームのデータを継続的に収集します。Amazon GameLift コンソールを使用すると、構築、フリート、ゲームセッション、プレイヤーセッションのこの情報を表示、管理、および分析できます。

トピック

現在の Amazon GameLift ステータスを表示する

Amazon GameLift ダッシュボードには、次のビューが表示されます。

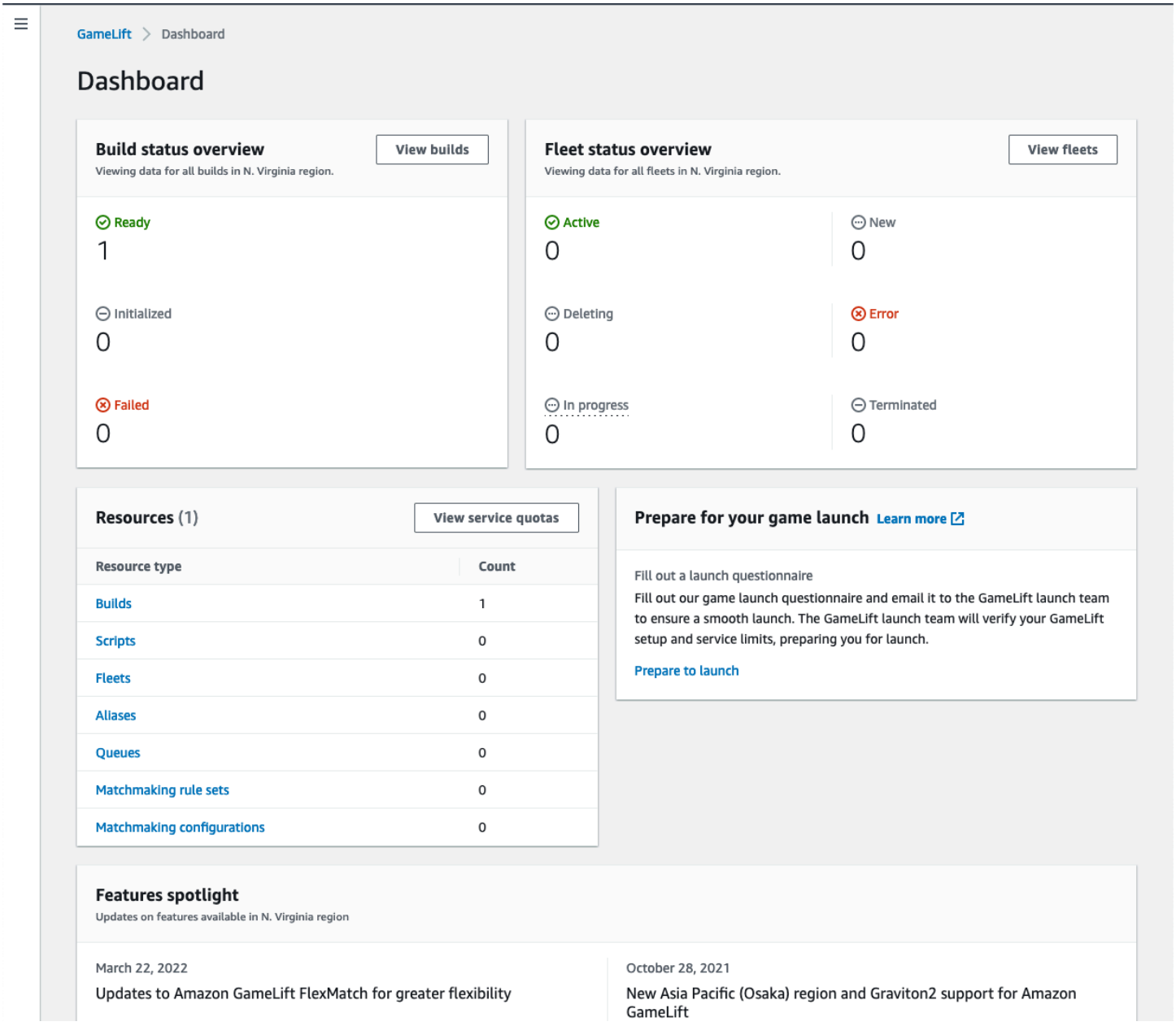
- [準備完了]、[初期化済み]]、[失敗] ステータスのビルドの数。現在のリージョンのビルドの詳細については、[ビルドを表示する] を選択してください。
- すべてのステータスのフリート数。現在のリージョンのフリートの詳細については、[フリートを表示する] を選択してください。
- 現在のリソース。
- 新しい特徴とサービスのお知らせ

Amazon GameLift ダッシュボードを開くには

- Amazon GameLift コンソール のナビゲーションペインで、ダッシュボード を選択します。

ダッシュボードからは、次の操作を行うことができます。

- [ローンチに備える] を選択し、対応するローンチアンケートに記入して、ゲームのローンチを準備します。
- [サービスクォータを表示] を選択し、ローンチに備えて、またはローンチに対応してサービスクォータの増加をリクエストします。
- [注目の機能] 内のリンクを選択すると、ブログ投稿や新機能に関する詳細情報を表示できます。



ビルドを表示する

Amazon GameLift コンソールの [ビルド] ページでは、Amazon GameLift にアップロードしたすべてのゲームサーバービルドに関する情報を表示し、管理できます。ナビゲーションペインで [ホスティング]、[ビルド] を選択します。

[ビルド] ページには、ビルドごとに以下の情報が表示されます。

> **Note**
>
> [ビルド] ページには、現在の AWS リージョンのビルドのみが表示されます。

- [名前] – アップロードしたビルドに関連付けられた名前。
- [ステータス] – ビルドのステータス。3 つのステータスメッセージのうちの 1 つを表示します。
  - [初期化済み] – アップロードはまだ開始されていないか、まだ進行中です。
  - [準備完了] – ビルドはフリートの作成の準備が整いました。
  - [失敗] – Amazon GameLift がバイナリを受信する前にビルドがタイムアウトしました。
- [作成日] – ビルドをAmazon GameLift にアップロードした日時。
- [ビルド ID]– アップロード時にビルドに割り当てられる一意の ID。
- [バージョン] – アップロードしたビルドに関連付けられたバージョンラベル付け。
- [オペレーティングシステム] – ビルドを実行するオペレーションシステム。ビルド OS により、Amazon GameLift はフリートのインスタンスにどのオペレーティングシステムをインストールするかを決定します。
- [サイズ] – Amazon GameLiftにアップロードされたビルドファイルのサイズ、単位はメガバイト (MB)。
- [フリート] – このビルドでデプロイされているフリートの数。

このページから、以下のいずれも行うことができます。

- ビルドの詳細を表示します。ビルド名を選択すると、ビルドの詳細ページが開きます。
- ビルドから新しいフリートを作成します。ビルドを選択し、[フリートを作成] を選択します。
- ビルドのリストのフィルタリングとソートをします。テーブルの先頭でコントロールを使用します。

- ビルドを削除します。ビルドを選択して、[削除] を選択します。

ビルドの詳細

[ビルド] ページで、ビルドの名前を選択し、詳細ページを開きます。詳細ページの [概要] セクションには、[ビルド] ページと同じビルドの概要情報が表示されます。[フリート] セクションには、[フリート] ページと同じ概要情報を含む、ビルドで作成されたフリートのリストが表示されます。

スクリプトを表示する

Amazon GameLift コンソールの [スクリプト] ページでは、Amazon GameLift にアップロードしたすべてのリアルタイムサーバースクリプトに関する情報を表示し、管理できます。ナビゲーションペインで [ホスティング]、[スクリプト] を選択します。

[スクリプト] ページには、スクリプトごとに以下の情報が表示されます。

> **Note**
> [スクリプト] ページには、現在の AWS リージョンのスクリプトのみが表示されます。

- [名前] – アップロードしたスクリプトに関連付けられた名前。
- [ID] – アップロード時にスクリプトに割り当てられる一意の ID。
- [バージョン] – アップロードしたスクリプトに関連付けられたバージョンラベル。
- [サイズ] – Amazon GameLiftにアップロードされたスクリプトファイルのサイズ、単位はメガバイト(MB)。
- [作成時刻] – スクリプトをAmazon GameLift にアップロードした日時。
- [フリート] – スクリプトを使用してデプロイされたフリートの数。

このページから、以下のいずれも行うことができます。

- スクリプトの詳細を表示します。ビルド名を選択すると、ビルドの詳細ページが開きます。
- スクリプトから新しいフリートを作成します。スクリプトを選択し、[フリートを作成] を選択します。
- スクリプトのリストのフィルタリングとソートをします。テーブルの先頭でコントロールを使用します。

- スクリプトを削除します。スクリプトを選択して、[削除] を選択します。

スクリプトの詳細

[スクリプト] ページで、スクリプトの名前を選択し、詳細ページを開きます。詳細ページの [概要] セクションには、[スクリプト] ページと同じビルドの概要情報が表示されます。[フリート] セクションには、[フリート] ページと同じ概要情報を含む、スクリプトで作成されたフリートのリストが表示されます。

フリートを表示する

AWSアカウントで Amazon GameLift のゲームをホストするために作成されたすべてのフリートに関する情報を表示できます。リストには、現在のリージョンで作成されたフリートが表示されます。[フリート] ページでは、新しいフリートを作成したり、フリートの詳細情報を表示したりできます。フリートの[詳細ページ] には、使用情報、メトリクス、ゲームセッションデータ、およびプレイヤーセッションデータが含まれます。フリートレコードを編集したり、フリートを削除したりすることもできます。

[フリート] ページを表示するには、ナビゲーションペインから [フリート] を選択します。

[フリート] ページには、以下の概要情報がデフォルトで表示されます。[設定] (歯車) ボタンを選択すると、表示される情報をカスタマイズできます。

- [フリート名] – フリートに付けられたわかりやすい名前です。
- [Status] (ステータス) - フリートのステータスとして、[New] (新規)、[Downloading] (ダウンロード中)、[Building] (構築)、[Active] (アクティブ) のいずれかを示します。
- [作成日] – フリートが作成された日時です。
- [コンピューティングタイプ] – ゲームのホストに使用されるコンピューティングのタイプ。フリートは Managed EC2 フリートでも Anywhere フリートでもかまいません。
- [インスタンスタイプ] – Amazon EC2 のインスタンスタイプです。タイプによってフリートのインスタンスの コンピューティングキャパシティが決まります。
- [Active instances] (アクティブなインスタンス) - フリートで使用中の EC2 インスタンスの数です。
- [目的のインスタンス] – アクティブに維持する EC2 インスタンスの数です。
- [ゲームセッション] – フリートで実行されているアクティブなゲームセッションの数です。データには 5 分の遅延が適用されています。

フリートの詳細の表示

[フリート] 詳細ページには、ダッシュボードから、または [フリート] ページでフリート名をクリックしてアクセスします。

フリート詳細ページでは、次のアクションを実行できます。

- フリートの属性、ポート設定、ランタイム 設定を更新します。
- フリートのロケーションを追加または削除します。
- フリート容量設定を変更します。
- ターゲット追跡のオートスケーリングを設定または変更します。
- フリートを削除します。

詳細

フリート設定

- [Fleet ID] (フリートID) – フリートに割り当てられた一意の識別子です。
- [名前] – フリートの名前。
- [ARN] – このフリートに割り当てられた識別子です。フリートの ARN は、それを Amazon GameLift リソースとして識別し、リージョンと AWS アカウントを指定して、それが一意の ID であることを確認します。
- [説明] – フリートに関する簡単な説明です。
- [Status] (ステータス) – フリートの現在のステータスとして、[New] (新規)、[Downloading] (ダウンロード中)、[Building] (構築中) および[Active] (アクティブ) のいずれかを示します。
- [作成時刻] – フリートが作成された日時です。
- [終了時間] – フリートが終了した日時です。フリートがまだアクティブな場合は空白です。
- [Fleet type] (フリートタイプ) – フリートでオンデマンド インスタンスを使用するかスポットインスタンスを使用するかを指定します。
- [EC2 type] (EC2タイプ) – フリート作成時に選択されたAmazon EC2の[instance type] (インスタンスタイプ) です。
- [インスタンスロールの ARN] – 他の AWS リソースへのアクセスを管理する AWS IAM ロール、フリートの作成時に指定した場合。

- [TLS 証明書] – フリートが、ゲームサーバーの認証とすべてのクライアント/サーバー通信の暗号化に TLS 証明書を使用するのに有効になっているか、無効になっているか。
- [メトリクスグループ] – このグループは、複数のフリートのメトリクスを集計するために使用されます。
- [ゲームスケーリング保護ポリシー] – フリートの [ゲームセッション保護] の現在の設定。
- [プレイヤーあたりの最大ゲームセッション数] – [ポリシー期間] 中にプレイヤーが作成できるセッションの最大数。
- [ポリシー期間] – プレイヤーが作成したセッション数をリセットするまでの待ち時間。

ビルドの詳細

[ビルドの詳細] セクションには、フリートでホストされているビルドが表示されます。ビルドの名前を選択すると、ビルドの全詳細ページが表示されます。

ランタイム設定

[ランタイム設定] セクションには、各インスタンスで起動するサーバープロセスが表示されます。これには、ゲームサーバー実行可能ファイルのパスとオプション起動パラメーターが含まれます。

ゲームセッションアクティベーション

[ゲームセッションアクティベーション] セクションには、同時に起動するサーバープロセスの数と、プロセスがアクティブ化されてから終了するまでの待ち時間が表示されます。

EC2 ポート設定

[ポート] タブには、IP アドレスやポート設定範囲など、フリートの接続アクセス許可が表示されます。

メトリクス

[メトリクス] タブには、時間の経過に伴うフリートメトリクスのグラフィカル表現が表示されます。Amazon GameLift のメトリクスの詳細については、「Amazon CloudWatch で Amazon GameLift をモニタリングする」を参照してください。

のイベント

[Events] タブには、イベントコード、メッセージ、タイムスタンプなど、フリートで発生したすべてのイベントのログが表示されます。Amazon GameLift API リファレンスの[Event] (イベント) を参照してください。

[Scaling] (スケーリング)

[スケーリング] タブには、現在のステータスや時間の経過に伴うキャパシティの変更など、フリートのキャパシティに関連する情報が含まれます。容量制限の更新と Auto Scaling の管理を行うツールも用意されています。

キャパシティのスケーリング

現在のフリートキャパシティの設定を、フリートの口ケーション別に表示します。制限とキャパシティの変更の詳細については、「Amazon GameLift ホスティングキャパシティのスケーリング」を参照してください。

- [AWS ロケーション] – フリートインスタンスがデプロイされる場所の名前。
- [Status] (ステータス) – フリートロケーションのホスティングステータス。ロケーションステータスはゲームをホストできるようにするために、ACTIVE である必要があります。
- [最小サイズ] – の場所にデプロイする必要があるインスタンスの最小数。
- [目的のインスタンス] – ロケーションを維持するアクティブインスタンスのターゲット数。アクティブなインスタンスと目的のインスタンスが同じでない場合、アクティブインスタンスが目的のインスタンスと等しくなるまで、必要に応じてインスタンスをスタートまたはシャットダウンするスケーリングイベントが開始されます。
- [最大サイズ] – そのロケーションにデプロイできるインスタンスの最大数。
- [利用可能] – インスタンスのサービス制限から、使用中のインスタンス数を引いた数です。この値は、ロケーションに追加できるインスタンスの最大数を示します。

[Auto Scaling policies] (オートスケーリングポリシー)

このセクションでは、フリートに適用される自動スケーリングポリシーについて説明します。ターゲットベースのポリシーを設定または更新できます。フリートのルールベースのポリシー。このポリシーは、AWS SDK または CLI がここに表示されます。(スケーリングの詳細については、「Amazon GameLift を使ったフリートキャパシティの自動スケーリング」を参照してください。)

[Scaling history] (スケーリング履歴)

時間の経過に伴うキャパシティの変化をグラフに表示します。

Locations

[Locations] (ロケーション) タブには、フリートインスタンスがデプロイされているすべてのロケーションが一覧表示されます。ロケーションには、フリートのホーム リージョンと、追加されたリモートロケーションが含まれます。このタブでは、直接ロケーションを追加または削除できます。

- [Location] (ロケーション) — フリートインスタンスがデプロイされる場所の名前。
- [Status] (ステータス) – フリートロケーションのホスティングステータス。ロケーションステータスは、ロケーション内の最初のインスタンスをアクティブ化するプロセスを追跡します。ロケーションステータスはゲームをホストできるようにするために、ACTIVE である必要があります。
- アクティブなインスタンス] – フリートロケーションにデプロイされたサーバープロセスが実行されているインスタンスの数。
- [アクティブなサーバー] – フリートロケーションでゲームセッションをホストできるゲームサーバープロセスの数。
- [ゲームセッション] – フリートロケーション内のインスタンスでアクティブなゲームセッションの数。
- [プレイヤーセッション] – フリートロケーションでアクティブなゲームセッションに参加しているプレイヤーセッション数 (個々のプレイヤーを表します)。

ゲームセッション

[Game sessions] タブには、フリートにホストされている過去と現在のゲームセッションと、いくつかの詳細情報がリストされます。ゲームセッション ID を選択すると、メディアプレイヤーセッションを含む追加のゲームセッション情報にアクセスできます。プレイヤーセッションの詳細については、「ゲームおよびプレイヤーセッションのデータを表示する」を参照してください。

ゲームおよびプレイヤーセッションのデータを表示する

フリゲームセッションと個々のプレイヤーに関する情報を表示できます。ゲームセッションとプレイヤーセッションの詳細については、『プレイヤーがゲームに接続する方法』を参照してください。

ゲームセッションとプレイヤーのデータの表示するには

1. [Amazon GameLift コンソール] のナビゲーションペインで、[フリート] を選択します。
2. ゲームセッションをホストしたフリートリストから [フリート] を選択します。

3. [ゲームセッション] タブを選択します。このタブには、フリートにホストされているすべてのゲームセッションと概要情報が表示されます。
4. ゲームセッションを選択し、ゲームセッションに関する追加情報と、ゲームに接続したプレイヤーのリストを表示します。

詳細

概要

このセクションには、ゲームセッション情報の概要が表示されます。

- [Status] (ステータス) – ゲーム セッションステータスです。
  - [アクティベート中] – インスタンスはゲームセッションを開始しています。
  - [アクティブ] – ゲームセッションが実行されており、プレイヤーを受け入れ可能です (セッションの [プレイヤー作成ポリシー] によります)。
  - [終了済み] – ゲームセッションが終了しました。
- [ARN] – ゲームセッションの Amazon リソースネーム。
- [名前] – ゲームセッションに対して生成された名前。
- [ロケーション] – Amazon GameLift がゲームセッションをホストしたロケーション。
- [作成時刻] – Amazon GameLift がストリームセッションを作成した日付と時刻。
- [終了時刻] – ゲームセッションが終了した日付と時刻。
- [DNS 名] – ゲームセッションのホスト名。
- [IP address] (IPアドレス) – ゲームセッションに指定された IP アドレスです。
- [Port] (ポート) – ゲームセッションへのConnectに使用されるポート番号です。
- [作成者 ID] – ゲームセッションを開始したプレイヤーの一意の識別子。
- [プレイヤーセッション作成ポリシー] – ゲームセッションが新しいプレイヤーを受け付けているかどうかを示します。
- [ゲームスケーリング保護ポリシー] – Amazon GameLift がフリートで開始するすべての新しいインスタンスに設定するゲームセッション保護のタイプ。

ゲームデータ

開始時にゲームセッションに送信される、適切にフォーマットされたデータ。

ゲームのプロパティ

ゲームセッションに影響するキーと値のペアのプロパティ。

マッチメーキングデータ

FlexMatch マッチメーカー JSON マッチメーカーを確認して編集するには、「マッチメーキング設定を表示する」を選択します。FlexMatch マッチメーキングの詳細については、「マッチメーカーを構築する」を参照してください。

プレイヤーセッション

ゲームセッションごとに次のプレイヤーセッションデータが収集されます。

- [プレイヤーセッション ID] – プレイヤーセッションに割り当てられた識別子。
- [Player ID] (プレイヤーID) – プレイヤーの一意の識別子です。この ID を選択して、追加のプレイヤー情報を取得します。
- [Status](ステータス) – プレイヤーセッションのステータスです。次のようなステータスがあります。
  - [予約済み] – プレイヤーセッションが予約されていますが、プレイヤーがまだ接続されていません。
  - [アクティブ] – プレイヤーセッションが現在ゲームサーバーに接続されています。
  - [Completed] (完了) – プレイヤーセッションが終了しました。プレイヤーは切断されました。
  - [タイムアウト] – プレーヤーが接続できませんでした。
- [開始時刻] – プレイヤーがゲームセッションにConnectした時間です。
- [終了時刻] – プレイヤーがゲームセッションから切断した時間です。
- [プレイヤーデータ] – プレイヤーセッションの作成中に提供されたプレイヤーに関する情報。

プレイヤー情報

現在のリージョンのすべてのフリート間でプレイヤーが接続したすべてのゲームのリストなど、選択したプレイヤーの追加情報を表示します。この情報には、各プレーヤーセッションのステータス、開始時刻と終了時刻、および合計接続時間が含まれます。選択すると、関連するゲームセッションとフリートのデータを表示できます。

エイリアスを表示する

[エイリアス] ページには、現在のリージョンで作成したフリートエイリアスに関する情報が表示されます。エイリアスページを表示するには、ナビゲーションペインで [エイリアス] を選択します。

この英国ページで、以下の操作を実行できます。

- 新しいエイリアスを作成する。[エイリアスの作成] を選択します。
- エイリアスのテーブルのフィルタリングとソートをする。テーブルの先頭でコントロールを使用します。
- エイリアスの詳細を表示する。エイリアス名を選択して、エイリアスの詳細ページを開く。
- エイリアスを削除する。エイリアスを選択し、[削除] を選択する。

エイリアスの詳細

エイリアスの詳細ページにはエイリアスに関する情報の概要が表示されます。

このページからできること:

- エイリアスを編集する。[Edit] (編集) を選択します。
- エイリアスに関連付けられているフリートを表示する。
- エイリアスを削除する。[Delete] (削除) をクリックします。

エイリアスの詳細情報を次に示します。

- [エイリアスID] – エイリアスを識別するために使用される一意の数値。
- [Description] (説明) – エイリアスの説明。
- [ARN] – エイリアスの Amazon リソースネーム。
- [作成] – エイリアスが作成された日時。
- [最終更新日] – エイリアスが最後に更新された日時のタイムスタンプ。
- [タイプ] – エイリアスのルーティングタイプで、以下のいずれかです。
  - [シンプル] – プレイヤートラフィックを指定されたフリート ID にルーティングします。エイリアスのフリート ID はいつでも更新できます。
  - [ターミナル] – クライアントにメッセージを返します。例えば、古いクライアントを使用しているプレイヤーを、アップデートを入手できるロケーションに誘導することができます。
- [タグ] – エイリアスの識別に使用されるキーと値のペア。

キューを表示する

既存のすべてのゲームセッション配置キュー に関する情報を表示できます。キューページには、現在のリージョンで作成されたキューが表示されます。[Queues] ページから、新しいキューの作成、既存のキューの削除、選択したキューの詳細ページの表示を行うことができます。各キューの詳細ページには、キューの設定とメトリクスデータが含まれます。キューの詳細については、「ゲームセッションプレイスメント用の Amazon GameLiftt キューのセットアップ」を参照してください。

キューのページには、各キューに関する次の概要情報が表示されます。

- [キュー名] - キューに割り当てられた名前です。新しいゲームセッションのリクエストはこの名前でキューを指定します。
- [キュータイムアウト] - ゲームセッション配置リクエストがキューに残る最長時間 (秒) です。この時間を経過するとタイムアウトします。
- [キューにある送信先] - キュー設定に一覧表示されるフリートの数です。Amazon GameLift は、キューにある任意のフリートの新しいゲームセッションを配置します。

キューの詳細を表示する

キューの設定やメトリクスなど、キューの詳細情報にアクセスできます。キューの詳細ページを開くには、[キュー] ページに移動し、キュー名を選択します。

キューの詳細ページには、概要テーブルと、追加情報を含むタブが表示されます。このページで、以下の操作を実行できます。

- キューの設定、送信先のリスト、プレイヤーレイテンシーポリシーを更新します。[Edit] (編集) を選択します。
- キューを削除する キューを削除すると、新規のゲームセッションでそのキュー名を参照するすべてのリクエストが失敗します。[Delete] (削除) をクリックします。

> **Note**
>
> 削除したキューを復元するには、削除したキューの名前で新しいキューを作成します。

詳細

概要

[概要] セクションには、キューの Amazon リソースネーム (ARN) と [タイムアウト] が表示されます。ARN は Amazon GameLift の他のアクションやエリアでキューを参照するときに使用できます。タイムアウトは、ゲームセッションプレイスメントリクエストがキューに残る最長時間 (秒) です。この時間を経過するとタイムアウトします。

イベントの通知

[イベント通知] セクションには、Amazon GameLift がイベント通知を発行する [SNS トピック] と、このキューによって作成されたすべてのイベントに追加される [イベントデータ] が一覧表示されます。

タグ

[タグ] テーブルには、リソースにタグを付けるために使用されるキーと値が表示されます。タグ付けの詳細については、「AWS リソースのタグ付け」を参照してください。

メトリクス

[Metrics] (メトリクス) タブには、時間の経過に伴うキューメトリクスのグラフィカル表現が表示されます。

キューメトリクスには、成功したプレイスメントのリージョン別内訳など、キュー全体のプレイスメントアクティビティに関する様々な情報が含まれます。リージョンデータを使用して、ゲームをホストしている場所を把握できます。キュー設計全体の問題を検出するのにリージョン別のプレイスメントメトリクスが役立つ場合があります。

キューメトリクスは、Amazon CloudWatch でも利用できます。利用可能なメトリクスの詳細については、「キューの Amazon GameLift メトリクス」を参照してください。

送信先

[Destinations] (送信先) タブに、キューにリストされたすべてのフリートまたはエイリアスが表示されます。

Amazon GameLift は、新規のゲームセッションをホストできるリソースの送信先を検索するとき、ここにリストされたデフォルトの順序で検索します。リストされた最初の送信先にキャパシティがある限り、Amazon GameLift は新規のゲームセッションをそこに配置します。プレイヤーのレイテンシーデータを提供することによって、個々のゲームセッションの配置リクエストにデフォルトの順序を上書きさせることができます。このデータは、Amazon GameLift に、平均プレイヤーレイテンシーが最も小さい、利用可能な送信先を検索するように指示します。キューの設計に関する詳細については、「ゲームセッションキューの設計」を参照してください。

セッションプレイスメント

プレイヤーレイテンシーポリシー

[プレイヤーレイテンシーポリシー] セクションには、キューが使用するすべてのポリシーが表示されます。表には、ポリシーが適用される順序で一覧表示されます。

Locations

[ロケーション] セクションには、このキューがゲームセッションを配置できるロケーションが表示されます。

優先度

[優先度] セクションには、キューがゲームセッションの詳細を評価する順序が表示されます。

ロケーションの順序

[ロケーションの順序] セクションには、ゲームセッションを配置する際にキューが使用するデフォルトの順序が表示されます。他に優先度を設定していない場合、キューはこの順序を使用します。

Amazon のモニタリング GameLift

Amazon EC2 のスタンドアロン機能として Amazon GameLift FleetIQ を使用している場合は、Amazon EC2 EC2 ユーザーガイド」のAmazon EC2 のセキュリティ」を参照してください。Amazon EC2

モニタリングは、Amazon およびその他の AWS ソリューションの信頼性、可用性、パフォーマンスを維持する GameLift 上で重要な部分です。Amazon のメトリクスには、主に 3 つの用途があります GameLift。システムの状態のモニタリングとアラームの設定、ゲームサーバーのパフォーマンスと使用状況の追跡、手動または自動スケーリングを使用した容量の管理です。

AWS には、Amazon を監視し GameLift、問題が発生したときに報告し、必要に応じて自動アクションを実行するための以下のモニタリングツールが用意されています。

- Amazon GameLift コンソール
- Amazon CloudWatch - – Amazon GameLift メトリクス、および AWS のサービスで実行している他の AWS リソースやアプリケーションのメトリクスをリアルタイムでモニタリングできます。は、カスタマイズされたダッシュボードを作成するツールや、メトリクスが指定されたしきい値に達したときに通知またはアクションを実行するアラームを設定する機能など、一連のモニタリング機能 CloudWatch を提供します。
- AWS CloudTrail – は、Amazon およびその他の AWS のサービスについて、アカウントによって、または AWS アカウントに代わって行われたすべての API コール GameLift および関連イベントをキャプチャします。データは、指定した Amazon S3 バケットにログファイルとして配信されます。を呼び出したユーザーとアカウント AWS、呼び出し元のソース IP アドレス、呼び出しが発生した日時を特定できます。
- ゲームセッションログ – ゲームセッションのカスタムサーバーメッセージを、Amazon S3 に保存されているログファイルに出力できます。

トピック

- Amazon CloudWatch で Amazon GameLift をモニタリングする
- AWS CloudTrail での Amazon GameLift API コールのログ記録
- Amazon GameLift でのサーバーメッセージのログ記録

Amazon CloudWatch で Amazon GameLift をモニタリングする

Amazon GameLift をモニタリングするには Amazon CloudWatch を使用できます、この AWS サービスは、raw データを収集し、読み取り可能なほぼリアルタイムのメトリクスに加工します。これらの統計は 15 か月間保持され、Amazon GameLift でホストしているゲームサーバーのパフォーマンスに関する履歴情報を提供します。特定のしきい値を監視するアラームを設定し、これらのしきい値に達したときに通知を送信したりアクションを実行したりできます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

以下の表は、Amazon GameLift のメトリクスとディメンションの一覧です。CloudWatch で利用可能なすべてのメトリクスは、Amazon GameLift コンソールでも利用可能です。このコンソールでは、データがカスタマイズ可能なグラフとして提供されます。ゲームの CloudWatch メトリクスにアクセスするには、AWS Management Console、AWS CLI または CloudWatch API を使用します。

メトリクスにロケーションがない場合は、ホームロケーションが使用されます。

Amazon GameLift メトリクスのディメンション

Amazon GameLift は、以下のディメンションでフィルタリングメトリクスをサポートします。

ディメンション	説明
`Location`	フリートのデプロイ ロケーションのメトリクスをフィルター処理します。メトリクスにロケーションがない場合は、ホームロケーションが使用されます。
`FleetId`	1 つのフリートのフィルタメトリックス。このディメンションは、インスタンス、サーバープロセス、ゲームセッション、プレイヤーセッションのすべてのフリートメトリクスで使用できます。
`MetricGroup`	フリートの集合のメトリクスをフィルタ処理します。フリートをフリートメトリクスグループに含めるには、メトリクスグループ名をフリートの属性 (「UpdateFleetAttributes ()」を参照) に追加します。このディメンションは、インスタンス、サーバープ

ディメンション	説明
	ロセス、ゲームセッション、プレイヤーセッションのすべてのフリートメトリクスで使用できます。
QueueName	1 つのキューのメトリクスをフィルタ処理します。このディメンションはゲームセッションキューのメトリクスでのみ使用されます。
ConfigurationName	1 つのマッチメーキング設定のフィルタメトリックス。このディメンションはマッチメーキング設定のメトリクスでのみ使用されます。
ConfigurationName-RuleName	マッチメーキング設定とマッチメーキングルールの交差のメトリクスをフィルタ処理します。このディメンションはマッチメーキングルールのメトリクスでのみ使用されます。
InstanceType	EC2 インスタンスタイプ (「c4.large」など) のメトリクスをフィルタ処理します。このディメンションはスポットインスタンスのメトリクスで使用されます。
OperatingSystem	インスタンスのオペレーティングシステムのメトリクスをフィルタ処理します。このディメンションはスポットインスタンスのメトリクスで使用されます。
GameServerGroup	ゲーム サーバーグループの FleetIQ メトリクスをフィルターする。

フリートの Amazon GameLift メトリクス

AWS/GameLift 名前空間には、フリート全体またはフリートグループにまたがるアクティビティに関する以下のメトリクスが含まれます。フリートはマネージド Amazon GameLift ソリューションで使用されます。Amazon GameLift サービスは CloudWatch に毎分メトリクスを送信します。

インスタンス

メトリクス	説明
ActiveInstances	ACTIVE ステータスのインスタンスの数。このステータスは、アクティブなサーバープロセスを実行中であることを示します。数値には、アイドル状態のインスタンスや、1 つ以上のゲームセッションをホストしているインスタンスが含まれます。このメトリクスでは、インスタンスの現在の総容量を測定します。このメトリクスは Auto Scaling で使用できます。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション
DesiredInstances	Amazon GameLift でフリートに保持しようとしているアクティブなインスタンスの目標数。Auto Scaling を使用している場合、この値は現在有効なスケーリングポリシーに基づいて決定されます。Auto Scaling を使用していない場合、この値は手動で設定します。このメトリクスは、フリートメトリクスグループのデータの表示には使用できません。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション
IdleInstances	現在ホストしているゲームセッション数が 0 (ゼロ) であるアクティブなインスタンス。このメトリクスでは、使用可能であるが使用されていない容量を測

メトリクス	説明
	定します。このメトリクスは Auto Scaling で使用できます。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション
MaxInstances	フリートで許容されるインスタンスの最大数。フリートのインスタンスの最大数により、手動または自動で容量をスケールアップする上限が決まります。このメトリクスは、フリートメトリクスグループのデータの表示には使用できません。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション
MinInstances	インスタンスで許容されるインスタンスの最小数。フリートのインスタンスの最小数により、手動または自動で容量をスケールダウンする下限が決まります。このメトリクスは、フリートメトリクスグループのデータの表示には使用できません。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション

メトリクス	説明
PercentIdleInstances	すべてのアイドル状態のアクティブなインスタンスのパーセント (IdleInstances / ActiveInstances で計算)。このメトリクスは Auto Scaling で使用できます。 単位: パーセント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション
RecycledInstances	リサイクルおよび置換されたスポットインスタンスの数。Amazon GameLift は、現在ゲームセッションをホストしておらず、中断の可能性が高いスポットインスタンスをリサイクルします。 単位はカウント 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: ロケーション
InstanceInterruptions	中断されたスポットインスタンス数。 単位はカウント 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: ロケーション

メトリクス	説明
CPUUtilization	EC2 メトリクス。Amazon GameLift の場合、このメトリクスはフリートロケーション内のすべてのアクティブインスタンスにおけるハードウェアパフォーマンスを表します。Amazon EC2 がインスタンスを実行するために使用する物理 CPU 時間の割合。これには、ユーザーコードと Amazon EC2 コードの両方を実行するために費やされた時間が含まれます。オペレーティングシステムのツールは CloudWatch と異なる割合を表示することがあります。これは、レガシーデバイスのシミュレーション、レガシーではないデバイスの設定、中断の多いワークロード、ライブ移行、ライブアップデートなどが原因です。 単位: パーセント
NetworkIn	EC2 メトリクス。Amazon GameLift の場合、このメトリクスはフリートロケーション内のすべてのアクティブインスタンスにおけるハードウェアパフォーマンスを表します。すべてのネットワークインターフェイスでの、このインスタンスによって受信されたバイトの数。このメトリクスは、1 つのインスタンス上での 1 つのアプリケーションへのネットワークトラフィックの量を表しています。 単位: バイト
NetworkOut	EC2 メトリクス。Amazon GameLift の場合、このメトリクスはフリートロケーション内のすべてのアクティブインスタンスにおけるハードウェアパフォーマンスを表します。すべてのネットワークインターフェイスでの、このインスタンスから送信されたバイトの数。このメトリクスは、1 つのインスタンス上での 1 つのアプリケーションからのネットワークトラフィックの量を表しています。 単位: バイト

メトリクス	説明
DiskReadBytes	EC2 メトリクス。Amazon GameLift の場合、このメトリクスはフリートロケーション内のすべてのアクティブインスタンスにおけるハードウェアパフォーマンスを表します。インスタンスで利用できるすべてのインスタンスストアボリュームから読み取られたバイト数。このメトリクスを使用すると、このインスタンスのハードディスクからアプリケーションが読み取るデータの量がわかります。これを利用すると、アプリケーションの速度を決定できます。 単位: バイト
DiskWriteBytes	EC2 メトリクス。Amazon GameLift の場合、このメトリクスはフリートロケーション内のすべてのアクティブインスタンスにおけるハードウェアパフォーマンスを表します。インスタンスで利用できるすべてのインスタンスストアボリュームに書き込まれたバイト数。このメトリクスを使用すると、このインスタンスのハードディスクにアプリケーションが書き込むデータの量がわかります。これを利用すると、アプリケーションの速度を決定できます。 単位: バイト
DiskReadOps	EC2 メトリクス。Amazon GameLift の場合、このメトリクスはフリートロケーション内のすべてのアクティブインスタンスにおけるハードウェアパフォーマンスを表します。指定された期間にインスタンスで利用できるすべてのインスタンスストアボリュームでの、完了した読み取り操作。その期間の 1 秒あたりの I/O 操作回数 (IOPS) の平均を算出するには、その期間の操作回数の合計をその期間の秒数で割ります。 単位はカウント

メトリクス	説明
DiskWriteOps	EC2 メトリクス。Amazon GameLift の場合、このメトリクスはフリートロケーション内のすべてのアクティブインスタンスにおけるハードウェアパフォーマンスを表します。指定された期間にインスタンスで利用できるすべてのインスタンスストアボリュームへの、完了した書き込み操作。その期間の 1 秒あたりの I/O 操作回数 (IOPS) の平均を算出するには、その期間の操作回数の合計をその期間の秒数で割ります。 単位はカウント

サーバープロセス

メトリクス	説明
ActiveServerProcesses	ACTIVE ステータスのサーバープロセス。このステータスは、プロセスが実行中でゲームセッションをホストできることを示します。数値には、アイドル状態のサーバープロセスやゲームセッションをホストしているサーバープロセスが含まれます。このメトリクスでは、サーバープロセスの現在の総容量を測定します。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション
HealthyServerProcesses	正常な状態を示しているアクティブなサーバープロセス。このメトリクスは、フリートのゲームサーバーの全体的な状態を追跡するのに役立ちます。 単位はカウント

メトリクス	説明
	関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション
PercentHealthyServerProcesses	正常な状態を示しているすべてのアクティブなサーバープロセスのパーセント (HealthyServerProcesses / ActiveServerProcesses で計算)。 単位: パーセント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション
ServerProcessAbnormalTerminations	前回のレポート以降に異常事態のためにシャットダウンされたサーバープロセス。このメトリクスには、Amazon GameLift サービスによって開始された終了が含まれます。これは、サーバープロセスが応答を停止した場合、継続的にヘルスチェックの失敗をレポートする場合、または ProcessEnding () の呼び出しで正常に終了しない場合に発生します。 単位はカウント 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: ロケーション

メトリクス	説明
ServerProcessActivations	前回のレポート以降に ACTIVATING から ACTIVE のステータスに正常に移行したサーバープロセス。サーバープロセスは、アクティブになるまでゲームセッションをホストすることはできません。 単位はカウント 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: ロケーション
ServerProcessTerminations	前回のレポート以降にシャットダウンされたサーバープロセス。これには、プロセスの正常または異常な終了を問わず、何らかの理由で TERMINATED ステータスに移行したすべてのサーバープロセスが含まれます。 単位はカウント 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: ロケーション

ゲームセッション

メトリクス	説明
ActivatingGameSessions	ACTIVATING ステータスのゲームセッション。このステータスは、セッションが起動中であることを示します。ゲームセッションは、アクティブになるまでプレイヤーをホストすることはできません。起動時間が長引く場合、セッションは ACTIVATING から ACTIVE ステータスに移行していない可能性があり

メトリクス	説明
	ます。このメトリクスは Auto Scaling で使用できます。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション
ActiveGameSessions	Active ステータスのゲームセッション。このステータスは、セッションでプレイヤーをホストできること、およびゼロ個以上のプレイヤーをホストしていることを示します。このメトリクスでは、現在ホストされているゲームセッションの総数を測定します。このメトリクスは Auto Scaling で使用できます。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション

メトリクス	説明
AvailableGameSessions	現在ゲームセッションのホストに使用されておらず、新しいサーバープロセスまたはインスタンスをスピンアップするために、遅滞なく新しいゲームセッションをスタートできる、アクティブで正常なサーバープロセス。このメトリクスは Auto Scaling で使用できます。 **Note** 同時ゲームセッションのアクティベーションを制限するフリートについては、メトリクス ConcurrentActivatableGameSessions を使用します。このメトリクスでは、遅延なしでスタートできる新しいゲームセッションの数をより正確に表します。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション

メトリクス	説明
ConcurrentActivatableGameSessions	現在ゲームセッションのホストに使用されていないアクティブで正常なサーバープロセスであり、新しいゲームセッションをすぐにスタートできます。 このメトリクスは、次の方法で AvailableGameSessions と異なります: ゲームセッションのアクティビティに制限があるため、現在新しいゲームセッションをアクティブ化できないサーバープロセスはカウントされません。(フリート RuntimeConfiguration オプション設定MaxConcurrentGameSessionActivations を参照してください)。ゲームセッションのアクティベーションを制限しないフリートの場合、このメトリクスは AvailableGameSessions と同一です。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: ロケーション
PercentAvailableGameSessions	現在使用されていないすべてのアクティブなサーバープロセス (正常または異常を問わない) のゲームセッションスロットのパーセント (AvailableGameSessions / [ActiveGameSessions + AvailableGameSessions + unhealthy server processes] で計算)。このメトリクスは Auto Scaling で使用できます。 単位: パーセント 関連する CloudWatch 統計: Maximum、Average ディメンション: ロケーション

メトリクス	説明
GameSessionInterruptions	中断されたスポットインスタンスのゲームセッション数。 単位はカウント 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: ロケーション

プレイヤーセッション

メトリクス	説明
CurrentPlayerSessions	ステータスが ACTIVE (プレイヤーはアクティブなゲームセッションに接続されている) であるか、RESERVED (プレイヤーはゲームセッションのスロットを与えられているが、まだ接続されていない) であるプレイヤーセッション。このメトリクスは Auto Scaling で使用できます。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum
PlayerSessionActivations	前回のレポート以降に RESERVED から ACTIVE ステータスに移行したプレイヤーセッション。これは、プレイヤーがアクティブなゲームセッションに正常に接続した場合に発生します。 単位はカウント 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum

キューの Amazon GameLift メトリクス

Amazon GameLift 名前空間には、ゲームセッション配置キュー全体のアクティビティに関する以下のメトリクスが含まれます。キューはマネージド Amazon GameLift ソリューションで使用されます。Amazon GameLift サービスは CloudWatch に毎分メトリクスを送信します。

メトリクス	説明
AverageWaitTime	ゲームセッション配置キューで PENDING ステータスのリクエストが実行されるまでの平均待機時間。 単位: 秒 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum ディメンション: ロケーション
FirstChoiceNotViable	第 1 選択のフリートが有効ではない (中断率の高いスポットフリートなどである) と判断されたために、別のフリートに正常に配置されたゲームセッション。このメトリクスは、レイテンシーではなくコストに基づいています。第 1 選択のフリートは、キューの先頭のフリートです。または、配置リクエストにプレイヤー レイテンシー データが含まれている場合は、[FleetIQ prioritization] (FleetIQ の優先度設定) によって最初に選択されたフリートです。有効なスポットフリートがない場合には、このリージョンの任意のフリートが選択されます。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Sum
FirstChoiceOutOfCapacity	第 1 選択のフリートに使用可能なリソースがないために、別のフリートに正常に配置されたゲームセッション。第 1 選択のフリートは、キューの先頭のフリートです。または、プレイスメントリクエストに

メトリクス	説明
	プレイヤーレイテンシーデータが含まれている場合は、定義した FleetIQ の優先度設定によって最初に選択されたフリートです。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum
LowestLatencyPlacement	プレイヤーに対してキューの最低のレイテンシーを提供するリージョンに正常に配置されたゲームセッション。このメトリクスは、配置リクエストにプレイヤーレイテンシーデータが含まれている場合にのみ生成されます。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum
LowestPricePlacement	選択した リージョン でキューの料金が最低であるフリートに正常に配置されたゲームセッション。このフリートは、スポットフリートであるか、キューにスポットインスタンスがない場合はオンデマンドインスタンスになります。このメトリクスは、配置リクエストにプレイヤーレイテンシーデータが含まれている場合にのみ生成されます。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum

メトリクス	説明
Placement <region name>	指定したリージョン内のフリートに正常に配置されたゲームセッション。このメトリクスは、PlacementsSucceeded メトリクスのリージョン別内訳を示します。 単位はカウント 関連する CloudWatch 統計: Sum
PlacementsCanceled	前回のレポート以降に、タイムアウト前にキャンセルされたゲームセッション配置リクエスト。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum
PlacementsFailed	前回のレポート以降に何らかの理由で失敗したゲームセッション配置リクエスト。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum
PlacementsStarted	前回のレポート以降にキューに追加された新しいゲームセッション配置リクエスト。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum

メトリクス	説明
PlacementsSucceeded	前回のレポート以降に新しいゲームセッションとなったゲームセッション配置リクエスト。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum
PlacementsTimedOut	前回のレポート以降に、キューのタイムアウト制限に達して実行されなかったゲームセッション配置リクエスト。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum
QueueDepth	キュー内で PENDING ステータスのゲームセッション配置リクエストの数。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum ディメンション: ロケーション

マッチメーキングの Amazon GameLift メトリクス

Amazon GameLift名前空間には、マッチメーキング設定とマッチメーキングルールの FlexMatc アクティビティに関するメトリクスが含まれます。FlexMatch マッチメーキングは、マネージド Amazon GameLift ソリューションで使用されます。Amazon GameLift サービスは CloudWatch に毎分メトリクスを送信します。

マッチメーキングアクティビティのシーケンスの詳細については、Amazon GameLift FlexMatch の仕組みを参照してください。

マッチメーキング設定

メトリクス	説明
CurrentTickets	マッチメーキングリクエストは現在処理中であるか、処理の待機中です。 単位はカウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum
MatchAcceptancesTimedOut	承諾を要求するマッチメーキング設定の場合、前回のレポート後に承諾プロセス中にタイムアウトしたマッチング案の数。 単位はカウント 関連する CloudWatch 統計: Sum
MatchesAccepted	承諾を要求するマッチメーキング設定の場合、前回のレポート後に承諾されたマッチング案の数。 単位はカウント 関連する CloudWatch 統計: Sum
MatchesCreated	前回のレポート後に作成された可能性のあるマッチング案の数。 単位はカウント 関連する CloudWatch 統計: Sum
MatchesPlaced	前回のレポート後にゲームセッションに正常に配置されたマッチング案の数。 単位はカウント 関連する CloudWatch 統計: Sum

メトリクス	説明
MatchesRejected	承諾を要求するマッチメーキング設定の場合、前回のレポート後に少なくも 1 人のプレイヤーによって却下された可能性のあるマッチング案の数。 単位はカウント 関連する CloudWatch 統計: Sum
PlayersStarted	前回のレポート後に追加されたマッチメーキングチケットのプレイヤーの数。 単位はカウント 関連する CloudWatch 統計: Sum
TicketsFailed	前回のレポート後に、マッチメーキングリクエストが失敗した数。 単位はカウント 関連する CloudWatch 統計: Sum
TicketsStarted	前回のレポート後に作成された新しいマッチメーキングリクエスト。 単位はカウント 関連する CloudWatch 統計: Sum
TicketsTimedOut	前回のレポート後に、タイムアウトしたマッチメーキングリクエストの数。 単位はカウント 関連する CloudWatch 統計: Sum

メトリクス	説明
TimeToMatch	前回のレポート以前にマッチング候補に配置されたマッチメーキングリクエストの場合、チケットの作成からマッチング候補の作成までの時間。 単位: 秒 関連する CloudWatch 統計: Data Samples、Average、Minimum、Maximum
TimeToTicketCancel	前回のレポート前にキャンセルされたマッチメーキングリクエストの場合、チケットの作成からキャンセルまでの時間。 単位: 秒 関連する CloudWatch 統計: Data Samples、Average、Minimum、Maximum
TimeToTicketSuccess	前回のレポート前に成功したマッチメーキングリクエストの場合、チケットの作成からマッチングが正常に配置されるまでの時間。 単位: 秒 関連する CloudWatch 統計: Data Samples、Average、Minimum、Maximum

マッチメーキングルール

メトリクス	説明
RuleEvaluationsPassed	前回のレポート後に、マッチメーキングプロセスで合格したルール評価数。このメトリクスは、上位 50 のルールに制限されます。 単位はカウント

メトリクス	説明
	関連する CloudWatch 統計: Sum
RuleEvaluationsFailed	前回のレポート後に、マッチメーキングで失敗したルール評価数。このメトリクスは、上位 50 のルールに制限されます。 単位はカウント 関連する CloudWatch 統計: Sum

FleetIQ の Amazon GameLift メトリクス

Amazon GameLift名前空間には、ゲームホスティングの FleetIQ スタンドアロンソリューションのパートとして、 ゲームサーバーグループとゲームサーバーのアクティビティに関するメトリクスが含まれます。Amazon GameLift サービスは CloudWatch に毎分メトリクスを送信します。また、Amazon EC2 Auto Scaling ユーザーガイドの「Amazon CloudWatch を使用した Auto Scaling グループとインスタンスのモニタリング」も参照してください。

メトリクス	説明
AvailableGameServers	ゲーム実行に使用可能で、ゲームプレイに現在使用されていないゲームサーバーの数。この数には、クレーム済みであるが、まだ AVAILABLE ステータスにあるゲームサーバーが含まれます。 単位はカウント 関連する CloudWatch 統計: Sum ディメンション: GameServerGroup
UtilizedGameServers	ゲームプレイに現在使用されているゲームサーバー。この数字には、[UTILIZED (使用中)] 状態のゲームサーバーも含まれます。 単位はカウント

メトリクス	説明
	関連する CloudWatch 統計: Sum ディメンション: GameServerGroup
DrainingAvailableGameServers	ゲームプレイを現在サポートしていない、終了予定のインスタンスのゲームサーバー。これらのゲームサーバーは、新しいクレームリクエストがあった場合に、クレームされる優先度が最も低いゲームサーバーです。 単位はカウント 関連する CloudWatch 統計: Sum ディメンション: GameServerGroup
DrainingUtilizedGameServers	ゲームプレイを現在サポートしている、終了予定のインスタンスのゲームサーバー。 単位はカウント 関連する CloudWatch 統計: Sum ディメンション: GameServerGroup
PercentUtilizedGameServers	ゲーム実行を現在サポートしているゲームサーバーの割合。このメトリクスは、ゲームサーバーキャパシティーの現在の使用量を示します。プレイヤーの需要に合わせてインスタンスを動的に追加および削除できる Auto Scaling ポリシーを実行する場合に役立つメトリクスです。 単位: パーセント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: GameServerGroup

メトリクス	説明
GameServerInterruptions	スポットの可用性が制限されているために中断されたスポットインスタンスのゲームサーバー。 単位はカウント 関連する CloudWatch 統計: Sum ディメンション: GameServerGroup、InstanceType
InstanceInterruptions	可用性が制限されているために中断されたスポットインスタンス。 単位はカウント 関連する CloudWatch 統計: Sum ディメンション: GameServerGroup、InstanceType

AWS CloudTrail での Amazon GameLift API コールのログ記録

Amazon GameLift は AWS CloudTrail と統合され、このサービスは Amazon GameLift 内のユーザー、ロール、または AWS サービスによって実行されたアクションのレコードを提供します。CloudTrail は、Amazon GameLift へのすべての API コールをイベントとしてキャプチャします。キャプチャされた呼び出しには、Amazon GameLiftコンソールからの呼び出しと、Amazon GameLift API オペレーションへのコード呼び出しが含まれます。証跡を作成する場合は、Amazon GameLift のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的デリバリーを有効にすることができます。追跡を設定しない場合でも、CloudTrail コンソールの [Event history] (イベント履歴) で最新のイベントを表示できます。CloudTrail により収集された情報を使用して、Amazon GameLift に対して行われたリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエストが行われた日時、および追加の詳細を特定することができます。

CloudTrail の詳細については、AWS CloudTrail ユーザーガイドを参照してください。

CloudTrail 内の Amazon GameLift 情報

CloudTrail は、アカウント作成時に AWS アカウント で有効になります。Amazon GameLift でアクティビティが発生すると、そのアクティビティは [イベント履歴] の AWS サービスのイベントととも

に CloudTrail イベントに記録されます。最近のイベントは、AWS アカウント で表示、検索、ダウンロードできます。詳細については、 CloudTrail イベント履歴でのイベントの表示を参照してください。

Amazon GameLift のイベントなど、AWS アカウント 内のイベントを継続的にレコードするには、追跡を作成します。追跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成するときに、証跡がすべての AWS リージョン に適用されます 証跡は、AWS パーティションのすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集したイベントデータをより詳細に分析し、それに基づく対応するためにその他の AWS のサービスを設定できます。詳細については、次を参照してください。

- 「追跡を作成するための概要」
- CloudTrail がサポートされているサービスと統合
- CloudTrail の Amazon SNS 通知の設定
- 複数のリージョンから CloudTrail ログファイルを受け取るおよび複数のアカウントから CloudTrail ログファイルを受け取る

すべての Amazon GameLift アクションは、CloudTrail により記録され、Amazon GameLift API リファレンスで文書化されます。例えば、`CreateGameSession`、`CreatePlayerSession`、`UpdateGameSession` の各アクションを呼び出すと、CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- リクエストが、ルート認証情報と AWS Identity and Access Management (IAM) ユーザー認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーティッドユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービスによって送信されたかどうか。

詳細については、「CloudTrail userIdentity エレメント」を参照してください。

Amazon GameLift ログファイルエントリを理解する

「トレイル」は、指定した Simple Storage Service (Amazon S3) バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルには、単一か複数のログエントリがあります。イベントはあらゆるソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストのパラメータなどの情報が含まれます。CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次は、`CreateFleet` と `DescribeFleetAttributes` のアクションを示す CloudTrail ログエントリの例です。

```
{
    "Records": [
        {
            "eventVersion": "1.04",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::111122223333:user/myUserName",
                "accountId": "111122223333",
                "accessKeyId": AKIAIOSFODNN7EXAMPLE",
                "userName": "myUserName"
            },
            "eventTime": "2015-12-29T23:40:15Z",
            "eventSource": "gamelift.amazonaws.com",
            "eventName": "CreateFleet",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "[]",
            "requestParameters": {
                "buildId": "build-92b6e8af-37a2-4c10-93bd-4698ea23de8d",
                "eC2InboundPermissions": [
                    {
                        "ipRange": "10.24.34.0/23",
                        "fromPort": 1935,
                        "protocol": "TCP",
                        "toPort": 1935
                    }
                ],
                "logPaths": [
                    "C:\\game\\serverErr.log",
```

```
                "C:\\game\\serverOut.log"
            ],
            "eC2InstanceType": "c5.large",
            "serverLaunchPath": "C:\\game\\MyServer.exe",
            "description": "Test fleet",
            "serverLaunchParameters": "-paramX=baz",
            "name": "My_Test_Server_Fleet"
        },
        "responseElements": {
            "fleetAttributes": {
                "fleetId": "fleet-0bb84136-4f69-4bb2-bfec-a9b9a7c3d52e",
                "serverLaunchPath": "C:\\game\\MyServer.exe",
                "status": "NEW",
                "logPaths": [
                    "C:\\game\\serverErr.log",
                    "C:\\game\\serverOut.log"
                ],
                "description": "Test fleet",
                "serverLaunchParameters": "-paramX=baz",
                "creationTime": "Dec 29, 2015 11:40:14 PM",
                "name": "My_Test_Server_Fleet",
                "buildId": "build-92b6e8af-37a2-4c10-93bd-4698ea23de8d"
            }
        },
        "requestID": "824a2a4b-ae85-11e5-a8d6-61d5cafb25f2",
        "eventID": "c8fbea01-fbf9-4c4e-a0fe-ad7dc205ce11",
        "eventType": "AwsApiCall",
        "recipientAccountId": "111122223333"
    },
    {
        "eventVersion": "1.04",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "AIDACKCEVSQ6C2EXAMPLE",
            "arn": "arn:aws:iam::111122223333:user/myUserName",
            "accountId": "111122223333",
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "userName": "myUserName"
        },
        "eventTime": "2015-12-29T23:40:15Z",
        "eventSource": "gamelift.amazonaws.com",
        "eventName": "DescribeFleetAttributes",
        "awsRegion": "us-west-2",
        "sourceIPAddress": "192.0.2.0",
```

```
            "userAgent": "[]",
            "requestParameters": {
                "fleetIds": [
                    "fleet-0bb84136-4f69-4bb2-bfec-a9b9a7c3d52e"
                ]
            },
            "responseElements": null,
            "requestID": "82e7f0ec-ae85-11e5-a8d6-61d5cafb25f2",
            "eventID": "11daabcb-0094-49f2-8b3d-3a63c8bad86f",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        },
    ]
}
```

Amazon GameLift でのサーバーメッセージのログ記録

Amazon GameLift サーバーからのカスタムサーバーメッセージをログファイルにキャプチャできます。ログ記録の設定方法は、カスタムサーバーとリアルタイムサーバーのどちらを使用するかによって異なります (この章の該当するサブセクションを参照)。

トピック

- サーバーメッセージのログ記録 (カスタムサーバー)
- サーバーメッセージのログ記録 (リアルタイムサーバー)

サーバーメッセージのログ記録 (カスタムサーバー)

Amazon カスタムサーバーからの GameLift カスタムサーバーメッセージをログファイルにキャプチャできます。リアルタイムサーバーのログ記録については、「サーバーメッセージのログ記録 (リアルタイムサーバー)」を参照してください。

> **Important**
>
> ゲームセッションあたりのログファイルのサイズには制限があります (「」の「Amazon GameLift エンドポイントとクォータ」を参照AWS 全般のリファレンス)。ゲームセッションが終了すると、Amazon はサーバーログを Amazon Simple Storage Service (Amazon S3 GameLift) にアップロードします。Amazon GameLift は、制限を超えるログをアップロード

しません。ログは急速に大きくなり、サイズ制限を超えることがあります。ログをモニタリングして、ログ出力を必要なメッセージのみに制限する必要があります。

カスタムサーバーのログ記録設定

Amazon GameLift カスタムサーバーでは、独自のコードを記述してログ記録を実行します。これはサーバープロセス設定の一部として設定します。Amazon GameLift は、ログ記録設定を使用して、各ゲームセッションの終了時に Amazon S3 にアップロードする必要があるファイルを識別します。

以下の手順は、簡単なコード例を使用してログ記録を設定する方法を示しています。

C++

ログ記録を設定するには (C++)

1. ゲームサーバーログファイルへのディレクトリパスである文字列のベクトルを作成します。

```
std::string serverLog("serverOut.log");        // Example server log file
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);
```

2. ベクトルを ProcessParameters オブジェクトLogParametersの として指定します。

```
Aws::GameLift::Server::ProcessParameters processReadyParameter =
 Aws::GameLift::Server::ProcessParameters(
    std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
    std::bind(&Server::onProcessTerminate, this),
    std::bind(&Server::OnHealthCheck, this),
    std::bind(&Server::OnUpdateGameSession, this),
    listenPort,
    Aws::GameLift::Server::LogParameters(logPaths));
```

3. ProcessReady() を呼び出すときに ProcessParameters オブジェクトを指定します。

```
Aws::GameLift::GenericOutcome outcome =
   Aws::GameLift::Server::ProcessReady(processReadyParameter);
```

完全な例については、「ProcessReady()」を参照してください。

C#

ログ記録を設定するには (C#)

1. ゲームサーバーログファイルへのディレクトリパスである文字列のリストを作成します。

```
List<string> logPaths = new List<string>();
logPaths.Add("C:\\game\\serverOut.txt");     // Example of a log file that the
 game server writes
```

2. オブジェクトLogParametersの としてリストを指定しますProcessParameters。

```
var processReadyParameter = new ProcessParameters(
    this.OnGameSession,
    this.OnProcessTerminate,
    this.OnHealthCheck,
    this.OnGameSessionUpdate,
    port,
    new LogParameters(logPaths));
```

3. ProcessReady() を呼び出すときに ProcessParameters オブジェクトを指定します。

```
var processReadyOutcome =
   GameLiftServerAPI.ProcessReady(processReadyParameter);
```

完全な例については、「ProcessReady()」を参照してください。

ログへの書き込み

ログファイルは、サーバープロセスが起動した後も存在します。ログには、ファイルに書き込む任意の方法で書き込むことができます。サーバーの標準出力とエラー出力をすべてキャプチャするには、次の例のように出力ストリームをログファイルに再マップします。

C++

```
std::freopen("serverOut.log", "w+", stdout);
std::freopen("serverErr.log", "w+", stderr);
```

C#

```
Console.SetOut(new StreamWriter("serverOut.txt"));
Console.SetError(new StreamWriter("serverErr.txt"));
```

サーバーログへのアクセス

ゲームセッションが終了すると、Amazon GameLift は自動的にログを Amazon S3 バケットに保存し、14 日間保持します。ゲームセッションのログの場所を取得するには、 GetGameSessionLogUrl API オペレーションを使用できます。ログをダウンロードするには、オペレーションが返す URL を使用します。

サーバーメッセージのログ記録 (リアルタイムサーバー)

リアルタイムアーバーからのカスタムサーバーメッセージをログファイルにキャプチャできます。カスタムサーバーのログ記録については、「サーバーメッセージのログ記録 (カスタムサーバー)」を参照してください。

ログファイルに出力できるメッセージにはさまざまな種類があります (サーバースクリプトでのメッセージのログ記録 を参照)。リアルタイムサーバーは、カスタムメッセージに加えて、同じメッセージタイプを使用してシステムメッセージを出力し、同じログファイルに書き込みます。フリートのログ記録レベルを調整して、サーバーが生成するログ記録メッセージの量を減らすことができます (ログ記録のレベルの調整 を参照)。

> **Important**
>
> ゲームセッションあたりのログファイルのサイズには制限があります (「」の「Amazon GameLift エンドポイントとクォータ」を参照AWS 全般のリファレンス) 。ゲームセッションが終了すると、Amazon はサーバーログを Amazon Simple Storage Service (Amazon S3 GameLift) にアップロードします。Amazon GameLift は、制限を超えるログをアップロードしません。ログは急速に大きくなり、サイズ制限を超えることがあります。ログをモニタリングして、ログ出力を必要なメッセージのみに制限する必要があります。

サーバースクリプトでのメッセージのログ記録

リアルタイムサーバーのスクリプトにカスタムメッセージを出力できます。以下の手順でサーバーメッセージをログファイルに送信します。

1. ロガーオブジェクトへの参照を保持する変数を作成します。

```
var logger;
```

2. init() 関数では、セッションオブジェクトからロガーを取得し、ロガー変数に割り当てます。

```
function init(rtSession) {
    session = rtSession;
    logger = session.getLogger();
}
```

3. ロガーの適切な関数を呼び出してメッセージを出力します。

   デバッグメッセージ

```
logger.debug("This is my debug message...");
```

   情報メッセージ

```
logger.info("This is my info message...");
```

   警告メッセージ

```
logger.warn("This is my warn message...");
```

   エラーメッセージ

```
logger.error("This is my error message...");
```

   致命的なエラーメッセージ

```
logger.fatal("This is my fatal error message...");
```

   カスタマーエクスペリエンスに関する致命的なエラーメッセージ

```
logger.cxfatal("This is my customer experience fatal error message...");
```

スクリプト内のログ記録ステートメントの例については、「リアルタイムサーバースクリプト例」を参照してください。

ログファイルの出力には、サンプルログの次の行に示すように、メッセージのタイプ (DEBUG、INFO、WARN、ERROR、FATAL、CXFATAL) が示されています。

```
09 Sep 2021 11:46:32,970 [INFO] (gamelift.js) 215: Calling GameLiftServerAPI.InitSDK...
09 Sep 2021 11:46:32,993 [INFO] (gamelift.js) 220: GameLiftServerAPI.InitSDK succeeded
09 Sep 2021 11:46:32,993 [INFO] (gamelift.js) 223: Waiting for Realtime server to
 start...
09 Sep 2021 11:46:33,15 [WARN] (index.js) 204: Connection is INSECURE. Messages will be
 sent/received as plaintext.
```

サーバーログへのアクセス

ゲームセッションが終了すると、Amazon GameLift は自動的にログを Amazon S3 に保存し、14 日間保持します。GetGameSessionLogUrl API コールを使用して、ゲームセッションのログの場所を取得できます。API コールによって返された URL を使用してログをダウンロードします。

ログ記録のレベルの調整

ログは急速に大きくなり、サイズ制限を超えることがあります。ログをモニタリングして、ログ出力を必要なメッセージのみに制限する必要があります。リアルタイムサーバーでは、フリートのランタイム設定にパラメーターを `loggingLevel:`*`LOGGING_LEVEL`* フォームに入力することでログ記録レベルを調整できます。ここで、*`LOGGING_LEVEL`* は次のいずれかの値を指定します。

1. `debug`
2. `info` (デフォルト)
3. `warn`
4. `error`
5. `fatal`
6. `cxfatal`

このリストは、最も重大度が低い (`debug`) から最も重大度が高い (`cxfatal`) の順になっています。1 つの `loggingLevel` を設定すると、サーバーはその重要度レベル以上のメッセージのみをログに記録します。例えば、`loggingLevel:error` を設定すると、フリート内のすべてのサーバーがログに `error`、`fatal`、`cxfatal` メッセージのみを書き込むようになります。

フリートのログ記録レベルは、フリートの作成時または実行後に設定できます。実行後にフリートのログ記録レベルを変更しても、更新後に作成されたゲームセッションのログにのみ影響します。既存のゲームセッションのログには影響しません。フリートの作成時にログ記録レベルを設定しない場合、サーバーはログ記録レベルをデフォルトで info に設定します。ログ記録レベルを設定する手順については、次のセクションを参照してください。

リアルタイムサーバーフリートを作成する際のログ記録レベルの設定 (コンソール)

Amazon GameLift マネージドフリートを作成する に記載されている手順に従ってフリートを作成します。さらに、以下を行います。

- [プロセス管理] ステップの [サーバープロセス割り当て] サブステップで、[起動パラメータ] の値としてログ記録レベルのキーと値のペア (`loggingLevel:error` など) を指定します。英数字以外の文字 (カンマを除く) を使用して、ログ記録レベルと追加のパラメータを区切ります (`loggingLevel:error +map Winter444` など)。

リアルタイムサーバーフリートを作成する際のログ記録レベルの設定 (AWS CLI)

Amazon GameLift マネージドフリートを作成する に記載されている手順に従ってフリートを作成します。さらに、以下を行います。

- `create-fleet` の `--runtime-configuration` パラメータの引数に、`Parameters` の値としてログ記録レベルのキーと値のペア (`loggingLevel:error` など) を指定します。英数字以外の文字 (カンマを除く) を使用して、ログ記録レベルと追加のパラメータを区切ります。次の例を参照してください。

```
--runtime-configuration "GameSessionActivationTimeoutSeconds=60,
                    MaxConcurrentGameSessionActivations=2,
                    ServerProcesses=[{LaunchPath=/local/game/myRealtimeLaunchScript.js,
                                      Parameters=loggingLevel:error +map Winter444,
                                      ConcurrentExecutions=10}]"
```

実行中のリアルタイムサーバーフリートのログ記録レベルの設定 (コンソール)

「」の手順に従ってフリート設定を更新する、Amazon GameLift コンソールを使用してフリートを更新し、以下を追加します。

- [フリートの編集] ページの [サーバープロセス割り当て] で、[起動パラメータ] の値としてログ記録レベルのキーと値のペア (`loggingLevel:error` など) を指定します。英数字以外の文字 (カンマ

を除く) を使用して、ログ記録レベルと追加のパラメータを区切ります (loggingLevel:error +map Winter444 など)。

実行中のリアルタイムサーバーフリートのログ記録レベルの設定 (AWS CLI)

フリート設定を更新する に記載されている手順に従って、AWS CLI を使用してフリートを更新します。さらに、以下も行います。

- update-runtime-configuration の --runtime-configuration パラメータの引数に、Parameters の値としてログ記録レベルのキーと値のペア (loggingLevel:error など) を指定します。英数字以外の文字 (カンマを除く) を使用して、ログ記録レベルと追加のパラメータを区切ります。次の例を参照してください。

```
--runtime-configuration "GameSessionActivationTimeoutSeconds=60,
                    MaxConcurrentGameSessionActivations=2,
                    ServerProcesses=[{LaunchPath=/local/game/myRealtimeLaunchScript.js,
                                    Parameters=loggingLevel:error +map Winter444,
                                    ConcurrentExecutions=10}]"
```

Amazon のセキュリティ GameLift

Amazon EC2 のスタンドアロン機能として Amazon GameLift FleetIQ を使用している場合は、Amazon EC2 EC2 ユーザーガイド」のAmazon EC2 のセキュリティ」を参照してください。Amazon EC2

のクラウドセキュリティが最優先事項 AWS です。 AWS のユーザーは、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャを利用できます。

セキュリティは、 AWS とユーザーの間で共有される責任です。責任共有モデルではこれを、クラウドのセキュリティ、およびクラウド内でのセキュリティと説明しています:

- クラウドのセキュリティ — クラウドで AWS サービスを実行するインフラストラクチャを保護する責任 AWS は にあります AWS 。 AWS また、 では、安全に使用できるサービスも提供しています。コンプライアンスAWS プログラムコンプライアンスプログラムコンプライアンスプログラムの一環として、サードパーティーの監査者は定期的にセキュリティの有効性をテストおよび検証します。Amazon に適用されるコンプライアンスプログラムの詳細については GameLift、「コンプライアンスプログラムAWS による対象範囲内の のサービス」、「コンプライアンスプログラム」を参照してください。
- クラウドのセキュリティ — お客様の責任は、使用する AWS サービスによって決まります。また、データの機密性、企業の要件、適用される l AWS および規制など、その他の要因についても責任を負います。

このドキュメントは、Amazon の使用時に責任共有モデルを適用する方法を理解するのに役立ちます GameLift。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成する GameLift ように Amazon を設定する方法について説明します。また、Amazon GameLift リソースのモニタリングや保護に役立つ他の AWS のサービスの使用方法についても説明します。

トピック

- Amazon でのデータ保護 GameLift
- Amazon の Identity and Access Management GameLift
- Amazon GameLift でのログ記録とモニタリング
- Amazon のコンプライアンス検証 GameLift
- Amazon の耐障害性 GameLift

- Amazon のインフラストラクチャセキュリティ GameLift
- Amazon での設定と脆弱性の分析 GameLift
- Amazon のセキュリティのベストプラクティス GameLift

Amazon でのデータ保護 GameLift

Amazon EC2 のスタンドアロン機能として Amazon GameLift FleetIQ を使用している場合は、Amazon EC2 EC2 ユーザーガイド」のAmazon EC2 のセキュリティ」を参照してください。Amazon EC2

責任 AWS 共有モデル、Amazon のデータ保護に適用されます GameLift。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された記事「AWS 責任共有モデルおよび GDPR」を参照してください。

データ保護の目的で、 認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須であり TLS 1.3 がお勧めです。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-2 検証済みの暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報は、タグ、または名前フィールドなどの自由形式のテキストフィールドに配置しないことを強くお勧めします。これは、コンソール、API、

GameLift または SDK を使用して Amazon AWS CLIまたは他の AWS のサービス を操作する場合も同様です。 AWS SDKs 名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

Amazon GameLift固有のデータは次のように処理されます。

- Amazon にアップロードするゲームサーバーのビルドとスクリプト GameLift はAmazon S3に保存されます。このデータがアップロードされると、お客様に直接アクセスすることはできません。許可されたユーザーは、ファイルをアップロードするための一時的なアクセス権を取得できますが、Amazon S3 ディレクトリのファイルを直接表示または更新することはできません。スクリプトとビルドを削除するには、Amazon GameLift コンソールまたはサービス API を使用します。
- ゲームセッションのログデータは、ゲームセッションの完了後に Amazon S3 に一定期間保存されます。許可されたユーザーは、Amazon GameLift コンソールのリンクから、またはサービス API の呼び出しを使用してログデータにアクセスできます。
- メトリクスとイベントデータは Amazon に保存 GameLift され、Amazon GameLift コンソールまたはサービス API の呼び出しからアクセスできます。データは、フリート、インスタンス、ゲームセッションの配置、マッチメーキングチケット、ゲームセッション、およびプレイヤーセッションで取得できます。データには、Amazon CloudWatch および CloudWatch Events からアクセスすることもできます。
- お客様が指定したデータは Amazon に保存されます GameLift 。許可されたユーザーは、サービス API を呼び出してアクセスできます。潜在的に機密性の高いデータには、プレイヤーデータ、プレイヤーセッションとゲームセッションデータ (接続情報を含む)、マッチメーカーデータなどがあります。

> **Note**
>
> リクエストにカスタムプレーヤー ID を指定する場合、これらの値は匿名化された UUID であり、識別されるプレーヤー情報が含まれていないことが想定されます。

データ保護の詳細については、AWS セキュリティブログのブログ投稿「AWS の責任共有モデルと GDPR」を参照してください。

保管中の暗号化

Amazon GameLift固有のデータの保管時の暗号化は、次のように処理されます。

- ゲームサーバーのビルドとスクリプトは、サーバー側の暗号化を使用して Amazon S3 バケットに保存されます。
- お客様が指定したデータは、暗号化 GameLift された形式で Amazon に保存されます。

送信中の暗号化

Amazon GameLift APIs への接続は、安全な (SSL) 接続を介して行われ、AWS 署名バージョン 4 を使用して認証されます (AWS CLI または AWS SDK を介して接続すると、署名は自動的に処理されます) 。認証は、接続の確立に使用されるセキュリティ認証情報の IAM 定義のアクセスポリシーを使用して管理されます。

ゲームクライアントとゲームサーバー間の直接的な通信は、以下の通りです。

- Amazon GameLift リソースでホストされているカスタムゲームサーバーの場合、通信には Amazon GameLift サービスは含まれません。この通信の暗号化は、お客様の責任となります。TLS 対応フリートを使用すると、接続時にゲームクライアントがゲームサーバーを認証したり、ゲームクライアントとゲームサーバー間のすべての通信を暗号化したりできます。
- TLS 証明書生成が有効なリアルタイムサーバーの場合、ゲーム クライアントとリアルタイムクライアント SDK を使用するリアルタイムサーバー間のトラフィックは処理中に暗号化されます。TCP トラフィックは TLS 1.2 を使用して暗号化され、UDP トラフィックは DTLS 1.2 を使用して暗号化されます。

インターネットトラフィックのプライバシー

Amazon GameLift インスタンスに安全にアクセスできます。Linux を使用するインスタンスの場合、SSH はリモートアクセス用の安全な通信チャネルを提供します。Windows を実行しているインスタンスの場合は、リモートデスクトッププロトコル (RDP) クライアントを使用します。Amazon GameLift FleetIQ では、 AWS Systems Manager Session Manager と Run Command を使用したインスタンスへのリモートアクセスは TLS 1.2 を使用して暗号化され、接続の作成リクエストは SigV4 を使用して署名されます。マネージド Amazon GameLift インスタンスへの接続については、「」を参照してくださいAmazon GameLift フリートインスタンスにリモート接続する。

Amazon の Identity and Access Management GameLift

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御するために役立つ AWS のサービス です。IAM 管理者は、誰を認証 (サインイン) し、誰に

Amazon GameLift リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、追加費用なしで使用できる AWS のサービスです。

トピック

対象者

AWS Identity and Access Management (IAM) の使用方法は、Amazon で行う作業によって異なります GameLift。

サービスユーザー – ジョブを実行するために Amazon GameLift サービスを使用する場合は、管理者から必要なアクセス許可と認証情報が与えられます。さらに多くの Amazon GameLift 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。Amazon の機能にアクセスできない場合は GameLift、「」を参照してくださいAmazon GameLift アイデンティティとアクセスのトラブルシューティング。

サービス管理者 – 社内の Amazon GameLift リソースを担当している場合は、通常、Amazon へのフルアクセスがあります GameLift。サービスのユーザーがどの Amazon GameLift 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社で Amazon で IAM を利用する方法の詳細については GameLift、「」を参照してくださいAmazon と IAM の GameLift 連携方法。

IAM 管理者 – IAM 管理者は、Amazon へのアクセスを管理するポリシーの作成方法の詳細について確認する場合があります GameLift。IAM で使用できる Amazon GameLift アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAmazon のアイデンティティベースのポリシーの例 GameLift。

アイデンティティによる認証

認証とは、アイデンティティ認証情報を使用して AWSにサインインする方法です。ユーザーは、AWS アカウントのルートユーザー として、または IAM ロールを引き受けることによって、認証済み (AWS にサインイン済み) である必要があります。

ID ソースから提供された認証情報を使用して、フェデレーティッドアイデンティティとして AWS にサインインできます。 AWS IAM Identity Centerフェデレーティッドアイデンティティの例としては、IAM アイデンティティセンターユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報などがあります。フェデレーティッドアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して AWSにアクセスする場合、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、AWS Management Console または AWS アクセスポータルにサインインできます。AWS へのサインインの詳細については、『AWS サインイン ユーザーガイド』の「AWS アカウント にサインインする方法」を参照してください。

プログラムで AWS にアクセスする場合、AWS は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) を提供し、認証情報でリクエストに暗号で署名します。AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに署名する推奨方法の使用については、『IAM ユーザーガイド』の「AWS API リクエストの署名」を参照してください。

使用する認証方法を問わず、追加のセキュリティ情報の提供が求められる場合もあります。例えば、AWS では多要素認証 (MFA) を使用してアカウントのセキュリティを高めることを推奨しています。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」 および「IAM ユーザーガイド」の「AWS での多要素認証 (MFA) の使用」を参照してください。

AWS アカウントのルートユーザー

AWS アカウント を作成する場合、このアカウントのすべての AWS のサービス とリソースに対して完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。このアイデンティティは AWS アカウントのルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることによってアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報を保護し、それらを使用してルートユーザーのみが実行できるタスクを実行してください。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーティッド ID

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに対し、ID プロバイダーとのフェデレーションを使用して、一時的な認証情報の使用により、AWS のサービスへのアクセスを要求します。

フェデレーテッドアイデンティティ は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリのユーザーか、または ID ソースから提供された認証情報を使用して AWS のサービス にアクセスするユーザーです。フェデレーティッド ID が AWS アカウント にアクセスすると、ロールが継承され、そのロールが一時的な認証情報を提供します。

アクセスを一元管理する場合は、AWS IAM Identity Center を使用することをお勧めします。IAM アイデンティティセンターでユーザーとグループを作成するか、すべての AWS アカウント とアプリケーションで使用するために、独自の ID ソースで一連のユーザーとグループに接続して同期することもできます。IAM アイデンティティセンター の詳細については、「AWS IAM Identity Center ユーザーガイド」の「What is IAM アイデンティティセンター?」(IAM アイデンティティセンター とは) を参照してください。

IAM ユーザーとグループ

IAM ユーザー は、1 人のユーザーまたは 1 つのアプリケーションに対して特定の許可を持つ AWS アカウント 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する権限を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳

細については、「IAM ユーザーガイド」の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロール は、特定の許可を持つ、AWS アカウント 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロールを切り替えることにより、AWS Management Console で一時的に IAM ロールを引き受けることができます。ロールを引き受けるには、AWS CLI または AWS API オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次のような状況で役立ちます。

- フェデレーティッドユーザーアクセス - フェデレーティッドアイデンティティに許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッドアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「サードパーティアイデンティティプロバイダー用のロールの作成」を参照してください。IAM Identity Center を使用する場合、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM アイデンティティセンターは、アクセス許可セットを IAM のロールに関連付けます。権限セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「権限セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースへのアクセスを別のアカウントの人物 (信頼できるプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の AWS のサービスでは、(ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス - 一部の AWS のサービスでは、他の AWS のサービスの機能を使用します。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの権限、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。

- 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービス に権限を委任するロールの作成」を参照してください。
- サービスにリンクされたロール - サービスにリンクされたロールは、AWS のサービス にリンクされたサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスにリンクされたロールは、AWS アカウント に表示され、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの許可を表示できますが、編集はできません。

- Amazon EC2 で実行されているアプリケーション - EC2 インスタンスで実行され、AWS CLI または AWS API 要求を行っているアプリケーションの一時的な認証情報を管理するには、IAM ロールを使用できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスに添付されたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用してアクセス許可を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、「IAM ユーザーガイド」の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」を参照してください。

ポリシーを使用したアクセス権の管理

AWS でアクセスを制御するには、ポリシーを作成して AWS アイデンティティまたはリソースにアタッチします。ポリシーは AWS のオブジェクトであり、アイデンティティやリソースに関連付けて、これらのアクセス許可を定義します。AWS は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシーを評価します。ポリシーでの権

限により、リクエストが許可されるか拒否されるかが決まります。大半のポリシーは JSON ドキュメントとして AWSに保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWSJSON ポリシーを使用して、だれが何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するためのアクセス許可をユーザーに付与するため、IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。このポリシーがあるユーザーは、AWS Management Console、AWS CLI、または AWS API からロール情報を取得できます。

アイデンティティベースポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件を制御します。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースのポリシーは、さらに インラインポリシー または マネージドポリシー に分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれます。管理ポリシーは、AWS アカウント 内の複数のユーザー、グループ、およびロールにアタッチできるスタンドアロンポリシーです。マネージドポリシーには、AWS マネージドポリシーとカスタマー管理ポリシーがあります。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「マネージドポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーの例には、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパル

には、アカウント、ユーザー、ロール、フェデレーションユーザー、または AWS のサービス を含めることができます。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは IAM の AWSマネージドポリシーは使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、AWS WAF、および Amazon VPC は、ACL をサポートするサービスの例です。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS では、他の一般的ではないポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与される最大の許可を設定できます。

- 権限の境界 - 権限の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる許可の上限を設定する高度な機能です。エンティティに権限の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとその権限の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、権限の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。権限の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティの権限の境界」を参照してください。
- サービスコントロールポリシー (SCP) - SCP は、AWS Organizations で組織や組織単位 (OU) の最大許可を指定する JSON ポリシーです。AWS Organizations は、ユーザーのビジネスが所有する複数の AWS アカウント をグループ化し、一元的に管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP はメンバーアカウントのエンティティに対する権限を制限します (各 AWS アカウントのルートユーザー など)。Organizations と SCP の詳細については、AWS Organizations ユーザーガイド の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーティッドユーザーの一時的なセッションをプログラムで作成する際に、パラメータとして渡す高度なポリシーです。結果としてセッションの権限の範囲は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合

もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」をご参照ください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関連するとき、リクエストを許可するかどうかを AWS が決定する方法の詳細については、『IAM ユーザーガイド』の「ポリシーの評価ロジック」を参照してください。

Amazon と IAM の GameLift 連携方法

IAM を使用して Amazon へのアクセスを管理する前に GameLift、Amazon で使用できる IAM 機能について学びます GameLift。

Amazon で使用できる IAM の機能 GameLift

IAM 機能	Amazon GameLift サポート
アイデンティティベースのポリシー	あり
リソースベースのポリシー	いいえ
ポリシーアクション	あり
ポリシーリソース	はい
ポリシー条件キー (サービス固有)	はい
ACL	なし
ABAC (ポリシー内のタグ)	はい
一時的な認証情報	あり
プリンシパル権限	あり
サービスロール	あり
サービスリンクロール	いいえ

Amazon GameLift およびその他の AWSのサービスがほとんどの IAM 機能と連携する方法の概要を把握するには、「IAM ユーザーガイド」のAWS「IAM と連携する のサービス」を参照してください。

Amazon のアイデンティティベースのポリシー GameLift

アイデンティティベースポリシーをサポートする	あり

アイデンティティベースのポリシーは、IAM ユーザー、ユーザーグループ、ロールなどのアイデンティティにアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件を制御します。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それがアタッチされているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素について学ぶには、『IAM ユーザーガイド』の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

Amazon のアイデンティティベースのポリシーの例 GameLift

Amazon GameLift アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAmazon のアイデンティティベースのポリシーの例 GameLift。

Amazon 内のリソースベースのポリシー GameLift

リソースベースのポリシーのサポート	なし

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーの例には、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義

されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーションユーザー、または AWS のサービス を含めることができます。

クロスアカウントアクセスを有効にするには、全体のアカウント、または別のアカウントの IAM エンティティを、リソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる AWS アカウント にある場合、信頼できるアカウントの IAM 管理者は、リソースへのアクセス許可をプリンシパルエンティティ (ユーザーまたはロール) に付与する必要もあります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーを追加する必要はありません。詳細については、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

Amazon のポリシーアクション GameLift

ポリシーアクションに対するサポート	あり

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーの`Action`要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連する AWS API オペレーションと同じです。一致する API オペレーションのない権限のみのアクションなど、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

Amazon GameLift アクションのリストについては、「サービス認証リファレンス」の「Amazon で定義されるアクション GameLift」を参照してください。

Amazon のポリシーアクションは、アクションの前に次のプレフィックス GameLift を使用します。

```
gamelift
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "gamelift:action1",
      "gamelift:action2"
         ]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、Describe という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "gamelift:Describe*"
```

Amazon GameLift アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAmazon のアイデンティティベースのポリシーの例 GameLift。

Amazon のポリシーリソース GameLift

ポリシーリソースに対するサポート	あり

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Resource JSON ポリシーの要素は、オブジェクトあるいはアクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource 要素を含める必要があります。ベストプラクティスとしては、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの権限と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

Amazon GameLift リソースのタイプとその ARNs」の「Amazon で定義されるリソース GameLift」を参照してください。 どのアクションで各リソースの ARN を指定できるかについては、「Amazon で定義されるアクション GameLift」を参照してください。

一部の Amazon GameLift リソースには ARN 値があり、リソースは IAM ポリシーを使用してアクセスを管理できます。Amazon GameLift フリートリソースには、次の構文の ARN があります。

```
arn:${Partition}:gamelift:${Region}:${Account}:fleet/${FleetId}
```

ARN の形式の詳細については、「AWS 全般のリファレンス」の「Amazon リソースネーム (ARN)」を参照してください。

たとえば、ステートメントで `fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa` フリートを指定するには、次の ARN を使用します。

```
"Resource": "arn:aws:gamelift:us-west-2:123456789012:fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
```

特定のアカウントに属するすべてのフリートを指定するには、ワイルドカード (*) を使用します。

```
"Resource": "arn:aws:gamelift:us-west-2:123456789012:fleet/*"
```

Amazon GameLift アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAmazon のアイデンティティベースのポリシーの例 GameLift。

Amazon のポリシー条件キー GameLift

サービス固有のポリシー条件キーのサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

`Condition` 要素 (または `Condition` ブロック) を使用すると、ステートメントが有効な条件を指定できます。`Condition` 要素はオプションです。equal や less than などの条件演算子を使用して条件式を作成することによって、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の `Condition` 要素が指定されている場合、または 1 つの `Condition` 要素に複数のキーが指定されている場合、AWS では `AND` 論理演算子を使用してそれらを評価します。単一の条件キーに複数の値が指定されている場合、AWS では `OR` 論理演算子を使用して条件を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる許可を付与できます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS はグローバル条件キーとサービス固有の条件キーをサポートしています。すべての AWS グローバル条件キーを確認するには、『IAM ユーザーガイド』の「AWS グローバル条件コンテキストキー」を参照してください。

Amazon GameLift の条件キーのリストについては、「サービス認証リファレンス」の「Amazon の条件キー GameLift」を参照してください。条件キーを使用できるアクションとリソースについては、「Amazon で定義されるアクション GameLift」を参照してください。

Amazon GameLift アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAmazon のアイデンティティベースのポリシーの例 GameLift。

Amazon ACLs GameLift

ACL のサポート	なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon での ABAC GameLift

ABAC のサポート (ポリシー内のタグ)	はい

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義するアクセス許可戦略です。AWS では、これらの属性はタグと呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール)、および多数の AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。次に、プリンシパルのタグがアクセスを試行するリソースのタグと一致したときにオペレーションを許可するよう、ABAC ポリシーを設計します。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを制御するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素 でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値は Yes です。サービスが一部のリソースタイプに対してのみ 3 つの条件キーすべてをサポートする場合、値は Partial です。

ABAC の詳細については、『IAM ユーザーガイド』の「ABAC とは?」を参照してください。ABAC をセットアップするステップを説明するチュートリアルについては、「IAM ユーザーガイド」の「属性に基づくアクセスコントロール (ABAC) を使用する」を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するアイデンティティベースのポリシーの例については、「タグに基づいて Amazon GameLift フリートを表示する」を参照してください。

Amazon での一時的な認証情報の使用 GameLift

一時的な認証情報のサポート	あり

AWS のサービス には、一時的な認証情報を使用してサインインしても機能しないものがあります。一時的な認証情報で機能する AWS のサービスなどの詳細については、「IAM ユーザーガイド」の「IAM と連携する AWS のサービス」を参照してください。

ユーザー名とパスワード以外の方法で AWS Management Console にサインインする場合は、一時的な認証情報を使用していることになります。例えば、会社のシングルサインオン (SSO) リンクを使用して AWS にアクセスすると、そのプロセスは自動的に一時的な認証情報を作成します。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ロールへの切り替え (コンソール)」を参照してください。

一時的な認証情報は、AWS CLI または AWS API を使用して手動で作成できます。作成後、一時的な認証情報を使用して AWS にアクセスできるようになります。AWS は、長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成することをお勧めします。詳細については、「IAM の一時的なセキュリティ認証情報」を参照してください。

Amazon のクロスサービスプリンシパル許可 GameLift

フォワードアクセスセッション (FAS) をサポート	はい

IAM ユーザーまたはロールを使用して AWSでアクションを実行するユーザーは、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行してから、別のサービスの別のアクションを開始することがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

Amazon のサービスロール GameLift

サービスロールに対するサポート	あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、『IAM ユーザーガイドの「AWS のサービス に権限を委任するロールの作成」を参照してください。

> **Warning**
>
> サービスロールのアクセス許可を変更すると、Amazon GameLift の機能が破損する可能性があります。Amazon が指示する場合以外 GameLift は、サービスロールを編集しないでください。

Amazon GameLiftがホストするゲームサーバーが、 AWS Lambda関数や Amazon DynamoDB データベースなどの他の AWSリソースにアクセスできるようにします。ゲームサーバーは Amazon が GameLift 管理するフリートでホストされるため、Amazon に他のAWSリソースへの GameLift 制限付きアクセスを許可するサービスロールが必要です。詳細については、「フリートの他の AWS リソースと通信する」を参照してください。

Amazon のサービスにリンクされたロール GameLift

サービスにリンクされたロールのサポート	いいえ

サービスにリンクされたロールは、AWS のサービスにリンクされているサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスにリンクされたロールは、AWS アカウント に表示され、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの許可を表示できますが、編集はできません。

サービスリンクロールの作成または管理の詳細については、「IAM ユーザーガイド」の「IAM と提携する AWS サービス」を参照してください。表の中から、[サービスにリンクされたロール] 列に Yes と記載されたサービスを見つけます。[あり] をクリックすると、該当するサービスにリンクされたロールに関するドキュメントを表示できます。

Amazon のアイデンティティベースのポリシーの例 GameLift

デフォルトでは、ユーザーおよびロールには Amazon GameLift リソースを作成または変更するアクセス許可はありません。また、AWS Management Console、AWS Command Line InterfaceAWS CLI、または AWS API を使用してタスクを実行することもできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者がロールに IAM ポリシーを追加すると、ユーザーはロールを引き受けることができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

各リソースタイプの ARN の形式など GameLift、Amazon で定義されるアクションとリソースタイプの詳細については、「サービス認証リファレンス」の「Amazon のアクション、リソース、および条件キー GameLift」を参照してください。 ARNs

トピック

- ポリシーのベストプラクティス
- Amazon GameLift コンソールの使用
- ユーザーが自分のアクセス許可を表示できるようにする方法
- ゲームセッションへのプレイヤーのアクセスを許可する
- 1 つの Amazon GameLift キューへのアクセスを許可する
- タグに基づいて Amazon GameLift フリートを表示する
- Amazon S3 でゲームビルドファイルにアクセスする

ポリシーのベストプラクティス

ID ベースのポリシーは、ユーザーのアカウントで誰かが Amazon GameLift リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、AWS アカウント に料金が発生する可能性があります。アイデンティティベースのポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください。

- AWS マネージドポリシーを使用して開始し、最小特権の権限に移行する – ユーザーとワークロードへの権限の付与を開始するには、多くの一般的なユースケースのために権限を付与する AWS マネージドポリシーを使用します。これらは AWS アカウントで使用できます。ユースケースに応じた AWS カスタマー管理ポリシーを定義することで、許可をさらに減らすことをお勧めします。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する - IAM ポリシーで許可を設定するときは、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権権限とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、「IAM ユーザーガイド」の「IAM でのポリシーと権限」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。また、AWS のサービス など特定の AWS CloudFormation を介して使用する場合、条件を使用してサービスアクションへのアクセスを許可することもできます。詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素: 条件」を参照してください。
- IAM アクセスアナライザーを使用して IAM ポリシーを検証し、安全で機能的な許可を確保する - IAM アクセスアナライザーは、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM アクセスアナライザーによるポリシーの検証」を参照してください。
- 多要素認証 (MFA) を要求する - AWS アカウント で IAM ユーザーまたはルートユーザーを要求するシナリオがある場合は、セキュリティを強化するために MFA をオンにします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、『IAM ユーザーガイド』の「IAM でのセキュリティのベストプラクティス」を参照してください。

Amazon GameLift コンソールの使用

Amazon GameLift コンソールにアクセスするには、一連の最小限のアクセス許可が必要です。これらのアクセス許可により、 の Amazon GameLift リソースの詳細をリストおよび表示できますAWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

これらのエンティティが引き続き Amazon GameLift コンソールを使用できるようにするには、次の例と の構文を使用してユーザーとグループにアクセス許可を追加します管理者アクセス許可の例。詳細については、「Amazon のユーザーアクセス許可を管理する GameLift」を参照してください。

AWS CLI または AWS API オペレーション GameLift を介して Amazon を操作するユーザーには、最小限のコンソールアクセス許可は必要ありません。代わりに、ユーザーが実行する必要のあるオペレーションのみにアクセスを制限できます。例えば、ゲームクライアントに代わって行動するプレイヤーユーザーは、ゲームセッションをリクエストしたり、プレイヤーをゲームに参加させたり、その他のタスクを実行するためのアクセスを必要とします。

すべての Amazon GameLift コンソール機能を使用するために必要なアクセス許可については、「」の「 管理者のアクセス許可構文」を参照してください管理者アクセス許可の例。

ユーザーが自分のアクセス許可を表示できるようにする方法

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を、IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI か AWS API を使用してプログラム的に、このアクションを完了する権限が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
```

```
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

ゲームセッションへのプレイヤーのアクセスを許可する

プレイヤーをゲームセッションに参加させるには、ゲームクライアントとバックエンドサービスにアクセス許可が必要です。これらのシナリオのポリシー例については、「プレイヤーユーザーアクセス許可の例」を参照してください。

1 つの Amazon GameLift キューへのアクセスを許可する

次の例では、特定の Amazon GameLift キューへのアクセスをユーザーに許可します。

このポリシーは、gamelift:UpdateGameSessionQueue、gamelift:DeleteGameSessionQueue および gamelift:DescribeGameSessionQueues のアクションでキューの送信先を追加、更新、削除するアクセス許可をユーザーに付与します。図に示すように、このポリシーは Resource 要素を使用して、単一のキュー gamesessionqueue/examplequeue123 へのアクセスを制限します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ViewSpecificQueueInfo",
```

```
         "Effect":"Allow",
         "Action":[
            "gamelift:DescribeGameSessionQueues"
         ],
         "Resource":"arn:aws:gamelift:::gamesessionqueue/examplequeue123"
      },
      {
         "Sid":"ManageSpecificQueue",
         "Effect":"Allow",
         "Action":[
            "gamelift:UpdateGameSessionQueue",
            "gamelift:DeleteGameSessionQueue"
         ],
         "Resource":"arn:aws:gamelift:::gamesessionqueue/examplequeue123"
      }
   ]
}
```

タグに基づいて Amazon GameLift フリートを表示する

アイデンティティベースのポリシーの条件を使用して、タグに基づいて Amazon GameLift リソースへのアクセスを制御できます。この例は、Owner タグがユーザーのユーザー名と一致した場合にフリートの表示を許可するポリシーを作成する方法を示しています。このポリシーでは、このオペレーションをコンソールで実行するために必要なアクセス許可も付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListFleetsInConsole",
            "Effect": "Allow",
            "Action": "gamelift:ListFleets",
            "Resource": "*"
        },
        {
            "Sid": "ViewFleetIfOwner",
            "Effect": "Allow",
            "Action": "gamelift:DescribeFleetAttributes",
            "Resource": "arn:aws:gamelift:*:*:fleet/*",
            "Condition": {
                "StringEquals": {"gamelift:ResourceTag/Owner": "${aws:username}"}
            }
        }
```

```
    ]
}
```

Amazon S3 でゲームビルドファイルにアクセスする

ゲームサーバーを Amazon と統合したら GameLift、ビルドファイルを Amazon S3 にアップロードします。Amazon がビルドファイル GameLift にアクセスするには、次のポリシーを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::bucket-name/object-name"
        }
    ]
}
```

Amazon GameLift ゲームファイルのアップロードの詳細については、「」を参照してくださいカスタムゲームサーバー構築を Amazon GameLift にアップロードする。

Amazon GameLift アイデンティティとアクセスのトラブルシューティング

以下の情報は、Amazon と AWS Identity and Access Management (IAM) の使用に伴って発生する可能性がある一般的な問題の診断 GameLift や修復に役立ちます。

トピック

- Amazon でアクションを実行する権限がない GameLift
- iam を実行する権限がありません。PassRole
- 自分の 以外のユーザーに Amazon GameLift リソースAWS アカウントへのアクセスを許可したい

Amazon でアクションを実行する権限がない GameLift

AWS Management Console から、アクションを実行する権限がないと通知された場合は、AWS アカウント管理者に問い合わせてサポートを依頼してください。管理者とは、サインイン認証情報を提供した担当者です。

以下の例のエラーは、mateojackson IAM ユーザーがコンソールを使用してキューの詳細を表示しようとするが、gamelift:DescribeGameSessionQueues アクセス許可がない場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 gamelift:DescribeGameSessionQueues on resource: examplequeue123
```

この場合、Mateo は、gamelift:DescribeGameSessionQueues アクションを使用して examplequeue123 リソースへの読み取りアクセスが許可されるように、管理者にポリシーの更新を依頼します。

iam を実行する権限がありません。PassRole

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Amazon にロールを渡すことができるようにする必要があります GameLift。

一部の AWS のサービスでは、新しいサービスロールやサービスリンクロールを作成せずに、既存のロールをサービスに渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、 という IAM marymajor ユーザーがコンソールを使用して Amazon でアクションを実行しようする場合に発生します GameLift。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新して Mary に iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。管理者とは、サインイン認証情報を提供した担当者です。

自分の 以外のユーザーに Amazon GameLift リソースAWS アカウントへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセス制御リスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Amazon がこれらの機能 GameLift をサポートしているかどうかを確認するには、「」を参照してくださいAmazon と IAM の GameLift 連携方法。
- 所有している AWS アカウント 全体のリソースへのアクセス権を提供する方法については、『IAM ユーザーガイド』の「所有している別の AWS アカウント アカウントへのアクセス権を IAM ユーザーに提供」を参照してください。
- サードパーティーの AWS アカウント にリソースへのアクセス権を提供する方法については、「IAM ユーザーガイド」の「サードパーティーが所有する AWS アカウント にアクセス権を提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、「IAM ユーザーガイド」の「外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

Amazon GameLift でのログ記録とモニタリング

モニタリングは、Amazon GameLift および AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。

AWS と Amazon GameLift は、ゲーム ホスティングリソースをモニタリングし、潜在的なインシデントに対応するためのいくつかのツールを提供します。

Amazon CloudWatch アラーム

Amazon CloudWatch アラームを使用して、指定した期間中、1 つのメトリクスをモニタリングします。メトリクスが特定のしきい値を超えると、Amazon SNS トピックまたはAWS Auto Scaling ポリ

シーに通知が送信されます。CloudWatch アラームは、状態が変化したときにトリガーされ、特定の状態になるのではなく、指定した期間だけ維持されます。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。

AWS CloudTrail ログ

CloudTrail では、Amazon GameLift のユーザー、ロール、または AWS のサービスによって実行されたアクションの記録を確認できます。CloudTrail により収集された情報を使用して、Amazon GameLift に対して行われたリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエストが行われた日時、および追加の詳細を特定することができます。詳細については、「AWS CloudTrail での Amazon GameLift API コールのログ記録」を参照してください。

Amazon のコンプライアンス検証 GameLift

Amazon GameLift は AWS コンプライアンスプログラムの対象ではありません。

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、コンプライアンスプログラムAWS のサービス による対象範囲内のコンプライアンスプログラムを参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「 でのレポートのダウンロード AWS Artifact」の」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 は、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャ — このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべて AWS のサービス HIPAA の対象となるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、ガイダンスを保護し AWS のサービス 、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめています。
- 「 デベロッパーガイド」の「 ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に確認できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- Amazon GuardDuty – これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境を監視することで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。 GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件への対応に役立ちます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

Amazon の耐障害性 GameLift

Amazon EC2 のスタンドアロン機能として Amazon GameLift FleetIQ を使用している場合は、Amazon EC2 EC2 ユーザーガイド」のAmazon EC2 のセキュリティ」を参照してください。 Amazon EC2

AWS グローバルインフラストラクチャは、 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。 AWS リージョンは、低レイテンシー、高スループット、および冗長性の高いネットワークで接続された、物理的に分離された複数のアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

Amazon GameLift では、 AWS グローバルインフラストラクチャに加えて、データの耐障害性のニーズをサポートするために以下の機能を提供しています。

- マルチリージョンキュー – Amazon GameLift ゲームセッションキューは、利用可能なホスティングリソースを持つ新しいゲームセッションを配置するために使用されます。複数のリージョンにまたがるキューは、リージョンが停止した場合にゲームセッションの配置をリダイレクトできます。ゲームセッションキューの作成のベストプラクティスについては、「ゲームセッションキューの設計」を参照してください。
- 自動キャパシティスケーリング — Amazon GameLift スケーリングツールを使用して、ホスティングリソースのヘルスと可用性を維持します。これらのツールには、ゲームやプレイヤーのニーズに合わせてフリートの容量を調整できるさまざまなオプションがあります。スケーリングの詳細については、「Amazon GameLift ホスティングキャパシティのスケーリング」を参照してください。
- インスタンス間の分散 – Amazon GameLift は、フリートのサイズに応じて、受信トラフィックを複数のインスタンスに分散します。ベストプラクティスとして、本稼働環境のゲームでは、インスタンスが異常または応答しなくなった場合に備えて、可用性を維持するために複数のインスタンスを持つ必要があります。
- Amazon S3 ストレージ – Amazon にアップロードされたゲームサーバーのビルドとスクリプト GameLift は、複数のデータセンターレプリケーションを使用して回復性を向上させる標準ストレージクラスを使用して Amazon S3 に保存されます。ゲーム セッション ログは、スタンダードストレージクラスを使用してAmazon S3にも保存されます。

Amazon のインフラストラクチャセキュリティ GameLift

Amazon EC2 のスタンドアロン機能として Amazon GameLift FleetIQ を使用している場合は、Amazon EC2 EC2 ユーザーガイド」のAmazon EC2 のセキュリティ」を参照してください。 Amazon EC2

マネージドサービスである Amazon GameLift は、ホワイトペーパー「Amazon Web Services: セキュリティプロセスの概要」に記載されている AWS グローバルネットワークセキュリティの手順で保護されています。 https://d0.awsstatic.com/whitepapers/Security/AWS_Security_Whitepaper.pdf

が AWS 公開している API コールを使用して、ネットワーク GameLift 経由で Amazon にアクセスします。クライアントは、Transport Layer Security (TLS) 1.2 以降をサポートする必要があります。TLS 1.3 以降が推奨されます。また、一時的ディフィー・ ヘルマン Ephemeral Diffie-Hellman

(DHE) や Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートもクライアントでサポートされている必要があります。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

Amazon GameLift サービスは、すべてのフリートを Amazon Virtual Private Cloud (VPCsに配置し、各フリートが AWS クラウド内の論理的に隔離されたエリアに存在するようにします。Amazon GameLift ポリシーを使用して、特定の VPC エンドポイントまたは特定の VPCsからのアクセスを制御できます。これにより、実質的にネットワーク内の特定の VPC からのみ、特定の Amazon GameLift リソースへの AWS ネットワークアクセスが分離されます。フリートを作成するときは、ポート番号と IP アドレスの範囲を指定します。これらの範囲は、インバウンドトラフィックがフリート VPC 上のホストされたゲームサーバーにアクセスする方法を制限します。フリートアクセス設定を選択するときは、標準セキュリティのベストプラクティスを使用します。

Amazon での設定と脆弱性の分析 GameLift

Amazon EC2 のスタンドアロン機能として Amazon GameLift FleetIQ を使用している場合は、Amazon EC2 EC2 ユーザーガイド」のAmazon EC2 のセキュリティ」を参照してください。Amazon EC2

構成および IT 管理は、 AWS 、お客様および弊社のお客様の間で共有される責任です。詳細については、 AWS 「 責任共有モデル」を参照してください。 は、ゲストオペレーティングシステム (OS) やデータベースのパッチ適用、ファイアウォール設定、ディザスタリカバリなどの基本的なセキュリティタスクを処理します。 AWS これらの手順は適切なサードパーティーによって確認され、認証されています。詳細については、「Amazon Web Services: セキュリティプロセスの概要」(ホワイトペーパー) を参照してください。

次のセキュリティのベストプラクティスは、Amazon の設定と脆弱性の分析にも対処します GameLift。

- お客様は、ゲームホスティングのために Amazon GameLift インスタンスにデプロイされるソフトウェアの管理に責任を負います。具体的には次のとおりです。
  - お客様が用意したゲームサーバーアプリケーションソフトウェアは、更新やセキュリティパッチを含めて管理する必要があります。ゲームサーバーソフトウェアを更新するには、Amazon に新

しいビルドをアップロードし GameLift、新しいフリートを作成して、トラフィックを新しいフリートにリダイレクトします。

- オペレーティングシステムを含む基本の Amazon マシンイメージ (AMI) は、新しいフリートが作成されたときにのみ更新されます。AMI の一部であるオペレーティングシステムやその他のアプリケーションにパッチを適用、更新、保護するには、ゲームサーバーの更新に関係なく、定期的にフリートをリサイクルします。

- お客様は、SDK、Amazon GameLift Server SDK、Amazon GameLift Client AWS SDK for Realtime Servers など、最新の SDK バージョンでゲームを定期的に更新することを検討する必要があります。

Amazon のセキュリティのベストプラクティス GameLift

Amazon EC2 のスタンドアロン機能として Amazon GameLift FleetIQ を使用している場合は、Amazon EC2 EC2 ユーザーガイド」のAmazon EC2 のセキュリティ」を参照してください。Amazon EC2

Amazon GameLift には、独自のセキュリティポリシーを開発および実装する際に考慮すべきさまざまなセキュリティ機能が用意されています。以下のベストプラクティスは一般的なガイドラインであり、完全なセキュリティソリューションを提供するものではありません。これらのベストプラクティスはお客様の環境に必ずしも適切または十分でない可能性があるため、処方箋ではなく、あくまで有用な検討事項とお考えください。

インターネットへのポートを開かないでください。

インターネットへのポートを開くことは、セキュリティ上のリスクとなるため、強くお勧めします。例えば、 UpdateFleetPortSettingsを使用して次のようなリモートデスクトップポートを開くとします。

```
{
  "FleetId": "<fleet identifier>",
  "InboundPermissionAuthorizations": [
      {
        "FromPort": 3389,
        "IpRange": "0.0.0.0/0",
        "Protocol": "RDP",
        "ToPort": 3389
      }
  ]
```

```
}
```

その後、インターネット上のすべてのユーザーがインスタンスにアクセスすることを許可します。

代わりに、特定の IP アドレスまたはアドレス範囲を使用してポートを開きます。例えば、次のようにします。

```
{
  "FleetId": "<fleet identifier>",
  "InboundPermissionAuthorizations": [
      {
        "FromPort": 3389,
        "IpRange": "54.186.139.221/32",
        "Protocol": "TCP",
        "ToPort": 3389
      }
  ]
}
```

詳細はこちら

Amazon をより GameLift 安全に利用する方法の詳細については、AWS Well-Architected Tool 「セキュリティの柱」を参照してください。

Amazon GameLift リファレンスガイド

このセクションには、Amazon GameLift を使用するためのリファレンスドキュメントが含まれます。

トピック

- Amazon GameLift サービス API リファレンス (AWS SDK)
- Amazon GameLift リアルタイムサーバーのリファレンス
- Amazon GameLift サーバー SDK リファレンス
- ゲームセッションプレイスメントイベント

Amazon GameLift サービス API リファレンス (AWS SDK)

このトピックでは、カスタム ゲーム サーバーやリアルタイムサーバーのホスティングなど、Amazon GameLift マネージド ホスティングソリューションで使用する API オペレーションのタスクベースの一覧を示します。これらのオペレーションは、`aws.gamelift` 名前空間の AWS SDK にパッケージ化されています。AWS SDK をダウンロードするか、Amazon GameLift API リファレンスドキュメントをご覧ください。

API には、マネージド ゲーム ホスティング用の 2 つのオペレーションセットが含まれています。

- Amazon GameLift ホスティングリソースをセットアップおよび管理する
- ゲームセッションをスタートし、プレイヤーを参加させる

Amazon GameLift サービス API には、他の Amazon GameLift ツールやソリューションで使用するためのオペレーションも含まれています。FleetIQ API のリストについては、[FleetIQ API actions](FleetIQ API アクション)を参照してください。マッチメイキング用の FlexMatch API のリストについては、FlexMatch API アクションを参照してください。

Amazon GameLift ホスティングリソースをセットアップおよび管理する

以下のオペレーションを呼び出して、ゲームサーバーのホスティングリソースの設定、プレイヤーの需要に合わせた容量のスケール、パフォーマンスと使用率のメトリクスへのアクセスなどを行います。以下の API オペレーションは、Amazon GameLift でホストされているゲームサーバー (リアルタイムサーバーなど) で使用されます。ほとんどのリソース管理タスクに [Amazon GameLift コ

ンソール] を使用できます。または、AWS Command Line Interface (AWS CLI) ツールまたは AWS SDK を使用してそのサービスを呼び出すことができます。

デプロイ用のゲームサーバーを準備する

ホスティングリソースでのデプロイと起動の準備として、ゲームのゲームサーバーコードをアップロードして構成します。

[Manage custom game server builds](カスタム ゲームサーバー構築の管理)

- [upload-build] – ローカルパスからビルドファイルをアップロードし、新しい Amazon GameLift ビルドリソースを作成します。このオペレーションは、AWS CLIコマンドのみが使用可能で、ゲーム サーバー構築をアップロードする最も一般的な方法です。
- [CreateBuild](Create構築) - Amazon S3 バケットに保存されているファイルを使用して新しい構築を作成します。
- [CreateBuild] – Amazon GameLift リージョンにアップロードされたすべてのビルドのリストを取得します。
- [DescribeBuild](Describe構築) - 構築に関連付けられた情報を取得します。
- [UpdateBuild]](Update構築)- 構築名とバージョンを含む構築メタデータを変更します。
- [DeleteBuild] – Amazon GameLift からビルドを削除します。

[Manage Realtime Servers configuration scripts](リアルタイムサーバー構成スクリプトの管理)

- [CreateScript] – JavaScript ファイルをアップロードし、新しい Amazon GameLift スクリプトリソースを作成します。
- [ListScripts] – Amazon GameLift リージョンにアップロードされたすべてのリアルタイムスクリプトのリストを取得します。
- [DescribeScript] （ 説明スクリプト ）- リアルタイムスクリプトに関連付けられた情報を取得します。
- [UpdateScript] （ 更新スクリプト ）（ - スクリプトメタデータを変更し、変更されたスクリプトコンテンツをアップロードします。
- [DeleteScript] – Amazon GameLift からリアルタイムスクリプトを削除します。

ホスティング用のコンピューティング リソースを設定する

ホスティングリソースを設定し、ゲーム サーバー構築またはリアル設定スクリプトとデプロイします。

[Create and manage fleets] (フリートの作成と管理)

- [CreateFleet] – ゲームサーバーを実行するためのコンピューティングリソースの新しい Amazon GameLift フリートを設定してデプロイします。デプロイされると、ゲームサーバーは設定に従って自動的に起動し、ゲームセッションをホストします。
- [ListFleets] – Amazon GameLift リージョン内のすべてのフリートのリストを取得します。
- [DeleteFleet] (削除フィート) - ゲームサーバーの実行やプレーヤーのホストをしなくなったフリートを終了します。
- フリートロケーションの表示/更新。
  - [CreateFleetLocations] (作成フリートロケーション) — 複数のロケーションをサポートする既存のフリートにリモートロケーションを追加する
  - [DescribeFleetLocationAttributes] (ディスクライリブフリートロケーションアトリビュート) — フリートのすべてのリモートロケーションのリストを取得し、各ロケーションの現在のステータスを表示します。
  - [DeleteFleetLocations] (削除フリートロケーション) — 複数のロケーションをサポートするフリートからリモートロケーションを削除します。
- フリート設定の表示/更新。
  - [DescribeFleetAttributes] (作成フリートアトリビュート) /UpdateFleetAttributes - ゲームセッション保護とリソース作成の制限に関するフリートのメタデータと設定を表示または変更します。
  - [DescribeFleetPortSettings] (作成フリートポート設定) /UpdateFleetPortSettings - フリートに許可されているインバウンドアクセス権限 (IP アドレスとポート設定範囲) を表示または変更します。
  - [DescribeRuntimeConfiguration] (作成ランタイム構成) /UpdateRuntimeConfiguration - フリート内の各インスタンスで実行するサーバープロセス (および数) を表示または変更します。

[フリートの容量を管理する

- [DescribeEC2InstanceLimits] (作成EC21インスタンス制限) - AWS アカウントと現在の使用レベルで許可されているインスタンスの最大数を取得します。

- [DescribeFleetCapacity]（作成フリート容量）- フリートのホーム リージョンの現在の容量設定を取得します。
- [DescribeFleetLocationCapacity]（作成フリートロケーション容量）— マルチロケーション フリートの各ロケーションの現在の容量設定を取得します。
- [UpdateFleetCapacity]（更新フリート容量）- フリートの容量設定を手動で調整します。
- Auto Scalingを設定する。
  - [PutScalingPolicy]（プットスケーリングポリシー）- ターゲットベースの Auto Scaling を有効にするか、カスタム Auto Scaling ポリシーを作成します。または、既存のポリシーを更新します。
  - [DescribeScalingPolicies]（作成スケーリングポリシー）- 既存の Auto Scaling ポリシーを取得します。
  - [DeleteScalingPolicy（削除スケーリングポリシー）- Auto Scaling ポリシーを削除し、フリートの容量に影響しないようにします。
  - [StartFleetActions]（スタートフリートアクション）- フリートのAuto Scaling ポリシーを再起動します。
  - [StopFleetActions（停止フリートアクション）- フリートの Auto Scaling ポリシーを停止します。

フリートのアクティビティをモニタリングします。

- [DescribeFleetUtilization]（作成フリート活用法）- フリートで現在アクティブなサーバープロセス、ゲームセッション、プレイヤーの数の統計を取得します。
- [DescribeFleetLocationUtilization]（作成フリートロケーション活用法）— マルチロケーション フリート内の各ロケーションの使用率統計を取得します。
- [DescribeFleetEvents]（作成フリートイベント）- 指定した期間中のフリートの記録されたイベントを表示します。
- [DescribeGameSessions]（作成ゲームセッション）- ゲームの実行時間や現在のプレイヤー数など、ゲームセッションのメタデータを取得します。

ゲームセッションの配置を最適化するためにキューを設定する

コスト、レイテンシー、回復性の点から最適なホスティングリソースにゲームセッションが配置されるようにマルチフリート、マルチリージョンキューを設定します。

- [CreateGameSessionQueue]（作成ゲームセッションキュー）- ゲームセッション配置のリクエストを処理するときに使用するキューを作成します。
- [DescribeGameSessionQueues] – Amazon GameLift リージョン で定義されているゲーム セッションキューを取得します。
- [UpdateGameSessionQueue]（更新ゲームセッションキュー）— ゲーム セッション キューの設定を変更します。
- [DeleteGameSessionQueue]（削除ゲームセッションキュー）- リージョン からゲーム セッションキューを削除します。

エイリアスの管理

エイリアスを使用してフリートを表すか、代わりのターミナル送信先を作成します。エイリアスは、ゲームサーバービルドの更新中など、フリート間でゲームアクティビティを移行するときに役立ちます。

- [CreateAlias]（作成エイリアス）- 新しいエイリアスを定義し、必要に応じてフリートに割り当てます。
- [ListAliases] – Amazon GameLift リージョン で定義されているすべてのフリート エイリアスを取得します。
- [DescribeAlias]（説明エイリアス）- 既存のエイリアスに関する情報を取得します。
- [UpdateAlias]（更新エイリアス）- エイリアスの設定 (フリートから別のフリートへのリダイレクトなど) を変更します。
- [DeleteAlias]（削除エイリアス）- リージョン からエイリアスを削除します。
- [ResolveAlias]（決定エイリアス）- 指定されたエイリアスが指すフリート ID を取得します。

ホスティングインスタンスにアクセスする

フリートの個々のインスタンスに関する情報を表示したり、トラブルシューティングのために指定したフリートインスタンスへのリモートアクセスをリクエストしたりします。

- [DescribeInstances]（説明インスタンス）- ID、IP アドレス、ロケーション、ステータスなど、フリート内の各インスタンスに関する情報を取得します。
- [GetInstanceAccess]（ゲットインスタンスアクセス）- フリート内の指定されたインスタンスにリモートConnectするために必要なアクセス認証情報をリクエストします。

VPC ピアリング接続のセットアップ

Amazon GameLift ホスティングリソースと他の AWS リソース間の VPC ピアリング接続を作成して管理します。

- [CreateVpcPeeringAuthorization]（VpcPeering認可の作成）- いずれかの VPC へのピアリング接続を許可します。
- [DescribeVpcPeeringAuthorizations]（作成VpcPeering認可の説明）- 有効なピアリング接続の承認を取得します。
- [DeleteVpcPeeringAuthorization]（VpcPeering認可の削除）- ピアリング接続の認可を削除します。
- [CreateVpcPeeringConnection] – Amazon GameLift フリートの VPC といずれかの VPC の間にピアリング接続を確立します。
- [DescribeVpcPeeringConnections] – Amazon GameLift フリートとのアクティブまたは保留中の VPC ピアリング接続に関する情報を取得します。
- [DeleteVpcPeeringConnection] – Amazon GameLift フリートとの VPC ピアリング接続を削除します。

ゲームセッションをスタートし、プレイヤーを参加させる

ゲームクライアントサービスからこれらのオペレーションを呼び出して、新しいゲームセッションをスタートし、既存のゲームセッションに関する情報を取得し、プレイヤーをゲームセッションに参加させます。以下のオペレーションは、Amazon GameLift でホストされているカスタムゲームサーバーを使用するゲーム用です。Realtime Server を使用している場合は、リアルタイムサーバー API (C#) リファレンスを使用しゲームセッションを管理します。

- 1 人以上のプレイヤーの新しいゲームセッションを開始します。
  - [StartGameSessionPlacement] – Amazon GameLift を使用して、利用可能な最適なホスティングリソースをみつけ、新しいゲームセッションをスタートします。これは、推奨する新しいゲームセッションの作成方法です。複数のリージョンでホスティングの可用性を追跡するためにゲームセッション キューに依存し、FleetIQ アルゴリズムを使用して、プレイヤーのレイテンシー、ホスティング コスト、ロケーションなどに基づいて配置の優先順位付けを行います。
  - [DescribeGameSessionPlacement]（スタートゲームセッションプレイスメント）- 配置リクエストの詳細とステータスを取得します。
  - [StopGameSessionPlacement]（停止ゲームセッションプレイスメント）- 配置リクエストをキャンセルします。

  - [CreateGameSession] - （ゲームセッションの作成）特定のフリートロケーションで新しい空のゲームセッションをスタートします。このオペレーションにより、FleetIQ を使用して配置オプションを評価する代わりに、ゲームセッションをスタートする場所を詳細に制御できます。別のステップで新しいゲームセッションにプレイヤーを追加する必要があります。
- [プレイヤーを既存のゲームに参加させます。] 使用可能なプレイヤースロットがある実行中のゲームセッションを検索し、新しいプレイヤー用に予約します。
  - [CreatePlayerSession]（プレイヤーセッションの作成）- プレイヤーがゲームセッションに参加できるように空きスロットを予約します。
  - [CreatePlayerSessions]（プレイヤーセッションの作成）- 複数のプレイヤーがゲームセッションに参加できるように空きスロットを予約します。
- ゲームセッションとプレイヤーセッションデータを操作します。ゲームセッションとプレイヤーセッションの情報を管理できます。
  - [SearchGameSessions]（ゲームセッションの探求）— 一連の検索条件に基づいて、アクティブなゲームセッションのリストを要求します。
  - [[DescribeGameSessions]（ゲームセッションの説明）- アクティブ時間の長さや現在のプレイヤー数など、ゲームセッションのメタデータを取得します。
  - [DescribeGameSessionDetails]（ゲームセッション詳細の説明）- 1 つ以上のゲームセッションのゲームセッション保護設定を含むメタデータを取得します。
  - [DescribePlayerSessions]（プレイヤーセッションの説明）- ステータス、プレイ時間、プレイヤーデータなど、プレイヤーアクティビティの詳細を取得します。
  - [UpdateGameSession]（ゲームセッションの更新）- 最大プレイヤー数や参加ポリシーなどのゲームセッション設定を変更します。
  - [GetGameSessionLogUrl]（ゲームセッションログURLの取得）- ゲームセッションのログが保存された場所を取得します。

Amazon GameLift リアルタイムサーバーのリファレンス

このセクションには、Amazon GameLift リアルタイムサーバー SDK のリファレンスドキュメントが含まれます。また、リアルタイム クライアント API と、リアルタイムサーバースクリプトを設定するためのガイダンスも含まれます。

トピック

- リアルタイムサーバー API (C#) リファレンス
- Amazon GameLift リアルタイムサーバースクリプトリファレンス

リアルタイムサーバー API (C#) リファレンス

リアルタイムクライアント API を使用して、Amazon GameLift リアルタイムサーバーで使用するマルチプレイヤーゲームクライアントを準備します。統合プロセスの詳細については、「リアルタイムサーバーを準備する」を参照してください。クライアント API には、ゲームクライアントがリアルタイムサーバーに接続し、サーバーを介して他のゲームクライアントとメッセージやデータを交換できるようにする同期 API 呼び出しと非同期コールバックのセットが含まれています。

この API は以下のライブラリで定義されています。

Client.cs

- 非同期アクション
- 非同期コールバック
- データ型

リアルタイムクライアント API をセットアップするには

1. Amazon GameLift リアルタイムクライアント SDK のダウンロード。
2. C# SDK ライブラリを構築します。ソリューションファイル `GameLiftRealtimeClientSdkNet45.sln` を探します。C# Server SDK の最小要件と追加の構築オプションについては、`README.md` ファイルを参照してください。IDE で、ソリューションファイルをロードします。SDK ライブラリを生成するには、NuGet パッケージを復元し、ソリューションを構築します。
3. リアルタイムクライアントライブラリをゲームクライアントプロジェクトに追加します。

リアルタイムサーバークライアント API (C#) リファレンス: アクション

この C# リアルタイムクライアント API リファレンスは、Amazon GameLift フリートにデプロイされたリアルタイムサーバーで使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「リアルタイムサーバーを準備する」を参照してください。

- 非同期アクション
- 非同期コールバック
- データ型

Client()

Realtime サーバーと通信するように新しいクライアントを初期化し、使用する接続のタイプを識別します。

構文

```
public Client(ClientConfiguration configuration)
```

パラメータ

clientConfiguration

クライアント/サーバー接続タイプを指定する設定の詳細。このパラメータを指定しないで Client() を呼び出すことを選択できます。ただしこのアプローチでは、デフォルトでセキュリティ保護なしの接続になります。

タイプ: ClientConfiguration

必須: いいえ

戻り値

Realtime サーバーとの通信に使用するリアルタイムクライアントのインスタンスを返します。

Connect()

ゲームセッションをホストしているサーバープロセスへの接続をリクエストします。

構文

```
public ConnectionStatus Connect(string endpoint, int remoteTcpPort, int listenPort,
 ConnectionToken token)
```

パラメータ

エンドポイント

接続するゲームセッションの DNS 名または IP アドレス。エンドポイントは GameSession オブジェクトで指定されます。このオブジェクトは AWSSDK Amazon GameLift API アクション StartGameSessionPlacement、CreateGameSession、または DescribeGameSessions のクライアント呼び出しに応じて返されます。

> **Note**
>
> TLS 証明書の生成が有効になっているフリートで Realtime サーバーが実行されている場合は、DNS 名を使用する必要があります。

型: 文字列

必須: はい

remoteTcpPort

ゲームセッションに割り当てられている TCP 接続のポート番号。この情報は、`GameSession` オブジェクトで指定されます。これは、StartGameSessionPlacement、CreateGameSession、または DescribeGameSession リクエストに応答して返されます。

型: 整数

有効な値: 1900 ~ 2000。

必須: はい

listenPort

ゲームクライアントが UDP チャンネルを使用して送信されたメッセージをリッスンしているポート番号。

型: 整数

有効な値: 33400 ~ 33500。

必須: はい

トークン

サーバープロセスにリクエストしているゲームクライアントを識別するオプションの情報。

タイプ: ConnectionToken

必須: はい

戻り値

クライアントの接続ステータスを示す ConnectionStatus 列挙値を返します。

Disconnect()

ゲームセッションに接続すると、ゲームクライアントをゲームセッションから切断します。

構文

```
public void Disconnect()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

このメソッドは何も返しません。

NewMessage()

指定されたオペレーションコードで新しいメッセージオブジェクトを作成します。メッセージオブジェクトが返されたら、ターゲットを指定し、配信方法を更新します。必要に応じてデータペイロードを追加して、メッセージの内容を完成させます。完了したら、SendMessage() を使用してメッセージを送信します。

構文

```
public RTMessage NewMessage(int opCode)
```

パラメータ

opCode

プレイヤーの移動やサーバーの通知など、ゲームのイベントやアクションを識別する開発者定義のオペレーションコード。

型: 整数

必須: はい

戻り値

指定されたオペレーションコードとデフォルトの配信方法を含む RTMessage オブジェクトを返します。配信インテントパラメータはデフォルトで FAST に設定されています。

SendMessage()

指定された配信方法を使用して、プレイヤーまたはグループにメッセージを送信します。

構文

```
public void SendMessage(RTMessage message)
```

パラメータ

message

ターゲット受信者、配信方法、およびメッセージの内容を指定するメッセージオブジェクト。

タイプ: RTMessage

必須: はい

戻り値

このメソッドは何も返しません。

JoinGroup()

指定されたグループのメンバーシップにプレイヤーを追加します。グループには、ゲームに接続しているすべてのプレイヤーを含めることができます。参加すると、プレイヤーはグループに送信された今後のメッセージをすべて受信し、グループ全体にメッセージを送信できます。

構文

```
public void JoinGroup(int targetGroup)
```

パラメータ

targetGroup

プレイヤーを追加するグループを識別する一意の ID。グループ ID は開発者定義です。

型: 整数

必須: はい

戻り値

このメソッドは何も返しません。このリクエストは信頼できる (TCP) 配信方法を使用して送信されるため、失敗したリクエストはコールバックをトリガーしますOnError()。

LeaveGroup()

指定されたグループのメンバーシップからプレイヤーを削除します。グループに参加しなくなったプレイヤーは、グループに送信されたメッセージを受信せず、グループ全体にメッセージを送信することもできません。

構文

```
public void LeaveGroup(int targetGroup)
```

パラメータ

targetGroup

プレイヤーを削除するグループを識別する一意の ID。グループ ID は開発者定義です。

型: 整数

必須: はい

戻り値

このメソッドは何も返しません。このリクエストは信頼できる (TCP) 配信方法を使用して送信されるため、失敗したリクエストはコールバックをトリガーしますOnError()。

RequestGroupMembership()

指定されたグループのプレイヤーのリストをゲームクライアントに送信するようにリクエストします。グループのメンバーであるかどうかにかかわらず、どのプレイヤーでもこの情報をリクエストできます。このリクエストに応答して、メンバーシップリストは OnGroupMembershipUpdated() コールバックを介してクライアントに送信されます。

構文

```
public void RequestGroupMembership(int targetGroup)
```

パラメータ

targetGroup

メンバーシップ情報を取得するグループを識別する一意の ID。グループ ID は開発者定義です。

型: 整数

必須: はい

戻り値

このメソッドは何も返しません。

リアルタイムサーバークライアント API (C#) リファレンス: 非同期コールバック

この C# リアルタイムクライアント API リファレンスを使用して、Amazon GameLift フリートにデプロイされているリアルタイムサーバーで使用するためのマルチプレイヤーゲームを準備してください。統合プロセスの詳細については、「リアルタイムサーバーを準備する」を参照してください。

- 非同期アクション
- 非同期コールバック
- データ型

ゲームクライアントは、イベントに応答するためにこれらのコールバックメソッドを実装する必要があります。リアルタイムサーバーはこれらのコールバックを呼び出して、ゲーム関連の情報をゲームクライアントに送信します。同じイベントのコールバックは、リアルタイムサーバースクリプトのカスタムゲームロジックでも実装できます。「リアルタイムサーバーのスクリプトコールバック」を参照してください。

コールバックメソッドは `ClientEvents.cs` で定義されています。

OnOpen()

サーバープロセスがゲームクライアントの接続リクエストを受け入れて接続を開くと呼び出されます。

構文

```
public void OnOpen()
```

パラメータ

このメソッドにはパラメータはありません。

戻り値

このメソッドは何も返しません。

OnClose()

ゲームセッションの終了後など、サーバープロセスがゲームクライアントとの接続を終了したときに呼び出されます。

構文

```
public void OnClose()
```

パラメータ

このメソッドにはパラメータはありません。

戻り値

このメソッドは何も返しません。

OnError()

リアルタイムクライアント API リクエストに障害が発生したときに呼び出されます。このコールバックは、さまざまな接続エラーを処理するようにカスタマイズできます。

構文

```
private void OnError(byte[] args)
```

パラメータ

このメソッドにはパラメータはありません。

戻り値

このメソッドは何も返しません。

OnDataReceived()

ゲームクライアントがリアルタイムサーバーからメッセージを受信したときに呼び出されます。これは、メッセージと通知がゲームクライアントによって受信される主な方法です。

構文

```
public void OnDataReceived(DataReceivedEventArgs dataReceivedEventArgs)
```

パラメータ

dataReceivedEventArgs

メッセージのアクティビティに関連する情報。

タイプ: DataReceivedEventArgs

必須: はい

戻り値

このメソッドは何も返しません。

OnGroupMembershipUpdated()

プレイヤーが属するグループのメンバーシップが更新されたときに呼び出されます。このコールバックは、クライアントが `RequestGroupMembership` を呼び出すときにも呼び出されます。

構文

```
public void OnGroupMembershipUpdated(GroupMembershipEventArgs groupMembershipEventArgs)
```

パラメータ

groupMembershipEventArgs

グループメンバーシップアクティビティに関連する情報。

タイプ: GroupMembershipEventArgs

必須: はい

戻り値

このメソッドは何も返しません。

リアルタイムサーバークライアント API (C#) リファレンス:データ型

この C# リアルタイムクライアント API リファレンスは、Amazon GameLift フリートにデプロイされたリアルタイムサーバーで使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「リアルタイムサーバーを準備する」を参照してください。

- 非同期アクション
- 非同期コールバック
- データ型

ClientConfiguration

ゲームクライアントがリアルタイムサーバーに接続する方法に関する情報。

目次

ConnectionType

使用するクライアント/サーバー接続のタイプ (セキュリティ保護ありまたはなし)。接続タイプを指定しない場合、デフォルトではセキュリティ保護なしになります。

> **Note**
>
> TLS 証明書の生成が有効になっているセキュリティ保護ありのフリート上の Realtime サーバーに接続する場合は、値 RT_OVER_WSS_DTLS_TLS12 を使用する必要があります。

型: `ConnectionType` 列挙値。

必須: いいえ

ConnectionToken

リアルタイムサーバーとの接続をリクエストしているゲームクライアントまたはプレイヤーに関する情報。

目次

playerSessionId

新しいプレイヤーセッションが作成されたときに Amazon GameLift によって発行される一意の ID。プレイヤーセッション ID は、`PlayerSession` オブジェクトです。これは、クライアントのGameLift API アクションのStartGameSessionPlacement、CreateGameSession、DescribeGameSessionPlacement、またはDescribePlayerSessions の呼び出しに応答して返されます。

型: 文字列

必須: はい

payload

接続時にリアルタイムサーバーに伝達されるデベロッパー定義の情報。これには、カスタムサインインメカニズムに使用できる任意のデータが含まれます。たとえば、ペイロードは、クライアントに接続を許可する前に、リアルタイムサーバースクリプトによって処理される認証情報を提供します。

タイプ: バイト配列

必須: いいえ

RTMessage

メッセージの内容と配信情報。メッセージはターゲットプレイヤーかターゲットグループのどちらかを指定する必要があります。

目次

opCode

プレイヤーの移動やサーバーの通知など、ゲームのイベントやアクションを識別する開発者定義のオペレーションコード。メッセージのオペコードは、提供されているデータペイロードのコンテキストを提供します。NewMessage() を使用して作成されたメッセージにはすでにオペレーションコードが設定されていますが、いつでも変更できます。

型: 整数

必須: はい

targetPlayer

送信されているメッセージの目的の受信者であるプレイヤーを識別する一意の ID。ターゲットは、サーバー自体 (サーバー ID を使用) または別のプレイヤー (プレイヤー ID を使用) のいずれかです。

型: 整数

必須: いいえ

targetGroup

送信されているメッセージの目的の受信者であるグループを識別する一意の ID。グループ ID は開発者定義です。

型: 整数

必須: いいえ

deliveryIntent

信頼性のある TCP 接続を使用してメッセージを送信するか、または高速 UDP チャンネルを使用してメッセージを送信するかを示します。NewMessage() を使用して作成したメッセージ。

タイプ: DeliveryIntent enum

有効な値: FAST | RELIABLE

必須: はい

payload

メッセージの内容。この情報は、必要に応じて付随するオペレーションコードに基づいてゲームクライアントによって処理されるように構成されています。ゲームクライアント間またはゲームクライアントとリアルタイムサーバーとの間で通信される必要があるゲームの状態データまたは他の情報が含まれています。

タイプ: バイト配列

必須: いいえ

DataReceivedEventArgs

OnDataReceived() コールバックで提供されるデータ

目次

送信者

メッセージを発信したエンティティ (プレイヤー ID またはサーバー ID) を識別する一意の ID。

型: 整数

必須: はい

opCode

プレイヤーの移動やサーバーの通知など、ゲームのイベントやアクションを識別する開発者定義のオペレーションコード。メッセージのオペコードは、提供されているデータペイロードのコンテキストを提供します。

型: 整数

必須: はい

データ

メッセージの内容。この情報は、必要に応じて付随するオペレーションコードに基づいてゲームクライアントによって処理されるように構成されています。ゲームクライアント間またはゲームクライアントとリアルタイムサーバーとの間で通信される必要があるゲームの状態データまたは他の情報が含まれています。

タイプ: バイト配列

必須: いいえ

GroupMembershipEventArgs

OnGroupMembershipUpdated() コールバックで提供されるデータ

目次

送信者

グループメンバーシップの更新をリクエストしたプレイヤーを識別する一意の ID。

型: 整数

必須: はい

opCode

ゲームのイベントまたはアクションを識別する開発者定義のオペレーションコード。

型: 整数

必須: はい

groupId

送信されているメッセージの目的の受信者であるグループを識別する一意の ID。グループ ID は開発者定義です。

型: 整数

必須: はい

playerId

指定されたグループの現在のメンバーであるプレイヤー ID のリスト。

タイプ: 整数配列

必須: はい

列挙型

リアルタイムクライアント SDK に定義された列挙値は次のように定義されます。

ConnectionStatus

- CONNECTED - ゲームクライアントは TCP 接続のみでリアルタイムサーバーに接続されています。配信目的に関係なく、すべてのメッセージは TCP 経由で送信されます。
- CONNECTED_SEND_FAST - ゲームクライアントは、TCP および UDP 接続でリアルタイムサーバーに接続されています。ただし、UDP を介してメッセージを受信する機能はまだ検証されていません。その結果、ゲームクライアントに送信されるすべてのメッセージは TCP を使用します。
- CONNECTED_SEND_AND_RECEIVE_FAST - ゲームクライアントは、TCP および UDP 接続でリアルタイムサーバーに接続されています。ゲームクライアントは、TCP または UDP を使用してメッセージを送受信できます。

- CONNECTING ゲームクライアントが接続リクエストを送信し、リアルタイムサーバーがそれを処理しています。
- DISCONNECTED_CLIENT_CALL - ゲームクライアントからの Disconnect() リクエストに応答して、ゲームクライアントがリアルタイムサーバーから切断されました。
- DISCONNECTED - ゲームクライアントが切断呼び出し以外の理由で、クライアントがリアルタイムサーバーから切断されました。

ConnectionType

- RT_OVER_WSS_DTLS_TLS12 - セキュリティ保護ありの接続タイプ。

  TLS 証明書の生成が有効になっている GameLift フリートで実行されている Realtime サーバーで使用します。セキュリティ保護ありの接続を使用する場合、TCP トラフィックは TLS 1.2 を使用して暗号化され、UDP トラフィックは DTLS 1.2 を使用して暗号化されます。
- RT_OVER_WS_UDP_UNSECURED - セキュリティ保護なしの接続タイプ。
- RT_OVER_WEBSOCKET - セキュリティ保護なしの接続タイプ。この値は推奨されなくなりました。

DeliveryIntent

- FAST - UDP チャンネルを使用して配信されました。
- RELIABLE - TCP 接続を使用して配信されました。

Amazon GameLift リアルタイムサーバースクリプトリファレンス

以下のリソースを使用して、リアルタイムスクリプトのカスタムロジックを構築します。

トピック

リアルタイムサーバーのスクリプトコールバック

リアルタイムスクリプトにこれらのコールバックを実装することで、イベントに応答するためのカスタムロジックを提供できます。

初期化

リアルタイムサーバーを初期化し、リアルタイムサーバーインターフェイスを受け取ります。

構文

```
init(rtsession)
```

onMessage

受信したメッセージがサーバーに送信されたときに呼び出されます。

構文

```
onMessage(gameMessage)
```

onHealthCheck

ゲームセッションの状態を設定するために呼び出されます。デフォルトでは、ヘルスステータスは正常 (true) です。このコールバックは、カスタムヘルスチェックを実行してステータスを返すために実装できます。

構文

```
onHealthCheck()
```

onStartGameSession

新しいゲームセッションが開始されてゲームセッションオブジェクトを渡されると呼び出されます。

構文

```
onStartGameSession(session)
```

onProcessTerminate

サーバープロセスが Amazon GameLift サービスによって終了中になると呼び出されます。これは、ゲームセッションから正常に終了するためのトリガーとして機能します。processEnding(). を呼び出す必要はありません

構文

```
onProcessTerminate()
```

onPlayerConnect

プレイヤーが接続をリクエストし、初期検証に合格したときに呼び出されます。

構文

```
onPlayerConnect(connectMessage)
```

onPlayerAccepted

プレイヤーの接続が受け入れられると呼び出されます。

構文

```
onPlayerAccepted(player)
```

onPlayerDisconnect

プレイヤーが切断リクエストを送信するか、または他の方法でゲームセッションから切断したときに呼び出されます。

構文

```
onPlayerDisconnect(peerId)
```

onProcessStarted

サーバープロセスの起動時に呼び出されます。このコールバックにより、スクリプトはゲームセッションをホストするための準備に必要なカスタムタスクを実行できます。

構文

```
onProcessStarted(args)
```

onSendToPlayer

あるプレイヤーから別のプレイヤーに配信されるメッセージがサーバー上で受信されたときに呼び出されます。このプロセスはメッセージが配信される前に実行されます。

構文

```
onSendToPlayer(gameMessage)
```

onSendToGroup

グループに配信されるメッセージが 1 人のプレイヤーからサーバー上で受信されたときに呼び出されます。このプロセスはメッセージが配信される前に実行されます。

構文

```
onSendToGroup(gameMessage))
```

onPlayerJoinGroup

プレイヤーがグループに参加するためのリクエストを送信したときに呼び出されます。

構文

```
onPlayerJoinGroup(groupId, peerId)
```

onPlayerLeaveGroup

プレイヤーがグループから脱退するためのリクエストを送信したときに呼び出されます。

構文

```
onPlayerLeaveGroup(groupId, peerId)
```

リアルタイムサーバーインターフェイス

リアルタイムスクリプトが初期化を実行すると、リアルタイムサーバーへのインターフェイスが返されます。このトピックでは、インターフェイスを通じて使用できるプロパティとメソッドについて説明します。リアルタイムスクリプトの書き込み方法と詳細なスクリプト例につついては「リアルタイムスクリプトの作成」を参照してください。

リアルタイムインターフェイスは、以下のオブジェクトへのアクセスを提供します。

- セッション
- player
- gameMessage
- 設定

リアルタイムセッションオブジェクト

以下のメソッドを使用して、サーバー関連の情報にアクセスし、サーバー関連のアクションを実行します。

getPlayers()

現在ゲームセッションに接続しているプレイヤーのピア ID のリストを取得します。プレイヤーオブジェクトの配列を返します。

構文

```
rtSession.getPlayers()
```

broadcastGroupMembershipUpdate()

更新されたグループメンバーシップリストのプレイヤーグループへの配信をトリガーします。ブロードキャストするメンバーシップ (groupIdToBroadcast) と更新を受け取るグループ (targetGroupId) を指定します。グループ ID は、すべてのグループを示す正の整数または「-1」でなければなりません。ユーザー定義のグループ ID の例は「リアルタイムサーバースクリプト例」を参照してください。

構文

```
rtSession.broadcastGroupMembershipUpdate(groupIdToBroadcast, targetGroupId)
```

getServerId()

メッセージをサーバーにルーティングするために使用される、サーバーの一意のピア ID 識別子を取得します。

構文

```
rtSession.getServerId()
```

getAllPlayersGroupId()

現在ゲームセッションに接続しているすべてのプレイヤーを含むデフォルトグループのグループ ID を取得します。

構文

```
rtSession.getAllPlayersGroupId()
```

processEnding()

ゲームサーバーを終了するためにリアルタイムサーバーをトリガーします。この関数は、ゲームセッションから正常に終了するためのリアルタイムスクリプトから呼び出す必要があります。

構文

```
rtSession.processEnding()
```

getGameSessionId()

現在実行中のゲームセッションの一意の ID を取得します。

構文

```
rtSession.getGameSessionId()
```

getLogger()

ログ記録用のインターフェイスを取得します。この関数を使用して、ゲームセッションログに収集されるステートメントを記録します。ロガーは「info」、「warn」、および「error」ステートメントの使用をサポートしています。例: `logger.info("<string>")`。

構文

```
rtSession.getLogger()
```

sendMessage()

`newTextGameMessage` または `newBinaryGameMessage` を使用して作成されたメッセージをリアルタイムサーバーからプレイヤー受信者に UDP チャネルで送信します。プレイヤーのピア ID を使用して受取人を識別します。

構文

```
rtSession.sendMessage(gameMessage, targetPlayer)
```

sendGroupMessage()

newTextGameMessage または newBinaryGameMessage を使用して作成されたメッセージをリアルタイムサーバーからプレイヤーグループ内のすべてのプレイヤーに UDP チャネルで送信します。グループ ID は、すべてのグループを示す正の整数または「-1」でなければなりません。ユーザー定義のグループ ID の例は「リアルタイムサーバースクリプト例」を参照してください。

構文

```
rtSession.sendGroupMessage(gameMessage, targetGroup)
```

sendReliableMessage()

newTextGameMessage または newBinaryGameMessage を使用して作成されたメッセージをリアルタイムサーバーからプレイヤー受信者に TCP チャネルで送信します。プレイヤーのピア ID を使用して受取人を識別します。

構文

```
rtSession.sendReliableMessage(gameMessage, targetPlayer)
```

sendReliableGroupMessage()

newTextGameMessage または newBinaryGameMessage を使用して作成されたメッセージをリアルタイムサーバーからプレイヤーグループ内のすべてのプレイヤーに TCP チャネルで送信します。グループ ID は、すべてのグループを示す正の整数または「-1」でなければなりません。ユーザー定義のグループ ID の例は「リアルタイムサーバースクリプト例」を参照してください。

構文

```
rtSession.sendReliableGroupMessage(gameMessage, targetGroup)
```

newTextGameMessage()

SendMessage 関数を使用してサーバーからプレイヤー受取人に送信される、テキストを含む新しいメッセージを作成します。メッセージ形式は、Realtime Client SDK で使用されている形式と同様です (「RTMessage」を参照)。gameMessage オブジェクトを返します。

構文

```
rtSession.newTextGameMessage(opcode, sender, payload)
```

newBinaryGameMessage()

SendMessage 関数を使用してサーバーからプレイヤー受取人に送信される、バイナリデータを含む新しいメッセージを作成します。メッセージ形式は、Realtime Client SDK で使用されている形式と同様です (「RTMessage」を参照)。gameMessage オブジェクトを返します。

構文

```
rtSession.newBinaryGameMessage(opcode, sender, binaryPayload)
```

プレイヤーオブジェクト

プレイヤー関連の情報にアクセスします。

player.peerId

ゲームクライアントがリアルタイムサーバーに接続してゲームセッションに参加したときに割り当てられる一意の ID。

player.playerSessionId

ゲームクライアントがリアルタイムサーバーに接続してゲームセッションに参加したときに参照したプレイヤーセッション ID。

ゲームメッセージオブジェクト

以下のメソッドを使用して、リアルタイムサーバーによって受信されたメッセージにアクセスします。ゲームクライアントから受信したメッセージは RTMessage 構造になっています。

getPayloadAsText()

ゲームメッセージのペイロードをテキストとして取得します。

構文

```
gameMessage.getPayloadAsText()
```

gameMessage.opcode

メッセージに含まれているオペレーションコード。

gameMessage.payload

メッセージに含まれているペイロード。テキストまたはバイナリです。

gameMessage.sender

メッセージを送信したゲームクライアントのピア ID。

gameMessage.reliable

メッセージが TCP (true) または UDP (false) のどちらで送信されたかを示すブール値。

設定オブジェクト

設定オブジェクトは、デフォルトの設定を上書きするために使用できます。

configuration.maxPlayers

リアルタイムサーバーで受け入れられるクライアント/サーバー接続の最大数。

デフォルトは 32 です。

configuration.pingIntervalTime

接続が正常であることを確認するために、サーバーが接続されているすべてのクライアントに ping を送信しようとする時間間隔 (ミリ秒) 。

デフォルトは 3000 ミリ秒です。

Amazon GameLift サーバー SDK リファレンス

このセクションには、Amazon GameLift サーバー SDK のリファレンスドキュメントが含まれます。Server SDK を使用してカスタムゲームサーバーを統合して Amazon GameLift と通信します。

トピック

- C++ 用 Amazon GameLift サーバー SDK 5 リファレンス
- C# 用 Amazon GameLift サーバー SDK リファレンス
- Go 用 Amazon GameLift サーバー SDK リファレンス
- Unreal Engine 用 Amazon GameLift サーバー SDK リファレンス

C++ 用 Amazon GameLift サーバー SDK 5 リファレンス

この Amazon GameLift C++ サーバー SDK リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

トピック

- C++ 用 Amazon GameLift サーバー SDK 5.x リファレンス
- Amazon GameLift C++ サーバー SDK 3.x リファレンス

C++ 用 Amazon GameLift サーバー SDK 5.x リファレンス

この Amazon GameLift C++ サーバー SDK 5.x リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

> **Note**
>
> このトピックでは、C++ 標準ライブラリ (std) を使用して構築するときに使用できる Amazon GameLift C++ API について説明します。特に、このドキュメントは `-DGAMELIFT_USE_STD=1` オプションを使用してコンパイルするコードが対象です。

トピック

- Amazon GameLift サーバー SDK (C++) 5.x リファレンス: アクション
- Amazon GameLift サーバー SDK (C++) リファレンス: データ型

Amazon GameLift サーバー SDK (C++) 5.x リファレンス: アクション

この Amazon GameLift C++ サーバー SDK リファレンスを使用すると、Amazon で使用するマルチプレイヤーゲームを準備するのに役立ちます GameLift。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

> **Note**
>
> このトピックでは、 GameLift C++ 標準ライブラリ () で構築するときに使用できる Amazon C++ API について説明しますstd。特に、このドキュメントは `-DGAMELIFT_USE_STD=1` オプションを使用してコンパイルするコードが対象です。

アクション

- GetSdkVersion()
- InitSDK()

- InitSDK()
- ProcessReady()
- ProcessReadyAsync()
- ProcessEnding()
- ActivateGameSession()
- UpdatePlayerSessionCreationPolicy()
- GetGameSessionId()
- GetTerminationTime()
- AcceptPlayerSession()
- RemovePlayerSession()
- DescribePlayerSessions()
- StartMatchBackfill()
- StopMatchBackfill()
- GetComputeCertificate()
- GetFleetRoleCredentials()
- Destroy()

GetSdkVersion()

サーバープロセスに組み込まれた SDK の現在のバージョン番号を返します。

Syntax

```
Aws::GameLift::AwsStringOutcome Server::GetSdkVersion();
```

戻り値

成功した場合、the section called “AwsStringOutcome” オブジェクトとして現在の SDK バージョンを返します。返されるオブジェクトには、バージョン番号が含まれます (例: `5.0.0`)。成功しなかった場合、エラーメッセージを返します。

例

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
 Aws::GameLift::Server::GetSdkVersion();
```

InitSDK()

マネージド EC2 フリートの Amazon GameLift SDK を初期化します。Amazon に関連する他の初期化 GameLift が発生する前に、起動時にこのメソッドを呼び出します。このメソッドは、ホスト環境からサーバーパラメータを読み取り、サーバーと Amazon GameLift サービス間の通信を設定します。

Syntax

```
Server::InitSDKOutcome Server::initSdkOutcome = InitSDK();
```

戻り値

サーバープロセスが ProcessReady() を呼び出す準備ができているかどうかを示す the section called “InitSDKOutcome” オブジェクトを返します。

例

```
//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
Aws::GameLift::Server::InitSDKOutcome initSdkOutcome =
  Aws::GameLift::Server::InitSDK();
```

InitSDK()

Anywhere フリートの Amazon GameLift SDK を初期化します。Amazon に関連する他の初期化 GameLift が発生する前に、起動時にこのメソッドを呼び出します。この方法では、サーバーと Amazon GameLift サービス間の通信を設定するには、明示的なサーバーパラメータが必要です。

Syntax

```
Server::InitSDKOutcome Server::initSdkOutcome = InitSDK(serverParameters);
```

パラメータ

ServerParameters

Amazon GameLift Anywhere フリートでゲームサーバーを初期化するには、次の情報を使用して `ServerParameters` オブジェクトを作成します。

- ゲームサーバーへの接続 WebSocket に使用される の URL。
- ゲームサーバーのホストに使用されるプロセスの ID。
- ゲームサーバープロセスをホスティングするコンピューティングの ID。
- Amazon コンピューティングを含む Amazon GameLift Anywhere GameLift フリートの ID。
- Amazon GameLift オペレーションによって生成された認証トークン。

戻り値

サーバープロセスが ProcessReady() を呼び出す準備ができているかどうかを示す the section called "InitSDKOutcome" オブジェクトを返します。

> **Note**
>
> Anywhere フリートにデプロイされたゲームビルドに対して InitSDK() への呼び出しが失敗する場合は、ビルドリソースの作成時に使用した ServerSdkVersion パラメータを確認してください。この値は、使用中のサーバー SDK バージョンに明示的に設定する必要があります。このパラメータのデフォルト値は 4.x で、互換性がありません。この問題を解決するには、新しいビルドを作成して新しいフリートにデプロイします。

例

Amazon GameLift Anywhere の例

```
//Define the server parameters
std::string websocketUrl = "wss://us-west-1.api.amazongamelift.com";
std::string processId = "PID1234";
std::string fleetId = "arn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa";
std::string hostId = "HardwareAnywhere";
std::string authToken = "1111aaaa-22bb-33cc-44dd-5555eeee66ff";
Aws::GameLift::Server::Model::ServerParameters serverParameters =
  Aws::GameLift::Server::Model::ServerParameters(webSocketUrl, authToken, fleetId,
 hostId, processId);

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
Aws::GameLift::Server::InitSDKOutcome initSdkOutcome =
 Aws::GameLift::Server::InitSDK(serverParameters);
```

ProcessReady()

サーバープロセス GameLift がゲームセッションをホストする準備ができたことを Amazon に通知します。InitSDK() を呼び出した後にこのメソッドを呼び出します このメソッドは、プロセスごとに 1 回だけ呼び出す必要があります。

Syntax

```
GenericOutcome ProcessReady(const Aws::GameLift::Server::ProcessParameters
&processParameters);
```

パラメータ

processParameters

サーバープロセスに関する以下の情報を伝える ProcessParameters オブジェクト。

- Amazon GameLift サービスがサーバープロセスと通信するために呼び出すゲームサーバーコードに実装されているコールバックメソッドの名前。
- サーバープロセスがリッスンするポートの番号。
- Amazon がキャプチャして保存するゲームセッション固有のファイル GameLift へのパス。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、ProcessReady() 呼び出しと委任関数の実装の両方を示します。

```
// Set parameters and call ProcessReady
std::string serverLog("serverOut.log");        // Example of a log file written by the
 game server
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);
int listenPort = 9339;

Aws::GameLift::Server::ProcessParameters processReadyParameter =
  Aws::GameLift::Server::ProcessParameters(
    std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
    std::bind(&Server::onProcessTerminate, this),
    std::bind(&Server::OnHealthCheck, this),
```

```
    std::bind(&Server::OnUpdateGameSession, this),
    listenPort,
    Aws::GameLift::Server::LogParameters(logPaths)
  );

Aws::GameLift::GenericOutcome outcome =
  Aws::GameLift::Server::ProcessReady(processReadyParameter);

// Implement callback functions
void Server::onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
  // game-specific tasks when starting a new game session, such as loading map
  GenericOutcome outcome =
    Aws::GameLift::Server::ActivateGameSession (maxPlayers);
}

void Server::onProcessTerminate()
{
  // game-specific tasks required to gracefully shut down a game session,
  // such as notifying players, preserving game state data, and other cleanup
  GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
}

bool Server::onHealthCheck()
{
  bool health;
  // complete health evaluation within 60 seconds and set health
  return health;
}
```

ProcessReadyAsync()

サーバープロセスがゲームセッションをホストする準備ができたことを Amazon GameLift サービスに通知します。サーバープロセスがゲームセッションをホストする準備ができたら、このメソッドを呼び出します。パラメータは、特定の状況で呼び GameLift 出す Amazon のコールバック関数名を指定します。ゲームサーバーコードは、これらの関数を実装する必要があります。

この呼び出しは非同期です。同期呼び出しを実行するには、ProcessReady() を使用します。詳細については、「サーバープロセスを初期化する」を参照してください。

Syntax

```
GenericOutcomeCallable ProcessReadyAsync(
```

```
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

パラメータ

processParameters

サーバープロセスに関する以下の情報を伝える ProcessParameters オブジェクト。

- Amazon GameLift サービスがサーバープロセスと通信するために呼び出すゲームサーバーコードに実装されているコールバックメソッドの名前。
- サーバープロセスがリッスンするポートの番号。
- Amazon がキャプチャして保存するゲームセッション固有のファイル GameLift へのパス。

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
// Set parameters and call ProcessReady
std::string serverLog("serverOut.log");        // This is an example of a log file
 written by the game server
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);
int listenPort = 9339;

Aws::GameLift::Server::ProcessParameters processReadyParameter =
  Aws::GameLift::Server::ProcessParameters(std::bind(&Server::onStartGameSession, this,
 std::placeholders::_1),
  std::bind(&Server::onProcessTerminate, this), std::bind(&Server::OnHealthCheck,
 this),
  std::bind(&Server::OnUpdateGameSession, this), listenPort,
 Aws::GameLift::Server::LogParameters(logPaths));

Aws::GameLift::GenericOutcomeCallable outcome =
  Aws::GameLift::Server::ProcessReadyAsync(processReadyParameter);

// Implement callback functions
void onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
```

```
  // game-specific tasks when starting a new game session, such as loading map
  GenericOutcome outcome = Aws::GameLift::Server::ActivateGameSession (maxPlayers);
}

void onProcessTerminate()
{
  // game-specific tasks required to gracefully shut down a game session,
  // such as notifying players, preserving game state data, and other cleanup
  GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
}

bool onHealthCheck()
{
  // perform health evaluation and complete within 60 seconds
  return health;
}
```

ProcessEnding()

サーバープロセスが終了中であることを Amazon に通知 GameLift します。他のすべてのクリーンアップタスク (アクティブなゲームセッションのシャットダウンを含む) の後、およびプロセスを終了する前に、このメソッドを呼び出します。ProcessEnding( ) の結果に応じて、プロセスは成功 (0) またはエラー (-1) で終了し、フリートイベントを生成します。プロセスがエラーで終了した場合、生成されるフリートイベントは ですSERVER_PROCESS_TERMINATED_UNHEALTHY。

Syntax

```
Aws::GameLift::GenericOutcome processEndingOutcome =
 Aws::GameLift::Server::ProcessEnding();
```

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、サーバープロセスを終了する前に、成功またはエラーの終了コードで ProcessEnding( ) と Destroy( ) を呼び出します。

```
Aws::GameLift::GenericOutcome processEndingOutcome =
 Aws::GameLift::Server::ProcessEnding();
Aws::GameLift::Server::Destroy();
```

```
// Exit the process with success or failure
if (processEndingOutcome.IsSuccess()) {
  exit(0);
}
else {
  cout << "ProcessEnding() failed. Error: " <<
 processEndingOutcome.GetError().GetErrorMessage();
  exit(-1);
}
```

ActivateGameSession()

サーバープロセスがゲームセッションをアクティブ化し、プレイヤー接続を受信する準備ができた GameLift ことを Amazon に通知します。このアクションは、すべてのゲームセッションの初期化の後、onStartGameSession() コールバック関数の一部として呼び出されます。

Syntax

```
Aws::GameLift::GenericOutcome activateGameSessionOutcome =
 Aws::GameLift::Server::ActivateGameSession();
```

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、onStartGameSession() 委任関数の一部として呼び出された ActivateGameSession() を示しています。

```
void onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
  // game-specific tasks when starting a new game session, such as loading map
  GenericOutcome outcome = Aws::GameLift::Server::ActivateGameSession();
}
```

UpdatePlayerSessionCreationPolicy()

現在のゲームセッションの機能を更新し、新しいプレイヤーセッションを承諾します。ゲームセッションは、新しいプレイヤーセッションをすべて受け入れるか拒否するかを設定できます。

Syntax

```
GenericOutcome
 UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCreationPolicy
 newPlayerSessionPolicy);
```

パラメータ

playerCreationSessionポリシー

型: `PlayerSessionCreationPolicy` 列挙値。

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例は、現在のゲームセッションの参加ポリシーを、すべてのプレイヤーを受け入れるように設定します。

```
Aws::GameLift::GenericOutcome outcome =

 Aws::GameLift::Server::UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCr
```

GetGameSessionId()

アクティブなサーバープロセスにホストされたゲームセッションの ID を取得します。

ゲームセッションでアクティブ化されていないアイドル状態のプロセスの場合、呼び出しは the section called “GameLiftError” を返します。

Syntax

```
AwsStringOutcome GetGameSessionId()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合、ゲームセッション ID を the section called “AwsStringOutcome” オブジェクトとして返します。成功しなかった場合、エラーメッセージを返します。

ゲームセッションでまだアクティブ化されていないアイドルプロセスの場合、呼び出しは Success=True および GameSessionId="" を返します。

例

```
Aws::GameLift::AwsStringOutcome sessionIdOutcome =
 Aws::GameLift::Server::GetGameSessionId();
```

GetTerminationTime()

終了時刻が判る場合に、サーバープロセスがシャットダウンを予定している時刻を返します。サーバープロセスは、Amazon から onProcessTerminate()コールバックを受信した後にアクションを実行します GameLift。Amazon onProcessTerminate() は、次の理由で を GameLift 呼び出します。

- サーバープロセスが正常性の低下を報告した場合、または Amazon に応答しなかった場合 GameLift。
- スケールダウンイベント中にインスタンスを終了する場合。
- スポットインスタンスの中断によりインスタンスが終了した場合。

Syntax

```
AwsDateTimeOutcome GetTerminationTime()
```

戻り値

成功した場合、終了時刻を AwsDateTimeOutcome オブジェクトとして返します。値は終了時間で、0001 00:00:00 以降の経過ティックで表現されます。例えば、日付時刻の値 2020-09-13 12:26:40 -000Z は、637355968000000000 ティックに等しくなります。終了時間がない場合は、エラーメッセージを返します。

プロセスが ProcessParameters.OnProcessTerminate() コールバックを受信していない場合は、エラーメッセージが返されます。サーバープロセスのシャットダウンの詳細については、「サーバープロセスのシャットダウン通知に応答する」を参照してください。

例

```
Aws::GameLift::AwsLongOutcome TermTimeOutcome =
 Aws::GameLift::Server::GetTerminationTime();
```

AcceptPlayerSession()

指定されたプレイヤーセッション ID を持つプレイヤーがサーバープロセスに接続し、検証が必要である GameLift ことを Amazon に通知します。Amazon は、プレイヤーセッション ID が有効 GameLift であることを確認します。プレイヤーセッションが検証されると、Amazon はプレイヤースロットのステータスを RESERVED から ACTIVE GameLift に変更します。

Syntax

```
GenericOutcome AcceptPlayerSession(String playerSessionId)
```

パラメータ

playerSessionId

新しいプレイヤーセッションの作成 GameLift 時に Amazon によって発行される一意の ID。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、無効なプレイヤーセッション ID の検証と拒否を含む接続リクエストを処理します。

```
void ReceiveConnectingPlayerSessionID (Connection& connection, const std::string&
 playerSessionId)
{
  Aws::GameLift::GenericOutcome connectOutcome =
 Aws::GameLift::Server::AcceptPlayerSession(playerSessionId);
  if(connectOutcome.IsSuccess())
  {
    connectionToSessionMap.emplace(connection, playerSessionId);
    connection.Accept();
  }
```

```
  else
  {
    connection.Reject(connectOutcome.GetError().GetMessage();
  }
}
```

RemovePlayerSession()

プレイヤーがサーバープロセスから切断された GameLift ことを Amazon に通知します。それに応じて、Amazon GameLift はプレイヤースロットを利用可能に変更します。

Syntax

```
GenericOutcome RemovePlayerSession(String playerSessionId)
```

パラメータ

playerSessionId

新しいプレイヤーセッションの作成 GameLift 時に Amazon によって発行される一意の ID。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
Aws::GameLift::GenericOutcome disconnectOutcome =
 Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

DescribePlayerSessions()

設定、セッションメタデータ、プレイヤーデータを含む、プレイヤーセッションデータを取得します。このメソッドを使用して、以下に関する情報を取得します。

- シングルプレイヤーセッション
- ゲームセッションのすべてのプレイヤーセッション
- 1 つのプレイヤー ID に関連付けられているすべてのプレイヤーセッション

Syntax

```
DescribePlayerSessionsOutcome DescribePlayerSessions(DescribePlayerSessionsRequest
 describePlayerSessionsRequest)
```

パラメータ

DescribePlayerSessionsRequest

取得するプレイヤーセッションを記述する the section called “DescribePlayerSessionsRequest” オブジェクト。

戻り値

成功した場合は、リクエストのパラメータに適合したプレイヤーセッションオブジェクトのセットを含む the section called “DescribePlayerSessionsOutcome” オブジェクトを返します。

例

この例は、指定したゲームセッションにアクティブに接続されているすべてのプレイヤーセッションのリクエストします。を省略NextTokenし、Limit 値を 10 に設定することで、Amazon はリクエストに一致する最初の 10 個のプレイヤーセッションレコード GameLift を返します。

```
// Set request parameters
Aws::GameLift::Server::Model::DescribePlayerSessionsRequest request;
request.SetPlayerSessionStatusFilter(Aws::GameLift::Server::Model::PlayerSessionStatusMapper::G
request.SetLimit(10);
request.SetGameSessionId("the game session ID");    // can use GetGameSessionId()

// Call DescribePlayerSessions
Aws::GameLift::DescribePlayerSessionsOutcome playerSessionsOutcome =
 Aws::GameLift::Server::DescribePlayerSessions(request);
```

StartMatchBackfill()

FlexMatch で作成されたゲームセッションの空きスロット用に新規プレイヤーを検索するリクエストを送信します。詳細については、FlexMatch 「バックフィル機能」を参照してください。

このアクションは非同期です。新しいプレイヤーがマッチングされると、Amazon はコールバック関数 を使用して更新されたマッチメーカーデータを GameLift 配信しますOnUpdateGameSession()。

サーバープロセスではアクティブなマッチバックフィルリクエストは一度に 1 つだけです。新しいリクエストを送信するには、まず StopMatchBackfill() を呼び出して元のリクエストをキャンセルする必要があります。

Syntax

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest
 startBackfillRequest);
```

パラメータ

StartMatchBackfillRequest

次の情報を伝える StartMatchBackfillRequest オブジェクト。

- バックフィルリクエストに割り当てるチケット ID。この情報はオプションです。ID が指定されていない場合、Amazon GameLift は ID を生成します。
- リクエストを送信するマッチメーカー。完全な設定 ARN が必要です。この値はゲームセッションのマッチメーカーデータに含まれています。
- バックフィルするゲームセッションの ID。
- ゲームセッションの現在のプレイヤーに利用可能なマッチメーキングデータ。

戻り値

the section called “StartMatchBackfillOutcome” オブジェクトを、マッチバックフィルチケット ID またはエラーメッセージを伴うエラーとともに返します。

例

```
// Build a backfill request
std::vector<Player> players;
Aws::GameLift::Server::Model::StartMatchBackfillRequest startBackfillRequest;
startBackfillRequest.SetTicketId("1111aaaa-22bb-33cc-44dd-5555eeee66ff");  // optional,
 autogenerated if not provided
startBackfillRequest.SetMatchmakingConfigurationArn("arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig"); //from the game
 session matchmaker data
startBackfillRequest.SetGameSessionArn("the game session ARN");        // can use
 GetGameSessionId()
startBackfillRequest.SetPlayers(players);                                  // from the
 game session matchmaker data
```

```
// Send backfill request
Aws::GameLift::StartMatchBackfillOutcome backfillOutcome =
  Aws::GameLift::Server::StartMatchBackfill(startBackfillRequest);

// Implement callback function for backfill
void Server::OnUpdateGameSession(Aws::GameLift::Server::Model::GameSession gameSession,
 Aws::GameLift::Server::Model::UpdateReason updateReason, std::string backfillTicketId)
{
  // handle status messages
  // perform game-specific tasks to prep for newly matched players
}
```

StopMatchBackfill()

アクティブなマッチバックフィルリクエストをキャンセルします。詳細については、FlexMatch「バックフィル機能」を参照してください。

Syntax

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

パラメータ

StopMatchBackfillRequest

キャンセルするマッチメーキングチケットを識別する StopMatchBackfillRequest オブジェクト：

- バックフィルリクエストに割り当てるチケット ID。
- バックフィルリクエストが送信されたマッチメーカー。
- バックフィルリクエストに関連付けられたゲームセッション。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
// Set backfill stop request parameters

Aws::GameLift::Server::Model::StopMatchBackfillRequest stopBackfillRequest;
stopBackfillRequest.SetTicketId("1111aaaa-22bb-33cc-44dd-5555eeee66ff");
```

```
stopBackfillRequest.SetGameSessionArn("the game session ARN"); // can use
 GetGameSessionId()
stopBackfillRequest.SetMatchmakingConfigurationArn("arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig");
// from the game session matchmaker data

Aws::GameLift::GenericOutcome stopBackfillOutcome =
  Aws::GameLift::Server::StopMatchBackfill(stopBackfillRequest);
```

GetComputeCertificate()

Amazon コンピューティングリソースと Amazon GameLift Anywhere 間のネットワーク接続の暗号化に使用される TLS 証明書へのパスを取得します GameLift。コンピューティングデバイスを Amazon GameLift Anywhere フリートに登録するときに、証明書パスを使用できます。詳細については、「」を参照してくださいRegisterCompute。

Syntax

```
GetComputeCertificateOutcome Server::GetComputeCertificate()
```

戻り値

戻り値は the section called “GetComputeCertificateOutcome”。

例

```
Aws::GameLift::GetComputeCertificateOutcome certificate =
 Aws::GameLift::Server::GetComputeCertificate();
```

GetFleetRoleCredentials()

Amazon が他の とやり取り GameLift することを許可する IAM ロール認証情報を取得しますAWS のサービス。詳細については、「フリートの他の AWS リソースと通信する」を参照してください。

構文

```
GetFleetRoleCredentialsOutcome GetFleetRoleCredentials(GetFleetRoleCredentialsRequest
 request);
```

パラメータ

GetFleetRoleCredentialsRequest

戻り値

the section called “GetFleetRoleCredentialsOutcome” オブジェクトを返します。

例

```
// form the fleet credentials request
Aws::GameLift::Server::Model::GetFleetRoleCredentialsRequest
 getFleetRoleCredentialsRequest;
getFleetRoleCredentialsRequest.SetRoleArn("arn:aws:iam::123456789012:role/service-role/
exampleGameLiftAction");

Aws::GameLift::GetFleetRoleCredentialsOutcome credentials =
 Aws::GameLift::Server::GetFleetRoleCredentials(getFleetRoleCredentialsRequest);
```

この例は、監査目的でオプションの RoleSessionName 値を使用して認証情報セッションに名前を割り当てる方法を示しています。ロールセッション名を指定しない場合、デフォルト値「*[fleet-id]-[host-id]*」が使用されます。

```
// form the fleet credentials request
Aws::GameLift::Server::Model::GetFleetRoleCredentialsRequest
 getFleetRoleCredentialsRequest;
getFleetRoleCredentialsRequest.SetRoleArn("arn:aws:iam::123456789012:role/service-role/
exampleGameLiftAction");
getFleetRoleCredentialsRequest.SetRoleSessionName("MyFleetRoleSession");

Aws::GameLift::GetFleetRoleCredentialsOutcome credentials =
 Aws::GameLift::Server::GetFleetRoleCredentials(getFleetRoleCredentialsRequest);
```

Destroy()

Amazon GameLift ゲームサーバー SDK をメモリから解放します。ベストプラクティスとして、ProcessEnding() の後、かつプロセスの終了前にこのメソッドを呼び出します。Anywhere フリートを使用していて、すべてのゲームセッション後にサーバープロセスを終了しない場合は、InitSDK()を呼び出しDestroy()てから再初期化してから、プロセスが でゲームセッションをホストする準備ができ GameLift たことを Amazon に通知しますProcessReady()。

Syntax

```
GenericOutcome Aws::GameLift::Server::Destroy();
```

パラメータ

パラメータはありません。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
Aws::GameLift::GenericOutcome processEndingOutcome =
 Aws::GameLift::Server::ProcessEnding();
Aws::GameLift::Server::Destroy();

// Exit the process with success or failure
if (processEndingOutcome.IsSuccess()) {
  exit(0);
}
else {
  cout << "ProcessEnding() failed. Error: " <<
 processEndingOutcome.GetError().GetErrorMessage();
  exit(-1);
}
```

Amazon GameLift サーバー SDK (C++) リファレンス: データ型

この Amazon GameLift C++ サーバー SDK リファレンスを使用すると、Amazon で使用するマルチプレイヤーゲームを準備するのに役立ちます GameLift。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

> **Note**
>
> このトピックでは、 GameLift C++ 標準ライブラリ () で構築するときに使用できる Amazon C++ API について説明しますstd。特に、このドキュメントは -DDGAMELIFT_USE_STD=1 オプションを使用してコンパイルするコードが対象です。

データ型

- LogParameters
- ProcessParameters

- UpdateGameSession
- GameSession
- ServerParameters
- StartMatchBackfillRequest
- プレイヤー
- DescribePlayerSessionsRequest
- StopMatchBackfillRequest
- AttributeValue
- GetFleetRoleCredentialsRequest
- AwsLongOutcome
- AwsStringOutcome
- DescribePlayerSessionsOutcome
- DescribePlayerSessionsResult
- GenericOutcome
- GenericOutcomeCallable
- PlayerSession
- StartMatchBackfillOutcome
- StartMatchBackfillResult
- GetComputeCertificateOutcome
- GetComputeCertificateResult
- GetFleetRoleCredentialsOutcome
- GetFleetRoleCredentialsResult
- InitSDKOutcome
- GameLiftError
- 列挙型

LogParameters

ゲームセッション中に生成されたファイルを識別 GameLift し、ゲームセッション
の終了後に Amazon がアップロードして保存するオブジェクト。ゲームサーバー
はLogParameters、ProcessReady()呼び出しのProcessParametersオブジェクト GameLift の一
部として Amazon に を提供します。

プロパティ	説明
LogPaths	Amazon が将来のアクセスのために GameLift 保存するゲームサーバーログファイルへのディレクトリパスのリスト。サーバープロセスは各ゲームセッション中にこれらのファイルを生成します。ファイルのパスと名前はゲームサーバーで定義し、ルートゲームビルドディレクトリに保存します。 ログパスは絶対パスである必要があります。例えば、ゲームビルドによって `MyGame\sessionLogs\` などのパスに保存されるゲームセッションログの場合、パスは `c:\game\MyGame\sessionLogs` (Windows インスタンスの場合) となります。 タイプ: `std:vector<std::string>` 必須: いいえ

ProcessParameters

このデータ型には、 GameLift で Amazon に送信されるパラメータのセットが含まれますProcessReady()。

プロパティ	説明
LogParameters	ゲームセッション中に生成されるファイルへのディレクトリパスを含むオブジェクト。Amazon は、将来のアクセスに備えてファイル GameLift をコピーして保存します。 タイプ: `Aws::GameLift::Server::` LogParameters 必須: いいえ

OnHealthCheck	サーバープロセスにヘルスステータスレポートをリクエストするために Amazon が GameLift 呼び出すコールバック関数。Amazon はこの関数を 60 秒ごとに GameLift 呼び出し、応答を 60 秒待機します。サーバープロセスは正常であれば TRUE を返し、正常でない場合は FALSE を返します。レスポンスが返されない場合、Amazon はサーバープロセスを正常でないものとして GameLift 記録します。 タイプ: `std::function<bool()> onHealthCheck` 必須: いいえ
OnProcessTerminate	サーバープロセスを強制シャットダウンするために Amazon が GameLift 呼び出すコールバック関数。この関数を呼び出した後、Amazon はサーバープロセスがシャットダウンするまで 5 分 GameLift 待ってからProcessEnding()、サーバープロセスをシャットダウンします。 タイプ: `std::function<void()> onProcessTerminate` 必須: はい
OnRefreshConnection	ゲームサーバーとの接続を更新するために Amazon が GameLift 呼び出すコールバック関数の名前。 タイプ: `void OnRefreshConnectionDelegate()` 必須: はい

OnStartGameSession	新しいゲームセッションをアクティブ化するために Amazon が GameLift 呼び出すコールバック関数。Amazon は、クライアントリクエスト に応答してこの関数を GameLift 呼び出しますCreateGameSession。コールバック関数は、Amazon GameLift API リファレンス で定義されているように GameSession オブジェクトを渡します。 タイプ: `const std::function<void (Aws::GameLift::Model::GameSession)> onStartGameSession` 必須: はい
OnUpdateGameSession	更新されたゲームセッションオブジェクトをサーバープロセスに渡すために Amazon が GameLift 呼び出すコールバック関数。Amazon は、マッチバックフィルリクエストが更新されたマッチメーカーデータを提供するために処理されたときに、この関数を GameLift 呼び出します。GameSession オブジェクト、ステータス更新 (updateReason) 、マッチバックフィルチケット ID を渡します。 タイプ: `std::function<void(Aws::GameLift::Server::Model::UpdateGameSession)> onUpdateGameSession` 必須: いいえ

ポート	サーバープロセスが新しいプレイヤーの接続をリスンするポート番号。値は、このゲームサーバービルドをデプロイするすべてのフリートで設定されているポート番号の範囲に含まれる必要があります。このポート番号は、ゲームセッションオブジェクトとプレイヤーセッションオブジェクトに含まれ、ゲームセッションがサーバープロセスに接続するときに使用します。 タイプ: `Integer` 必須: はい

UpdateGameSession

このデータ型はゲームセッションオブジェクトに更新されます。これには、ゲームセッションが更新された理由と、バックフィルを使用してゲームセッション内のプレイヤーセッションを埋めるための関連するバックフィルチケット ID が含まれます。

プロパティ	説明
GameSession	Amazon GameLift API で定義されるGameSessionオブジェクト。`GameSession` オブジェクトにはゲームセッションを説明するプロパティが含まれています。 タイプ: `Aws::GameLift::Server::GameSession` 必須: はい
UpdateReason	ゲームセッションが更新されている理由。 タイプ: `Aws::GameLift::Server::UpdateReason` 必須: はい

プロパティ	説明
BackfillTicketId	ゲームセッションの更新を試みるバックフィルチケットの ID。 タイプ: `std::string` 必須: いいえ

GameSession

このデータ型はゲームセッションの詳細を提供します。

プロパティ	説明
GameSessionId	ゲームセッションの一意の識別子。ゲームセッション ARN の形式 は `arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token>` です。 タイプ: `std::string` 必須: いいえ
名前	ゲームセッションについて説明するラベル。 タイプ: `std::string` 必須: いいえ
FleetId	ゲームセッションが実行されているフリートの一意の識別子。 タイプ: `std::string` 必須: いいえ
MaximumPlayerSessionCount	ゲームセッションへのプレーヤー接続の最大数。

プロパティ	説明
	タイプ: `int` 必須: いいえ
ポート	ゲームセッションのポート番号。Amazon GameLift ゲームサーバーに接続するには、アプリに IP アドレスとポート番号の両方が必要です。 タイプ: `in` 必須: いいえ
IpAddress	ゲームセッションの IP アドレス。Amazon GameLift ゲームサーバーに接続するには、アプリに IP アドレスとポート番号の両方が必要です。 タイプ: `std::string` 必須: いいえ
GameSessionData	単一の文字列値としてフォーマットされたカスタムゲームセッションプロパティのセット。 タイプ: `std::string` 必須: いいえ

プロパティ	説明
MatchmakerData	ゲームセッションの作成に使用されたマッチメーキングプロセスに関する情報。JSON 構文で、文字列としてフォーマットされています。使用されたマッチメーキング設定に加えて、プレイヤー属性やチーム割り当てなど、マッチに割り当てられた全プレイヤーに関するデータが含まれます。 タイプ: `std::string` 必須: いいえ
GameProperties	ゲームセッションのカスタムプロパティのセットで、キーと値のペアとしてフォーマットされます。これらのプロパティは、新しいゲームセッションを開始するリクエストとともに渡されます。 タイプ: `std :: vector < GameProperty >` 必須: いいえ

プロパティ	説明
DnsName	ゲームセッションを実行しているインスタンスに割り当てられた DNS 識別子。値の形式は次のとおりです。 • TLS 対応フリート: `<unique identifier>.<region identifier>.amazongamelift.com` 。 • TLS 対応でないフリート: `ec2-<unique identifier>.compute.amazonaws.com` 。 TLS 対応フリートで実行しているゲームセッションに接続する場合、IP アドレスではなく DNS 名を使用する必要があります。 タイプ: `std::string` 必須: いいえ

ServerParameters

Amazon GameLift Anywhere フリート上のゲームサーバーと Amazon GameLift サービス間の接続を維持するために使用される情報。この情報は、InitSDK() で新しいサーバープロセスを起動するときに使用されます。Amazon GameLift マネージド EC2 インスタンスでホストされているサーバーの場合は、空の オブジェクトを使用します。

プロパティ	説明
webSocketUrl	`GameLiftServerSdkEndpoint` Amazon は GameLift Anywhere、Amazon コンピューティングリソース`RegisterCompute` に対して を GameLift 返す。 タイプ: `std::string`

プロパティ	説明
	必須: はい
processId	ゲームをホストするサーバープロセスに登録された固有の識別子。 タイプ: `std::string` 必須: はい
hostId	`HostID` はコンピューティングを登録したときに使用される `ComputeName` です。詳細については、「」を参照してくださいRegisterCompute。 タイプ: `std::string` 必須: はい
fleetId	コンピューティングが登録されているフリートの固有識別子。詳細については、「」を参照してくださいRegisterCompute。 タイプ: `std::string` 必須: はい
authToken	サーバーを Amazon に対して認証 GameLift する Amazon によって生成された認証トークン GameLift。詳細については、「」を参照してくださいGetComputeAuthToken。 タイプ: `std::string` 必須: はい

StartMatchBackfillRequest

マッチメーキングバックフィルリクエストの作成に使用される情報。ゲームサーバーは、この情報を StartMatchBackfill()呼び出し GameLift で Amazon に伝えます。

プロパティ	説明
GameSessionArn	一意のゲームセッション識別子。API オペレーション GetGameSessionId は ARN 形式の識別子を返します。 タイプ: `std::string` 必須: はい
MatchmakingConfigurationArn	このリクエストに使用されるマッチメーカーの ARN 形式の一意な識別子。元のゲームセッションののマッチメーカー ARN は、マッチメーカーデータプロパティのゲームセッションオブジェクトにあります。マッチメーカーデータの詳細については「マッチメーカーデータの処理」を参照してください。 タイプ: `std::string` 必須: はい
プレイヤー	ゲームセッションに参加しているすべてのプレイヤーを表すデータのセット。マッチメーカーはこの情報を使用して、現在のプレイヤーとマッチする新しいプレイヤーを検索します。 タイプ: `std::vector<Player>` 必須: はい
TicketId	マッチメーキングまたはバックフィルリクエストチケットの一意の識別子。値を指定しない場合、Amazon は値 GameLift を生成します。この識別子を使用してマッチバックフィルチケッ

プロパティ	説明
	トのステータスを追跡したり、必要に応じてリクエストをキャンセルしたりします。 タイプ: `std::string` 必須: いいえ

プレイヤー

このデータ型はマッチメーキングのプレイヤーを表します。マッチメーキングリクエストを開始すると、プレイヤーはプレイヤー ID、属性、場合によってはレイテンシーデータを保有します。Amazon は、マッチングが行われた後にチーム情報 GameLift を追加します。

プロパティ	説明
LatencyInMS	プレイヤーがロケーションに接続したときに発生するレイテンシーの量を示すミリ秒単位の値のセット。 このプロパティを使用すると、プレーヤーはリストに表示されている場所でのみマッチングされます。マッチメーカーにプレイヤーレイテンシーを評価するルールがある場合、プレイヤーはレイテンシーを報告しないとマッチングされません。 タイプ: `Dictionary<string,int>` 必須: いいえ
PlayerAttributes	マッチメーキングに使用するプレイヤー情報を含むキーと値のペアの集合。プレイヤー属性キーは、マッチメーキングルールセット PlayerAttributes で使用される と一致する必要があります。

プロパティ	説明
	プレイヤー属性の詳細については、「」を参照してくださいAttributeValue。 タイプ: `std::map<std::string,AttributeValue>` 必須: いいえ
PlayerId	プレイヤーを表す一意の識別子。 タイプ: `std::string` 必須: いいえ
Team	マッチでプレーヤーが割り当てられるチームの名前。チーム名はマッチメーキングルールセットで定義します。 タイプ: `std::string` 必須: いいえ

DescribePlayerSessionsRequest

取得するプレイヤーセッションを指定するオブジェクト。サーバープロセスは、Amazon へのDescribePlayerSessions()呼び出しでこの情報を提供します GameLift。

プロパティ	説明
GameSessionId	一意のゲームセッション識別子。このパラメータを使用して、指定したゲームセッションのすべてのプレイヤーセッションをリクエストします。 ゲームセッション ID の形式は `arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>` で

プロパティ	説明
	す。GameSessionID はカスタム ID 文字列または タイプ: std::string 必須: いいえ
PlayerSessionId	プレイヤーセッションを表す一意の識別子。このパラメータを使用して、特定の 1 つのプレイヤーセッションをリクエストします。 タイプ: std::string 必須: いいえ
PlayerId	プレイヤーの一意識別子。このパラメータを使用して、特定の 1 人のプレイヤーに対するすべてのプレイヤーセッションをリクエストします。プレイヤー ID を生成する を参照してください。 タイプ: std::string 必須: いいえ

プロパティ	説明
PlayerSessionStatusFilter	結果をフィルタリングするプレイヤーセッションステータス。可能なプレイヤーセッションステータスには以下が含まれます。 • RESERVED – プレイヤーセッションリクエストは受領されましたが、プレイヤーはサーバープロセスに接続していないか、または検証はまだ行われていません。 • ACTIVE – プレイヤーはサーバープロセスによって検証され、接続されています。 • COMPLETED – プレイヤー接続は削除されました。 • TIMEDOUT – プレイヤーセッションリクエストは受領されましたが、タイムアウト制限 (60 秒) 内でのプレイヤーの接続や検証は行われていません。 タイプ: `std::string` 必須: いいえ
NextToken	結果の次のページの先頭を示すトークン。結果セットの先頭を指定するには、値を指定しないでください。プレイヤーセッション ID を提供する場合、このパラメータは無視されます。 タイプ: `std::string` 必須: いいえ

プロパティ	説明
制限	返される結果の最大数。プレイヤーセッション ID を提供する場合、このパラメータは無視されます。 タイプ: `int` 必須: いいえ

StopMatchBackfillRequest

マッチメーキングバックフィルリクエストのキャンセルに使用される情報。ゲームサーバーは、この情報を StopMatchBackfill()呼び出しで Amazon GameLift サービスに伝えます。

プロパティ	説明
GameSessionArn	キャンセルされるリクエストの一意のゲームセッション識別子。 タイプ: `char[]` 必須: いいえ
MatchmakingConfigurationArn	このリクエストが送信されたマッチメーカーの一意の識別子。 タイプ: `char[]` 必須: いいえ
TicketId	キャンセルされるバックフィルリクエストチケットの一意の識別子。 タイプ: `char[]` 必須: いいえ

AttributeValue

これらの値を プレイヤー 属性のキーと値のペアで使用します。このオブジェクトでは、文字列、数値、文字列配列、データマップのいずれかの有効なデータ型を使用して属性値を指定できます。各 `AttributeValue` オブジェクトは S、N、SL、または SDM の使用可能なプロパティのうちの 1 つだけを使用する必要があります。

プロパティ	説明
AttrType	属性値のタイプを指定します。可能な属性値のタイプは次のとおりです。 • なし • STRING • DOUBLE • STRING_LIST • STRING_DOUBLE_MAP 必須: いいえ
S	文字列の属性値を表します。 タイプ: `std::string` 必須: いいえ
N	数値の属性値を表します。 タイプ: `double` 必須: いいえ
SL	文字列の属性値の配列を表します。 タイプ: `std::vector<std::string>` 必須: いいえ

プロパティ	説明
SDM	文字列キーと二重値のディクショナリを表します。 タイプ: `std::map<std::string, double>` 必須: いいえ

GetFleetRoleCredentialsRequest

このデータ型により、ゲームサーバーは他の AWS リソースへのアクセスが制限されます。詳細については、「Amazon の IAM サービスロールを設定する GameLift」を参照してください。

プロパティ	説明
RoleArn	AWS リソースへの制限付きアクセスを拡張するサービスロールの Amazon リソースネーム (ARN)。 タイプ: `std::string` 必須: いいえ
RoleSessionName	セッションを一意に識別するために使用できるロールAWS Security Token ServiceAssumeRoleセッション名。この名前は、 の監査ログなどで公開されます CloudTrail。 タイプ: `std::string` 必須: いいえ

AwsLongOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 タイプ: `long` 必須: いいえ
ResultWithOwnership	アクションの結果を rvalue としてキャストし、呼び出し元のコードがオブジェクトの所有権を取得できるようにします。 タイプ: `long&&` 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “GameLiftError” 必須: いいえ

AwsStringOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 タイプ: `std::string` 必須: いいえ

プロパティ	説明
ResultWithOwnership	アクションの結果を rvalue としてキャストし、呼び出し元のコードがオブジェクトの所有権を取得できるようにします。 タイプ: `long&&` 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “GameLiftError” 必須: いいえ

DescribePlayerSessionsOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 タイプ: the section called “DescribePlayerSessionsResult” 必須: いいえ
ResultWithOwnership	アクションの結果を rvalue としてキャストし、呼び出し元のコードがオブジェクトの所有権を取得できるようにします。

プロパティ	説明
	タイプ: `Aws::GameLift::Server::Model::DescribePlayerSessionsResult&&` 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “GameLiftError” 必須: いいえ

DescribePlayerSessionsResult

リクエストに一致する各プレイヤーセッションのプロパティを含むオブジェクトの集合。

プロパティ	説明
NextToken	結果の次の順次ページの先頭を示すトークン。このオペレーションの以前の呼び出しで返されたトークンを使用します。結果セットの先頭で開始するには、値を指定しないでください。プレイヤーセッション ID を指定した場合、このパラメータは無視されます。 タイプ: `std::string` 必須: はい
PlayerSessions	タイプ: `IList<`the section called “PlayerSession” `>`

プロパティ	説明
	必須:
ResultWithOwnership	アクションの結果を rvalue としてキャストし、呼び出し元のコードがオブジェクトの所有権を取得できるようにします。 タイプ: `std::string&&` 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “GameLiftError” 必須: いいえ

GenericOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “GameLiftError” 必須: いいえ

GenericOutcomeCallable

このデータ型は非同期の一般的な結果です。以下のプロパティがあります。

プロパティ	説明
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “GameLiftError” 必須: いいえ

PlayerSession

このデータ型は、Amazon がゲームサーバーに GameLift 渡すプレイヤーセッションを表します。詳細については、「」を参照してくださいPlayerSession。

プロパティ	説明
CreationTime	タイプ: `long` 必須: いいえ
FleetId	タイプ: `std::string` 必須: いいえ
GameSessionId	タイプ: `std::string` 必須: いいえ
IpAddress	タイプ: `std::string` 必須: いいえ

プロパティ	説明
PlayerData	タイプ: `std::string` 必須: いいえ
PlayerId	タイプ: `std::string` 必須: いいえ
PlayerSessionId	タイプ: `std::string` 必須: いいえ
ポート	タイプ: `int` 必須: いいえ
ステータス	結果をフィルタリングするプレイヤーセッションステータス。 PlayerSessionId または PlayerId が指定されている場合、 PlayerSessionStatusFilter はレスポンスには影響しません。 Type: A `PlayerSessionStatus` enum. 以下に示しているのは、可能な値です。 • ACTIVE • COMPLETED • NOT_SET • RESERVED • TIMEDOUT 必須: いいえ
TerminationTime	タイプ: `long` 必須: いいえ

プロパティ	説明
DnsName	タイプ: `std::string` 必須: いいえ

StartMatchBackfillOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 タイプ: the section called “StartMatchBackfillResult” 必須: いいえ
ResultWithOwnership	アクションの結果を rvalue としてキャストし、呼び出し元のコードがオブジェクトの所有権を取得できるようにします。 タイプ: `StartMatchBackfillResult&&` 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “GameLiftError” 必須: いいえ

StartMatchBackfillResult

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
TicketId	マッチメーキングチケットの一意の識別子。ここでチケット ID が指定されていない場合、Amazon GameLift は UUID の形式でチケット ID を生成します。この識別子を使用して、マッチバックフィルチケットのステータスを追跡し、マッチ結果を取得します。 タイプ: `std::string` 必須: いいえ

GetComputeCertificateOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 タイプ: the section called “GetComputeCertificateResult” 必須: いいえ
ResultWithOwnership	アクションの結果を rvalue としてキャストし、呼び出し元のコードがオブジェクトの所有権を取得できるようにします。 タイプ: `Aws::GameLift::Server::Model::GetComputeCertificateResult&&` 必須: いいえ

プロパティ	説明
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “GameLiftError” 必須: いいえ

GetComputeCertificateResult

コンピューティングの TLS 証明書へのパスとコンピューティングのホスト名。

プロパティ	説明
CertificatePath	コンピューティングリソースの TLS 証明書へのパス。Amazon GameLift マネージドフリートを使用する場合、このパスには以下が含まれます。 • `certificate.pem` : エンドユーザー証明書。証明書チェーン全体は、この証明書に追加された `certificateChain.pem` を組み合わせたものです。 • `certificateChain.pem` : ルート証明書と中間証明書を含む証明書チェーン。 • `rootCertificate.pem` : ルート証明書。 • `privateKey.pem` : エンドユーザー証明書のプライベートキー。 タイプ: `std::string`

プロパティ	説明
	必須: いいえ
ComputeName	コンピューティングリソースの名前。 タイプ: `std::string` 必須: いいえ

GetFleetRoleCredentialsOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 タイプ: the section called “GetFleetRoleCredentialsResult” 必須: いいえ
ResultWithOwnership	アクションの結果を rvalue としてキャストし、呼び出し元のコードがオブジェクトの所有権を取得できるようにします。 タイプ: `Aws::GameLift::Server::Model::GetFleetRoleCredentialsResult` 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。

プロパティ	説明
	タイプ: the section called “GameLiftError” 必須: いいえ

GetFleetRoleCredentialsResult

プロパティ	説明
AccessKeyId	AWS へのアクセスを認証して提供するためのアクセスキー ID。 タイプ: `string` 必須: いいえ
AssumedRoleId	サービスロールが属するユーザーの ID。 タイプ: `string` 必須: いいえ
AssumedRoleUserArn	サービスロールが属するユーザーの Amazon リソースネーム (ARN)。 タイプ: `string` 必須: いいえ
有効期限	セッション認証情報の有効期限が切れるまでの時間。 タイプ: `DateTime` 必須: いいえ
SecretAccessKey	認証のためのシークレットアクセスキー ID。 タイプ: `string`

プロパティ	説明
	必須: いいえ
SessionToken	AWS リソースとやり取りしている現在のアクティブなセッションを識別するトークン。 タイプ: `string` 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “GameLiftError” 必須: いいえ

InitSDKOutcome

> **Note**
>
> `InitSDKOutcome` は `std` フラグを付けて SDK を構築した場合にのみ返されます。`nostd` フラグを付けて構築すると、代わりに the section called “GenericOutcome” が返されます。

プロパティ	説明
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。

プロパティ	説明
	タイプ: the section called “GameLiftError” 必須: いいえ

GameLiftError

プロパティ	説明
ErrorType	エラーのタイプ。 型: A `GameLiftErrorType` enum。 必須: いいえ
ErrorName	エラーの名前。 タイプ: `std::string` 必須: いいえ
ErrorMessage	エラーメッセージです。 タイプ: `std::string` 必須: いいえ

列挙型

Amazon Server GameLift SDK (C++) に定義された列挙型は、次のように定義されます。

GameLiftErrorType

エラータイプを示す文字列値。有効な値を次に示します。

- BAD_REQUEST_EXCEPTION
- GAMESESSION_ID_NOT_SET – ゲームセッション ID が設定されていません。
- INTERNAL_SERVICE_EXCEPTION
- LOCAL_CONNECTION_FAILED – Amazon へのローカル接続 GameLift に失敗しました。

- NETWORK_NOT_INITIALIZED – ネットワークは初期化されていません。
- SERVICE_CALL_FAILED – AWS サービスへの呼び出しが失敗しました。
- WEBSOCKET_CONNECT_FAILURE
- WEBSOCKET_CONNECT_FAILURE_FORBIDDEN
- WEBSOCKET_CONNECT_FAILURE_INVALID_URL
- WEBSOCKET_CONNECT_FAILURE_TIMEOUT
- ALREADY_INITIALIZED – Amazon GameLift サーバーまたはクライアントはすでに Initialize() で初期化されています。
- FLEET_MISMATCH – ターゲットフリートがゲームセッションまたはプレイヤーセッションのフリートと一致しません。
- GAMiliFT_CLIENT_NOT_INITIALIZED – Amazon GameLift クライアントは初期化されていません。
- GAMiliFT_SERVER_NOT_INITIALIZED – Amazon GameLift サーバーは初期化されていません。
- GAME_SESSION_ENDED_FAILED – Amazon GameLift Server SDK はサービスに連絡してゲームセッションが終了したことを報告できませんでした。
- GAME_SESSION_NOT_READY – Amazon GameLift サーバーゲームセッションはアクティブ化されていません。
- GAME_SESSION_READY_FAILED – Amazon GameLift Server SDK はサービスに連絡してゲームセッションの準備ができたことを報告することができませんでした。
- INITIALIZATION_MISMATCH – Server:: Initialize() の後にクライアントメソッドが呼び出されました。その逆も同様です。
- NOT_INITIALIZED – Amazon GameLift サーバーまたはクライアントが Initialize() で初期化されていません。
- NO_TARGET_ALIASID_SET – ターゲットのエイリアスが設定されていません。
- NO_TARGET_FLEET_SET – ターゲットフリートが設定されていません。
- PROCESS_ENDING_FAILED – Amazon GameLift Server SDK は、プロセスが終了していることを報告するためにサービスに接続できませんでした。
- PROCESS_NOT_ACTIVE – サーバープロセスはまだアクティブではなく、 にバインドされておらず GameSession、 を受け入れたり処理したりすることはできません PlayerSessions。
- PROCESS_NOT_READY – サーバープロセスをまだアクティブ化する準備ができていません。

- PROCESS_READY_FAILED – Amazon GameLift Server SDK は、プロセスの準備が完了したことを報告するために サービスに接続できませんでした。
- SDK_VERSION_DETECTION_FAILED – SDK バージョン検出に失敗しました。
- STX_CALL_FAILED – XSTX サーバーのバックエンドコンポーネントへの呼び出しが失敗しました。
- STX_INITIALIZATION_FAILED – XSTX サーバーのバックエンドコンポーネントが初期化に失敗しました。
- UNEXPECTED_PLAYER_SESSION – 未登録のプレイヤーセッションがサーバーによって検出されました。
- WEBSOCKET_CONNECT_FAILURE
- WEBSOCKET_CONNECT_FAILURE_FORBIDDEN
- WEBSOCKET_CONNECT_FAILURE_INVALID_URL
- WEBSOCKET_CONNECT_FAILURE_TIMEOUT
- WEBSOCKET_RETRIABLE_SEND_MESSAGE_FAILURE – GameLift サービスへのメッセージの送信に失敗しました WebSocket。
- WEBSOCKET_SEND_MESSAGE_FAILURE – GameLift サービスへのメッセージの送信に失敗しました WebSocket。
- MATCH_BACKFILL_REQUEST_VALIDATION – リクエストの検証に失敗しました。
- PLAYER_SESSION_REQUEST_VALIDATION – リクエストの検証に失敗しました。

PlayerSessionCreationPolicy

ゲームセッションで新しいプレイヤーを承諾するかどうかを示す文字列値。有効な値を次に示します。

- ACCEPT_ALL – すべての新しいプレイヤーセッションを承諾します。
- DENY_ALL – すべての新しいプレイヤーセッションを拒否します。
- NOT_SET – ゲームセッションは、新規プレイヤーセッションを受け入れたり拒否したりするように設定されていません。

Amazon GameLift C++ サーバー SDK 3.x リファレンス

この Amazon GameLift C++ サーバー SDK 3.x リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

トピック

- Amazon GameLift サーバー SDK (C++) リファレンス: アクション
- Amazon GameLift サーバー SDK (C++) リファレンス: データ型

Amazon GameLift サーバー SDK (C++) リファレンス: アクション

この Amazon GameLift C++ サーバー SDK リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

アクション

- AcceptPlayerSession()
- ActivateGameSession()
- DescribePlayerSessions()
- GetGameSessionId()
- GetInstanceCertificate()
- GetSdkVersion()
- GetTerminationTime()
- InitSDK()
- ProcessEnding()
- ProcessReady()
- ProcessReadyAsync()
- RemovePlayerSession()
- StartMatchBackfill()
- StopMatchBackfill()
- TerminateGameSession()
- UpdatePlayerSessionCreationPolicy()
- Destroy()

AcceptPlayerSession()

指定されたプレイヤーセッション ID のプレイヤーがサーバープロセスに接続し、検証が必要であることを Amazon GameLift サービスに通知します。Amazon GameLift は、プレイヤーセッション

ID が有効であること、つまり、そのプレイヤー ID でゲームセッションにプレイヤースロットが予約されていることを確認します。検証できたら、Amazon GameLift はプレーヤースロットの状態を RESERVED から ACTIVE に変更します。

構文

```
GenericOutcome AcceptPlayerSession(const std::string& playerSessionId);
```

パラメータ

playerSessionId

AWS SDK Amazon GameLift API アクションのCreatePlayerSessionへの呼び出しに応答して Amazon GameLift サービスによって発行された一意の ID。ゲームクライアントは、サーバープロセスに接続するときにこの ID をリファレンスします。

Type: std::string

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、無効なプレイヤーセッション ID の検証や拒否を含む、接続リクエストを処理するための関数を示します。

```
void ReceiveConnectingPlayerSessionID (Connection& connection, const std::string&
 playerSessionId){
    Aws::GameLift::GenericOutcome connectOutcome =
        Aws::GameLift::Server::AcceptPlayerSession(playerSessionId);
    if(connectOutcome.IsSuccess())
    {
        connectionToSessionMap.emplace(connection, playerSessionId);
        connection.Accept();
    }
    else
    {
        connection.Reject(connectOutcome.GetError().GetMessage();
    }
```

```
}
```

ActivateGameSession()

サーバープロセスがゲームセッションを開始し、プレイヤーの接続を受ける準備ができていることを Amazon GameLift サービスに通知します。このアクションは、すべてのゲームセッションの初期化が完了した後、onStartGameSession() コールバック関数の一部として呼び出されます。

構文

```
GenericOutcome ActivateGameSession();
```

パラメータ

このアクションにはパラメータがありません。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、ActivateGameSession() が onStartGameSession() コールバック関数の一部として呼び出されていることを示しています。

```
void onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
   // game-specific tasks when starting a new game session, such as loading map
   GenericOutcome outcome = Aws::GameLift::Server::ActivateGameSession();
}
```

DescribePlayerSessions()

設定、セッションメタデータ、プレイヤーデータを含む、プレイヤーセッションデータを取得します。このアクションを使用して、単一のプレイヤーセッション、ゲームセッション内のすべてのプレイヤーセッション、または単一のプレイヤー ID に関連付けられたすべてのプレイヤーセッションに関する情報を取得します。

構文

```
DescribePlayerSessionsOutcome DescribePlayerSessions (
```

```
    const Aws::GameLift::Server::Model::DescribePlayerSessionsRequest
&describePlayerSessionsRequest);
```

パラメータ

describePlayerSessionsRequest

取得するプレイヤーセッションを記述する DescribePlayerSessionsRequest オブジェクト。

必須: はい

戻り値

成功した場合は、リクエストのパラメータに適合したプレイヤーセッションオブジェクトのセットを含む DescribePlayerSessionsOutcome オブジェクトを返します。プレイヤーセッションオブジェクトは、AWS SDK Amazon GameLift API PlayerSession データ型と同じ構造を持ちます。

例

この例は、指定したゲームセッションにアクティブに接続されているすべてのプレイヤーセッションのリクエストを示しています。NextToken を省略し、Limit 値を 10 に設定すると、Amazon GameLift はリクエストに一致する最初の 10 個のプレイヤーセッションを返します。

```
// Set request parameters
Aws::GameLift::Server::Model::DescribePlayerSessionsRequest request;
request.SetPlayerSessionStatusFilter(Aws::GameLift::Server::Model::PlayerSessionStatusMapper::G
request.SetLimit(10);
request.SetGameSessionId("the game session ID");    // can use GetGameSessionId()

// Call DescribePlayerSessions
Aws::GameLift::DescribePlayerSessionsOutcome playerSessionsOutcome =
    Aws::GameLift::Server::DescribePlayerSessions(request);
```

GetGameSessionId()

サーバープロセスがアクティブな場合、サーバープロセスが現在ホストしているゲームセッションの一意の識別子を取得します。識別子は ARN 形式で返されます: arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>

ゲームセッションでまだアクティブ化されていないアイドルプロセスの場合、コールはSuccess=TrueそしてGameSessionId=""(空の文字列)を返します。

構文

```
AwsStringOutcome GetGameSessionId();
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合、ゲームセッション ID を AwsStringOutcome オブジェクトとして返します。成功しなかった場合、エラーメッセージを返します。

例

```
Aws::GameLift::AwsStringOutcome sessionIdOutcome =
    Aws::GameLift::Server::GetGameSessionId();
```

GetInstanceCertificate()

フリートとそのインスタンスに関連付けられている pem エンコードされた TLS 証明書のファイルの場所を取得します。AWS Certificate Manager は、証明書設定を GENERATED に設定して新しいフリートを作成するとこの証明書が生成されます。この証明書を使用して、ゲームクライアントとのセキュリティ保護ありの接続を確立し、クライアント/サーバー通信を暗号化します。

構文

```
GetInstanceCertificateOutcome GetInstanceCertificate();
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功すると、インスタンスに保存されているフリートの TLS 証明書ファイルの場所と証明書チェーンを含む GetInstanceCertificateOutcome オブジェクトを返します。証明書チェーンから抽出されたルート証明書ファイルもインスタンスに保存されます。成功しなかった場合、エラーメッセージを返します。

証明書と証明書チェーンデータの詳細については、AWS Certificate Manager API リファレンスの「GetCertificate レスポンス要素」を参照してください。

例

```
Aws::GameLift::GetInstanceCertificateOutcome certificateOutcome =
    Aws::GameLift::Server::GetInstanceCertificate();
```

GetSdkVersion()

使用中の SDK の現在のバージョン番号を返します。

構文

```
AwsStringOutcome GetSdkVersion();
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合、AwsStringOutcome オブジェクトとして現在の SDK バージョンを返します。返される文字列は、バージョン番号のみを含みます(例: 3.1.5)。成功しなかった場合、エラーメッセージを返します。

例

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
    Aws::GameLift::Server::GetSdkVersion();
```

GetTerminationTime()

終了時刻が判る場合に、サーバープロセスがシャットダウンを予定している時刻を返します。サーバープロセスは、Amazon GameLift サービスから onProcessTerminate() コールバックを受信した後にこのアクションを実行します。Amazon GameLift が onProcessTerminate() を呼び出すには、以下の理由が考えられます。(1) サーバープロセスが異常を報告した場合、もしくは Amazon GameLift に応答しなかった場合、(2) スケールダウンイベント中にインスタンスを終了した場合、(3)スポット中断のためインスタンスが終了した場合。

プロセスが onProcessTerminate() コールバックを受信した場合、戻り値は予想終了時刻です。プロセスが onProcessTerminate() コールバックを受信していない場合、エラーメッセージが返されます。サーバープロセスのシャットダウンの詳細を確認してください。

構文

```
AwsLongOutcome GetTerminationTime();
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合、終了時刻をAwsLongOutcomeオブジェクトとして返します。値は終了時間で、0001 00:00:00 以降の経過ティックで表現されます。たとえば、日付時刻の値 2020-09-13 12:26:40 -000Z は、637355968000000000 ティックに等しくなります。終了時間がない場合は、エラーメッセージを返します。

例

```
Aws::GameLift::AwsLongOutcome TermTimeOutcome =
    Aws::GameLift::Server::GetTerminationTime();
```

InitSDK()

Amazon GameLift SDK を初期化する。このメソッドは起動時に他の Amazon GameLift 関連の初期化が実行される前に呼び出す必要があります。

構文

```
InitSDKOutcome InitSDK();
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合は、サーバープロセスが ProcessReady() を呼び出す準備ができていることを示す InitSdkOutcome オブジェクトを返します。

例

```
Aws::GameLift::Server::InitSDKOutcome initOutcome =
```

```
    Aws::GameLift::Server::InitSDK();
```

ProcessEnding()

サーバープロセスがシャットダウンしていることを Amazon GameLift サービスに通知します。このメソッドは、すべてのアクティブゲームセッションのシャットダウンを含む他のすべてのクリーンアップタスクの後に呼び出されます。このメソッドは、終了コード 0 で終了します。0 以外の終了コードでは、処理が問題なく終了しなかったというイベントメッセージが発生します。

メソッドがコード 0 で終了すると、成功した終了コードでプロセスを終了できます。エラーコードでプロセスを終了することもできます。エラーコードで終了すると、フリートイベントはプロセスが異常終了したことを示します (SERVER_PROCESS_TERMINATED_UNHEALTHY)。

構文

```
GenericOutcome ProcessEnding();
```

パラメータ

このアクションにはパラメータがありません。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
Aws::GameLift::GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
if (outcome.Success)
    exit(0);  // exit with success
// otherwise, exit with error code
exit(errorCode);
```

ProcessReady()

サーバープロセスがゲームセッションをホストする準備ができたことを Amazon GameLift サービスに通知します。このメソッドは、InitSDK() の呼び出しが成功して必要な設定タスクが完了した後、サーバープロセスがゲームセッションをホストできるようになる前に呼び出す必要があります。このメソッドは、プロセスごとに 1 回だけ呼び出す必要があります。

この呼び出しは同期です。非同期呼び出しを実行するには、ProcessReadyAsync() を使用します。詳細については、「サーバープロセスを初期化する」を参照してください。

構文

```
GenericOutcome ProcessReady(
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

パラメータ

processParameters

サーバープロセスに関する以下の情報を伝える ProcessParameters オブジェクト。

- サーバープロセスと通信するために Amazon GameLift サービスが呼び出す、ゲームサーバーコードで実装されたコールバックメソッドの名前。
- サーバープロセスがリッスンするポートの番号。
- Amazon GameLift でキャプチャして保存するゲームセッション固有のファイルへのパス。

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、ProcessReady() 呼び出しとコールバック関数の実装の両方を示します。

```
// Set parameters and call ProcessReady
std::string serverLog("serverOut.log");        // Example of a log file written by the
 game server
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);

int listenPort = 9339;

Aws::GameLift::Server::ProcessParameters processReadyParameter =
 Aws::GameLift::Server::ProcessParameters(
    std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
    std::bind(&Server::onProcessTerminate, this),
    std::bind(&Server::OnHealthCheck, this),
    std::bind(&Server::OnUpdateGameSession, this),
```

```
    listenPort,
    Aws::GameLift::Server::LogParameters(logPaths));

Aws::GameLift::GenericOutcome outcome =
   Aws::GameLift::Server::ProcessReady(processReadyParameter);

// Implement callback functions
void Server::onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
   // game-specific tasks when starting a new game session, such as loading map
   GenericOutcome outcome =
       Aws::GameLift::Server::ActivateGameSession (maxPlayers);
}

void Server::onProcessTerminate()
{
   // game-specific tasks required to gracefully shut down a game session,
   // such as notifying players, preserving game state data, and other cleanup
   GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
}

bool Server::onHealthCheck()
{
    bool health;
    // complete health evaluation within 60 seconds and set health
    return health;
}
```

ProcessReadyAsync()

サーバープロセスがゲームセッションをホストする準備ができたことを Amazon GameLift サービスに通知します。サーバープロセスがゲームセッションをホストする準備ができたら、このメソッドを呼び出します。パラメータは、特定の状況で呼び出す Amazon GameLift のコールバック関数名を指定します。ゲームサーバーコードは、これらの関数を実装する必要があります。

この呼び出しは非同期です。同期呼び出しを実行するには、ProcessReady() を使用します。詳細については、「サーバープロセスを初期化する」を参照してください。

構文

```
GenericOutcomeCallable ProcessReadyAsync(
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

パラメータ

processParameters

サーバープロセスに関する以下の情報を伝える ProcessParameters オブジェクト。

- サーバープロセスと通信するために Amazon GameLift サービスが呼び出す、ゲームサーバーコードで実装されたコールバックメソッドの名前。
- サーバープロセスがリッスンするポートの番号。
- Amazon GameLift でキャプチャして保存するゲームセッション固有のファイルへのパス。

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
// Set parameters and call ProcessReady
std::string serverLog("serverOut.log");        // This is an example of a log file
 written by the game server
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);

int listenPort = 9339;

Aws::GameLift::Server::ProcessParameters processReadyParameter =
 Aws::GameLift::Server::ProcessParameters(
    std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
    std::bind(&Server::onProcessTerminate, this),
    std::bind(&Server::OnHealthCheck, this),
    std::bind(&Server::OnUpdateGameSession, this),
    listenPort,
    Aws::GameLift::Server::LogParameters(logPaths));

Aws::GameLift::GenericOutcomeCallable outcome =
   Aws::GameLift::Server::ProcessReadyAsync(processReadyParameter);

// Implement callback functions
void onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
   // game-specific tasks when starting a new game session, such as loading map
```

```
   GenericOutcome outcome = Aws::GameLift::Server::ActivateGameSession (maxPlayers);
}

void onProcessTerminate()
{
   // game-specific tasks required to gracefully shut down a game session,
   // such as notifying players, preserving game state data, and other cleanup
   GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
}

bool onHealthCheck()
{
    // perform health evaluation and complete within 60 seconds
    return health;
}
```

RemovePlayerSession()

指定されたプレイヤーセッション ID のプレイヤーがサーバープロセスから切断されたことを Amazon GameLift サービスに通知します。それに応じて、Amazon GameLift はプレイヤースロットを新しいプレイヤーに割り当てられるよう利用可能に変更します。

構文

```
GenericOutcome RemovePlayerSession(
    const std::string& playerSessionId);
```

パラメータ

playerSessionId

AWS SDK Amazon GameLift API アクションのCreatePlayerSessionへの呼び出しに応答して Amazon GameLift サービスによって発行された一意の ID。ゲームクライアントは、サーバープロセスに接続するときにこの ID をリファレンスします。

Type: std::string

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
Aws::GameLift::GenericOutcome disconnectOutcome =
    Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

StartMatchBackfill()

FlexMatch で作成されたゲームセッションの空きスロット用に新規プレイヤーを検索するリクエストを送信します。AWS SDK アクション StartMatchBackfill() を参照してください。このアクションを使用すると、ゲームセッションをホストするゲームサーバーのプロセスによってマッチバックフィルリクエストを開始できます。FlexMatch バックフィルの特徴の詳細はこちら。

このアクションは非同期です。新規プレイヤーが正常にマッチングされると、Amazon GameLift サービスはコールバック関数 OnUpdateGameSession() を使用して更新済みマッチメーカーデータを送信します。

サーバープロセスではアクティブなマッチバックフィルリクエストは一度に 1 つだけです。新しいリクエストを送信するには、まず StopMatchBackfill() を呼び出して元のリクエストをキャンセルする必要があります。

構文

```
StartMatchBackfillOutcome StartMatchBackfill (
    const Aws::GameLift::Server::Model::StartMatchBackfillRequest
 &startBackfillRequest);
```

パラメータ

StartMatchBackfillRequest

次の情報を通信する StartMatchBackfillRequest オブジェクト。

- バックフィルリクエストに割り当てるチケット ID。この情報はオプションです。ID が指定されていない場合は Amazon GameLift が ID を 1 つ自動生成します。
- リクエストを送信するマッチメーカー。完全な設定 ARN が必要です。この値は、ゲームセッションのマッチメーカーデータから取得できます。
- バックフィルされるゲームセッションの ID。
- ゲームセッションの現在のプレイヤーに利用可能なマッチメーキングデータ。

必須: はい

戻り値

StartMatchBackfillOutcome オブジェクトを、マッチバックフィルチケットまたはエラーメッセージを伴うエラーとともに返します。チケットのステータスは、AWS SDK アクション DescribeMatchmaking() を使用して追跡できます。

例

```
// Build a backfill request
std::vector<Player> players;
Aws::GameLift::Server::Model::StartMatchBackfillRequest startBackfillRequest;
startBackfillRequest.SetTicketId("a ticket ID");
  //optional, autogenerated if not provided
startBackfillRequest.SetMatchmakingConfigurationArn("the matchmaker configuration
 ARN"); //from the game session matchmaker data
startBackfillRequest.SetGameSessionArn("the game session ARN");
  // can use GetGameSessionId()
startBackfillRequest.SetPlayers(players);
    //from the game session matchmaker data

// Send backfill request
Aws::GameLift::StartMatchBackfillOutcome backfillOutcome =
    Aws::GameLift::Server::StartMatchBackfill(startBackfillRequest);

// Implement callback function for backfill
void Server::OnUpdateGameSession(Aws::GameLift::Server::Model::GameSession gameSession,
 Aws::GameLift::Server::Model::UpdateReason updateReason, std::string backfillTicketId)
{
   // handle status messages
   // perform game-specific tasks to prep for newly matched players
}
```

StopMatchBackfill()

StartMatchBackfill() とともに作成されたアクティブなマッチバックフィルリクエストをキャンセルします。AWS SDK アクション StopMatchmaking() も参照してください。FlexMatch バックフィルの特徴の詳細はこちら。

構文

```
GenericOutcome StopMatchBackfill (
    const Aws::GameLift::Server::Model::StopMatchBackfillRequest &stopBackfillRequest);
```

パラメータ

StopMatchBackfillRequest

キャンセルするマッチメーキングチケットを識別する StopMatchBackfillRequestオブジェクト:

- キャンセルされるバックフィルリクエストに割り当てられたチケット ID
- バックフィルリクエストが送信されるマッチメーカー
- バックフィルリクエストに関連付けられたゲームセッション

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
// Set backfill stop request parameters

Aws::GameLift::Server::Model::StopMatchBackfillRequest stopBackfillRequest;
stopBackfillRequest.SetTicketId("the ticket ID");
stopBackfillRequest.SetGameSessionArn("the game session ARN");
  // can use GetGameSessionId()
stopBackfillRequest.SetMatchmakingConfigurationArn("the matchmaker configuration ARN");
  // from the game session matchmaker data

Aws::GameLift::GenericOutcome stopBackfillOutcome =
    Aws::GameLift::Server::StopMatchBackfillRequest(stopBackfillRequest);
```

TerminateGameSession()

このメソッドは、バージョン 4.0.1 で非推奨となりました。代わりに、サーバープロセスはゲームセッションが終了した後に ProcessEnding() を呼び出す必要があります。

サーバープロセスがゲームセッションをシャットダウンしたことを　Amazon GameLift サービスに通知します。このアクションは、サーバープロセスがアクティブなままになり、新しいゲームセッションをホストするための準備ができたときに呼び出されます。これは、新しいゲームセッションをホストするためにサーバープロセスがすぐに利用可能であることを Amazon GameLift に通知するため、ゲームセッション終了手順が完了した後にのみ呼び出す必要があります。

ゲームセッションが停止した後にサーバープロセスがシャットダウンされる場合は、このアクションは呼び出されません。代わりに、 ProcessEnding() を呼び出し、ゲームセッションとサーバープロセスの両方が終了していることを通知します。

構文

```
GenericOutcome TerminateGameSession();
```

パラメータ

このアクションにはパラメータがありません。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

UpdatePlayerSessionCreationPolicy()

現在のゲームセッションの機能を更新し、新しいプレイヤーセッションを承諾します。ゲームセッションは、新しいプレイヤーセッションをすべて受け入れるか拒否するかを設定できます。AWS SDK アクション UpdateGameSession() も参照してください。

構文

```
GenericOutcome UpdatePlayerSessionCreationPolicy(
    Aws::GameLift::Model::PlayerSessionCreationPolicy newPlayerSessionPolicy);
```

パラメータ

newPlayerSessionPolicy

ゲームセッションで新しいプレイヤーを承諾するかどうかを示す文字列値。

型: Aws::GameLift::Model::PlayerSessionCreationPolicy 列挙。有効な値を次に示します。

- ACCEPT_ALL – すべての新しいプレイヤーセッションを承諾します。
- DENY_ALL – すべての新しいプレイヤーセッションを拒否します。

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例は、現在のゲームセッションの参加ポリシーを、すべてのプレイヤーを受け入れるように設定します。

```
Aws::GameLift::GenericOutcome outcome =
 Aws::GameLift::Server::UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCr
```

Destroy()

ゲームサーバーの初期化中に initSDK () によって割り当てられたメモリをクリーンアップします。このメソッドは、ゲームサーバープロセスを終了した後に使用すると、サーバーメモリの浪費を防ぐことができます。

構文

```
GenericOutcome Aws::GameLift::Server::Destroy();
```

パラメータ

パラメータはありません。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、ゲームサーバープロセスの終了後に InitSDK によって割り当てられたメモリをクリーンアップします。

```
if (Aws::GameLift::Server::ProcessEnding().IsSuccess()) {
  Aws::GameLift::Server::Destroy();
  exit(0);
}
```

Amazon GameLift サーバー SDK (C++) リファレンス: データ型

この Amazon GameLift C++ サーバー SDK リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

この API は、GameLiftServerAPI.h、LogParameters.h、および ProcessParameters.h で定義されています。

- [Actions] (アクション)
- データ型

DescribePlayerSessionsRequest

このデータ型は、取得するプレイヤーセッションを指定するのに使用されます。このデータ型は次のように使用できます。

- 特定のプレーヤーセッションをリクエストする PlayerSessionId を提供します。
- 指定したゲームセッションのすべてのプレイヤーセッションをリクエストする GameSessionId を提供します。
- 指定したプレイヤーのすべてのプレイヤーセッションをリクエストする PlayerId を提供します。

プレイヤーセッション数が多い場合は、ページ分割パラメータを使用して結果を順次ブロックとして取得します。

目次

GameSessionId

一意のゲームセッション識別子。このパラメータを使用して、指定したゲームセッションのすべてのプレイヤーセッションをリクエストします。ゲームセッション ID の形式は、arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string> です。<ID string> の値は、カスタム ID 文字列または (ゲームセッション作成時に指定した場合) 生成された文字列のいずれかです。

型: 文字列

必須: いいえ

制限

返される結果の最大数。このパラメータを NextToken とともに使用して、結果を一連の順次ページとして取得します。プレイヤーセッション ID を指定した場合、このパラメータは無視されます。

型: 整数

必須: いいえ

NextToken

結果において次の順次ページの開始を示すトークン。このアクションの以前の呼び出しで返されたトークンを使用します。結果セットの先頭を指定するには、値を指定しないでください。プレイヤーセッション ID を指定した場合、このパラメータは無視されます。

型: 文字列

必須: いいえ

PlayerId

プレイヤーを表す一意の識別子。プレイヤー ID は開発者によって定義されます。「プレイヤー ID を生成する」を参照してください。

型: 文字列

必須: いいえ

PlayerSessionId

プレイヤーセッションを表す一意の識別子。

型: 文字列

必須: いいえ

PlayerSessionStatusFilter

結果をフィルタリングするプレイヤーセッションステータス。可能なプレイヤーセッションステータスとして以下のステータスがあります。

- RESERVED – プレイヤーセッションリクエストは受領されましたが、プレイヤーのサーバープロセスへの接続や検証はまだ行われていません。
- ACTIVE – プレイヤーはサーバープロセスによって検証され、現時点で接続されています。
- COMPLETED – プレイヤー接続は削除されました。
- TIMEDOUT – プレイヤーセッションリクエストは受領されましたが、タイムアウト制限 (60 秒) 内でのプレイヤーの接続や検証は行われていません。

型: 文字列

必須: いいえ

LogParameters

このデータ型は、ゲームセッション中に生成されたファイルのうち、ゲームセッション終了時に Amazon GameLift でアップロードして保存するファイルを識別するのに使用されます。この情報は、ProcessReady() 呼び出しで Amazon GameLift サービスに伝えられます。

目次

logPaths

Amazon GameLift で将来のアクセスに備えて保存するゲームサーバーログファイルへのディレクトリパス。これらのファイルは、各ゲームセッション中に生成されます。ファイルのパスと名前はゲームサーバーで定義され、ルートゲームビルドディレクトリに保存されます。ログパスは絶対パスである必要があります。たとえば、ゲームビルドによって `MyGame\sessionlogs\` などのパスに保存されるゲームセッションログのログパスは `c:\game\MyGame\sessionLogs` (Windows インスタンスの場合) または `/local/game/MyGame/sessionLogs` (Linux インスタンスの場合) となります。

型: std:vector<std::string>

必須: いいえ

ProcessParameters

このデータ型には、ProcessReady() 呼び出しで Amazon GameLift サービスに送られたパラメータセットが含まれます。

目次

port

サーバープロセスが新しいプレイヤーの接続をリスンするポート番号。値は、このゲームサーバービルドをデプロイするすべてのフリートで設定されているポート番号の範囲に含まれる必要があります。このポート番号は、ゲームセッションオブジェクトとプレイヤーセッションオブジェクトに含まれ、ゲームセッションがサーバープロセスに接続するときに使用します。

型: 整数

必須: はい

logParameters

ゲームセッションログファイルへのディレクトリパスのリストを含むオブジェクト。

タイプ: Aws::GameLift::Server::LogParameters

必須: いいえ

onStartGameSession

Amazon GameLift サービスが新しいゲームセッションをアクティブにするために呼び出すコールバック関数名。Amazon GameLift はクライアントのリクエスト CreateGameSession に応じ、この関数を呼び出します。コールバック関数は GameSession オブジェクト (Amazon GameLift サービス API リファレンスで定義) を渡します。

タイプ: `const std::function<void(Aws::GameLift::Model::GameSession)> onStartGameSession`

必須: はい

onProcessTerminate

Amazon GameLift サービスがサーバープロセスを強制シャットダウンするために呼び出すコールバック関数名。この関数を呼び出した後、Amazon GameLift はサーバープロセスがシャットダウンするのを 5 分間待ち、ProcessEnding() 呼び出しで応答します。応答がない場合は、受信サーバープロセスをシャットダウンします。

タイプ: `std::function<void()> onProcessTerminate`

必須: いいえ

onHealthCheck

Amazon GameLift サービスがサーバープロセスにヘルスステータスレポートをリクエストするために呼び出すコールバック関数名。Amazon GameLift は、この関数を 60 秒ごとに呼び出します。この関数を呼び出した後、Amazon GameLift はレスポンスを 60 秒ほど待ちます。レスポンスがなければ、サーバープロセスを異常と記録します。

タイプ: `std::function<bool()> onHealthCheck`

必須: いいえ

onUpdateGameSession

Amazon GameLift サービスが更新されたゲームセッションオブジェクトを提供するために呼び出すコールバック関数名。Amazon GameLift はこの関数を、更新されたマッチメーカーデータを提供するためにマッチバックフィルリクエストを処理するときに呼び出します。GameSession オブジェクト、ステータス更新 (updateReason)、およびマッチバックフィルチケット ID が渡されます。

タイプ:
`std::function<void(Aws::GameLift::Server::Model::UpdateGameSession)>`
`onUpdateGameSession`

必須: いいえ

StartMatchBackfillRequest

このデータ型はマッチメーキングバックフィルリクエストの送信に使用します。この情報は、StartMatchBackfill() 呼び出しで Amazon GameLift サービスに伝えられます。

目次

GameSessionArn

一意のゲームセッション識別子。API アクション GetGameSessionId() は ARN 形式の識別子を返します。

型: 文字列

必須: はい

MatchmakingConfigurationArn

このリクエストに使用されるマッチメーカーの ARN 形式の一意な識別子。元のゲームセッションの作成に使用されたマッチメーカーを見つけるには、ゲームセッションオブジェクトのマッチメーカーデータプロパティを確認します。マッチメーカーデータの詳細については「マッチメーカーデータの処理」を参照してください。

型: 文字列

必須: はい

プレイヤー

現在ゲームセッションに参加しているすべてのプレイヤーを表すデータのセット。マッチメーカーはこの情報を使用して、現在のプレイヤーとマッチする新しいプレイヤーを検索します。プレイヤーオブジェクトの形式の説明については、Amazon GameLift API リファレンスガイドを参照してください。プレイヤー属性、ID、チームの割り当てを見つけるには、マッチメーカーデータプロパティのゲームセッションオブジェクトを参照してください。マッチメーカーでレイテンシーが使用されている場合は、現在のリージョンの更新されたレイテンシーを収集し、それを各プレイヤーのデータに含めます。

型: std:vector<player>

必須: はい

TicketId

マッチメーキングまたはバックフィルリクエストチケットの一意の識別子。ここで値を指定しない場合、Amazon GameLift によって UUID 形式で自動的に生成されます。この識別子を使用してマッチバックフィルチケットのステータスを追跡したり、必要に応じてリクエストをキャンセルしたりします。

型: 文字列

必須: いいえ

StopMatchBackfillRequest

このデータ型はマッチメーキングバックフィルリクエストのキャンセルに使用します。この情報は、StopMatchBackfill() 呼び出しで Amazon GameLift サービスに伝えられます。

目次

GameSessionArn

キャンセルされるリクエストに関連付けられた一意のゲームセッション識別子。

型: 文字列

必須: はい

MatchmakingConfigurationArn

このリクエストが送信されたマッチメーカーの一意の識別子。

型: 文字列

必須: はい

TicketId

キャンセルされるバックフィルリクエストチケットの一意の識別子。

型: 文字列

必須: はい

C# 用 Amazon GameLift サーバー SDK リファレンス

この Amazon GameLift C# サーバー SDK リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

トピック

- C# と Unity 用 Amazon GameLift サーバー SDK 5.x リファレンス
- C# 用 Amazon GameLift サーバー SDK 4.x リファレンス

C# と Unity 用 Amazon GameLift サーバー SDK 5.x リファレンス

この Amazon GameLift C++ サーバー SDK 5.x リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。Unity 用 C# サーバー SDK プラグインの使用に関する情報については、「Unity プロジェクトに Amazon GameLift を統合する」を参照してください。C# 用 Amazon GameLift サーバー SDK 5.x は .NET 4.6 と.NET 6 をサポートしています。

トピック

- C# および Unity の Amazon GameLift サーバー SDK リファレンス: アクション
- C# および Unity 用 Amazon GameLift サーバー SDK リファレンス: データ型

C# および Unity の Amazon GameLift サーバー SDK リファレンス: アクション

この Amazon GameLift C# サーバー SDK リファレンスは、Amazon で使用するマルチプレイヤーゲームを準備するのに役立ちます GameLift。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。Unity 用 C# サーバー SDK プラグインの使用に関する情報については、「Unity プロジェクトに Amazon GameLift を統合する」を参照してください。

アクション

- GetSdkVersion()
- InitSDK()
- InitSDK()
- ProcessReady()

- ProcessEnding()
- ActivateGameSession()
- UpdatePlayerSessionCreationPolicy()
- GetGameSessionId()
- GetTerminationTime()
- AcceptPlayerSession()
- RemovePlayerSession()
- DescribePlayerSessions()
- StartMatchBackfill()
- StopMatchBackfill()
- GetComputeCertificate()
- GetFleetRoleCredentials()
- Destroy()

GetSdkVersion()

サーバープロセスに組み込まれた SDK の現在のバージョン番号を返します。

Syntax

```
AwsStringOutcome GetSdkVersion();
```

戻り値

成功した場合、the section called “AwsStringOutcome” オブジェクトとして現在の SDK バージョンを返します。返される文字列は、バージョン番号のみを含みます。(例: `5.0.0`) 成功しなかった場合、エラーメッセージを返します。

例

```
var getSdkVersionOutcome = GameLiftServerAPI.GetSdkVersion();
```

InitSDK()

マネージド EC2 フリートの Amazon GameLift SDK を初期化します。Amazon に関連する他の初期化 GameLift が発生する前に、起動時にこのメソッドを呼び出します。このメソッドは、ホスト環

境からサーバーパラメータを読み取り、サーバーと Amazon GameLift サービス間の通信を設定します。

Syntax

```
GenericOutcome InitSDK();
```

戻り値

成功すると、サーバープロセスが を呼び出す準備ができていることを示す InitSdkOutcome オブジェクトが返されますProcessReady()。

例

```
//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
GenericOutcome initSDKOutcome = GameLiftServerAPI.InitSDK();
```

InitSDK()

Anywhere フリートの Amazon GameLift SDK を初期化します。Amazon に関連する他の初期化 GameLift が発生する前に、起動時にこのメソッドを呼び出します。この方法では、サーバーと Amazon GameLift サービス間の通信を設定するには、明示的なサーバーパラメータが必要です。

Syntax

```
GenericOutcome InitSDK(ServerParameters serverParameters);
```

パラメータ

ServerParameters

Amazon GameLift Anywhere フリートでゲームサーバーを初期化するには、次の情報を使用して ServerParameters オブジェクトを作成します。

- ゲームサーバーへの接続 WebSocket に使用される の URL。
- ゲームサーバーのホストに使用されるプロセスの ID。
- ゲームサーバープロセスをホスティングするコンピューティングの ID。
- Amazon コンピューティングを含む Amazon GameLift Anywhere GameLift フリートの ID。
- Amazon GameLift オペレーションによって生成された認証トークン。

戻り値

成功すると、サーバープロセスが を呼び出す準備ができていることを示す InitSdkOutcome オブジェクトが返されますProcessReady()。

> **Note**
>
> Anywhere フリートにデプロイされたゲームビルドに対して InitSDK() への呼び出しが失敗する場合は、ビルドリソースの作成時に使用した ServerSdkVersion パラメータを確認してください。この値は、使用中のサーバー SDK バージョンに明示的に設定する必要があります。このパラメータのデフォルト値は 4.x で、互換性がありません。この問題を解決するには、新しいビルドを作成して新しいフリートにデプロイします。

例

```
//Define the server parameters
string websocketUrl = "wss://us-west-1.api.amazongamelift.com";
string processId = "PID1234";
string fleetId = "aarn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa";
string hostId = "HardwareAnywhere";
string authToken = "1111aaaa-22bb-33cc-44dd-5555eeee66ff";
ServerParameters serverParameters =
  new ServerParameters(webSocketUrl, processId, hostId, fleetId, authToken);

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
GenericOutcome initSDKOutcome = GameLiftServerAPI.InitSDK(serverParameters);
```

ProcessReady()

サーバープロセス GameLift がゲームセッションをホストする準備ができたことを Amazon に通知します。InitSDK() を呼び出した後にこのメソッドを呼び出します このメソッドは、プロセスごとに 1 回だけ呼び出す必要があります。

Syntax

```
GenericOutcome ProcessReady(ProcessParameters processParameters)
```

パラメータ

ProcessParameters

ProcessParameters オブジェクトにはサーバープロセスに関する情報が保持されます。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、メソッドと委任関数の実装の両方を示します。

```
// Set parameters and call ProcessReady
ProcessParameters processParams = new ProcessParameters(
  this.OnStartGameSession,
  this.OnProcessTerminate,
  this.OnHealthCheck,
  this.OnUpdateGameSession,
  port,
  new LogParameters(new List<string>()
  // Examples of log and error files written by the game server
  {
    "C:\\game\\logs",
    "C:\\game\\error"
  })
);
GenericOutcome processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);
```

ProcessEnding()

サーバープロセスが終了中であることを Amazon に通知 GameLift します。他のすべてのクリーンアップタスク (アクティブなゲームセッションのシャットダウンを含む) の後、およびプロセスを終了する前に、このメソッドを呼び出します。ProcessEnding( ) の結果に応じて、プロセスは成功 (0) またはエラー (-1) で終了し、フリートイベントを生成します。プロセスがエラーで終了した場合、生成されるフリートイベントは ですSERVER_PROCESS_TERMINATED_UNHEALTHY。

Syntax

```
GenericOutcome ProcessEnding()
```

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、サーバープロセスを終了する前に、成功またはエラーの終了コードで ProcessEnding() と Destroy() を呼び出します。

```
GenericOutcome processEndingOutcome = GameLiftServerAPI.ProcessEnding();
GameLiftServerAPI.Destroy();

if (processEndingOutcome.Success)
  {
    Environment.Exit(0);
  }
else
  {
    Console.WriteLine("ProcessEnding() failed. Error: " +
 processEndingOutcome.Error.ToString());
    Environment.Exit(-1);
  }
```

ActivateGameSession()

サーバープロセスがゲームセッションをアクティブ化し、プレイヤー接続を受信する準備ができた GameLift ことを Amazon に通知します。このアクションは、すべてのゲームセッションの初期化の後、onStartGameSession() コールバック関数の一部として呼び出されます。

Syntax

```
GenericOutcome ActivateGameSession()
```

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、ActivateGameSession() が onStartGameSession() 委任関数の一部として呼び出されていることを示しています。

```
void OnStartGameSession(GameSession gameSession)
{
  // game-specific tasks when starting a new game session, such as loading map
  // When ready to receive players
  GenericOutcome activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}
```

UpdatePlayerSessionCreationPolicy()

現在のゲームセッションの機能を更新し、新しいプレイヤーセッションを承諾します。ゲームセッションは、新しいプレイヤーセッションをすべて受け入れるか拒否するかを設定できます。

Syntax

```
GenericOutcome UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy
 playerSessionPolicy)
```

パラメータ

playerSessionPolicy

ゲームセッションで新しいプレイヤーを承諾するかどうかを示す文字列値。

有効な値を次に示します。

- ACCEPT_ALL – すべての新しいプレイヤーセッションを承諾します。
- DENY_ALL – すべての新しいプレイヤーセッションを拒否します。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例は、現在のゲームセッションの参加ポリシーを、すべてのプレイヤーを受け入れるように設定します。

```
GenericOutcome updatePlayerSessionPolicyOutcome =

 GameLiftServerAPI.UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy.ACCEPT_ALL);
```

GetGameSessionId()

アクティブなサーバープロセスにホストされたゲームセッションの ID を取得します。

ゲームセッションでアクティブ化されていないアイドル状態のプロセスの場合、呼び出しは the section called “GameLiftError” を返します。

Syntax

```
AwsStringOutcome GetGameSessionId()
```

戻り値

成功した場合、ゲームセッション ID を the section called “AwsStringOutcome” オブジェクトとして返します。成功しなかった場合、エラーメッセージを返します。

例

```
AwsStringOutcome getGameSessionIdOutcome = GameLiftServerAPI.GetGameSessionId();
```

GetTerminationTime()

終了時刻が判る場合に、サーバープロセスがシャットダウンを予定している時刻を返します。サーバープロセスは、Amazon から onProcessTerminate()コールバックを受信した後、このアクションを実行します GameLift。Amazon onProcessTerminate() は、次の理由で を GameLift 呼び出します。

- サーバープロセスが正常性の低下を報告した場合、または Amazon に応答しなかった場合 GameLift。
- スケールダウンイベント中にインスタンスを終了する場合。
- スポットインスタンスの中断によりインスタンスが終了した場合。

Syntax

```
AwsDateTimeOutcome GetTerminationTime()
```

戻り値

成功した場合、終了時刻を the section called “AwsDateTimeOutcome” オブジェクトとして返します。値は終了時間で、0001 00:00:00 以降の経過ティックで表現されます。例えば、日付時刻の

値 2020-09-13 12:26:40 -000Z は、637355968000000000 ティックに等しくなります。終了時間がない場合は、エラーメッセージを返します。

例

```
AwsDateTimeOutcome getTerminationTimeOutcome = GameLiftServerAPI.GetTerminationTime();
```

AcceptPlayerSession()

指定されたプレイヤーセッション ID を持つプレイヤーがサーバープロセスに接続し、検証が必要である GameLift ことを Amazon に通知します。Amazon は、プレイヤーセッション ID が有効 GameLift であることを確認します。プレイヤーセッションが検証されると、Amazon はプレイヤースロットのステータスを RESERVED から ACTIVE GameLift に変更します。

Syntax

```
GenericOutcome AcceptPlayerSession(String playerSessionId)
```

パラメータ

playerSessionId

新しいプレイヤーセッションの作成 GameLift 時に によって発行される一意の ID。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、無効なプレイヤーセッション ID の検証や拒否を含む、接続リクエストを処理するための関数を示します。

```
void ReceiveConnectingPlayerSessionID (Connection connection, String playerSessionId)
{
  GenericOutcome acceptPlayerSessionOutcome =
 GameLiftServerAPI.AcceptPlayerSession(playerSessionId);
  if(acceptPlayerSessionOutcome.Success)
  {
    connectionToSessionMap.emplace(connection, playerSessionId);
    connection.Accept();
  }
```

```
  else
  {
    connection.Reject(acceptPlayerSessionOutcome.Error.ErrorMessage);
  }
}
```

RemovePlayerSession()

プレイヤーがサーバープロセスから切断された GameLift ことを Amazon に通知します。それに応じて、Amazon GameLift はプレイヤースロットを利用可能に変更します。

Syntax

```
GenericOutcome RemovePlayerSession(String playerSessionId)
```

パラメータ

playerSessionId

新しいプレイヤーセッションの作成 GameLift 時に Amazon によって発行される一意の ID。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
GenericOutcome removePlayerSessionOutcome =
 GameLiftServerAPI.RemovePlayerSession(playerSessionId);
```

DescribePlayerSessions()

設定、セッションメタデータ、プレイヤーデータを含む、プレイヤーセッションデータを取得します。このアクションを使用して、単一のプレイヤーセッション、ゲームセッション内のすべてのプレイヤーセッション、または単一のプレイヤー ID に関連付けられたすべてのプレイヤーセッションに関する情報を取得します。

Syntax

```
DescribePlayerSessionsOutcome DescribePlayerSessions(DescribePlayerSessionsRequest
 describePlayerSessionsRequest)
```

パラメータ

DescribePlayerSessionsRequest

取得するプレイヤーセッションを記述する the section called “DescribePlayerSessionsRequest” オブジェクト。

戻り値

成功した場合は、リクエストのパラメータに適合したプレイヤーセッションオブジェクトのセットを含む the section called “DescribePlayerSessionsOutcome” オブジェクトを返します。

例

この例は、指定したゲームセッションにアクティブに接続されているすべてのプレイヤーセッションのリクエストを示しています。を省略NextTokenし、Limit 値を 10 に設定することで、Amazon GameLift はリクエストに一致する最初の 10 個のプレイヤーセッションレコードを返します。

```
// Set request parameters
DescribePlayerSessionsRequest describePlayerSessionsRequest = new
 DescribePlayerSessionsRequest()
{
  GameSessionId = GameLiftServerAPI.GetGameSessionId().Result,    //gets the ID for the
 current game session
  Limit = 10,
  PlayerSessionStatusFilter =
    PlayerSessionStatusMapper.GetNameForPlayerSessionStatus(PlayerSessionStatus.ACTIVE)
};
// Call DescribePlayerSessions
DescribePlayerSessionsOutcome describePlayerSessionsOutcome =
  GameLiftServerAPI.DescribePlayerSessions(describePlayerSessionsRequest);
```

StartMatchBackfill()

FlexMatch で作成されたゲームセッションの空きスロット用に新規プレイヤーを検索するリクエストを送信します。詳細については、 FlexMatch「バックフィル機能」を参照してください。

このアクションは非同期です。新しいプレイヤーがマッチングされると、Amazon はコールバック関数 を使用して更新されたマッチメーカーデータを GameLift 配信しますOnUpdateGameSession()。

サーバープロセスではアクティブなマッチバックフィルリクエストは一度に 1 つだけです。新しいリクエストを送信するには、まず StopMatchBackfill() を呼び出して元のリクエストをキャンセルする必要があります。

Syntax

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest
 startBackfillRequest);
```

パラメータ

StartMatchBackfillRequest

StartMatchBackfillRequest オブジェクトにはバックフィルリクエストに関する情報が保持されます。

戻り値

the section called “StartMatchBackfillOutcome” オブジェクトを、マッチバックフィルチケット ID またはエラーメッセージを伴うエラーとともに返します。

例

```
// Build a backfill request
StartMatchBackfillRequest startBackfillRequest = new StartMatchBackfillRequest()
{
  TicketId = "1111aaaa-22bb-33cc-44dd-5555eeee66ff", //optional
  MatchmakingConfigurationArn = "arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig",
  GameSessionId = GameLiftServerAPI.GetGameSessionId().Result,    // gets ID for
 current game session
  MatchmakerData matchmakerData =
    MatchmakerData.FromJson(gameSession.MatchmakerData),  // gets matchmaker data for
 current players
  // get matchmakerData.Players
  // remove data for players who are no longer connected
  Players = ListOfPlayersRemainingInTheGame
};

// Send backfill request
StartMatchBackfillOutcome startBackfillOutcome =
 GameLiftServerAPI.StartMatchBackfill(startBackfillRequest);
```

```
// Implement callback function for backfill
void OnUpdateGameSession(GameSession myGameSession)
{
  // game-specific tasks to prepare for the newly matched players and update matchmaker
 data as needed
}
```

StopMatchBackfill()

アクティブなマッチバックフィルリクエストをキャンセルします。詳細については、FlexMatch「バックフィル機能」を参照してください。

Syntax

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

パラメータ

StopMatchBackfillRequest

停止しているマッチメーキングチケットに関する詳細を提供する StopMatchBackfillRequest オブジェクト。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
// Set backfill stop request parameters
StopMatchBackfillRequest stopBackfillRequest = new StopMatchBackfillRequest(){
  TicketId = "1111aaaa-22bb-33cc-44dd-5555eeee66ff", //optional, if not provided one is
 autogenerated
  MatchmakingConfigurationArn = "arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig",
  GameSessionId = GameLiftServerAPI.GetGameSessionId().Result    //gets the ID for the
 current game session
};
GenericOutcome stopBackfillOutcome =
 GameLiftServerAPI.StopMatchBackfillRequest(stopBackfillRequest);
```

GetComputeCertificate()

ゲームサーバーとゲームクライアント間のネットワーク接続を暗号化するために使用される TLS 証明書へのパスを取得します。コンピューティングデバイスを Amazon GameLift Anywhere フリートに登録するときに、証明書パスを使用できます。詳細については、「」を参照してくださいRegisterCompute。

Syntax

```
GetComputeCertificateOutcome GetComputeCertificate();
```

戻り値

以下を含む GetComputeCertificateResponse オブジェクトを返します。

- CertificatePath: コンピューティングリソースの TLS 証明書へのパス。Amazon GameLift マネージドフリートを使用する場合、このパスには以下が含まれます。
  - `certificate.pem`: エンドユーザー証明書。証明書チェーン全体は、この証明書に追加された `certificateChain.pem` を組み合わせたものです。
  - `certificateChain.pem`: ルート証明書と中間証明書を含む証明書チェーン。
  - `rootCertificate.pem`: ルート証明書。
  - `privateKey.pem`: エンドユーザー証明書のプライベートキー。
- ComputeName: コンピューティングリソースの名前。

例

```
GetComputeCertificateOutcome getComputeCertificateOutcome =
 GameLiftServerAPI.GetComputeCertificate();
```

GetFleetRoleCredentials()

Amazon が他の とやり取り GameLift することを許可する IAM ロール認証情報を取得しますAWS のサービス。詳細については、「フリートの他の AWS リソースと通信する」を参照してください。

構文

```
GetFleetRoleCredentialsOutcome GetFleetRoleCredentials(GetFleetRoleCredentialsRequest
 request);
```

パラメータ

GetFleetRoleCredentialsRequest

AWS リソースへの制限付きアクセスをゲームサーバーにまで拡張するロール認証情報。

戻り値

the section called “GetFleetRoleCredentialsOutcome” オブジェクトを返します。

例

```
// form the fleet credentials request
GetFleetRoleCredentialsRequest getFleetRoleCredentialsRequest = new
 GetFleetRoleCredentialsRequest(){
  RoleArn = "arn:aws:iam::123456789012:role/service-role/exampleGameLiftAction"
};
GetFleetRoleCredentialsOutcome GetFleetRoleCredentialsOutcome credentials =
 GetFleetRoleCredentials(getFleetRoleCredentialsRequest);
```

Destroy()

Amazon GameLift ゲームサーバー SDK をメモリから解放します。ベストプラクティスとして、ProcessEnding() の後、かつプロセスの終了前にこのメソッドを呼び出します。Anywhere フリートを使用していて、すべてのゲームセッション後にサーバープロセスを終了しない場合は、InitSDK()を呼び出しDestroy()てから再初期化してから、プロセスが でゲームセッションをホストする準備ができ GameLift たことを Amazon に通知しますProcessReady()。

Syntax

```
GenericOutcome Destroy()
```

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
// Operations to end game sessions and the server process
GenericOutcome processEndingOutcome = GameLiftServerAPI.ProcessEnding();
```

```
// Shut down and destroy the instance of the GameLift Game Server SDK
GenericOutcome destroyOutcome = GameLiftServerAPI.Destroy();

// Exit the process with success or failure
if (processEndingOutcome.Success)
  {
    Environment.Exit(0);
  }
else
  {
    Console.WriteLine("ProcessEnding() failed. Error: " +
 processEndingOutcome.Error.ToString());
    Environment.Exit(-1);
  }
```

C# および Unity 用 Amazon GameLift サーバー SDK リファレンス: データ型

この Amazon GameLift C# サーバー SDK リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。Unity 用 C# サーバー SDK プラグインの使用に関する情報については、「Unity プロジェクトに Amazon GameLift を統合する」を参照してください。

データ型

- LogParameters
- ProcessParameters
- UpdateGameSession
- GameSession
- ServerParameters
- StartMatchBackfillRequest
- プレイヤー
- DescribePlayerSessionsRequest
- StopMatchBackfillRequest
- GetFleetRoleCredentialsRequest
- AttributeValue
- AwsStringOutcome

- GenericOutcome
- DescribePlayerSessionsOutcome
- DescribePlayerSessionsResult
- PlayerSession
- StartMatchBackfillOutcome
- StartMatchBackfillResult
- GetComputeCertificateOutcome
- GetComputeCertificateResult
- GetFleetRoleCredentialsOutcome
- GetFleetRoleCredentialsResult
- AwsDateTimeOutcome
- GameLiftError
- 列挙型

LogParameters

このデータ型は、ゲームセッション中に生成されたファイルのうち、ゲームセッション終了時にAmazon GameLiftにアップロードするファイルを識別するのに使用されます。この情報は、ProcessReady() 呼び出しで `LogParameters` to Amazon GameLift に伝えられます。

[Properties] (プロパティ)	説明
LogPaths	Amazon GameLift で将来のアクセスに備えて保存するゲームサーバーログファイルへのディレクトリパスのリスト。サーバープロセスは各ゲームセッション中にこれらのファイルを生成します。ファイルのパスと名前はゲームサーバーで定義し、ルートゲームビルドディレクトリに保存します。 ログパスは絶対パスである必要があります。例えば、ゲームビルドによって `MyGame\sessionLogs\` などのパスに保存されるゲームセッションログの場合、パスは `c:\game\M`

	yGame\sessionLogs (Windows インスタンスの場合) となります。 [Type] (タイプ): `List<String>` 必須: いいえ

ProcessParameters

このデータ型には、ProcessReady() で呼び出しで Amazon GameLift に送られたパラメータセットが含まれます。

[Properties] (プロパティ)	説明
LogParameters	ゲームセッションログファイルへのディレクトリパスのリストを含むオブジェクト。 [Type] (タイプ): `Aws::GameLift::Server:: LogParameters` 必須: はい
OnHealthCheck	Amazon GameLift がサーバープロセスにヘルスステータスレポートをリクエストするために呼び出すコールバック関数名。Amazon GameLift は、この関数を 60 秒ごとに呼び出します。この関数を呼び出した後、Amazon GameLift はレスポンスを 60 秒ほど待ちます。Amazon GameLift は、レスポンスがなければサーバープロセスを異常と記録します。 [Type] (タイプ): `void OnHealthCheckDelegate()` 必須: はい
OnProcessTerminate	Amazon GameLift がサーバープロセスを強制シャットダウンするために呼び出すコールバック関数名。この関数を呼び出すと、Amazon

	GameLift はサーバープロセスがシャットダウンするために 5 分間待ち、サーバープロセスをシャットダウンする前に ProcessEnding() 呼び出しで応答します。 [Type] (タイプ): `void OnProcessTerminateDelegate()` 必須: はい
OnStartGameSession	Amazon GameLift が新しいゲームセッションをアクティブにするために呼び出すコールバック関数名。Amazon GameLift はクライアントのリクエスト CreateGameSession に応じ、この関数を呼び出します。コールバック関数は、Amazon GameLift API リファレンスで定義された GameSession オブジェクトを渡します。 [Type] (タイプ): `void OnStartGameSessionDelegate( GameSession )` 必須: はい
OnUpdateGameSession	Amazon GameLift が更新されたゲームセッションオブジェクトを提供するために呼び出すコールバック関数名。Amazon GameLift はこの関数を、更新されたマッチメーカーデータを提供するためにマッチバックフィルリクエストを処理するときに呼び出します。GameSession オブジェクト、ステータス更新 (`updateReason`)、およびマッチバックフィルチケット ID が渡されます。 タイプ: void OnUpdateGameSessionDelegate (UpdateGameSession) 必須: いいえ

ポート	サーバープロセスが新しいプレイヤーの接続をリスンするポート番号。値は、このゲームサーバービルドをデプロイするすべてのフリートで設定されているポート番号の範囲に含まれる必要があります。このポート番号は、ゲームセッションオブジェクトとプレイヤーセッションオブジェクトに含まれ、ゲームセッションがサーバープロセスに接続するときに使用します。 [Type] (タイプ): `Integer` 必須: はい

UpdateGameSession

ゲームセッションオブジェクトの更新情報。ゲームセッションが更新された理由も含まれます。更新がマッチバックフィルアクションに関連する場合、このデータタイプにはバックフィルチケット ID が含まれます。

プロパティ	説明
GameSession	Amazon GameLift API によって定義された GameSession オブジェクト。`GameSession` オブジェクトにはゲームセッションを説明するプロパティが含まれています。 [Type] (タイプ): `GameSession GameSession()` 必須: はい
UpdateReason	ゲームセッションが更新されている理由。 [Type] (タイプ): `UpdateReason UpdateReason()` 必須: はい

プロパティ	説明
BackfillTicketId	ゲームセッションの更新を試みるバックフィルチケットの ID。 [Type] (タイプ): `String` 必須: はい

GameSession

ゲームセッションの詳細。

プロパティ	説明
GameSessionId	ゲームセッションの一意の識別子。ゲームセッション ARN の形式 は `arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token>` です。 [Type] (タイプ): `String` 必須: いいえ
名前	ゲームセッションについて説明するラベル。 [Type] (タイプ): `String` 必須: いいえ
FleetId	ゲームセッションが実行されているフリートの一意の識別子。 [Type] (タイプ): `String` 必須: いいえ
MaximumPlayerSessionCount	ゲームセッションへのプレーヤー接続の最大数。

プロパティ	説明
	[Type] (タイプ): `Integer` 必須: いいえ
ポート	ゲームセッションのポート番号。Amazon GameLift ゲームサーバーに接続するには、アプリに IP アドレスとポート番号の両方が必要です。 [Type] (タイプ): `Integer` 必須: いいえ
IP アドレス	ゲームセッションの IP アドレス。Amazon GameLift ゲームサーバーに接続するには、アプリに IP アドレスとポート番号の両方が必要です。 [Type] (タイプ): `String` 必須: いいえ
GameSessionData	単一の文字列値としてフォーマットされたカスタムゲームセッションプロパティのセット。 [Type] (タイプ): `String` 必須: いいえ

プロパティ	説明
MatchmakerData	ゲームセッションの作成に使用されたマッチメーキングプロセスに関する情報。JSON 構文で、文字列としてフォーマットされています。使用されたマッチメーキング設定に加えて、プレイヤー属性やチーム割り当てなど、マッチに割り当てられた全プレイヤーに関するデータが含まれます。 [Type] (タイプ): `String` 必須: いいえ
GameProperties	ゲームセッションのカスタムプロパティのセットで、キーと値のペアとしてフォーマットされます。これらのプロパティは、新しいゲームセッションを開始するリクエストとともに渡されます。 [Type] (タイプ): `Dictionary<string, string>` 必須: いいえ

プロパティ	説明
DnsName	ゲームセッションを実行しているインスタンスに割り当てられた DNS 識別子。値の形式は次のとおりです。 • TLS 対応フリート: `<unique identifier>.<region identifier>.amazongamelift.com` 。 • TLS 対応でないフリート: `ec2-<unique identifier>.compute.amazonaws.com` 。 TLS 対応フリートで実行しているゲームセッションに接続する場合、IP アドレスではなく DNS 名を使用する必要があります。 [Type] (タイプ): `String` 必須: いいえ

ServerParameters

Amazon GameLift Anywhere サーバーと Amazon GameLift サービス間の接続を維持するために使用される情報。この情報は、InitSDK() で新しいサーバープロセスを起動するときに使用されます。Amazon GameLift マネージド EC2 インスタンスでホストされているサーバーには、空のオブジェクトを使用してください。

プロパティ	説明
WebSocketUrl	Amazon GameLift Anywhere の一部として `RegisterCompute` したときに返された `GameLiftServerSdkEndpoint` 。 [Type] (タイプ): `String` 必須: はい

プロパティ	説明
ProcessId	ゲームをホストするサーバープロセスに登録された固有の識別子。 [Type] (タイプ): `String` 必須: はい
HostId	ゲームをホストするサーバープロセスのホスト固有の識別子。hostId はコンピューティングを登録したときに使用される ComputeName です。詳細については、「RegisterCompute」を参照してください。 [Type] (タイプ): `String` 必須: はい
FleetId	コンピューティングが登録されているフリートのフリート ID。詳細については、「RegisterCompute」を参照してください。 [Type] (タイプ): `String` 必須: はい
AuthToken	Amazon GameLift によって生成された認証トークンで、Amazon GameLift に対してサーバーを認証します。詳細については、「GetComputeAuthToken」を参照してください。 [Type] (タイプ): `String` 必須: はい

StartMatchBackfillRequest

マッチメーキングバックフィルリクエストの作成に使用される情報。この情報は、StartMatchBackfill() 呼び出しで Amazon GameLift に伝えられます。

プロパティ	説明
GameSessionArn	一意のゲームセッション識別子。API オペレーション GetGameSessionId は ARN 形式の識別子を返します。 [Type] (タイプ): `String` 必須: はい
MatchmakingConfigurationArn	このリクエストに使用されるマッチメーカーの ARN 形式の一意な識別子。元のゲームセッションののマッチメーカー ARN は、マッチメーカーデータプロパティのゲームセッションオブジェクトにあります。マッチメーカーデータの詳細については「マッチメーカーデータの処理」を参照してください。 [Type] (タイプ): `String` 必須: はい
プレイヤー	現在ゲームセッションに参加しているすべてのプレイヤーを表すデータのセット。マッチメーカーはこの情報を使用して、現在のプレイヤーとマッチする新しいプレイヤーを検索します。 [Type] (タイプ): `List<Player>` 必須: はい
TicketId	マッチメーキングまたはバックフィルリクエストチケットの一意の識別子。値を入力していない場合、Amazon GameLift によって値が生成されます。この識別子を使用してマッチバック

プロパティ	説明
	フィルチケットのステータスを追跡したり、必要に応じてリクエストをキャンセルしたりします。 [Type] (タイプ): `String` 必須: いいえ

プレイヤー

マッチメーキングのプレイヤーを表します。マッチメーキングリクエストを開始すると、プレイヤーはプレイヤー ID、属性、場合によってはレイテンシーデータを保有します。Amazon GameLift は、マッチが行われた後にチーム情報を追加します。

プロパティ	説明
LatencyInMS	プレイヤーがロケーションに接続したときに発生するレイテンシーの量を示すミリ秒単位の値のセット。 このプロパティを使用すると、プレーヤーはリストに表示されている場所でのみマッチングされます。マッチメーカーにプレイヤーレイテンシーを評価するルールがある場合、プレイヤーはレイテンシーを報告しないとマッチングされません。 [Type] (タイプ): `Dictionary<string, int>` 必須: いいえ
PlayerAttributes	マッチメーキングに使用するプレイヤー情報を含むキーと値のペアの集合。プレイヤー属性キーは、マッチメーキングルールセットで使用されているプレイヤー属性と一致する必要があります。

プロパティ	説明
	プレイヤー属性の詳細については、「Attribute Value」を参照してください。 [Type] (タイプ): `Dictionary<string, AttributeValue` 必須: いいえ
PlayerId	プレイヤーを表す一意の識別子。 [Type] (タイプ): `String` 必須: いいえ
Team	マッチでプレーヤーが割り当てられるチームの名前。チーム名はマッチメーキングルールセットで定義します。 [Type] (タイプ): `String` 必須: いいえ

DescribePlayerSessionsRequest

このデータ型は、取得するプレイヤーセッションを指定するのに使用されます。複数の方法で使用できます。(1) 特定のプレイヤーセッションをリクエストする PlayerSessionId; を使用します。(2) 指定したゲームセッションのすべてのプレイヤーをリクエストするには、GameSessionId を指定します。または、(3) 指定したプレイヤーのすべてのプレイヤーセッションをリクエストするには、PlayerId を指定します。プレイヤーセッション数が多い場合は、ページ分割パラメータを使用して結果を順次ページとして取得します。

プロパティ	説明
GameSessionId	一意のゲームセッション識別子。このパラメータを使用して、指定したゲームセッションのすべてのプレイヤーセッションをリクエストします。ゲームセッション ID の形式

プロパティ	説明
	は、arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string> です。<ID string> の値は、カスタム ID 文字列または (ゲームセッション作成時に指定した場合) 生成された文字列のいずれかです。 [Type] (タイプ): String 必須: いいえ
PlayerSessionId	プレイヤーセッションを表す一意の識別子。 [Type] (タイプ): String 必須: いいえ
PlayerId	プレイヤーの一意識別子。「プレイヤー ID を生成する」を参照してください。 [Type] (タイプ): String 必須: いいえ

プロパティ	説明
PlayerSessionStatusFilter	結果をフィルタリングするプレイヤーセッションステータス。可能なプレイヤーセッションステータスとして以下のステータスがあります。 • RESERVED – プレイヤーセッションリクエストは受領されましたが、プレイヤーのサーバープロセスへの接続や検証はまだ行われていません。 • ACTIVE – プレイヤーはサーバープロセスによって検証され、現時点で接続されています。 • COMPLETED – プレイヤー接続は削除されました。 • TIMEDOUT – プレイヤーセッションリクエストは受領されましたが、タイムアウト制限 (60 秒) 内でのプレイヤーの接続や検証は行われていません。 [Type] (タイプ): `String` 必須: いいえ
NextToken	結果の次のページの先頭を示すトークン。結果セットの先頭を指定するには、値を指定しないでください。プレイヤーセッション ID を提供する場合、このパラメータは無視されます。 [Type] (タイプ): `String` 必須: いいえ

プロパティ	説明
制限	返される結果の最大数。プレイヤーセッション ID を提供する場合、このパラメータは無視されます。 [Type] (タイプ): `int` 必須: いいえ

StopMatchBackfillRequest

マッチメーキングバックフィルリクエストのキャンセルに使用される情報。この情報は、StopMatchBackfill() 呼び出しで Amazon GameLift サービスに伝えられます。

プロパティ	説明
GameSessionArn	キャンセルされるリクエストの一意のゲームセッション識別子。 [Type] (タイプ): `string` 必須: はい
MatchmakingConfigurationArn	このリクエストが送信されたマッチメーカーの一意の識別子。 [Type] (タイプ): `string` 必須: はい
TicketId	キャンセルされるバックフィルリクエストチケットの一意の識別子。 [Type] (タイプ): `string` 必須: はい

GetFleetRoleCredentialsRequest

このデータ型により、ゲームサーバーは他の AWS リソースへのアクセスが制限されます。詳細については、「Amazon の IAM サービスロールを設定する GameLift」を参照してください。

プロパティ	説明
RoleArn	AWS リソースへの制限付きアクセスを拡張するサービスロールの Amazon リソースネーム (ARN)。 [Type] (タイプ): `string` 必須: はい
RoleSessionName	ロール認証情報の使用を説明するセッションの名前。 [Type] (タイプ): `string` 必須: いいえ

AttributeValue

これらの値を プレイヤー 属性のキーと値のペアで使用します。このオブジェクトでは、文字列、数値、文字列配列、データマップのいずれかの有効なデータ型を使用して属性値を指定できます。各 `AttributeValue` オブジェクトは、使用可能なプロパティのうちの 1 つだけを使用できます。

プロパティ	説明
attrType	属性値のタイプを指定します。 型: `AttrType` enum 値。 必須: いいえ
S	文字列の属性値を表します。 [Type] (タイプ): `string`

プロパティ	説明
	必須: はい
N	数値の属性値を表します。 [Type] (タイプ): `double` 必須: はい
SL	文字列の属性値の配列を表します。 [Type] (タイプ): `string[]` 必須: はい
SDM	文字列キーと二重値のディクショナリを表します。 [Type] (タイプ): `Dictionary<string, double>` 必須: はい

AwsStringOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 [Type] (タイプ): `string` 必須: いいえ
成功	アクションが成功したかどうか。 [Type] (タイプ): `bool` 必須: はい

プロパティ	説明
エラー	アクションが失敗した場合に発生したエラー。 [Type] (タイプ): the section called “GameLift Error” 必須: いいえ

GenericOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
成功	アクションが成功したかどうか。 [Type] (タイプ): bool 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 [Type] (タイプ): the section called “GameLift Error” 必須: いいえ

DescribePlayerSessionsOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 [Type] (タイプ): the section called “DescribePlayerSessionsResult”

プロパティ	説明
	必須: いいえ
成功	アクションが成功したかどうか。 [Type] (タイプ): bool 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 [Type] (タイプ): the section called “GameLift Error” 必須: いいえ

DescribePlayerSessionsResult

プロパティ	説明
NextToken	結果の次のページの先頭を示すトークン。結果セットの先頭を指定するには、値を指定しないでください。プレイヤーセッション ID を提供する場合、このパラメータは無視されます。 [Type] (タイプ): string 必須: はい
PlayerSessions	リクエストに一致する各プレイヤーセッションのプロパティを含むオブジェクトの集合。 [Type] (タイプ): IList<the section called “PlayerSession” > 必須:
成功	アクションが成功したかどうか。

プロパティ	説明
	[Type] (タイプ): `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 [Type] (タイプ): the section called “GameLift Error” 必須: いいえ

PlayerSession

プロパティ	説明
CreationTime	[Type] (タイプ): `long` 必須: はい
FleetId	[Type] (タイプ): `string` 必須: はい
GameSessionId	[Type] (タイプ): `string` 必須: はい
IP アドレス	[Type] (タイプ): `string` 必須: はい
PlayerData	[Type] (タイプ): `string` 必須: はい
PlayerId	[Type] (タイプ): `string` 必須: はい

プロパティ	説明
PlayerSessionId	[Type] (タイプ): `string` 必須: はい
ポート	[Type] (タイプ): `int` 必須: はい
ステータス	Type: A `PlayerSessionStatus` enum. 必須: はい
TerminationTime	[Type] (タイプ): `long` 必須: はい
DnsName	[Type] (タイプ): `string` 必須: はい

StartMatchBackfillOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 [Type] (タイプ): the section called “StartMatchBackfillResult” 必須: いいえ
成功	アクションが成功したかどうか。 [Type] (タイプ): `bool` 必須: はい

プロパティ	説明
エラー	アクションが失敗した場合に発生したエラー。 [Type] (タイプ): the section called “GameLift Error” 必須: いいえ

StartMatchBackfillResult

プロパティ	説明
TicketId	[Type] (タイプ): `string` 必須: はい

GetComputeCertificateOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 [Type] (タイプ): the section called “GetComputeCertificateResult” 必須: いいえ
成功	アクションが成功したかどうか。 [Type] (タイプ): `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。

プロパティ	説明
	[Type] (タイプ): the section called “GameLift Error” 必須: いいえ

GetComputeCertificateResult

コンピューティングの TLS 証明書へのパスとコンピューティングのホスト名。

プロパティ	説明
CertificatePath	[Type] (タイプ): `string` 必須: はい
ComputeName	[Type] (タイプ): `string` 必須: はい

GetFleetRoleCredentialsOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 [Type] (タイプ): the section called “GetFleetRoleCredentialsResult” 必須: いいえ
成功	アクションが成功したかどうか。 [Type] (タイプ): `bool` 必須: はい

プロパティ	説明
エラー	アクションが失敗した場合に発生したエラー。 [Type] (タイプ): the section called “GameLift Error” 必須: いいえ

GetFleetRoleCredentialsResult

プロパティ	説明
AccessKeyId	AWS へのアクセスを認証して提供するためのアクセスキー ID。 [Type] (タイプ): `string` 必須: いいえ
AssumedRoleId	サービスロールが属するユーザーの ID。 [Type] (タイプ): `string` 必須: いいえ
AssumedRoleUserArn	サービスロールが属するユーザーの Amazon リソースネーム (ARN)。 [Type] (タイプ): `string` 必須: いいえ
有効期限	セッション認証情報の有効期限が切れるまでの時間。 [Type] (タイプ): `DateTime` 必須: いいえ

プロパティ	説明
SecretAccessKey	認証のためのシークレットアクセスキー ID。 [Type] (タイプ): `string` 必須: いいえ
[SessionToken]	AWS リソースとやり取りしている現在のアクティブなセッションを識別するトークン。 [Type] (タイプ): `string` 必須: いいえ
成功	アクションが成功したかどうか。 [Type] (タイプ): `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 [Type] (タイプ): the section called “GameLift Error” 必須: いいえ

AwsDateTimeOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 [Type] (タイプ): `DateTime` 必須: いいえ
成功	アクションが成功したかどうか。

プロパティ	説明
	[Type] (タイプ): `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 [Type] (タイプ): the section called “GameLift Error” 必須: いいえ

GameLiftError

プロパティ	説明
ErrorType	エラーのタイプ。 Type: A `GameLiftErrorType` enum. 必須: いいえ
ErrorName	エラータイプの名前。 [Type] (タイプ): `string` 必須: いいえ
ErrorMessage	エラーメッセージです。 [Type] (タイプ): `string` 必須: いいえ

列挙型

Amazon GameLift サーバー SDK (C#) に定義された列挙値は次のように定義されます。

AttrType

- なし
- STRING
- DOUBLE
- STRING_LIST
- STRING_DOUBLE_MAP

GameLiftErrorType

エラータイプを示す文字列値。有効な値を次に示します。

- SERVICE_CALL_FAILED – AWS サービスへの呼び出しが失敗しました。
- LOCAL_CONNECTION_FAILED – Amazon GameLift へのローカル接続に失敗しました。
- NETWORK_NOT_INITIALIZED – ネットワークは初期化されていません。
- GAMESESSION_ID_NOT_SET – ゲームセッション ID が設定されていません。
- BAD_REQUEST_EXCEPTION
- INTERNAL_SERVICE_EXCEPTION
- ALREADY_INITIALIZED – Amazon GameLift サーバーまたはクライアントはすでに Initialize() で初期化されています。
- FLEET_MISMATCH – ターゲットフリートがゲームセッションまたはプレイヤーセッションのフリートと一致しません。
- GAMELIFT_CLIENT_NOT_INITIALIZED – Amazon GameLift クライアントは初期化されていません。
- GAMELIFT_SERVER_NOT_INITIALIZED – Amazon GameLift クライアントは初期化されていません。
- GAME_SESSION_ENDED_FAILED – Amazon GameLift サーバー SDK はサービスにアクセスしてゲームセッションが終了したことを報告できませんでした。
- GAME_SESSION_ENDED_FAILED – Amazon GameLift サーバーのゲームセッションがアクティブ化されませんでした。
- GAME_SESSION_READY_FAILED – Amazon GameLift サーバー SDK は、サービスにアクセスしてゲームセッションの準備が完了したことを報告できませんでした。
- INITIALIZATION_MISMATCH – Server:: Initialize() の後にクライアントメソッドが呼び出されました。その逆も同様です。

- NOT_INITIALIZED – Amazon GameLift サーバーまたはクライアントはすでに Initialize() で初期化されていません。
- NO_TARGET_ALIASID_SET – ターゲットのエイリアスが設定されていません。
- NO_TARGET_FLEET_SET – ターゲットフリートが設定されていません。
- PROCESS_ENDING_FAILED – Amazon GameLift サーバー SDK はサービスにアクセスしてプロセスの終了を報告できませんでした。
- PROCESS_NOT_ACTIVE – サーバープロセスはまだアクティブではなく、GameSession にバインドされていないため、プレイヤーセッションを受け入れたり処理したりすることはできません。
- PROCESS_NOT_READY – サーバープロセスをまだアクティブ化する準備ができていません。
- PROCESS_READY_FAILED – Amazon GameLift サーバー SDK はサービスにアクセスしてプロセスの準備が完了したことを報告できませんでした。
- SDK_VERSION_DETECTION_FAILED – SDK バージョン検出に失敗しました。
- STX_CALL_FAILED – XSTX サーバーのバックエンドコンポーネントへの呼び出しが失敗しました。
- STX_INITIALIZATION_FAILED – XSTX サーバーのバックエンドコンポーネントが初期化に失敗しました。
- UNEXPECTED_PLAYER_SESSION – 未登録のプレイヤーセッションがサーバーによって検出されました。
- WEBSOCKET_CONNECT_FAILURE
- WEBSOCKET_CONNECT_FAILURE_FORBIDDEN
- WEBSOCKET_CONNECT_FAILURE_INVALID_URL
- WEBSOCKET_CONNECT_FAILURE_TIMEOUT
- WEBSOCKET_RETRIABLE_SEND_MESSAGE_FAILURE – GameLift Service WebSocket にメッセージを送信する際に再試行可能な障害が発生しました。
- WEBSOCKET_SEND_MESSAGE_FAILURE – GameLift Service WebSocket へのメッセージの送信に失敗しました。
- MATCH_BACKFILL_REQUEST_VALIDATION – リクエストの検証に失敗しました。
- PLAYER_SESSION_REQUEST_VALIDATION – リクエストの検証に失敗しました。

PlayerSessionCreationPolicy

ゲームセッションで新しいプレイヤーを承諾するかどうかを示す文字列値。有効な値を次に示します。

- ACCEPT_ALL – すべての新しいプレイヤーセッションを承諾します。
- DENY_ALL – すべての新しいプレイヤーセッションを拒否します。
- NOT_SET – ゲームセッションは、新規プレイヤーセッションを受け入れたり拒否したりするように設定されていません。

PlayerSessionStatus

- ACTIVE
- COMPLETED
- NOT_SET
- RESERVED
- TIMEDOUT

C# 用 Amazon GameLift サーバー SDK 4.x リファレンス

この Amazon GameLift C# サーバー SDK 4.x リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

トピック

- Amazon GameLift サーバー SDK (C#) リファレンス: アクション
- Amazon GameLift サーバー SDK (C#) リファレンス: データ型

Amazon GameLift サーバー SDK (C#) リファレンス: アクション

この Amazon GameLift C# サーバー SDK リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

- アクション
- データ型

AcceptPlayerSession()

指定されたプレイヤーセッション ID のプレイヤーがサーバープロセスに接続し、検証が必要であることを Amazon GameLift サービスに通知します。Amazon GameLift は、プレイヤーセッション

ID が有効であること、つまり、そのプレイヤー ID でゲームセッションにプレイヤースロットが予約されていることを確認します。検証できたら、Amazon GameLift はプレーヤースロットの状態を RESERVED から ACTIVE に変更します。

構文

```
GenericOutcome AcceptPlayerSession(String playerSessionId)
```

パラメータ

playerSessionId

新しいプレイヤーセッションが作成されたときに Amazon GameLift によって発行される一意の ID。プレイヤーセッション ID は、PlayerSession オブジェクトです。これは、クライアントのGameLift API アクションのStartGameSessionPlacement、CreateGameSession、DescribeGameSessionPlacement、またはDescribePlayerSessions の呼び出しに応答して返されます。

型: 文字列

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、無効なプレイヤーセッション ID の検証や拒否を含む、接続リクエストを処理するための関数を示します。

```
void ReceiveConnectingPlayerSessionID (Connection connection, String playerSessionId){
    var acceptPlayerSessionOutcome =
 GameLiftServerAPI.AcceptPlayerSession(playerSessionId);
     if(acceptPlayerSessionOutcome.Success)
    {
        connectionToSessionMap.emplace(connection, playerSessionId);
        connection.Accept();
    }
     else
```

```
    {
        connection.Reject(acceptPlayerSessionOutcome.Error.ErrorMessage);    }
}
```

ActivateGameSession()

サーバープロセスがゲームセッションをアクティブにし、プレイヤーの接続を受ける準備ができていることを Amazon GameLift サービスに通知します。このアクションは、すべてのゲームセッションの初期化が完了した後、onStartGameSession() コールバック関数の一部として呼び出されます。

構文

```
GenericOutcome ActivateGameSession()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、ActivateGameSession() が onStartGameSession() 委任関数の一部として呼び出されていることを示しています。

```
void OnStartGameSession(GameSession gameSession)
{
    // game-specific tasks when starting a new game session, such as loading map

    // When ready to receive players
    var activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}
```

DescribePlayerSessions()

設定、セッションメタデータ、プレイヤーデータを含む、プレイヤーセッションデータを取得します。このアクションを使用して、単一のプレイヤーセッション、ゲームセッション内のすべてのプレ

イヤーセッション、または単一のプレイヤー ID に関連付けられたすべてのプレイヤーセッションに関する情報を取得します。

構文

```
DescribePlayerSessionsOutcome DescribePlayerSessions(DescribePlayerSessionsRequest
 describePlayerSessionsRequest)
```

パラメータ

describePlayerSessionsRequest

取得するプレイヤーセッションを記述する DescribePlayerSessionsRequest オブジェクト。

必須: はい

戻り値

成功した場合は、リクエストのパラメータに適合したプレイヤーセッションオブジェクトのセットを含む DescribePlayerSessionsOutcome オブジェクトを返します。プレイヤーセッションオブジェクトは、AWS SDK Amazon GameLift API PlayerSession データ型と同じ構造を持ちます。

例

この例は、指定したゲームセッションにアクティブに接続されているすべてのプレイヤーセッションのリクエストを示しています。NextToken を省略し、[制限] 値を 10 に設定すると、Amazon GameLift は、リクエストに一致するプレイヤーセッションレコードの最初の 10 個を返します。

```
// Set request parameters
var describePlayerSessionsRequest = new
 Aws.GameLift.Server.Model.DescribePlayerSessionsRequest()
{
    GameSessionId = GameLiftServerAPI.GetGameSessionId().Result,    //gets the ID for
 the current game session
    Limit = 10,
    PlayerSessionStatusFilter =
 PlayerSessionStatusMapper.GetNameForPlayerSessionStatus(PlayerSessionStatus.ACTIVE)
};
// Call DescribePlayerSessions
Aws::GameLift::DescribePlayerSessionsOutcome playerSessionsOutcome =
    Aws::GameLift::Server::Model::DescribePlayerSessions(describePlayerSessionRequest);
```

GetGameSessionId()

サーバープロセスがアクティブな場合、サーバープロセスが現在ホストしているゲームセッションの ID を取得します。

ゲームセッションでまだアクティブ化されていないアイドルプロセスの場合、コールは、Success = True そして GameSessionId = "" (空の文字列) を返します。

構文

```
AwsStringOutcome GetGameSessionId()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合、ゲームセッション ID を AwsStringOutcome オブジェクトとして返します。成功しなかった場合、エラーメッセージを返します。

例

```
var getGameSessionIdOutcome = GameLiftServerAPI.GetGameSessionId();
```

GetInstanceCertificate()

フリートとそのインスタンスに関連付けられている pem エンコードされた TLS 証明書のファイルの場所を取得します。AWS Certificate Manager は、証明書設定を GENERATED に設定して新しいフリートを作成するとこの証明書が生成されます。この証明書を使用して、ゲームクライアントとのセキュリティ保護ありの接続を確立し、クライアント/サーバー通信を暗号化します。

構文

```
GetInstanceCertificateOutcome GetInstanceCertificate();
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功すると、インスタンスに保存されているフリートの TLS 証明書ファイルの場所と証明書チェーンを含む GetInstanceCertificateOutcome オブジェクトを返します。証明書チェーンから抽出されたルート証明書ファイルもインスタンスに保存されます。成功しなかった場合、エラーメッセージを返します。

証明書と証明書チェーンデータの詳細については、AWS Certificate Manager API リファレンスの「GetCertificate レスポンス要素」を参照してください。

例

```
var getInstanceCertificateOutcome = GameLiftServerAPI.GetInstanceCertificate();
```

GetSdkVersion()

サーバープロセスに組み込まれた SDK の現在のバージョン番号を返します。

構文

```
AwsStringOutcome GetSdkVersion()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合、AwsStringOutcome オブジェクトとして現在の SDK バージョンを返します。返される文字列は、バージョン番号のみを含みます(例: 3.1.5)。成功しなかった場合、エラーメッセージを返します。

例

```
var getSdkVersionOutcome = GameLiftServerAPI.GetSdkVersion();
```

GetTerminationTime()

終了時刻が判る場合に、サーバープロセスがシャットダウンを予定している時刻を返します。サーバープロセスは、Amazon GameLift サービスから onProcessTerminate() コールバックを受信

した後にこのアクションを実行します。Amazon GameLift が onProcessTerminate() を呼び出すには、以下の理由が考えられます。(1) サーバープロセスがポートヘルスを報告した場合、もしくは Amazon GameLift に応答しなかった場合、(2) スケールダウンイベント中にインスタンスを終了した場合、(3) スポットインスタンス のためインスタンスが終了した場合。

プロセスが onProcessTerminate() コールバックを受信した場合、戻り値は予想終了時刻です。プロセスが onProcessTerminate() コールバックを受信していない場合、エラーメッセージが返されます。サーバープロセスのシャットダウンの詳細を確認してください。

構文

```
AwsDateTimeOutcome GetTerminationTime()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合、終了時刻をAwsDateTimeOutcomeオブジェクトとして返します。値は終了時間で、0001 00:00:00 以降の経過ティックで表現されます。たとえば、日付時刻の値 2020-09-13 12:26:40 -000Z は、637355968000000000 ティックに等しくなります。終了時間がない場合は、エラーメッセージを返します。

例

```
var getTerminationTimeOutcome = GameLiftServerAPI.GetTerminationTime();
```

InitSDK()

Amazon GameLift SDK を初期化する。このメソッドは起動時に他の Amazon GameLift 関連の初期化が実行される前に呼び出す必要があります。

構文

```
InitSDKOutcome InitSDK()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合は、サーバープロセスが ProcessReady() を呼び出す準備ができていることを示す InitSdkOutcome オブジェクトを返します。

例

```
var initSDKOutcome = GameLiftServerAPI.InitSDK();
```

ProcessEnding()

サーバープロセスがシャットダウンしていることを Amazon GameLift サービスに通知します。このメソッドは、すべてのアクティブゲームセッションのシャットダウンを含む他のすべてのクリーンアップタスクの後に呼び出されます。このメソッドは、終了コード 0 で終了します。0 以外の終了コードでは、処理が問題なく終了しなかったというイベントメッセージが発生します。

メソッドがコード 0 で終了すると、成功した終了コードでプロセスを終了できます。エラーコードでプロセスを終了することもできます。エラーコードで終了すると、フリートイベントはプロセスが異常終了したことを示します (SERVER_PROCESS_TERMINATED_UNHEALTHY)。

構文

```
GenericOutcome ProcessEnding()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();
if (processReadyOutcome.Success)
   Environment.Exit(0);
// otherwise, exit with error code
Environment.Exit(errorCode);
```

ProcessReady()

サーバープロセスがゲームセッションをホストする準備ができたことを Amazon GameLift サービスに通知します。このメソッドは、InitSDK() の呼び出しが成功して必要な設定タスクが完了した後、サーバープロセスがゲームセッションをホストできるようになる前に呼び出す必要があります。このメソッドは、プロセスごとに 1 回だけ呼び出す必要があります。

構文

```
GenericOutcome ProcessReady(ProcessParameters processParameters)
```

パラメータ

processParameters

サーバープロセスに関する以下の情報を伝える ProcessParameters オブジェクト。

- サーバープロセスと通信するために Amazon GameLift サービスが呼び出す、ゲームサーバーコードで実装されたコールバックメソッドの名前。
- サーバープロセスがリッスンするポートの番号。
- Amazon GameLift でキャプチャして保存するゲームセッション固有のファイルへのパス。

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、ProcessReady() 呼び出しと委任関数の実装の両方を示します。

```
// Set parameters and call ProcessReady
var processParams = new ProcessParameters(
   this.OnGameSession,
   this.OnProcessTerminate,
   this.OnHealthCheck,
   this.OnGameSessionUpdate,
   port,
   new LogParameters(new List<string>()          // Examples of log and error files
 written by the game server
```

```
   {
      "C:\\game\\logs",
      "C:\\game\\error"
   })
);

var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);

// Implement callback functions
void OnGameSession(GameSession gameSession)
{
   // game-specific tasks when starting a new game session, such as loading map
   // When ready to receive players
   var activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}

void OnProcessTerminate()
{
   // game-specific tasks required to gracefully shut down a game session,
   // such as notifying players, preserving game state data, and other cleanup
    var ProcessEndingOutcome = GameLiftServerAPI.ProcessEnding();
}

bool OnHealthCheck()
{
    bool isHealthy;
    // complete health evaluation within 60 seconds and set health
    return isHealthy;
}
```

RemovePlayerSession()

指定されたプレイヤーセッション ID のプレイヤーがサーバープロセスから切断されたことを Amazon GameLift サービスに通知します。それに応じて、Amazon GameLift はプレイヤースロットを新しいプレイヤーに割り当てられるよう利用可能に変更します。

構文

```
GenericOutcome RemovePlayerSession(String playerSessionId)
```

パラメータ

playerSessionId

新しいプレイヤーセッションが作成されたときに Amazon GameLift によって発行される一意の ID。プレイヤーセッション ID は、PlayerSession オブジェクトです。これは、クライアントのGameLift API アクションのStartGameSessionPlacement、CreateGameSession、DescribeGameSessionPlacement、またはDescribePlayerSessions の呼び出しに応答して返されます。

型: 文字列

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
Aws::GameLift::GenericOutcome disconnectOutcome =
    Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

StartMatchBackfill()

FlexMatch で作成されたゲームセッションの空きスロット用に新規プレイヤーを検索するリクエストを送信します。AWS SDK アクション StartMatchBackfill() を参照してください。このアクションを使用すると、ゲームセッションをホストするゲームサーバーのプロセスによってマッチバックフィルリクエストを開始できます。FlexMatch バックフィルの特徴の詳細はこちら。

このアクションは非同期です。新規プレイヤーが正常にマッチングされると、Amazon GameLift サービスはコールバック関数 OnUpdateGameSession() を使用して更新済みマッチメーカーデータを送信します。

サーバープロセスではアクティブなマッチバックフィルリクエストは一度に 1 つだけです。新しいリクエストを送信するには、まず StopMatchBackfill() を呼び出して元のリクエストをキャンセルする必要があります。

構文

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest
 startBackfillRequest);
```

パラメータ

StartMatchBackfillRequest

次の情報を通信する StartMatchBackfillRequest オブジェクト。

- バックフィルリクエストに割り当てるチケット ID。この情報はオプションです。ID が指定されていない場合は Amazon GameLift が ID を 1 つ自動生成します。
- リクエストを送信するマッチメーカー。完全な設定 ARN が必要です。この値は、ゲームセッションのマッチメーカーデータから取得できます。
- バックフィルされるゲームセッションの ID。
- ゲームセッションの現在のプレイヤーに利用可能なマッチメーキングデータ。

必須: はい

戻り値

StartMatchBackfillOutcome オブジェクトを、マッチバックフィルチケット ID またはエラーメッセージを伴うエラーとともに返します。

例

```
// Build a backfill request
var startBackfillRequest = new AWS.GameLift.Server.Model.StartMatchBackfillRequest()
{
    TicketId = "a ticket ID", //optional
    MatchmakingConfigurationArn = "the matchmaker configuration ARN",
    GameSessionId = GameLiftServerAPI.GetGameSessionId().Result,    // gets ID for
 current game session
        //get player data for all currently connected players
            MatchmakerData matchmakerData =
              MatchmakerData.FromJson(gameSession.MatchmakerData);  // gets matchmaker
 data for current players
            // get matchmakerData.Players
            // remove data for players who are no longer connected
    Players = ListOfPlayersRemainingInTheGame
```

```
};

// Send backfill request
var startBackfillOutcome = GameLiftServerAPI.StartMatchBackfill(startBackfillRequest);

// Implement callback function for backfill
void OnUpdateGameSession(GameSession myGameSession)
{
   // game-specific tasks to prepare for the newly matched players and update
 matchmaker data as needed
}
```

StopMatchBackfill()

StartMatchBackfill() とともに作成されたアクティブなマッチバックフィルリクエストをキャンセルします。AWS SDK アクション StopMatchmaking() も参照してください。FlexMatch バックフィルの特徴の詳細はこちら。

構文

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

パラメータ

StopMatchBackfillRequest

キャンセルするマッチメーキングチケットを識別する StopMatchBackfillRequestオブジェクト:

- キャンセルされるバックフィルリクエストに割り当てられたチケット ID
- バックフィルリクエストが送信されるマッチメーカー
- バックフィルリクエストに関連付けられたゲームセッション

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
// Set backfill stop request parameters
```

```
var stopBackfillRequest = new AWS.GameLift.Server.Model.StopMatchBackfillRequest()
{
    TicketId = "a ticket ID", //optional, if not provided one is autogenerated
    MatchmakingConfigurationArn = "the matchmaker configuration ARN", //from the game
 session matchmaker data
    GameSessionId = GameLiftServerAPI.GetGameSessionId().Result    //gets the ID for
 the current game session
};

var stopBackfillOutcome =
 GameLiftServerAPI.StopMatchBackfillRequest(stopBackfillRequest);
```

TerminateGameSession()

このメソッドは、バージョン 4.0.1 で非推奨となりました。代わりに、サーバープロセスはゲームセッションが終了した後に ProcessEnding() を呼び出す必要があります。

サーバープロセスがゲームセッションをシャットダウンしたことを　Amazon GameLift サービスに通知します。このアクションは、サーバープロセスがアクティブなままになり、新しいゲームセッションをホストするための準備ができたときに呼び出されます。これは、新しいゲームセッションをホストするためにサーバープロセスがすぐに利用可能であることを Amazon GameLift に通知するため、ゲームセッション終了手順が完了した後にのみ呼び出す必要があります。

ゲームセッションが停止した後にサーバープロセスがシャットダウンされる場合は、このアクションは呼び出されません。代わりに、 ProcessEnding() を呼び出し、ゲームセッションとサーバープロセスの両方が終了していることを通知します。

構文

```
GenericOutcome TerminateGameSession()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例は、ゲームセッションの終了時のサーバープロセスを示しています。

```
// game-specific tasks required to gracefully shut down a game session,
// such as notifying players, preserving game state data, and other cleanup

var terminateGameSessionOutcome = GameLiftServerAPI.TerminateGameSession();
var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);
```

UpdatePlayerSessionCreationPolicy()

現在のゲームセッションの機能を更新し、新しいプレイヤーセッションを承諾します。ゲームセッションは、新しいプレイヤーセッションをすべて受け入れるか拒否するかを設定できます。(Amazon GameLift サービス API リファレンスの「UpdateGameSession()」アクションも参照してください)。

構文

```
GenericOutcome UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy
 playerSessionPolicy)
```

パラメータ

newPlayerSessionPolicy

ゲームセッションで新しいプレイヤーを承諾するかどうかを示す文字列値。

型: PlayerSessionCreationPolicy 列挙。有効な値を次に示します。

- ACCEPT_ALL – すべての新しいプレイヤーセッションを承諾します。
- DENY_ALL – すべての新しいプレイヤーセッションを拒否します。

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例は、現在のゲームセッションの参加ポリシーを、すべてのプレイヤーを受け入れるように設定します。

```
var updatePlayerSessionCreationPolicyOutcomex =
```

```
GameLiftServerAPI.UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy.ACCEPT_ALL);
```

Amazon GameLift サーバー SDK (C#) リファレンス: データ型

この Amazon GameLift C# サーバー SDK リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

- [Actions] (アクション)
- データ型

LogParameters

このデータ型は、ゲームセッション中に生成されたファイルのうち、ゲームセッション終了時に Amazon GameLift でアップロードして保存するファイルを識別するのに使用されます。この情報は、ProcessReady() 呼び出しで Amazon GameLift サービスに伝えられます。

目次

logPaths

Amazon GameLift で将来のアクセスに備えて保存するゲームサーバーログファイルへのディレクトリパスのリスト。これらのファイルは各ゲームセッション中にサーバープロセスによって生成されます。ファイルのパスと名前は、ゲームサーバー内に定義され、ルートゲームビルドディレクトリに保存されます。ログパスは絶対パスである必要があります。たとえば、ゲームビルドによって `MyGame\sessionlogs\` などのパスに保存されるゲームセッションログのログパスは `c:\game\MyGame\sessionLogs` (Windows インスタンスの場合) または `/local/game/MyGame/sessionLogs` (Linux インスタンスの場合) となります。

型: List<String>

必須: いいえ

DescribePlayerSessionsRequest

このデータ型は、取得するプレイヤーセッションを指定するのに使用されます。複数の方法で使用できます。(1) 特定のプレイヤーセッションをリクエストする PlayerSessionId; を使用します。(2) 指定したゲームセッションのすべてのプレイヤーをリクエストするには、GameSessionId を指

定します。または、(3) 指定したプレイヤーのすべてのプレイヤーセッションをリクエストするには、PlayerId を指定します。プレイヤーセッション数が多い場合は、ページ分割パラメータを使用して結果を順次ページとして取得します。

目次

GameSessionId

一意のゲームセッション識別子。このパラメータを使用して、指定したゲームセッションのすべてのプレイヤーセッションをリクエストします。ゲームセッション ID の形式は、`arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>` です。<ID string> の値は、カスタム ID 文字列または (ゲームセッション作成時に指定した場合) 生成された文字列のいずれかです。

型: 文字列

必須: いいえ

制限

返される結果の最大数。このパラメータを NextToken とともに使用して、結果を一連の順次ページとして取得します。プレイヤーセッション ID を指定した場合、このパラメータは無視されます。

型: 整数

必須: いいえ

NextToken

結果において次の順次ページの開始を示すトークン。このアクションの以前の呼び出しで返されたトークンを使用します。結果セットの先頭を指定するには、値を指定しないでください。プレイヤーセッション ID を指定した場合、このパラメータは無視されます。

型: 文字列

必須: いいえ

PlayerId

プレイヤーを表す一意の識別子。プレイヤー ID は開発者によって定義されます。「プレイヤー ID を生成する」を参照してください。

型: 文字列

必須: いいえ

PlayerSessionId

プレイヤーセッションを表す一意の識別子。

型: 文字列

必須: いいえ

PlayerSessionStatusFilter

結果をフィルタリングするプレイヤーセッションステータス。可能なプレイヤーセッションステータスとして以下のステータスがあります。

- RESERVED – プレイヤーセッションリクエストは受領されましたが、プレイヤーのサーバープロセスへの接続や検証はまだ行われていません。
- ACTIVE – プレイヤーはサーバープロセスによって検証され、現時点で接続されています。
- COMPLETED – プレイヤー接続は削除されました。
- TIMEDOUT – プレイヤーセッションリクエストは受領されましたが、タイムアウト制限 (60 秒) 内でのプレイヤーの接続や検証は行われていません。

型: 文字列

必須: いいえ

ProcessParameters

このデータ型には、ProcessReady() 呼び出しで Amazon GameLift サービスに送られたパラメータセットが含まれます。

目次

port

サーバープロセスが新しいプレイヤーの接続をリスニングするポート番号。値は、このゲームサーバービルドをデプロイするすべてのフリートで設定されているポート番号の範囲に含まれる必要があります。このポート番号は、ゲームセッションオブジェクトとプレイヤーセッションオブジェクトに含まれ、ゲームセッションがサーバープロセスに接続するときに使用します。

型: 整数

必須: はい

logParameters

ゲームセッションログファイルへのディレクトリパスのリストを含むオブジェクト。

タイプ: `Aws::GameLift::Server::LogParameters`

必須: はい

onStartGameSession

Amazon GameLift サービスが新しいゲームセッションをアクティブにするために呼び出すコールバック関数名。Amazon GameLift はクライアントのリクエスト CreateGameSession に応じ、この関数を呼び出します。コールバック関数は GameSession オブジェクト (Amazon GameLift サービス API リファレンスで定義) を取得します。

タイプ: `void OnStartGameSessionDelegate(GameSession gameSession)`

必須: はい

onProcessTerminate

Amazon GameLift サービスがサーバープロセスを強制シャットダウンするために呼び出すコールバック関数名。この関数を呼び出すと、Amazon GameLift はサーバープロセスがシャットダウンするために 5 分間待ち、サーバープロセスをシャットダウンする前に ProcessEnding() 呼び出しで応答します。

タイプ: `void OnProcessTerminateDelegate()`

必須: はい

onHealthCheck

Amazon GameLift サービスがサーバープロセスにヘルスステータスレポートをリクエストするために呼び出すコールバック関数名。Amazon GameLift は、この関数を 60 秒ごとに呼び出します。この関数を呼び出した後、Amazon GameLift はレスポンスを 60 秒ほど待ちます。レスポンスがなければ、サーバープロセスを異常と記録します。

タイプ: `bool OnHealthCheckDelegate()`

必須: はい

onUpdateGameSession

Amazon GameLift サービスが更新されたゲームセッションオブジェクトを提供するために呼び出すコールバック関数名。Amazon GameLift はこの関数を、更新されたマッチメーカーデータを提

供するためにマッチバックフィルリクエストを処理するときに呼び出します。GameSession オブジェクト、ステータス更新 (updateReason)、およびマッチバックフィルチケット ID が渡されます。

タイプ: `void OnUpdateGameSessionDelegate ( UpdateGameSession updateGameSession )`

必須: いいえ

StartMatchBackfillRequest

このデータ型はマッチメーキングバックフィルリクエストの送信に使用します。この情報は、StartMatchBackfill() 呼び出しで Amazon GameLift サービスに伝えられます。

目次

GameSessionArn

一意のゲームセッション識別子。SDK メソッド GetGameSessionId() は ARN 形式の識別子を返します。

型: 文字列

必須: はい

MatchmakingConfigurationArn

このリクエストに使用されるマッチメーカーの ARN 形式の一意な識別子。元のゲームセッションの作成に使用されたマッチメーカーを見つけるには、ゲームセッションオブジェクトのマッチメーカーデータプロパティを確認します。マッチメーカーデータの詳細については「マッチメーカーデータの処理」を参照してください。

型: 文字列

必須: はい

プレイヤー

現在ゲームセッションに参加しているすべてのプレイヤーを表すデータのセット。マッチメーカーはこの情報を使用して、現在のプレイヤーとマッチする新しいプレイヤーを検索します。プレイヤーオブジェクトの形式の説明については、Amazon GameLift API リファレンスガイドを参照してください。プレイヤー属性、ID、チームの割り当てを見つけるには、マッチメーカーデー

タプロパティのゲームセッションオブジェクトを参照してください。マッチメーカーでレイテンシーが使用されている場合は、現在のリージョンの更新されたレイテンシーを収集し、それを各プレイヤーのデータに含めます。

型: プレイヤー[ ]

必須: はい

TicketId

マッチメーキングまたはバックフィルリクエストチケットの一意の識別子。ここで値を指定しない場合、Amazon GameLift によって UUID 形式で自動的に生成されます。この識別子を使用してマッチバックフィルチケットのステータスを追跡したり、必要に応じてリクエストをキャンセルしたりします。

型: 文字列

必須: いいえ

StopMatchBackfillRequest

このデータ型はマッチメーキングバックフィルリクエストのキャンセルに使用します。この情報は、StopMatchBackfill() 呼び出しで Amazon GameLift サービスに伝えられます。

目次

GameSessionArn

キャンセルされるリクエストに関連付けられた一意のゲームセッション識別子。

型: 文字列

必須: はい

MatchmakingConfigurationArn

このリクエストが送信されたマッチメーカーの一意の識別子。

型: 文字列

必須: はい

TicketId

キャンセルされるバックフィルリクエストチケットの一意の識別子。

型: 文字列

必須: はい

Go 用 Amazon GameLift サーバー SDK リファレンス

この Amazon GameLift Go サーバー SDK リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

トピック

- Amazon GameLift サーバー SDK (Go) リファレンス: アクション
- Amazon GameLift サーバー SDK (Go) リファレンス: データ型

Amazon GameLift サーバー SDK (Go) リファレンス: アクション

この Amazon GameLift Go サーバー SDK リファレンスを使用すると、Amazon で使用するマルチプレイヤーゲームを準備するのに役立ちます GameLift。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

`GameLiftServerAPI.go` は、Go サーバー SDK アクションを定義します。

アクション

- GetSdkVersion()
- InitSDK()
- ProcessReady()
- ProcessEnding()
- ActivateGameSession()
- UpdatePlayerSessionCreationPolicy()
- GetGameSessionId()
- GetTerminationTime()
- AcceptPlayerSession()
- RemovePlayerSession()
- DescribePlayerSessions()
- StartMatchBackfill()

- StopMatchBackfill()
- GetComputeCertificate()
- GetFleetRoleCredentials()
- Destroy()

GetSdkVersion()

サーバープロセスに組み込まれた SDK の現在のバージョン番号を返します。

Syntax

```
func GetSdkVersion() (string, error)
```

戻り値

成功した場合、文字列として現在の SDK バージョンを返します。返される文字列は、バージョン番号のみを含みます。(例: `5.0.0`) 成功しなかった場合、`common.SdkVersionDetectionFailed` などのエラーメッセージを返します。

例

```
version, err := server.GetSdkVersion()
```

InitSDK()

Amazon GameLift SDK を初期化します。Amazon に関連する他の初期化 GameLift が発生する前に、起動時にこのメソッドを呼び出します。このメソッドは、サーバーと Amazon GameLift サービス間の通信を設定します。

Syntax

```
func InitSDK(params ServerParameters) error
```

パラメータ

ServerParameters

Amazon GameLift Anywhere フリートでゲームサーバーを初期化するには、次の情報を使用して `ServerParameters` オブジェクトを作成します。

- ゲームサーバーへの接続 WebSocket に使用される の URL。
- ゲームサーバーのホストに使用されるプロセスの ID。
- ゲームサーバープロセスをホスティングするコンピューティングの ID。
- Amazon コンピューティングを含む Amazon GameLift Anywhere GameLift フリートの ID。
- Amazon GameLift オペレーションによって生成された認証トークン。

Amazon GameLift マネージド EC2 フリートでゲームサーバーを初期化するには、パラメータなしで `ServerParameters` オブジェクトを構築します。この呼び出しにより、Amazon GameLift エージェントはコンピューティング環境を設定し、自動的に Amazon GameLift サービスに接続します。

戻り値

成功した場合は、サーバープロセスが ProcessReady() を呼び出す準備ができていることを示す `nil` エラーを返します。

> **Note**
>
> Anywhere フリートにデプロイされたゲームビルドに対して `InitSDK()` への呼び出しが失敗する場合は、ビルドリソースの作成時に使用した `ServerSdkVersion` パラメータを確認してください。この値は、使用中のサーバー SDK バージョンに明示的に設定する必要があります。このパラメータのデフォルト値は 4.x で、互換性がありません。この問題を解決するには、新しいビルドを作成して新しいフリートにデプロイします。

例

Amazon GameLift Anywhere の例

```
//Define the server parameters
serverParameters := ServerParameters {
  WebSocketURL: "wss://us-west-1.api.amazongamelift.com",
  ProcessID: "PID1234",
  HostID: "HardwareAnywhere",
  FleetID: "aarn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa",
  AuthToken: "1111aaaa-22bb-33cc-44dd-5555eeee66ff"
}
```

```
//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
err := server.InitSDK(serverParameters)
```

Amazon GameLift マネージド EC2 の例

```
//Define the server parameters
serverParameters := ServerParameters {}

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
err := server.InitSDK(serverParameters)
```

ProcessReady()

サーバープロセス GameLift がゲームセッションをホストする準備ができたことを Amazon に通知します。InitSDK() を呼び出した後にこのメソッドを呼び出します このメソッドは、プロセスごとに 1 回だけ呼び出す必要があります。

Syntax

```
func ProcessReady(param ProcessParameters) error
```

パラメータ

ProcessParameters

サーバープロセスに関する以下の情報を伝える ProcessParameters オブジェクト。

- Amazon GameLift サービスがサーバープロセスと通信するために呼び出すゲームサーバーコードに実装されているコールバックメソッドの名前。
- サーバープロセスがリッスンするポートの番号。
- Amazon が GameLift キャプチャして保存するゲームセッション固有のファイルへのパスを含むLogParametersデータ型。

戻り値

メソッドが失敗すると、エラーとエラーメッセージを返します。メソッドが成功した場合は、`nil` を返します。

例

この例では、ProcessReady() 呼び出しと委任関数の実装の両方を示します。

```
// Define the process parameters
processParams := ProcessParameters {
  OnStartGameSession: gameProcess.OnStartGameSession,
  OnUpdateGameSession: gameProcess.OnGameSessionUpdate,
  OnProcessTerminate: gameProcess.OnProcessTerminate,
  OnHealthCheck: gameProcess.OnHealthCheck,
  Port: port,
  LogParameters: LogParameters {    // logging and error example
    []string {"C:\\game\\logs", "C:\\game\\error"}
  }
}

err := server.ProcessReady(processParams)
```

ProcessEnding()

サーバープロセスが終了中であることを Amazon に通知 GameLift します。他のすべてのクリーンアップタスク (アクティブなゲームセッションのシャットダウンを含む) の後、およびプロセスを終了する前に、このメソッドを呼び出します。ProcessEnding( ) の結果に応じて、プロセスは成功 (0) またはエラー (-1) で終了し、フリートイベントを生成します。プロセスがエラーで終了した場合、生成されるフリートイベントは ですSERVER_PROCESS_TERMINATED_UNHEALTHY。

Syntax

```
func ProcessEnding() error
```

戻り値

0 のエラーコードまたは定義済みのエラーコードを返します。

例

```
// operations to end game sessions and the server process
defer func() {
  err := server.ProcessEnding()
  server.Destroy()
  if err != nil {
```

```
    fmt.Println("ProcessEnding() failed. Error: ", err)
    os.Exit(-1)
  } else {
    os.Exit(0)
  }
}
```

ActivateGameSession()

サーバープロセスがゲームセッションをアクティブ化し、プレイヤー接続を受信する準備ができた GameLift ことを Amazon に通知します。このアクションは、すべてのゲームセッションの初期化の後、onStartGameSession() コールバック関数の一部として呼び出されます。

Syntax

```
func ActivateGameSession() error
```

戻り値

メソッドが失敗すると、エラーとエラーメッセージを返します。

例

この例では、onStartGameSession() 委任関数の一部として呼び出された ActivateGameSession() を示しています。

```
func OnStartGameSession(GameSession gameSession) {
  // game-specific tasks when starting a new game session, such as loading map
  // Activate when ready to receive players
  err := server.ActivateGameSession();
}
```

UpdatePlayerSessionCreationPolicy()

現在のゲームセッションの機能を更新し、新しいプレイヤーセッションを承諾します。ゲームセッションは、新しいプレイヤーセッションをすべて受け入れるか拒否するかを設定できます。

Syntax

```
func UpdatePlayerSessionCreationPolicy(policy model.PlayerSessionCreationPolicy) error
```

パラメータ

playerSessionCreationポリシー

ゲームセッションで新しいプレイヤーを承諾するかどうかを示す文字列値。

有効な値を次に示します。

- **model.AcceptAll** – すべての新しいプレイヤーセッションを受け入れます。
- **model.DenyAll** – すべての新しいプレイヤーセッションを拒否します。

戻り値

失敗が発生すると、エラーとエラーメッセージを返します。

例

この例は、現在のゲームセッションの参加ポリシーを、すべてのプレイヤーを受け入れるように設定します。

```
err := server.UpdatePlayerSessionCreationPolicy(model.AcceptAll)
```

GetGameSessionId()

アクティブなサーバープロセスにホストされたゲームセッションの ID を取得します。

Syntax

```
func GetGameSessionID() (string, error)
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合、ゲームセッション ID を nil エラーを返します。ゲームセッションでアクティブ化されていないアイドルプロセスの場合、呼び出しは空の文字列と nil エラーを返します。

例

```
gameSessionID, err := server.GetGameSessionID()
```

GetTerminationTime()

終了時刻が判る場合に、サーバープロセスがシャットダウンを予定している時刻を返します。サーバープロセスは、Amazon から onProcessTerminate()コールバックを受信した後、このアクションを実行します GameLift。Amazon onProcessTerminate() は、次の理由で を GameLift 呼び出します。

- サーバープロセスが正常性の低下を報告した場合、または Amazon に応答しなかった場合 GameLift。
- スケールダウンイベント中にインスタンスを終了する場合。
- スポットインスタンスの中断によりインスタンスが終了した場合。

Syntax

```
func GetTerminationTime() (int64, error)
```

戻り値

成功すると、サーバープロセスのシャットダウンが予定されているタイムスタンプ (エポック秒単位) と nil エラー終了を返します。値は終了時間で、0001 00:00:00 からの経過ティックで表現されます。例えば、日付時刻の値 2020-09-13 12:26:40 -000Z は、637355968000000000 ティックに等しくなります。終了時間がない場合は、エラーメッセージを返します。

例

```
terminationTime, err := server.GetTerminationTime()
```

AcceptPlayerSession()

指定されたプレイヤーセッション ID を持つプレイヤーがサーバープロセスに接続し、検証が必要である GameLift ことを Amazon に通知します。Amazon は、プレイヤーセッション ID が有効 GameLift であることを確認します。プレイヤーセッションが検証されると、Amazon はプレイヤースロットのステータス GameLift を から RESERVEDに変更しますACTIVE。

Syntax

```
func AcceptPlayerSession(playerSessionID string) error
```

パラメータ

playerSessionId

新しいプレイヤーセッションの作成 GameLift 時に Amazon によって発行される一意の ID。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、無効なプレイヤーセッション ID の検証と拒否を含む接続リクエストを処理します。

```
func ReceiveConnectingPlayerSessionID(conn Connection, playerSessionID string) {
    err := server.AcceptPlayerSession(playerSessionID)
    if err != nil {
        connection.Accept()
    } else {
        connection.Reject(err.Error())
    }
}
```

RemovePlayerSession()

プレイヤーがサーバープロセスから切断された GameLift ことを Amazon に通知します。それに応じて、Amazon GameLift はプレイヤースロットを利用可能に変更します。

Syntax

```
func RemovePlayerSession(playerSessionID string) error
```

パラメータ

playerSessionId

新しいプレイヤーセッションの作成 GameLift 時に Amazon によって発行される一意の ID。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
err := server.RemovePlayerSession(playerSessionID)
```

DescribePlayerSessions()

設定、セッションメタデータ、プレイヤーデータを含む、プレイヤーセッションデータを取得します。このメソッドを使用して、以下に関する情報を取得します。

- シングルプレイヤーセッション
- ゲームセッションのすべてのプレイヤーセッション
- 1 つのプレイヤー ID に関連付けられているすべてのプレイヤーセッション

Syntax

```
func DescribePlayerSessions(req request.DescribePlayerSessionsRequest)
 (result.DescribePlayerSessionsResult, error) {
 return srv.describePlayerSessions(&req)
}
```

パラメータ

DescribePlayerSessionsRequest

取得するプレイヤーセッションを記述する DescribePlayerSessionsRequest オブジェクト。

戻り値

成功した場合は、リクエストのパラメータに適合したプレイヤーセッションオブジェクトのセットを含む DescribePlayerSessionsResult オブジェクトを返します。

例

この例は、指定したゲームセッションにアクティブに接続されているすべてのプレイヤーセッションのリクエストします。を省略NextTokenし、Limit 値を 10 に設定することで、Amazon はリクエストに一致する最初の 10 個のプレイヤーセッションレコード GameLift を返します。

```
// create request
```

```
describePlayerSessionsRequest := request.NewDescribePlayerSessions()
describePlayerSessionsRequest.GameSessionID, _ = server.GetGameSessionID() // get ID
 for the current game session
describePlayerSessionsRequest.Limit = 10                                  // return the
 first 10 player sessions
describePlayerSessionsRequest.PlayerSessionStatusFilter = "ACTIVE"          // Get all
 player sessions actively connected to the game session

describePlayerSessionsResult, err :=
 server.DescribePlayerSessions(describePlayerSessionsRequest)
```

StartMatchBackfill()

FlexMatch で作成されたゲームセッションの空きスロット用に新規プレイヤーを検索するリクエストを送信します。詳細については、FlexMatch 「バックフィル機能」を参照してください。

このアクションは非同期です。新しいプレイヤーがマッチングされると、Amazon はコールバック関数 を使用して更新されたマッチメーカーデータを GameLift 配信しますOnUpdateGameSession()。

サーバープロセスではアクティブなマッチバックフィルリクエストは一度に 1 つだけです。新しいリクエストを送信するには、まず StopMatchBackfill() を呼び出して元のリクエストをキャンセルする必要があります。

Syntax

```
func StartMatchBackfill(req request.StartMatchBackfillRequest)
 (result.StartMatchBackfillResult, error)
```

パラメータ

StartMatchBackfillRequest

StartMatchBackfillRequest オブジェクトは、次の情報を通信します。

- バックフィルリクエストに割り当てるチケット ID。この情報はオプションです。ID が指定されていない場合、Amazon は ID GameLift を生成します。
- リクエストを送信するマッチメーカー。完全な設定 ARN が必要です。この値はゲームセッションのマッチメーカーデータに含まれています。
- バックフィルするゲームセッションの ID。
- ゲームセッションの現在のプレイヤーに利用可能なマッチメーキングデータ。

戻り値

StartMatchBackfillResult オブジェクトを、マッチバックフィルチケット ID またはエラーメッセージを伴うエラーとともに返します。

例

```
// form the request
startBackfillRequest := request.NewStartMatchBackfill()
startBackfillRequest.RequestID = "1111aaaa-22bb-33cc-44dd-5555eeee66ff"          //
 optional
startBackfillRequest.MatchmakingConfigurationArn = "arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig"
var matchMaker model.MatchmakerData
if err := matchMaker.UnmarshalJSON([]byte(gameSession.MatchmakerData)); err != nil {

    return
}
startBackfillRequest.Players = matchMaker.Players
res, err := server.StartMatchBackfill(startBackfillRequest)

// Implement callback function for backfill
func OnUpdateGameSession(myGameSession model.GameSession) {
    // game-specific tasks to prepare for the newly matched players and update
 matchmaker data as needed
}
```

StopMatchBackfill()

アクティブなマッチバックフィルリクエストをキャンセルします。詳細については、FlexMatch「バックフィル機能」を参照してください。

Syntax

```
func StopMatchBackfill(req request.StopMatchBackfillRequest) error
```

パラメータ

StopMatchBackfillRequest

キャンセルするマッチメーキングチケットを識別する StopMatchBackfillRequest オブジェクト：

- バックフィルリクエストに割り当てるチケット ID。
- バックフィルリクエストが送信されたマッチメーカー。
- バックフィルリクエストに関連付けられたゲームセッション。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
stopBackfillRequest := request.NewStopMatchBackfill()  // Use this function to create
 request
stopBackfillRequest.TicketID = "1111aaaa-22bb-33cc-44dd-5555eeee66ff"
stopBackfillRequest.MatchmakingConfigurationArn = "arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig"

//error
err := server.StopMatchBackfill(stopBackfillRequest)
```

GetComputeCertificate()

ゲームサーバーとゲームクライアント間のネットワーク接続を暗号化するために使用される TLS 証明書へのパスを取得します。コンピューティングデバイスを Amazon GameLift Anywhere フリートに登録するときに、証明書パスを使用できます。詳細については、「」を参照してくださいRegisterCompute。

Syntax

```
func GetComputeCertificate() (result.GetComputeCertificateResult, error)
```

戻り値

以下が含まれる `GetComputeCertificateResult` オブジェクトを返します。

- CertificatePath: コンピューティングリソースの TLS 証明書へのパス。Amazon GameLift マネージドフリートを使用する場合、このパスには以下が含まれます。
  - `certificate.pem`: エンドユーザー証明書。証明書チェーン全体は、この証明書に追加された `certificateChain.pem` を組み合わせたものです。

- certificateChain.pem: ルート証明書と中間証明書を含む証明書チェーン。
- rootCertificate.pem: ルート証明書。
- privateKey.pem: エンドユーザー証明書のプライベートキー。

- ComputeName: コンピューティングリソースの名前。

例

```
tlsCertificate, err := server.GetFleetRoleCredentials(getFleetRoleCredentialsRequest)
```

GetFleetRoleCredentials()

作成したサービスロールの認証情報を取得し、アクセス許可を他の AWS のサービスに拡張します GameLift。これらの認証情報により、ゲームサーバーは AWS リソースを使用できます。詳細については、「Amazon の IAM サービスロールを設定する GameLift」を参照してください。

構文

```
func GetFleetRoleCredentials(
  req request.GetFleetRoleCredentialsRequest,
) (result.GetFleetRoleCredentialsResult, error) {
  return srv.getFleetRoleCredentials(&req)
}
```

パラメータ

GetFleetRoleCredentialsRequest

AWS リソースへの制限付きアクセスをゲームサーバーにまで拡張するロール認証情報。

戻り値

以下が含まれる GetFleetRoleCredentialsResult オブジェクトを返します。

- AssumedRoleUserArn - サービスロールが属するユーザーの Amazon リソースネーム (ARN)。
- AssumedRoleId - サービスロールが属するユーザーの ID。
- AccessKeyId - AWSリソースへの認証およびアクセスを提供するアクセスキー ID。
- SecretAccessKey - 認証用のシークレットアクセスキー ID。

- SessionToken - AWSリソースと対話している現在アクティブなセッションを識別するトークン。
- Expiration - セッション認証情報の有効期限が切れるまでの時間。

例

```
// form the customer credentials request
getFleetRoleCredentialsRequest := request.NewGetFleetRoleCredentials()
getFleetRoleCredentialsRequest.RoleArn = "arn:aws:iam::123456789012:role/service-role/
exampleGameLiftAction"

credentials, err := server.GetFleetRoleCredentials(getFleetRoleCredentialsRequest)
```

Destroy()

Amazon GameLift ゲームサーバー SDK をメモリから解放します。ベストプラクティスとして、ProcessEnding() の後、かつプロセスの終了前にこのメソッドを呼び出します。Anywhere フリートを使用していて、すべてのゲームセッション後にサーバープロセスを終了しない場合は、InitSDK()を呼び出しDestroy()てから再初期化してから、プロセスが でゲームセッションをホストする準備ができ GameLift たことを Amazon に通知しますProcessReady()。

Syntax

```
func Destroy() error {
 return srv.destroy()
}
```

戻り値

メソッドが失敗すると、エラーとエラーメッセージを返します。

例

```
// operations to end game sessions and the server process
defer func() {
  err := server.ProcessEnding()
  server.Destroy()
  if err != nil {
    fmt.Println("ProcessEnding() failed. Error: ", err)
    os.Exit(-1)
```

```
  } else {
    os.Exit(0)
  }
}
```

Amazon GameLift サーバー SDK (Go) リファレンス: データ型

この Amazon GameLift Go サーバー SDK リファレンスを使用すると、Amazon で使用するマルチプレイヤーゲームを準備するのに役立ちます GameLift。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

データ型

- LogParameters
- ProcessParameters
- UpdateGameSession
- GameSession
- ServerParameters
- StartMatchBackfillRequest
- プレイヤー
- DescribePlayerSessionsRequest
- StopMatchBackfillRequest
- GetFleetRoleCredentialsRequest

LogParameters

ゲームセッション中に生成されたファイルを識別 GameLift し、ゲームセッションの終了後に Amazon がアップロードして保存するオブジェクト。ゲームサーバーは`LogParameters`、ProcessReady()呼び出しの`ProcessParameters`オブジェクト GameLift の一部として Amazon に を提供します。

プロパティ	説明
`LogPaths`	Amazon が将来のアクセスのために GameLift 保存するゲームサーバーログファイルへのディレクトリパスのリスト。サーバープロセスは各ゲームセッション中にこれらのファイルを生成します。ファイルのパスと名前はゲームサーバーで定義し、ルートゲームビルドディレクトリに保存します。

	ログパスは絶対パスである必要があります。例えば、ゲームビルドによって MyGame\sessionLogs\ などのパスに保存されるゲームセッションログの場合、パスは c:\game\MyGame\sessionLogs (Windows インスタンスの場合) となります。 タイプ: []string 必須: いいえ

ProcessParameters

サーバープロセスと Amazon 間の通信を記述するオブジェクト GameLift。サーバープロセスは、 への呼び出し GameLift でこの情報を Amazon に提供しますProcessReady()。

プロパティ	説明
LogParameters	ゲームセッション中に生成されるファイルへのディレクトリパスを含むオブジェクト。Amazon は、将来のアクセスに備えてファイル GameLift をコピーして保存します。 タイプ: LogParameters 必須: いいえ
OnHealthCheck	サーバープロセスにヘルスステータスレポートをリクエストするために Amazon が GameLift 呼び出すコールバック関数。Amazon はこの関数を 60 秒ごとに GameLift 呼び出し、応答を 60 秒待機します。サーバープロセスは正常であれば TRUE を返し、正常でない場合は FALSE を返します。レスポンスが返されない場合、Amazon はサーバープロセスを正常でないものとして GameLift 記録します。 タイプ: OnHealthCheck func() bool 必須: いいえ
OnProcessTerminate	サーバープロセスを強制シャットダウンするために Amazon が GameLift 呼び出すコールバック関数。この関数を呼び出した後、Amazon はサーバープロセスがシャットダウンするまで 5 分 GameLift 待ってからProcessEnding()、サーバープロセスをシャットダウンします。

	タイプ: `OnProcessTerminate func()` 必須: はい
`OnStartGameSession`	更新されたゲームセッションオブジェクトをサーバープロセスに渡すために Amazon が GameLift 呼び出すコールバック関数。Amazon は、マッチバックフィルリクエストが更新されたマッチメーカーデータを提供するために処理されたときに、この関数を GameLift 呼び出します。GameSession オブジェクト、ステータス更新 (`updateReason `) 、マッチバックフィルチケット ID を渡します。 タイプ: `OnStartGameSession func (model.GameSession )` 必須: はい
`OnUpdateGameSession`	更新されたゲームセッション情報をサーバープロセスに渡すために Amazon が GameLift 呼び出すコールバック関数。Amazon は、マッチバックフィルリクエストを処理した後にこの関数を GameLift 呼び出して、更新されたマッチメーカーデータを提供します。 タイプ: `OnUpdateGameSession func (model.UpdateGameSession)` 必須: いいえ
`Port`	サーバープロセスが新しいプレイヤーの接続をリスンするポート番号。値は、このゲームサーバービルドをデプロイするすべてのフリートで設定されているポート番号の範囲に含まれる必要があります。このポート番号は、ゲームセッションオブジェクトとプレイヤーセッションオブジェクトに含まれ、ゲームセッションがサーバープロセスに接続するときに使用します。 タイプ: `int` 必須: はい

UpdateGameSession

ゲームセッションオブジェクトの更新。これには、ゲームセッションが更新された理由と、バックフィルを使用してゲームセッション内のプレイヤーセッションを埋めるための関連するバックフィルチケット ID が含まれます。

プロパティ	説明
GameSession	Amazon GameLift API で定義されるGameSessionオブジェクト。GameSession オブジェクトにはゲームセッションを説明するプロパティが含まれています。 タイプ: GameSession GameSession() 必須: はい
UpdateReason	ゲームセッションが更新されている理由。 タイプ: UpdateReason UpdateReason() 必須: はい
BackfillTicketId	ゲームセッションの更新を試みるバックフィルチケットの ID。 タイプ: String 必須: いいえ

GameSession

ゲームセッションの詳細。

プロパティ	説明
GameSessionId	ゲームセッションの一意の識別子。ゲームセッション Amazon リソースネーム (ARN) には arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token> 形式があります。 タイプ: String

プロパティ	説明
	必須: いいえ
名前	ゲームセッションについて説明するラベル。 タイプ: `String` 必須: いいえ
FleetId	ゲームセッションが実行されているフリートの一意の識別子。 タイプ: `String` 必須: いいえ
MaximumPlayerSessionCount	ゲームセッションへのプレーヤー接続の最大数。 タイプ: `Integer` 必須: いいえ
ポート	ゲームセッションのポート番号。Amazon GameLift ゲームサーバーに接続するには、アプリに IP アドレスとポート番号の両方が必要です。 タイプ: `Integer` 必須: いいえ
IpAddress	ゲームセッションの IP アドレス。Amazon GameLift ゲームサーバーに接続するには、アプリに IP アドレスとポート番号の両方が必要です。 タイプ: `String` 必須: いいえ
GameSessionData	単一の文字列値としてフォーマットされたカスタムゲームセッションプロパティのセット。 タイプ: `String` 必須: いいえ

プロパティ	説明
MatchmakerData	ゲームセッションの作成に使用されたマッチメーキングプロセスに関する情報。JSON 構文で、文字列としてフォーマットされています。使用されたマッチメーキング設定に加えて、プレイヤー属性やチーム割り当てなど、マッチに割り当てられた全プレイヤーに関するデータが含まれます。 タイプ: `String` 必須: いいえ
GameProperties	ゲームセッションのカスタムプロパティのセットで、キーと値のペアとしてフォーマットされます。これらのプロパティは、新しいゲームセッションを開始するリクエストとともに渡されます。 タイプ: `map[string] string` 必須: いいえ
DnsName	ゲームセッションを実行しているインスタンスに割り当てられた DNS 識別子。値の形式は次のとおりです。 • TLS 対応フリート: `<unique identifier>.<region identifier>.amazongamelift.com` 。 • TLS 対応でないフリート: `ec2-<unique identifier>.compute.amazonaws.com` 。 TLS 対応フリートで実行しているゲームセッションに接続する場合、IP アドレスではなく DNS 名を使用する必要があります。 タイプ: `String` 必須: いいえ

ServerParameters

Amazon GameLift Anywhere サーバーと Amazon GameLift サービス間の接続を維持するために使用される情報。この情報は、InitSDK() で新しいサーバープロセスを起動するときに使用されま

す。Amazon GameLift マネージド EC2 インスタンスでホストされているサーバーの場合は、空のオブジェクトを使用します。

プロパティ	説明
`WebSocketURL`	`GameLiftServerSdkEndpoint` Amazon は GameLift Anywhere、Amazon コンピューティングリソース`RegisterCompute` に対して を GameLift 返す。 タイプ: `string` 必須: はい
`ProcessID`	ゲームをホストするサーバープロセスに登録された固有の識別子。 タイプ: `string` 必須: はい
`HostID`	新しいサーバープロセスをホストしているコンピュートリソースの一意の識別子。 `HostID` はコンピューティングを登録したときに使用される `ComputeName` です。詳細については、「」を参照してくださいRegisterCompute。 タイプ: `string` 必須: はい
`FleetID`	コンピューティングが登録されているフリートの固有識別子。詳細については、「」を参照してくださいRegisterCompute。 タイプ: `string` 必須: はい
`AuthToken`	Amazon が生成し、Amazon に対してサーバーを認証 GameLift する認証トークン GameLift。詳細については、「」を参照してくださいGetComputeAuthToken。 タイプ: `string`

プロパティ	説明
	必須: はい

StartMatchBackfillRequest

マッチメーキングバックフィルリクエストの作成に使用される情報。ゲームサーバーは、この情報を StartMatchBackfill()呼び出し GameLift で Amazon に伝えます。

プロパティ	説明
GameSessionArn	一意のゲームセッション識別子。API オペレーション GetGameSessionId は ARN 形式の識別子を返します。 タイプ: `String` 必須: はい
MatchmakingConfigurationArn	このリクエストに使用されるマッチメーカーの ARN 形式の一意な識別子。元のゲームセッションののマッチメーカー ARN は、マッチメーカーデータプロパティのゲームセッションオブジェクトにあります。マッチメーカーデータの詳細については「マッチメーカーデータの処理」を参照してください。 タイプ: `String` 必須: はい
プレイヤー	現在ゲームセッションに参加しているすべてのプレイヤーを表すデータのセット。マッチメーカーはこの情報を使用して、現在のプレイヤーとマッチする新しいプレイヤーを検索します。 タイプ: `[]model.Player` 必須: はい
TicketId	マッチメーキングまたはバックフィルリクエストチケットの一意の識別子。値を指定しない場合、Amazon は値 GameLift を生成します。この識別子を使用してマッチバックフィルチケットのステータスを追跡したり、必要に応じてリクエストをキャンセルしたりします。

プロパティ	説明
	タイプ: `String` 必須: いいえ

プレイヤー

マッチメーキングでプレイヤーを表すオブジェクト。マッチメーキングリクエストを開始すると、プレイヤーはプレイヤー ID、属性、場合によってはレイテンシーデータを保有します。Amazon は、マッチング後にチーム情報 GameLift を追加します。

プロパティ	説明
LatencyInMS	プレイヤーがロケーションに接続したときに発生するレイテンシーの量を示すミリ秒単位の値のセット。 このプロパティを使用すると、プレーヤーはリストに表示されている場所でのみマッチングされます。マッチメーカーにプレイヤーレイテンシーを評価するルールがある場合、プレイヤーはレイテンシーを報告しないとマッチングされません。 タイプ: `map[string] int` 必須: いいえ
PlayerAttributes	マッチメーキングに使用するプレイヤー情報を含むキーと値のペアの集合。プレイヤー属性キーは、マッチメーキングルールセット PlayerAttributes で使用される と一致する必要があります。 プレイヤー属性の詳細については、「」を参照してくださいAttributeValue。 タイプ: `map[string] AttributeValue` 必須: いいえ
PlayerId	プレイヤーを表す一意の識別子。 タイプ: `String`

プロパティ	説明
	必須: いいえ
Team	マッチでプレーヤーが割り当てられるチームの名前。チーム名はマッチメーキングルールセットで定義します。 タイプ: String 必須: いいえ

DescribePlayerSessionsRequest

取得するプレイヤーセッションを指定するオブジェクト。サーバープロセスは、Amazon へのDescribePlayerSessions()呼び出しでこの情報を提供します GameLift。

プロパティ	説明
GameSessionID	一意のゲームセッション識別子。このパラメータを使用して、指定したゲームセッションのすべてのプレイヤーセッションをリクエストします。 ゲームセッション ID の形式は arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string> です。GameSessionID はカスタム ID 文字列または生成された文字列です。 タイプ: String 必須: いいえ
PlayerSessionID	プレイヤーセッションを表す一意の識別子。このパラメータを使用して、特定の 1 つのプレイヤーセッションをリクエストします。 タイプ: String 必須: いいえ
PlayerID	プレイヤーの一意識別子。このパラメータを使用して、特定の 1 人のプレイヤーに対するすべてのプレイヤーセッションをリクエストします。プレイヤー ID を生成する を参照してください。

プロパティ	説明
	タイプ: String 必須: いいえ
PlayerSessionStatusFilter	結果をフィルタリングするプレイヤーセッションステータス。可能なプレイヤーセッションステータスには以下が含まれます。 • RESERVED – プレイヤーセッションリクエストは受領されましたが、プレイヤーはサーバープロセスに接続していないか、または検証はまだ行われていません。 • ACTIVE – プレイヤーはサーバープロセスによって検証され、接続されています。 • COMPLETED – プレイヤー接続は削除されました。 • TIMEDOUT – プレイヤーセッションリクエストは受領されましたが、タイムアウト制限 (60 秒) 内でのプレイヤーの接続や検証は行われていません。 タイプ: String 必須: いいえ
NextToken	結果の次のページの先頭を示すトークン。結果セットの先頭を指定するには、値を指定しないでください。プレイヤーセッション ID を提供する場合、このパラメータは無視されます。 タイプ: String 必須: いいえ
Limit	返される結果の最大数。プレイヤーセッション ID を提供する場合、このパラメータは無視されます。 タイプ: int 必須: いいえ

StopMatchBackfillRequest

マッチメーキングバックフィルリクエストのキャンセルに使用される情報。ゲームサーバーは、この情報を StopMatchBackfill()呼び出しで Amazon GameLift サービスに伝えます。

プロパティ	説明
GameSessionArn	キャンセルされるリクエストの一意のゲームセッション識別子。 タイプ: `string` 必須: いいえ
MatchmakingConfigurationArn	このリクエストが送信されたマッチメーカーの一意の識別子。 タイプ: `string` 必須: いいえ
TicketId	キャンセルされるバックフィルリクエストチケットの一意の識別子。 タイプ: `string` 必須: いいえ

GetFleetRoleCredentialsRequest

AWS リソースへの制限付きアクセスをゲームサーバーにまで拡張するロール認証情報。詳細については、「Amazon の IAM サービスロールを設定する GameLift」を参照してください。

プロパティ	説明
RoleArn	AWS リソースへの制限付きアクセスを拡張するサービスロールの ARN。 タイプ: `string` 必須: はい
RoleSessionName	ロール認証情報の使用を説明するセッションの名前。 タイプ: `string`

プロパティ	説明
	必須: はい

Unreal Engine 用 Amazon GameLift サーバー SDK リファレンス

この Amazon GameLift サーバー SDK リファレンスは、Unreal Engine のゲームプロジェクトを Amazon GameLift で使用するために準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

この API は、`GameLiftServerSDK.h` と `GameLiftServerSDKModels.h` で定義されます。

Unreal Engine プラグインを設定するには、コードサンプル「Amazon GameLift を Unreal Engine プロジェクトに統合する」を参照してください。

トピック

- Amazon GameLift Unreal Engine サーバー SDK 5.x リファレンス
- Amazon GameLift Unreal Engine サーバー SDK 3.x リファレンス

Amazon GameLift Unreal Engine サーバー SDK 5.x リファレンス

この Amazon GameLift Unreal Engine サーバー SDK 5.x リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を、Unreal SDK サーバープラグインの使用方法については「Amazon GameLift を Unreal Engine プロジェクトに統合する」を参照してください。

トピック

- Amazon GameLift サーバー SDK (Unreal) 5.x リファレンス: アクション
- Amazon GameLift サーバー SDK (Unreal) リファレンス: データ型

Amazon GameLift サーバー SDK (Unreal) 5.x リファレンス: アクション

この Amazon GameLift Unreal サーバー SDK リファレンスを使用すると、Amazon で使用するマルチプレイヤーゲームを準備するのに役立ちます GameLift。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を、Unreal SDK サーバープラグインの使用方法については「Amazon GameLift を Unreal Engine プロジェクトに統合する」を参照してください。

アクション

- GetSdkVersion()
- InitSDK()
- InitSDK()
- ProcessReady()
- ProcessEnding()
- ActivateGameSession()
- UpdatePlayerSessionCreationPolicy()
- GetGameSessionId()
- GetTerminationTime()
- AcceptPlayerSession()
- RemovePlayerSession()
- DescribePlayerSessions()
- StartMatchBackfill()
- StopMatchBackfill()
- GetComputeCertificate()
- GetFleetRoleCredentials()

> **Note**
>
> このトピックでは、Unreal Engine 用に構築するときに使用できる Amazon GameLift C++ API について説明します。特に、このドキュメントは `-DBUILD_FOR_UNREAL=1` オプションを使用してコンパイルするコードが対象です。

GetSdkVersion()

サーバープロセスに組み込まれた SDK の現在のバージョン番号を返します。

Syntax

```
FGameLiftStringOutcome GetSdkVersion();
```

戻り値

成功した場合、the section called “FGameLiftStringOutcome” オブジェクトとして現在の SDK バージョンを返します。返されるオブジェクトには、バージョン番号が含まれます (例: `5.0.0`)。成功しなかった場合、エラーメッセージを返します。

例

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
 Aws::GameLift::Server::GetSdkVersion();
```

InitSDK()

マネージド EC2 フリートの Amazon GameLift SDK を初期化します。Amazon に関連する他の初期化 GameLift が発生する前に、起動時にこのメソッドを呼び出します。このメソッドは、ホスト環境からサーバーパラメータを読み取り、サーバーと Amazon GameLift サービス間の通信を設定します。

Syntax

```
FGameLiftGenericOutcome InitSDK()
```

戻り値

成功した場合は、サーバープロセスが ProcessReady() を呼び出す準備ができていることを示す `InitSdkOutcome` オブジェクトを返します。

例

```
//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
FGameLiftGenericOutcome initSdkOutcome = gameLiftSdkModule->InitSDK();
```

InitSDK()

Anywhere フリートの Amazon GameLift SDK を初期化します。Amazon に関連する他の初期化 GameLift が発生する前に、起動時にこのメソッドを呼び出します。この方法では、サーバーと Amazon GameLift サービス間の通信を設定するには、明示的なサーバーパラメータが必要です。

Syntax

```
FGameLiftGenericOutcome InitSDK(serverParameters)
```

パラメータ

FServerParameters

Amazon GameLift Anywhere フリートでゲームサーバーを初期化するには、次の情報を使用して ServerParameters オブジェクトを作成します。

- ゲームサーバーへの接続 WebSocket に使用される の URL。
- ゲームサーバーのホストに使用されるプロセスの ID。
- ゲームサーバープロセスをホスティングするコンピューティングの ID。
- Amazon コンピューティングを含む Amazon GameLift Anywhere GameLift フリートの ID。
- Amazon GameLift オペレーションによって生成された認証トークン。

戻り値

成功した場合は、サーバープロセスが ProcessReady() を呼び出す準備ができていることを示す InitSdkOutcome オブジェクトを返します。

> **Note**
>
> Anywhere フリートにデプロイされたゲームビルドに対して InitSDK() への呼び出しが失敗する場合は、ビルドリソースの作成時に使用した ServerSdkVersion パラメータを確認してください。この値は、使用中のサーバー SDK バージョンに明示的に設定する必要があります。このパラメータのデフォルト値は 4.x で、互換性がありません。この問題を解決するには、新しいビルドを作成して新しいフリートにデプロイします。

例

```
//Define the server parameters
FServerParameters serverParameters;
parameters.m_authToken = "1111aaaa-22bb-33cc-44dd-5555eeee66ff";
parameters.m_fleetId = "arn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa";
```

```
parameters.m_hostId = "HardwareAnywhere";
parameters.m_processId = "PID1234";
parameters.m_webSocketUrl = "wss://us-west-1.api.amazongamelift.com";

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
FGameLiftGenericOutcome initSdkOutcome = gameLiftSdkModule->InitSDK(serverParameters);
```

ProcessReady()

サーバープロセス GameLift がゲームセッションをホストする準備ができたことを Amazon に通知します。InitSDK() を呼び出した後にこのメソッドを呼び出します このメソッドは、プロセスごとに 1 回だけ呼び出す必要があります。

Syntax

```
GenericOutcome ProcessReady(const Aws::GameLift::Server::ProcessParameters
&processParameters);
```

パラメータ

processParameters

サーバープロセスに関する以下の情報を伝える FProcessParameters オブジェクト。

- Amazon GameLift サービスがサーバープロセスと通信するために呼び出すゲームサーバーコードに実装されているコールバックメソッドの名前。
- サーバープロセスがリッスンするポートの番号。
- Amazon がキャプチャして保存するゲームセッション固有のファイル GameLift へのパス。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、ProcessReady() 呼び出しと委任関数の実装の両方を示します。

```
//Calling ProcessReady tells GameLift this game server is ready to receive incoming
 game sessions!
UE_LOG(GameServerLog, Log, TEXT("Calling Process Ready"));
```

```
FGameLiftGenericOutcome processReadyOutcome = gameLiftSdkModule-
>ProcessReady(*params);
```

ProcessEnding()

サーバープロセスが終了中であることを Amazon に通知 GameLift します。他のすべてのクリーンアップタスク (アクティブなゲームセッションのシャットダウンを含む) の後、およびプロセスを終了する前に、このメソッドを呼び出します。ProcessEnding( ) の結果に応じて、プロセスは成功 (0) またはエラー (-1) で終了し、フリートイベントを生成します。プロセスがエラーで終了した場合、生成されるフリートイベントは SERVER_PROCESS_TERMINATED_UNHEALTHY です。

Syntax

```
FGameLiftGenericOutcome ProcessEnding()
```

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
//OnProcessTerminate callback. GameLift will invoke this callback before shutting down
 an instance hosting this game server.
//It gives this game server a chance to save its state, communicate with services,
 etc., before being shut down.
//In this case, we simply tell GameLift we are indeed going to shutdown.
params->OnTerminate.BindLambda([=]() {
  UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
  gameLiftSdkModule->ProcessEnding();
});
```

ActivateGameSession()

サーバープロセスがゲームセッションをアクティブ化し、プレイヤー接続を受信する準備ができた GameLift ことを Amazon に通知します。このアクションは、すべてのゲームセッションの初期化の後、onStartGameSession( ) コールバック関数の一部として呼び出されます。

Syntax

```
FGameLiftGenericOutcome ActivateGameSession()
```

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、onStartGameSession() 委任関数の一部として呼び出された ActivateGameSession() を示しています。

```
//When a game session is created, GameLift sends an activation request to the game
 server and passes along the game session object containing game properties and other
 settings.
//Here is where a game server should take action based on the game session object.
//Once the game server is ready to receive incoming player connections, it should
 invoke GameLiftServerAPI.ActivateGameSession()
auto onGameSession = [=](Aws::GameLift::Server::Model::GameSession gameSession)
{
  FString gameSessionId = FString(gameSession.GetGameSessionId());
  UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"), *gameSessionId);
  gameLiftSdkModule->ActivateGameSession();
};
```

UpdatePlayerSessionCreationPolicy()

現在のゲームセッションの機能を更新し、新しいプレイヤーセッションを承諾します。ゲームセッションは、新しいプレイヤーセッションをすべて受け入れるか拒否するかを設定できます。

Syntax

```
FGameLiftGenericOutcome UpdatePlayerSessionCreationPolicy(EPlayerSessionCreationPolicy
 policy)
```

パラメータ

playerCreationSessionポリシー

ゲームセッションで新しいプレイヤーを承諾するかどうかを示す文字列値。

有効な値を次に示します。

- ACCEPT_ALL – すべての新しいプレイヤーセッションを承諾します。
- DENY_ALL – すべての新しいプレイヤーセッションを拒否します。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例は、現在のゲームセッションの参加ポリシーを、すべてのプレイヤーを受け入れるように設定します。

```
FGameLiftGenericOutcome outcome = gameLiftSdkModule-
>UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::EPlayerSessionCreationPolicy::ACCEPT_A
```

GetGameSessionId()

アクティブなサーバープロセスにホストされたゲームセッションの ID を取得します。

ゲームセッションでアクティブ化されていないアイドル状態のプロセスの場合、呼び出しは the section called “FGameLiftError” を返します。

Syntax

```
FGameLiftStringOutcome GetGameSessionId()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合、ゲームセッション ID を the section called “FGameLiftStringOutcome” オブジェクトとして返します。成功しなかった場合、エラーメッセージを返します。

ゲームセッションでまだアクティブ化されていないアイドルプロセスの場合、呼び出しは `Success=True` および `GameSessionId=""` を返します。

例

```
//When a game session is created, GameLift sends an activation request to the game
 server and passes along the game session object containing game properties and other
 settings.
//Here is where a game server should take action based on the game session object.
```

```
//Once the game server is ready to receive incoming player connections, it should
 invoke GameLiftServerAPI.ActivateGameSession()
auto onGameSession = [=](Aws::GameLift::Server::Model::GameSession gameSession)
{
  FString gameSessionId = FString(gameSession.GetGameSessionId());
  UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"), *gameSessionId);
  gameLiftSdkModule->ActivateGameSession();
};
```

GetTerminationTime()

終了時刻が判る場合に、サーバープロセスがシャットダウンを予定している時刻を返します。サーバープロセスは、Amazon から onProcessTerminate()コールバックを受信した後にアクションを実行します GameLift。Amazon onProcessTerminate() は、次の理由で を GameLift 呼び出します。

- サーバープロセスが正常性の低下を報告した場合、または Amazon に応答しなかった場合 GameLift。
- スケールダウンイベント中にインスタンスを終了する場合。
- スポットインスタンスの中断によりインスタンスが終了した場合。

Syntax

```
AwsDateTimeOutcome GetTerminationTime()
```

戻り値

成功した場合、終了時刻を AwsDateTimeOutcome オブジェクトとして返します。値は終了時間で、0001 00:00:00 以降の経過ティックで表現されます。例えば、日付時刻の値 2020-09-13 12:26:40 -000Z は、637355968000000000 ティックに等しくなります。終了時間がない場合は、エラーメッセージを返します。

プロセスが ProcessParameters.OnProcessTerminate() コールバックを受信していない場合、エラーメッセージが返されます。サーバープロセスのシャットダウンの詳細については、「サーバープロセスのシャットダウン通知に応答する」を参照してください。

例

```
AwsDateTimeOutcome TermTimeOutcome = gameLiftSdkModule->GetTerminationTime();
```

AcceptPlayerSession()

指定されたプレイヤーセッション ID を持つプレイヤーがサーバープロセスに接続し、検証が必要である GameLift ことを Amazon に通知します。Amazon は、プレイヤーセッション ID が有効 GameLift であることを確認します。プレイヤーセッションが検証されると、Amazon はプレイヤースロットのステータスを RESERVED から ACTIVE GameLift に変更します。

Syntax

```
FGameLiftGenericOutcome AcceptPlayerSession(const FString& playerSessionId)
```

パラメータ

playerSessionId

新しいプレイヤーセッションの作成 GameLift 時に Amazon によって発行される一意の ID。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

この例では、無効なプレイヤーセッション ID の検証と拒否を含む接続リクエストを処理します。

```
bool GameLiftManager::AcceptPlayerSession(const FString& playerSessionId, const
 FString& playerId)
{
  #if WITH_GAMELIFT
  UE_LOG(GameServerLog, Log, TEXT("Accepting GameLift PlayerSession: %s . PlayerId:
 %s"), *playerSessionId, *playerId);
  FString gsId = GetCurrentGameSessionId();
  if (gsId.IsEmpty()) {
    UE_LOG(GameServerLog, Log, TEXT("No GameLift GameSessionId. Returning early!"));
    return false;
  }

  if (!gameLiftSdkModule->AcceptPlayerSession(playerSessionId).IsSuccess()) {
    UE_LOG(GameServerLog, Log, TEXT("PlayerSession not Accepted."));
    return false;
  }

  // Add PlayerSession from internal data structures keeping track of connected players
```

```
  connectedPlayerSessionIds.Add(playerSessionId);
  idToPlayerSessionMap.Add(playerSessionId, PlayerSession{ playerId,
 playerSessionId });
  return true;
  #else
  return false;
  #endif
}
```

RemovePlayerSession()

プレイヤーがサーバープロセスから切断された GameLift ことを Amazon に通知します。それに応じて、Amazon GameLift はプレイヤースロットを利用可能に変更します。

Syntax

```
FGameLiftGenericOutcome RemovePlayerSession(const FString& playerSessionId)
```

パラメータ

playerSessionId

新しいプレイヤーセッションの作成 GameLift 時に Amazon によって発行される一意の ID。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
bool GameLiftManager::RemovePlayerSession(const FString& playerSessionId)
{
  #if WITH_GAMELIFT
  UE_LOG(GameServerLog, Log, TEXT("Removing GameLift PlayerSession: %s"),
 *playerSessionId);

  if (!gameLiftSdkModule->RemovePlayerSession(playerSessionId).IsSuccess()) {
    UE_LOG(GameServerLog, Log, TEXT("PlayerSession Removal Failed"));
    return false;
  }

  // Remove PlayerSession from internal data structures that are keeping track of
 connected players
```

```
  connectedPlayerSessionIds.Remove(playerSessionId);
  idToPlayerSessionMap.Remove(playerSessionId);

  // end the session if there are no more players connected
  if (connectedPlayerSessionIds.Num() == 0) {
    EndSession();
  }

  return true;
  #else
  return false;
  #endif
}
```

DescribePlayerSessions()

設定、セッションメタデータ、プレイヤーデータを含む、プレイヤーセッションデータを取得します。このメソッドを使用して、以下に関する情報を取得します。

- シングルプレイヤーセッション
- ゲームセッションのすべてのプレイヤーセッション
- 1 つのプレイヤー ID に関連付けられているすべてのプレイヤーセッション

Syntax

```
FGameLiftDescribePlayerSessionsOutcome DescribePlayerSessions(const
 FGameLiftDescribePlayerSessionsRequest &describePlayerSessionsRequest)
```

パラメータ

FGameLiftDescribePlayerSessionsRequest

取得するプレイヤーセッションを記述する the section called “FGameLiftDescribePlayerSessionsRequest” オブジェクト。

戻り値

成功した場合は、リクエストのパラメータに適合したプレイヤーセッションオブジェクトのセットを含む the section called “FGameLiftDescribePlayerSessionsOutcome” オブジェクトを返します。

例

この例は、指定したゲームセッションにアクティブに接続されているすべてのプレイヤーセッションのリクエストします。を省略NextTokenし、Limit 値を 10 に設定することで、Amazon はリクエストに一致する最初の 10 個のプレイヤーセッションレコード GameLift を返します。

```
void GameLiftManager::DescribePlayerSessions()
{
  #if WITH_GAMELIFT
  FString localPlayerSessions;
  for (auto& psId : connectedPlayerSessionIds)
  {
    PlayerSession ps = idToPlayerSessionMap[psId];
    localPlayerSessions += FString::Printf(TEXT("%s : %s  ; "), *(ps.playerSessionId),
 *(ps.playerId));
  }
  UE_LOG(GameServerLog, Log, TEXT("LocalPlayerSessions: %s"), *localPlayerSessions);

  UE_LOG(GameServerLog, Log, TEXT("Describing PlayerSessions in this GameSession"));
  FGameLiftDescribePlayerSessionsRequest request;
  request.m_gameSessionId = GetCurrentGameSessionId();

  FGameLiftDescribePlayerSessionsOutcome outcome = gameLiftSdkModule-
>DescribePlayerSessions(request);
  LogDescribePlayerSessionsOutcome(outcome);
  #endif
}
```

StartMatchBackfill()

FlexMatch で作成されたゲームセッションの空きスロット用に新規プレイヤーを検索するリクエストを送信します。詳細については、FlexMatch 「バックフィル機能」を参照してください。

このアクションは非同期です。新しいプレイヤーがマッチングされると、Amazon はコールバック関数 を使用して更新されたマッチメーカーデータを GameLift 配信しますOnUpdateGameSession()。

サーバープロセスではアクティブなマッチバックフィルリクエストは一度に 1 つだけです。新しいリクエストを送信するには、まず StopMatchBackfill() を呼び出して元のリクエストをキャンセルする必要があります。

Syntax

```
FGameLiftStringOutcome StartMatchBackfill (FStartMatchBackfillRequest
 &startBackfillRequest);
```

パラメータ

FStartMatchBackfillRequest

次の情報を伝える StartMatchBackfillRequest オブジェクト。

- バックフィルリクエストに割り当てるチケット ID。この情報はオプションです。ID が指定されていない場合、Amazon GameLift は ID を生成します。
- リクエストを送信するマッチメーカー。完全な設定 ARN が必要です。この値はゲームセッションのマッチメーカーデータに含まれています。
- バックフィルするゲームセッションの ID。
- ゲームセッションの現在のプレイヤーに利用可能なマッチメーキングデータ。

戻り値

StartMatchBackfillOutcome オブジェクトを、マッチバックフィルチケット ID またはエラーメッセージを伴うエラーとともに返します。

例

```
FGameLiftStringOutcome FGameLiftServerSDKModule::StartMatchBackfill(const
 FStartMatchBackfillRequest& request)
{
  #if WITH_GAMELIFT
  Aws::GameLift::Server::Model::StartMatchBackfillRequest sdkRequest;
  sdkRequest.SetTicketId(TCHAR_TO_UTF8(*request.m_ticketId));
  sdkRequest.SetGameSessionArn(TCHAR_TO_UTF8(*request.m_gameSessionArn));

 sdkRequest.SetMatchmakingConfigurationArn(TCHAR_TO_UTF8(*request.m_matchmakingConfigurationArn
  for (auto player : request.m_players) {
    Aws::GameLift::Server::Model::Player sdkPlayer;
    sdkPlayer.SetPlayerId(TCHAR_TO_UTF8(*player.m_playerId));
    sdkPlayer.SetTeam(TCHAR_TO_UTF8(*player.m_team));
    for (auto entry : player.m_latencyInMs) {
      sdkPlayer.WithLatencyMs(TCHAR_TO_UTF8(*entry.Key), entry.Value);
    }
```

```
   std::map<std::string, Aws::GameLift::Server::Model::AttributeValue>
sdkAttributeMap;
   for (auto attributeEntry : player.m_playerAttributes) {
     FAttributeValue value = attributeEntry.Value;
     Aws::GameLift::Server::Model::AttributeValue attribute;
     switch (value.m_type) {
       case FAttributeType::STRING:
         attribute =
Aws::GameLift::Server::Model::AttributeValue(TCHAR_TO_UTF8(*value.m_S));
       break;
       case FAttributeType::DOUBLE:
         attribute = Aws::GameLift::Server::Model::AttributeValue(value.m_N);
       break;
       case FAttributeType::STRING_LIST:
         attribute =
Aws::GameLift::Server::Model::AttributeValue::ConstructStringList();
         for (auto sl : value.m_SL) {
           attribute.AddString(TCHAR_TO_UTF8(*sl));
         };
       break;
       case FAttributeType::STRING_DOUBLE_MAP:
         attribute =
Aws::GameLift::Server::Model::AttributeValue::ConstructStringDoubleMap();
         for (auto sdm : value.m_SDM) {
           attribute.AddStringAndDouble(TCHAR_TO_UTF8(*sdm.Key), sdm.Value);
         };
       break;
     }
     sdkPlayer.WithPlayerAttribute((TCHAR_TO_UTF8(*attributeEntry.Key)), attribute);
   }
   sdkRequest.AddPlayer(sdkPlayer);
 }
 auto outcome = Aws::GameLift::Server::StartMatchBackfill(sdkRequest);
 if (outcome.IsSuccess()) {
   return FGameLiftStringOutcome(outcome.GetResult().GetTicketId());
 }
 else {
   return FGameLiftStringOutcome(FGameLiftError(outcome.GetError()));
 }
 #else
 return FGameLiftStringOutcome("");
 #endif
```

```
}
```

StopMatchBackfill()

アクティブなマッチバックフィルリクエストをキャンセルします。詳細については、FlexMatch「バックフィル機能」を参照してください。

Syntax

```
FGameLiftGenericOutcome StopMatchBackfill (FStopMatchBackfillRequest
 &stopBackfillRequest);
```

パラメータ

FStopMatchBackfillRequest

キャンセルするマッチメーキングチケットを識別する StopMatchBackfillRequest オブジェクト：

- バックフィルリクエストに割り当てるチケット ID。
- バックフィルリクエストが送信されたマッチメーカー。
- バックフィルリクエストに関連付けられたゲームセッション。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

```
FGameLiftGenericOutcome FGameLiftServerSDKModule::StopMatchBackfill(const
 FStopMatchBackfillRequest& request)
{
  #if WITH_GAMELIFT
  Aws::GameLift::Server::Model::StopMatchBackfillRequest sdkRequest;
  sdkRequest.SetTicketId(TCHAR_TO_UTF8(*request.m_ticketId));
  sdkRequest.SetGameSessionArn(TCHAR_TO_UTF8(*request.m_gameSessionArn));

 sdkRequest.SetMatchmakingConfigurationArn(TCHAR_TO_UTF8(*request.m_matchmakingConfigurationArn
  auto outcome = Aws::GameLift::Server::StopMatchBackfill(sdkRequest);
  if (outcome.IsSuccess()) {
    return FGameLiftGenericOutcome(nullptr);
  }
  else {
```

```
    return FGameLiftGenericOutcome(FGameLiftError(outcome.GetError()));
  }
  #else
  return FGameLiftGenericOutcome(nullptr);
  #endif
}
```

GetComputeCertificate()

Amazon コンピューティングリソースと Amazon GameLift Anywhere 間のネットワーク接続の暗号化に使用される TLS 証明書へのパスを取得します GameLift。コンピューティングデバイスを Amazon GameLift Anywhere フリートに登録するときに、証明書パスを使用できます。詳細については、「」を参照してくださいRegisterCompute。

Syntax

```
FGameLiftGetComputeCertificateOutcome FGameLiftServerSDKModule::GetComputeCertificate()
```

戻り値

以下を含む GetComputeCertificateResponse オブジェクトを返します。

- CertificatePath: コンピューティングリソースの TLS 証明書へのパス。
- HostName: コンピューティングリソースのホスト名。

例

```
FGameLiftGetComputeCertificateOutcome FGameLiftServerSDKModule::GetComputeCertificate()
{
  #if WITH_GAMELIFT
  auto outcome = Aws::GameLift::Server::GetComputeCertificate();
  if (outcome.IsSuccess()) {
    auto& outres = outcome.GetResult();
    FGameLiftGetComputeCertificateResult result;
    result.m_certificate_path = UTF8_TO_TCHAR(outres.GetCertificatePath());
    result.m_computeName = UTF8_TO_TCHAR(outres.GetComputeName());
    return FGameLiftGetComputeCertificateOutcome(result);
  }
  else {
    return FGameLiftGetComputeCertificateOutcome(FGameLiftError(outcome.GetError()));
  }
  #else
```

```
  return FGameLiftGetComputeCertificateOutcome(FGameLiftGetComputeCertificateResult());
  #endif
}
```

GetFleetRoleCredentials()

Amazon が他の とやり取り GameLift することを許可する IAM ロール認証情報を取得しますAWS のサービス。詳細については、「フリートの他の AWS リソースと通信する」を参照してください。

構文

```
FGameLiftGetFleetRoleCredentialsOutcome
 FGameLiftServerSDKModule::GetFleetRoleCredentials(const
 FGameLiftGetFleetRoleCredentialsRequest &request)
```

パラメータ

FGameLiftGetFleetRoleCredentialsRequest

戻り値

the section called “FGameLiftGetFleetRoleCredentialsOutcome” オブジェクトを返します。

例

```
FGameLiftGetFleetRoleCredentialsOutcome
 FGameLiftServerSDKModule::GetFleetRoleCredentials(const
 FGameLiftGetFleetRoleCredentialsRequest &request)
{
  #if WITH_GAMELIFT
  Aws::GameLift::Server::Model::GetFleetRoleCredentialsRequest sdkRequest;
  sdkRequest.SetRoleArn(TCHAR_TO_UTF8(*request.m_roleArn));
  sdkRequest.SetRoleSessionName(TCHAR_TO_UTF8(*request.m_roleSessionName));

  auto outcome = Aws::GameLift::Server::GetFleetRoleCredentials(sdkRequest);

  if (outcome.IsSuccess()) {
    auto& outres = outcome.GetResult();
    FGameLiftGetFleetRoleCredentialsResult result;
    result.m_assumedUserRoleArn = UTF8_TO_TCHAR(outres.GetAssumedUserRoleArn());
    result.m_assumedRoleId = UTF8_TO_TCHAR(outres.GetAssumedRoleId());
    result.m_accessKeyId = UTF8_TO_TCHAR(outres.GetAccessKeyId());
    result.m_secretAccessKey = UTF8_TO_TCHAR(outres.GetSecretAccessKey());
    result.m_sessionToken = UTF8_TO_TCHAR(outres.GetSessionToken());
```

```
    result.m_expiration = FDateTime::FromUnixTimestamp(outres.GetExpiration());
    return FGameLiftGetFleetRoleCredentialsOutcome(result);
  }
  else {
    return FGameLiftGetFleetRoleCredentialsOutcome(FGameLiftError(outcome.GetError()));
  }
  #else
  return
 FGameLiftGetFleetRoleCredentialsOutcome(FGameLiftGetFleetRoleCredentialsResult());
  #endif
}
```

Amazon GameLift サーバー SDK (Unreal) リファレンス: データ型

この Amazon GameLift Unreal サーバー SDK リファレンスを使用すると、Amazon で使用するマルチプレイヤーゲームを準備するのに役立ちます GameLift。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を、Unreal SDK サーバープラグインの使用方法については「Amazon GameLift を Unreal Engine プロジェクトに統合する」を参照してください。

データ型

- FProcessParameters
- UpdateGameSession
- GameSession
- FServerParameters
- FStartMatchBackfillRequest
- FPlayer
- FGameLiftDescribePlayerSessionsRequest
- FStopMatchBackfillRequest
- FAttributeValue
- FGameLiftGetFleetRoleCredentialsRequest
- FGameLiftLongOutcome
- FGameLiftStringOutcome
- FGameLiftDescribePlayerSessionsOutcome
- FGameLiftDescribePlayerSessionsResult
- FGenericOutcome
- FGameLiftPlayerSession

- FGameLiftGetComputeCertificateOutcome
- FGameLiftGetComputeCertificateResult
- FGameLiftGetFleetRoleCredentialsOutcome
- FGetFleetRoleCredentialsResult
- FGameLiftError
- 列挙型

> **Note**
>
> このトピックでは、Unreal Engine 用に構築するときに使用できる Amazon GameLift C++ API について説明します。特に、このドキュメントは `-DBUILD_FOR_UNREAL=1` オプションを使用してコンパイルするコードが対象です。

FProcessParameters

このデータ型には、 GameLift で Amazon に送信されるパラメータのセットが含まれますProcessReady()。

プロパティ	説明
LogParameters	ゲームセッション中に生成されるファイルへのディレクトリパスを含むオブジェクト。Amazon は、将来のアクセスに備えてファイル GameLift をコピーして保存します。 タイプ: `TArray<FString>` 必須: いいえ
OnHealthCheck	サーバープロセスにヘルスステータスレポートをリクエストするために Amazon が GameLift 呼び出すコールバック関数。Amazon はこの関数を 60 秒ごとに GameLift 呼び出し、応答を 60 秒待機します。サーバープロセスは正常であれば `TRUE` を返し、正常でない場合は `FALSE` を返します。レスポンスが返されない

	場合、Amazon はサーバープロセスを正常でないものとして GameLift 記録します。 このプロパティは、DECLARE_DELEGATE_RetVal(bool, FOnHealthCheck) として定義されるデリゲート関数です。 タイプ: FOnHealthCheck 必須: いいえ
OnProcessTerminate	サーバープロセスを強制シャットダウンするために Amazon が GameLift 呼び出すコールバック関数。この関数を呼び出した後、Amazon はサーバープロセスがシャットダウンするまで 5 分 GameLift 待ってからProcessEnding()、サーバープロセスをシャットダウンします。 タイプ: FSimpleDelegate 必須: はい

OnStartGameSession	新しいゲームセッションをアクティブ化するために Amazon が GameLift 呼び出すコールバック関数。Amazon は、クライアントリクエスト に応答してこの関数を GameLift 呼び出しますCreateGameSession。コールバック関数は、Amazon GameLift API リファレンス で定義されているように GameSession オブジェクトを渡します。 このプロパティは、`DECLARE_DELEGATE_OneParam(FOnStartGameSession, Aws::GameLift::Server::Model::GameSession);` として定義されるデリゲート関数です タイプ: `FOnStartGameSession` 必須: はい
OnUpdateGameSession	更新されたゲームセッションオブジェクトをサーバープロセスに渡すために Amazon が GameLift 呼び出すコールバック関数。Amazon は、マッチバックフィルリクエストが更新されたマッチメーカーデータを提供するために処理されたときに、この関数を GameLift 呼び出します。GameSession オブジェクト、ステータス更新 (updateReason) 、マッチバックフィルチケット ID を渡します。 このプロパティは、`DECLARE_DELEGATE_OneParam(FOnUpdateGameSession, Aws::GameLift::Server::Model::UpdateGameSession);` として定義されるデリゲート関数です タイプ: `FOnUpdateGameSession` 必須: いいえ

ポート	サーバープロセスが新しいプレイヤーの接続をリスンするポート番号。値は、このゲームサーバービルドをデプロイするすべてのフリートで設定されているポート番号の範囲に含まれる必要があります。このポート番号は、ゲームセッションオブジェクトとプレイヤーセッションオブジェクトに含まれ、ゲームセッションがサーバープロセスに接続するときに使用します。 タイプ: `int` 必須: はい

UpdateGameSession

このデータ型はゲームセッションオブジェクトに更新されます。これには、ゲームセッションが更新された理由と、バックフィルを使用してゲームセッション内のプレイヤーセッションを埋めるための関連するバックフィルチケット ID が含まれます。

プロパティ	説明
GameSession	Amazon GameLift API で定義されるGameSessionオブジェクト。`GameSession` オブジェクトにはゲームセッションを説明するプロパティが含まれています。 タイプ: `Aws::GameLift::Server::GameSession` 必須: いいえ
UpdateReason	ゲームセッションが更新されている理由。 タイプ: `enum class UpdateReason` • MATCHMAKING_DATA_UPDATED • BACKFILL_FAILED • BACKFILL_TIMED_OUT

プロパティ	説明
	• BACKFILL_CANCELLED 必須: いいえ
BackfillTicketId	ゲームセッションの更新を試みるバックフィルチケットの ID。 タイプ: char[] 必須: いいえ

GameSession

このデータ型はゲームセッションの詳細を提供します。

プロパティ	説明
GameSessionId	ゲームセッションの一意の識別子。ゲームセッション ARN の形式 は arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token> です。 タイプ: char[] 必須: いいえ
名前	ゲームセッションについて説明するラベル。 タイプ: char[] 必須: いいえ
FleetId	ゲームセッションが実行されているフリートの一意の識別子。 タイプ: char[]

プロパティ	説明
	必須: いいえ
MaximumPlayerSessionCount	ゲームセッションへのプレーヤー接続の最大数。 タイプ: `int` 必須: いいえ
ポート	ゲームセッションのポート番号。Amazon GameLift ゲームサーバーに接続するには、アプリに IP アドレスとポート番号の両方が必要です。 タイプ: `int` 必須: いいえ
IpAddress	ゲームセッションの IP アドレス。Amazon GameLift ゲームサーバーに接続するには、アプリに IP アドレスとポート番号の両方が必要です。 タイプ: `char[]` 必須: いいえ
GameSessionData	単一の文字列値としてフォーマットされたカスタムゲームセッションプロパティのセット。 タイプ: `char[]` 必須: いいえ

プロパティ	説明
MatchmakerData	ゲームセッションの作成に使用されたマッチメーキングプロセスに関する情報。JSON 構文で、文字列としてフォーマットされています。使用されたマッチメーキング設定に加えて、プレイヤー属性やチーム割り当てなど、マッチに割り当てられた全プレイヤーに関するデータが含まれます。 タイプ: `char[]` 必須: いいえ
GameProperties	ゲームセッションのカスタムプロパティのセットで、キーと値のペアとしてフォーマットされます。これらのプロパティは、新しいゲームセッションを開始するリクエストとともに渡されます。 タイプ: `GameProperty[]` 必須: いいえ

プロパティ	説明
DnsName	ゲームセッションを実行しているインスタンスに割り当てられた DNS 識別子。値の形式は次のとおりです。 • TLS 対応フリート: `<unique identifier>.<region identifier>.amazongamelift.com` 。 • TLS 対応でないフリート: `ec2-<unique identifier>.compute.amazonaws.com` 。 TLS 対応フリートで実行しているゲームセッションに接続する場合、IP アドレスではなく DNS 名を使用する必要があります。 タイプ: `char[]` 必須: いいえ

FServerParameters

Amazon GameLift Anywhere サーバーと Amazon GameLift サービス間の接続を維持するために使用される情報。この情報は、InitSDK() で新しいサーバープロセスを起動するときに使用されます。Amazon GameLift マネージド EC2 インスタンスでホストされているサーバーの場合は、空のオブジェクトを使用します。

プロパティ	説明
webSocketUrl	`GameLiftServerSdkEndpoint` Amazon は、Amazon コンピューティングリソースRegisterCompute に対して GameLift Anywhere を GameLift 返す。 タイプ: `char[]`

プロパティ	説明
	必須: はい
processId	ゲームをホストするサーバープロセスに登録された固有の識別子。 タイプ: `char[]` 必須: はい
hostId	`HostID` はコンピューティングを登録したときに使用される `ComputeName` です。詳細については、「」を参照してくださいRegisterCompute。 タイプ: `char[]` 必須: はい
fleetId	コンピューティングが登録されているフリートの固有識別子。詳細については、「」を参照してくださいRegisterCompute。 タイプ: `char[]` 必須: はい
authToken	サーバーを Amazon に対して認証 GameLift する Amazon によって生成された認証トークン GameLift。詳細については、「」を参照してくださいGetComputeAuthToken。 タイプ: `char[]` 必須: はい

FStartMatchBackfillRequest

マッチメーキングバックフィルリクエストの作成に使用される情報。ゲームサーバーは、この情報を StartMatchBackfill()呼び出し GameLift で Amazon に伝えます。

プロパティ	説明
GameSessionArn	一意のゲームセッション識別子。API オペレーション `GetGameSessionId` は ARN 形式の識別子を返します。 タイプ: `char[]` 必須: はい
MatchmakingConfigurationArn	このリクエストに使用されるマッチメーカーの ARN 形式の一意な識別子。元のゲームセッションののマッチメーカー ARN は、マッチメーカーデータプロパティのゲームセッションオブジェクトにあります。マッチメーカーデータの詳細については「マッチメーカーデータの処理」を参照してください。 タイプ: `char[]` 必須: はい
プレイヤー	ゲームセッションに参加しているすべてのプレイヤーを表すデータのセット。マッチメーカーはこの情報を使用して、現在のプレイヤーとマッチする新しいプレイヤーを検索します。 タイプ: `TArray<FPlayer>` 必須: はい
TicketId	マッチメーキングまたはバックフィルリクエストチケットの一意の識別子。値を指定しない場合、Amazon は値 GameLift を生成します。この識別子を使用してマッチバックフィルチケッ

プロパティ	説明
	トのステータスを追跡したり、必要に応じてリクエストをキャンセルしたりします。 タイプ: `char[]` 必須: いいえ

FPlayer

このデータ型はマッチメーキングのプレイヤーを表します。マッチメーキングリクエストを開始すると、プレイヤーはプレイヤー ID、属性、場合によってはレイテンシーデータを保有します。Amazon は、マッチング後にチーム情報 GameLift を追加します。

プロパティ	説明
LatencyInMS	プレイヤーがロケーションに接続したときに発生するレイテンシーの量を示すミリ秒単位の値のセット。 このプロパティを使用すると、プレーヤーはリストに表示されている場所でのみマッチングされます。マッチメーカーにプレイヤーレイテンシーを評価するルールがある場合、プレイヤーはレイテンシーを報告しないとマッチングされません。 タイプ: `TMap>FString, int32<` 必須: いいえ
PlayerAttributes	マッチメーキングに使用するプレイヤー情報を含むキーと値のペアの集合。プレイヤー属性キーは、マッチメーキングルールセット PlayerAttributes で使用される と一致する必要があります。

プロパティ	説明
	プレイヤー属性の詳細については、「」を参照してくださいAttributeValue。 タイプ: `TMap>FString, FAttributeValue<` 必須: いいえ
PlayerId	プレイヤーを表す一意の識別子。 タイプ: `std::string` 必須: いいえ
Team	マッチでプレーヤーが割り当てられるチームの名前。チーム名はマッチメーキングルールセットで定義します。 タイプ: `FString` 必須: いいえ

FGameLiftDescribePlayerSessionsRequest

取得するプレイヤーセッションを指定するオブジェクト。サーバープロセスは、Amazon へのDescribePlayerSessions()呼び出しでこの情報を提供します GameLift。

プロパティ	説明
GameSessionId	一意のゲームセッション識別子。このパラメータを使用して、指定したゲームセッションのすべてのプレイヤーセッションをリクエストします。 ゲームセッション ID の形式は `FString` です。`GameSessionID` はカスタム ID 文字列または

プロパティ	説明
	タイプ: `std::string` 必須: いいえ
PlayerSessionId	プレイヤーセッションを表す一意の識別子。このパラメータを使用して、特定の 1 つのプレイヤーセッションをリクエストします。 タイプ: `FString` 必須: いいえ
PlayerId	プレイヤーの一意識別子。このパラメータを使用して、特定の 1 人のプレイヤーに対するすべてのプレイヤーセッションをリクエストします。プレイヤー ID を生成する を参照してください。 タイプ: `FString` 必須: いいえ

プロパティ	説明
PlayerSessionStatusFilter	結果をフィルタリングするプレイヤーセッションステータス。可能なプレイヤーセッションステータスには以下が含まれます。 • RESERVED – プレイヤーセッションリクエストは受領されましたが、プレイヤーはサーバープロセスに接続していないか、または検証はまだ行われていません。 • ACTIVE – プレイヤーはサーバープロセスによって検証され、接続されています。 • COMPLETED – プレイヤー接続は削除されました。 • TIMEDOUT – プレイヤーセッションリクエストは受領されましたが、タイムアウト制限 (60 秒) 内でのプレイヤーの接続や検証は行われていません。 タイプ: `FString` 必須: いいえ
NextToken	結果の次のページの先頭を示すトークン。結果セットの先頭を指定するには、値を指定しないでください。プレイヤーセッション ID を提供する場合、このパラメータは無視されます。 タイプ: `FString` 必須: いいえ

プロパティ	説明
制限	返される結果の最大数。プレイヤーセッション ID を提供する場合、このパラメータは無視されます。 タイプ: `int` 必須: いいえ

FStopMatchBackfillRequest

マッチメーキングバックフィルリクエストのキャンセルに使用される情報。ゲームサーバーは、この情報を StopMatchBackfill()呼び出しで Amazon GameLift サービスに伝えます。

プロパティ	説明
GameSessionArn	キャンセルされるリクエストの一意のゲームセッション識別子。 タイプ: `FString` 必須: はい
MatchmakingConfigurationArn	このリクエストが送信されたマッチメーカーの一意の識別子。 タイプ: `FString` 必須: はい
TicketId	キャンセルされるバックフィルリクエストチケットの一意の識別子。 タイプ: `FString` 必須: はい

FAttributeValue

これらの値を FPlayer 属性のキーと値のペアで使用します。このオブジェクトでは、文字列、数値、文字列配列、データマップのいずれかの有効なデータ型を使用して属性値を指定できます。各 AttributeValue オブジェクトは、使用可能なプロパティのうちの 1 つだけを使用できます。

プロパティ	説明
attrType	属性値のタイプを指定します。 型: FAttributeType enum 値。 必須: いいえ
S	文字列の属性値を表します。 タイプ: FString 必須: いいえ
N	数値の属性値を表します。 タイプ: double 必須: いいえ
SL	文字列の属性値の配列を表します。 タイプ: TArray<FString> 必須: いいえ
SDM	文字列キーと二重値のディクショナリを表します。 タイプ: TMap<FString, double> 必須: いいえ

FGameLiftGetFleetRoleCredentialsRequest

このデータタイプは、AWS リソースへの制限付きアクセスをゲームサーバーにまで拡張するロール認証情報を提供します。詳細については、「Amazon の IAM サービスロールを設定する GameLift」を参照してください。

プロパティ	説明
RoleArn	AWS リソースへの制限付きアクセスを拡張するサービスロールの Amazon リソースネーム (ARN)。 タイプ: `FString` 必須: いいえ
RoleSessionName	ロール認証情報の使用を説明するセッションの名前。 タイプ: `FString` 必須: いいえ

FGameLiftLongOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 タイプ: `long` 必須: いいえ
ResultWithOwnership	アクションの結果を rvalue としてキャストし、呼び出し元のコードがオブジェクトの所有権を取得できるようにします。 タイプ: `long&&`

プロパティ	説明
	必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “FGameLiftError” 必須: いいえ

FGameLiftStringOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 タイプ: `FString` 必須: いいえ
ResultWithOwnership	アクションの結果を rvalue としてキャストし、呼び出し元のコードがオブジェクトの所有権を取得できるようにします。 タイプ: `FString&&` 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい

プロパティ	説明
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “FGameLiftError” 必須: いいえ

FGameLiftDescribePlayerSessionsOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 タイプ: the section called “FGameLiftDescribePlayerSessionsResult” 必須: いいえ
ResultWithOwnership	アクションの結果を rvalue としてキャストし、呼び出し元のコードがオブジェクトの所有権を取得できるようにします。 タイプ: `FGameLiftDescribePlayerSessionsResult&&` 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “FGameLiftError”

プロパティ	説明
	必須: いいえ

FGameLiftDescribePlayerSessionsResult

プロパティ	説明
PlayerSessions	タイプ: `TArray<FGameLiftPlayerSession>` 必須: はい
NextToken	結果の次のページの先頭を示すトークン。結果セットの先頭を指定するには、値を指定しないでください。プレイヤーセッション ID を提供する場合、このパラメータは無視されます。 タイプ: `FString` 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “FGameLiftError” 必須: いいえ

FGenericOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “FGameLiftError” 必須: いいえ

FGameLiftPlayerSession

プロパティ	説明
CreationTime	タイプ: `long` 必須: はい
FleetId	タイプ: `FString` 必須: はい
GameSessionId	タイプ: `FString` 必須: はい
IpAddress	タイプ: `FString` 必須: はい
PlayerData	タイプ: `FString` 必須: はい
PlayerId	タイプ: `FString`

プロパティ	説明
	必須: はい
PlayerSessionId	タイプ: `FString` 必須: はい
ポート	タイプ: `int` 必須: はい
ステータス	型: A `PlayerSessionStatus` enum。 必須: はい
TerminationTime	タイプ: `long` 必須: はい
DnsName	タイプ: `FString` 必須: はい

FGameLiftGetComputeCertificateOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。 タイプ: the section called “FGameLiftGetComputeCertificateResult” 必須: いいえ
ResultWithOwnership	アクションの結果を rvalue としてキャストし、呼び出し元のコードがオブジェクトの所有権を取得できるようにします。

プロパティ	説明
	タイプ: `FGameLiftGetComputeCertificateResult&&` 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “FGameLiftError” 必須: いいえ

FGameLiftGetComputeCertificateResult

コンピューティングの TLS 証明書へのパスとコンピューティングのホスト名。

プロパティ	説明
CertificatePath	タイプ: `FString` 必須: はい
ComputeName	タイプ: `FString` 必須: はい

FGameLiftGetFleetRoleCredentialsOutcome

このデータ型はアクションの結果で、以下のプロパティを持つオブジェクトを生成します。

プロパティ	説明
結果	アクションの結果。

プロパティ	説明
	タイプ: the section called “FGetFleetRoleCredentialsResult” 必須: いいえ
ResultWithOwnership	アクションの結果を rvalue としてキャストし、呼び出し元のコードがオブジェクトの所有権を取得できるようにします。 タイプ: `FGameLiftGetFleetRoleCredentialsResult&&` 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: `bool` 必須: はい
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “FGameLiftError” 必須: いいえ

FGetFleetRoleCredentialsResult

プロパティ	説明
AccessKeyId	AWS へのアクセスを認証して提供するためのアクセスキー ID。 タイプ: `FString` 必須: いいえ
AssumedRoleId	サービスロールが属するユーザーの ID。

プロパティ	説明
	タイプ: FString 必須: いいえ
AssumedRoleUserArn	サービスロールが属するユーザーの Amazon リソースネーム (ARN)。 タイプ: FString 必須: いいえ
有効期限	セッション認証情報の有効期限が切れるまでの時間。 タイプ: FDateTime 必須: いいえ
SecretAccessKey	認証のためのシークレットアクセスキー ID。 タイプ: FString 必須: いいえ
SessionToken	AWS リソースとやり取りしている現在のアクティブなセッションを識別するトークン。 タイプ: FString 必須: いいえ
成功	アクションが成功したかどうか。 タイプ: bool 必須: はい

プロパティ	説明
エラー	アクションが失敗した場合に発生したエラー。 タイプ: the section called “GameLiftError” 必須: いいえ

FGameLiftError

プロパティ	説明
ErrorType	エラーのタイプ。 型: A `GameLiftErrorType` enum。 必須: いいえ
ErrorName	エラーの名前。 タイプ: `std::string` 必須: いいえ
ErrorMessage	エラーメッセージです。 タイプ: `std::string` 必須: いいえ

列挙型

Amazon GameLift サーバー SDK (Unreal) に定義された列挙型は、次のように定義されます。

FAttributeType

- なし
- STRING
- DOUBLE

- STRING_LIST
- STRING_DOUBLE_MAP

GameLiftErrorType

エラータイプを示す文字列値。有効な値を次に示します。

- SERVICE_CALL_FAILED – AWS サービスへの呼び出しが失敗しました。
- LOCAL_CONNECTION_FAILED – Amazon へのローカル接続 GameLift に失敗しました。
- NETWORK_NOT_INITIALIZED – ネットワークは初期化されていません。
- GAMESESSION_ID_NOT_SET – ゲームセッション ID が設定されていません。
- BAD_REQUEST_EXCEPTION
- INTERNAL_SERVICE_EXCEPTION
- ALREADY_INITIALIZED – Amazon GameLift サーバーまたはクライアントはすでに Initialize() で初期化されています。
- FLEET_MISMATCH – ターゲットフリートがゲームセッションまたはプレイヤーセッションのフリートと一致しません。
- GAMiliFT_CLIENT_NOT_INITIALIZED – Amazon GameLift クライアントは初期化されていません。
- GAMiliFT_SERVER_NOT_INITIALIZED – Amazon GameLift サーバーは初期化されていません。
- GAME_SESSION_ENDED_FAILED – Amazon GameLift Server SDK はサービスに連絡してゲームセッションが終了したことを報告できませんでした。
- GAME_SESSION_NOT_READY – Amazon GameLift サーバーゲームセッションはアクティブ化されていません。
- GAME_SESSION_READY_FAILED – Amazon GameLift Server SDK はサービスに連絡してゲームセッションの準備ができたことを報告することができませんでした。
- INITIALIZATION_MISMATCH – Server:: Initialize() の後にクライアントメソッドが呼び出されました。その逆も同様です。
- NOT_INITIALIZED – Amazon GameLift サーバーまたはクライアントが Initialize() で初期化されていません。
- NO_TARGET_ALIASID_SET – ターゲットのエイリアスが設定されていません。
- NO_TARGET_FLEET_SET – ターゲットフリートが設定されていません。
- PROCESS_ENDING_FAILED – Amazon GameLift Server SDK は、プロセスが終了していることを報告するためにサービスに接続できませんでした。

- PROCESS_NOT_ACTIVE – サーバープロセスはまだアクティブではなく、 にバインドされておらず GameSession、 を受け入れたり処理したりすることはできません PlayerSessions。
- PROCESS_NOT_READY – サーバープロセスをまだアクティブ化する準備ができていません。
- PROCESS_READY_FAILED – Amazon GameLift Server SDK は、プロセスの準備が完了したことを報告するためにサービスに接続できませんでした。
- SDK_VERSION_DETECTION_FAILED – SDK バージョン検出に失敗しました。
- STX_CALL_FAILED – XSTX サーバーのバックエンドコンポーネントへの呼び出しが失敗しました。
- STX_INITIALIZATION_FAILED – XSTX サーバーのバックエンドコンポーネントが初期化に失敗しました。
- UNEXPECTED_PLAYER_SESSION – 未登録のプレイヤーセッションがサーバーによって検出されました。
- WEBSOCKET_CONNECT_FAILURE
- WEBSOCKET_CONNECT_FAILURE_FORBIDDEN
- WEBSOCKET_CONNECT_FAILURE_INVALID_URL
- WEBSOCKET_CONNECT_FAILURE_TIMEOUT
- WEBSOCKET_RETRIABLE_SEND_MESSAGE_FAILURE – GameLift サービスへのメッセージの送信に失敗しました WebSocket。
- WEBSOCKET_SEND_MESSAGE_FAILURE – GameLift サービス にメッセージを送信できません WebSocket。
- MATCH_BACKFILL_REQUEST_VALIDATION – リクエストの検証に失敗しました。
- PLAYER_SESSION_REQUEST_VALIDATION – リクエストの検証に失敗しました。

EPlayerSessionCreationPolicy

ゲームセッションで新しいプレイヤーを承諾するかどうかを示す文字列値。有効な値を次に示します。

- ACCEPT_ALL – すべての新しいプレイヤーセッションを承諾します。
- DENY_ALL – すべての新しいプレイヤーセッションを拒否します。
- NOT_SET – ゲームセッションは、新規プレイヤーセッションを受け入れたり拒否したりするように設定されていません。

EPlayerSessionStatus

- ACTIVE

- COMPLETED
- NOT_SET
- RESERVED
- TIMEDOUT

Amazon GameLift Unreal Engine サーバー SDK 3.x リファレンス

この Amazon GameLift Unreal Engine サーバー SDK 3.x リファレンスは、Amazon GameLift で使用するマルチプレイヤーゲームを準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

トピック

- Unreal Engine アクション用 Amazon GameLift サーバー SDK リファレンス
- Unreal Engine の Amazon GameLift サーバー SDK リファレンス: データ型

Unreal Engine アクション用 Amazon GameLift サーバー SDK リファレンス

この Amazon GameLift サーバー SDK リファレンスは、Unreal Engine のゲームプロジェクトを Amazon GameLift で使用するために準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

この API は、`GameLiftServerSDK.h` と `GameLiftServerSDKModels.h` で定義されます。

Unreal Engine プラグインを設定するには、コードサンプル「Amazon GameLift を Unreal Engine プロジェクトに統合する」を参照してください。

- アクション
- データ型

AcceptPlayerSession()

指定されたプレイヤーセッション ID のプレイヤーがサーバープロセスに接続し、検証が必要であることを Amazon GameLift サービスに通知します。Amazon GameLift は、プレイヤーセッション ID が有効であること、つまり、そのプレイヤー ID でゲームセッションにプレイヤースロットが予約されていることを確認します。検証できたら、Amazon GameLift はプレーヤースロットの状態を RESERVED から ACTIVE に変更します。

構文

```
FGameLiftGenericOutcome AcceptPlayerSession(const FString& playerSessionId)
```

パラメータ

playerSessionId

AWS SDK Amazon GameLift API アクションのCreatePlayerSessionへの呼び出しに応答して Amazon GameLift サービスによって発行された一意の ID。ゲームクライアントは、サーバープロセスに接続するときにこの ID をリファレンスします。

タイプ: FString

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

ActivateGameSession()

サーバープロセスがゲームセッションをアクティブにし、プレイヤーの接続を受ける準備ができていることを Amazon GameLift サービスに通知します。このアクションは、すべてのゲームセッションの初期化が完了した後、onStartGameSession() コールバック関数の一部として呼び出されます。

構文

```
FGameLiftGenericOutcome ActivateGameSession()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

DescribePlayerSessions()

設定、セッションメタデータ、プレイヤーデータを含む、プレイヤーセッションデータを取得します。このアクションを使用して、単一のプレイヤーセッション、ゲームセッション内のすべてのプレ

イヤーセッション、または単一のプレイヤー ID に関連付けられたすべてのプレイヤーセッションに関する情報を取得します。

構文

```
FGameLiftDescribePlayerSessionsOutcome DescribePlayerSessions(const
 FGameLiftDescribePlayerSessionsRequest &describePlayerSessionsRequest)
```

パラメータ

describePlayerSessionsRequest

取得するプレイヤーセッションを記述する FDescribePlayerSessionsRequest オブジェクト。

必須: はい

戻り値

成功した場合は、リクエストのパラメータに適合したプレイヤーセッションオブジェクトのセットを含む FDescribePlayerSessionsRequest オブジェクトを返します。プレイヤーセッションオブジェクトは、AWS SDK Amazon GameLift API PlayerSession データ型と同じ構造を持ちます。

GetGameSessionId()

サーバープロセスがアクティブな場合、サーバープロセスが現在ホストしているゲームセッションの ID を取得します。

構文

```
FGameLiftStringOutcome GetGameSessionId()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合、ゲームセッション ID を FGameLiftStringOutcome オブジェクトとして返します。成功しなかった場合、エラーメッセージを返します。

GetInstanceCertificate()

フリートとそのインスタンスに関連付けられている pem エンコードされた TLS 証明書のファイルの場所を取得します。AWS Certificate Manager は、証明書設定を GENERATED に設定して新しいフリートを作成するとこの証明書が生成されます。この証明書を使用して、ゲームクライアントとのセキュリティ保護ありの接続を確立し、クライアント/サーバー通信を暗号化します。

構文

```
FGameLiftGetInstanceCertificateOutcome GetInstanceCertificate()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功すると、インスタンスに保存されているフリートの TLS 証明書ファイルの場所と証明書チェーンを含む GetInstanceCertificateOutcome オブジェクトを返します。証明書チェーンから抽出されたルート証明書ファイルもインスタンスに保存されます。成功しなかった場合、エラーメッセージを返します。

証明書と証明書チェーンデータの詳細については、AWS Certificate Manager API リファレンスの「GetCertificate レスポンス要素」を参照してください。

GetSdkVersion()

サーバープロセスに組み込まれた SDK の現在のバージョン番号を返します。

構文

```
FGameLiftStringOutcome GetSdkVersion();
```

パラメータ

このアクションにはパラメータがありません。

戻り値

成功した場合、FGameLiftStringOutcome オブジェクトとして現在の SDK バージョンを返します。返される文字列は、バージョン番号のみを含みます(例: 3.1.5)。成功しなかった場合、エラーメッセージを返します。

例

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
    Aws::GameLift::Server::GetSdkVersion();
```

InitSDK()

Amazon GameLift SDK を初期化する。このメソッドは起動時に他の Amazon GameLift 関連の初期化が実行される前に呼び出す必要があります。

構文

```
FGameLiftGenericOutcome InitSDK()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

ProcessEnding()

サーバープロセスがシャットダウンしていることを Amazon GameLift サービスに通知します。このメソッドは、すべてのアクティブゲームセッションのシャットダウンを含む他のすべてのクリーンアップタスクの後に呼び出されます。このメソッドは、終了コード 0 で終了します。0 以外の終了コードでは、処理が問題なく終了しなかったというイベントメッセージが発生します。

構文

```
FGameLiftGenericOutcome ProcessEnding()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

ProcessReady()

サーバープロセスがゲームセッションをホストする準備ができたことを Amazon GameLift サービスに通知します。このメソッドは、InitSDK() の呼び出しが成功して必要な設定タスクが完了した後、サーバープロセスがゲームセッションをホストできるようになる前に呼び出す必要があります。このメソッドは、プロセスごとに 1 回だけ呼び出す必要があります。

構文

```
FGameLiftGenericOutcome ProcessReady(FProcessParameters &processParameters)
```

パラメータ

FProcessParameters

サーバープロセスに関する以下の情報を伝える FProcessParameters オブジェクト。

- サーバープロセスと通信するために Amazon GameLift サービスが呼び出す、ゲームサーバーコードで実装されたコールバックメソッドの名前。
- サーバープロセスがリッスンするポートの番号。
- Amazon GameLift でキャプチャして保存するゲームセッション固有のファイルへのパス。

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

例

Using the Unreal Engine Plugin にあるサンプルコードを参照してください。

RemovePlayerSession()

指定されたプレイヤーセッション ID のプレイヤーがサーバープロセスから切断されたことを Amazon GameLift サービスに通知します。それに応じて、Amazon GameLift はプレイヤースロットを新しいプレイヤーに割り当てられるよう利用可能に変更します。

構文

```
FGameLiftGenericOutcome RemovePlayerSession(const FString& playerSessionId)
```

パラメータ

playerSessionId

AWS SDK Amazon GameLift API アクションのCreatePlayerSessionへの呼び出しに応答して Amazon GameLift サービスによって発行された一意の ID。ゲームクライアントは、サーバープロセスに接続するときにこの ID をリファレンスします。

タイプ: FString

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

StartMatchBackfill()

FlexMatch で作成されたゲームセッションの空きスロット用に新規プレイヤーを検索するリクエストを送信します。AWS SDK アクション StartMatchBackfill() を参照してください。このアクションを使用すると、ゲームセッションをホストするゲームサーバーのプロセスによってマッチバックフィルリクエストを開始できます。FlexMatch バックフィルの特徴の詳細はこちら。

このアクションは非同期です。新規プレイヤーが正常にマッチングされると、Amazon GameLift サービスはコールバック関数 OnUpdateGameSession() を使用して更新済みマッチメーカーデータを送信します。

サーバープロセスではアクティブなマッチバックフィルリクエストは一度に 1 つだけです。新しいリクエストを送信するには、まず StopMatchBackfill() を呼び出して元のリクエストをキャンセルする必要があります。

構文

```
FGameLiftStringOutcome StartMatchBackfill (FStartMatchBackfillRequest
 &startBackfillRequest);
```

パラメータ

FStartMatchBackfillRequest

次の情報を通信する FStartMatchBackfillRequest オブジェクト。

- バックフィルリクエストに割り当てるチケット ID。この情報はオプションです。ID が指定されていない場合は Amazon GameLift が ID を 1 つ自動生成します。
- リクエストを送信するマッチメーカー。完全な設定 ARN が必要です。この値は、ゲームセッションのマッチメーカーデータから取得できます。
- バックフィルされるゲームセッションの ID。
- ゲームセッションの現在のプレイヤーに利用可能なマッチメーキングデータ。

必須: はい

戻り値

成功した場合、マッチバックフィルチケットを FGameLiftStringOutcome オブジェクトとして返します。成功しなかった場合、エラーメッセージを返します。チケットのステータスは、AWS SDK アクション DescribeMatchmaking() を使用して追跡できます。

StopMatchBackfill()

StartMatchBackfill() とともに作成されたアクティブなマッチバックフィルリクエストをキャンセルします。AWS SDK アクション StopMatchmaking() も参照してください。FlexMatch バックフィルの特徴の詳細はこちら。

構文

```
FGameLiftGenericOutcome StopMatchBackfill (FStopMatchBackfillRequest
 &stopBackfillRequest);
```

パラメータ

StopMatchBackfillRequest

キャンセルするマッチメーキングチケットを識別する FStopMatchBackfillRequestオブジェクト:

- キャンセルされるバックフィルリクエストに割り当てられたチケット ID
- バックフィルリクエストが送信されるマッチメーカー
- バックフィルリクエストに関連付けられたゲームセッション

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

TerminateGameSession()

このメソッドは、バージョン 4.0.1 で非推奨となりました。代わりに、サーバープロセスはゲームセッションが終了した後に ProcessEnding() を呼び出す必要があります。

サーバープロセスがゲームセッションをシャットダウンしたことを　Amazon GameLift サービスに通知します。このアクションは、サーバープロセスがアクティブなままになり、新しいゲームセッションをホストするための準備ができたときに呼び出されます。これは、新しいゲームセッションをホストするためにサーバープロセスがすぐに利用可能であることを Amazon GameLift に通知するため、ゲームセッション終了手順が完了した後にのみ呼び出す必要があります。

ゲームセッションが停止した後にサーバープロセスがシャットダウンされる場合は、このアクションは呼び出されません。代わりに、 ProcessEnding() を呼び出し、ゲームセッションとサーバープロセスの両方が終了していることを通知します。

構文

```
FGameLiftGenericOutcome TerminateGameSession()
```

パラメータ

このアクションにはパラメータがありません。

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

UpdatePlayerSessionCreationPolicy()

現在のゲームセッションの機能を更新し、新しいプレイヤーセッションを承諾します。ゲームセッションは、新しいプレイヤーセッションをすべて受け入れるか拒否するかを設定できます。(Amazon GameLift サービス API リファレンスの「UpdateGameSession()」アクションを参照してください。)

構文

```
FGameLiftGenericOutcome UpdatePlayerSessionCreationPolicy(EPlayerSessionCreationPolicy
 policy)
```

パラメータ

ポリシー

ゲームセッションで新しいプレイヤーを承諾するかどうかを示す値。

タイプ: `EPlayerSessionCreationPolicy` enum。有効な値を次に示します。

- ACCEPT_ALL – すべての新しいプレイヤーセッションを承諾します。
- DENY_ALL – すべての新しいプレイヤーセッションを拒否します。

必須: はい

戻り値

正常またはエラーメッセージを伴うエラーの、一般的な結果を返します。

Unreal Engine の Amazon GameLift サーバー SDK リファレンス: データ型

この Amazon GameLift サーバー SDK リファレンスは、Unreal Engine のゲームプロジェクトを Amazon GameLift で使用するために準備するのに役立ちます。統合プロセスの詳細については、「Amazon GameLift をゲームサーバーに追加する」を参照してください。

この API は、`GameLiftServerSDK.h` と `GameLiftServerSDKModels.h` で定義されます。

Unreal Engine プラグインを設定するには、コードサンプル「Amazon GameLift を Unreal Engine プロジェクトに統合する」を参照してください。

- [Actions] (アクション)
- データ型

FDescribePlayerSessionsRequest

このデータ型は、取得するプレイヤーセッションを指定するのに使用されます。このデータ型は次のように使用できます。

- 特定のプレーヤーセッションをリクエストする PlayerSessionId を提供します。
- 指定したゲームセッションのすべてのプレイヤーセッションをリクエストする GameSessionId を提供します。
- 指定したプレイヤーのすべてのプレイヤーセッションをリクエストする PlayerId を提供します。

プレイヤーセッション数が多い場合は、ページ分割パラメータを使用して結果を順次ブロックとして取得します。

目次

GameSessionId

一意のゲームセッション識別子。このパラメータを使用して、指定したゲームセッションのすべてのプレイヤーセッションをリクエストします。ゲームセッション ID の形式は、`arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>` です。<ID string> の値は、カスタム ID 文字列または (ゲームセッション作成時に指定した場合) 生成された文字列のいずれかです。

型: 文字列

必須: いいえ

制限

返される結果の最大数。このパラメータを NextToken とともに使用して、結果を一連の順次ページとして取得します。プレイヤーセッション ID を指定した場合、このパラメータは無視されます。

型: 整数

必須: いいえ

NextToken

結果において次の順次ページの開始を示すトークン。このアクションの以前の呼び出しで返されたトークンを使用します。結果セットの先頭を指定するには、値を指定しないでください。プレイヤーセッション ID を指定した場合、このパラメータは無視されます。

型: 文字列

必須: いいえ

PlayerId

プレイヤーを表す一意の識別子。プレイヤー ID は開発者によって定義されます。「プレイヤー ID を生成する」を参照してください。

型: 文字列

必須: いいえ

PlayerSessionId

プレイヤーセッションを表す一意の識別子。

型: 文字列

必須: いいえ

PlayerSessionStatusFilter

結果をフィルタリングするプレイヤーセッションステータス。可能なプレイヤーセッションステータスとして以下のステータスがあります。

- RESERVED – プレイヤーセッションリクエストは受領されましたが、プレイヤーのサーバープロセスへの接続や検証はまだ行われていません。
- ACTIVE – プレイヤーはサーバープロセスによって検証され、現時点で接続されています。
- COMPLETED – プレイヤー接続は削除されました。
- TIMEDOUT – プレイヤーセッションリクエストは受領されましたが、タイムアウト制限 (60 秒) 内でのプレイヤーの接続や検証は行われていません。

型: 文字列

必須: いいえ

FProcessParameters

このデータ型には、ProcessReady() 呼び出しで Amazon GameLift サービスに送られたパラメータセットが含まれます。

目次

port

サーバープロセスが新しいプレイヤーの接続をリスニングするポート番号。値は、このゲームサーバービルドをデプロイするすべてのフリートで設定されているポート番号の範囲に含まれる必要があります。このポート番号は、ゲームセッションオブジェクトとプレイヤーセッションオブジェクトに含まれ、ゲームセッションがサーバープロセスに接続するときに使用します。

型: 整数

必須: はい

logParameters

ゲームセッションログファイルへのディレクトリパスのリストを含むオブジェクト。

型: TArray<FString>

必須: いいえ

onStartGameSession

Amazon GameLift サービスが新しいゲームセッションをアクティブにするために呼び出すコールバック関数名。Amazon GameLift はクライアントのリクエスト CreateGameSession に応じ、この関数を呼び出します。コールバック関数は GameSession オブジェクト (Amazon GameLift サービス API リファレンスで定義) を取得します。

型: FOnStartGameSession

必須: はい

onProcessTerminate

Amazon GameLift サービスがサーバープロセスを強制シャットダウンするために呼び出すコールバック関数名。この関数を呼び出すと、Amazon GameLift はサーバープロセスがシャットダウンするために 5 分間待ち、サーバープロセスをシャットダウンする前に ProcessEnding() 呼び出しで応答します。

型: FSimpleDelegate

必須: いいえ

onHealthCheck

Amazon GameLift サービスがサーバープロセスにヘルスステータスレポートをリクエストするために呼び出すコールバック関数名。Amazon GameLift は、この関数を 60 秒ごとに呼び出します。この関数を呼び出した後、Amazon GameLift はレスポンスを 60 秒ほど待ちます。レスポンスがなければ、サーバープロセスを異常と記録します。

型: FOnHealthCheck

必須: いいえ

onUpdateGameSession

Amazon GameLift サービスが更新されたゲームセッションオブジェクトを提供するために呼び出すコールバック関数名。Amazon GameLift はこの関数を、更新されたマッチメーカーデータを提

供するためにマッチバックフィルリクエストを処理するときに呼び出します。GameSession オブジェクト、ステータス更新 (updateReason)、およびマッチバックフィルチケット ID が渡されます。

タイプ: FOnUpdateGameSession

必須: いいえ

FStartMatchBackfillRequest

このデータ型はマッチメーキングバックフィルリクエストの送信に使用します。この情報は、StartMatchBackfill() 呼び出しで Amazon GameLift サービスに伝えられます。

目次

GameSessionArn

一意のゲームセッション識別子。API アクション GetGameSessionId() は ARN 形式の識別子を返します。

タイプ: FString

必須: はい

MatchmakingConfigurationArn

このリクエストに使用されるマッチメーカーの ARN 形式の一意な識別子。元のゲームセッションの作成に使用されたマッチメーカーを見つけるには、ゲームセッションオブジェクトのマッチメーカーデータプロパティを確認します。マッチメーカーデータの詳細については「マッチメーカーデータの処理」を参照してください。

タイプ: FString

必須: はい

プレイヤー

現在ゲームセッションに参加しているすべてのプレイヤーを表すデータのセット。マッチメーカーはこの情報を使用して、現在のプレイヤーとマッチする新しいプレイヤーを検索します。プレイヤーオブジェクトの形式の説明については、Amazon GameLift API リファレンスガイドを参照してください。プレイヤー属性、ID、チームの割り当てを見つけるには、マッチメーカーデータプロパティのゲームセッションオブジェクトを参照してください。マッチメーカーでレイテン

シーが使用されている場合は、現在のリージョンの更新されたレイテンシーを収集し、それを各プレイヤーのデータに含めます。

型: TArray<FPlayer>

必須: はい

TicketId

マッチメーキングまたはバックフィルリクエストチケットの一意の識別子。ここで値を指定しない場合、Amazon GameLift によって UUID 形式で自動的に生成されます。この識別子を使用してマッチバックフィルチケットのステータスを追跡したり、必要に応じてリクエストをキャンセルしたりします。

タイプ: FString

必須: いいえ

FStopMatchBackfillRequest

このデータ型はマッチメーキングバックフィルリクエストのキャンセルに使用します。この情報は、StopMatchBackfill() 呼び出しで Amazon GameLift サービスに伝えられます。

目次

GameSessionArn

キャンセルされるリクエストに関連付けられた一意のゲームセッション識別子。

タイプ: FString

必須: はい

MatchmakingConfigurationArn

このリクエストが送信されたマッチメーカーの一意の識別子。

タイプ: FString

必須: はい

TicketId

キャンセルされるバックフィルリクエストチケットの一意の識別子。

タイプ: FString

必須: はい

ゲームセッションプレイスメントイベント

Amazon GameLift は、ゲームセッションプレイスメントリクエストが処理されるたびにイベントを発行します。「ゲームセッション配置のイベン通知を設定」の説明に従って、これらのイベントを Amazon SNS トピックに発行できます。これらのイベントは、ほぼリアルタイムで、ベストエフォートベースで Amazon CloudWatch Events にも出力されます。

このトピックでは、ゲームセッション配置イベントの構造について説明し、各イベントタイプの例を示します。ゲームセッション配置リクエストのステータスの詳細については、Amazon GameLift API リファレンスのGameSessionPlacement「」を参照してください。

プレイスメントイベント構文

イベントは、JSON オブジェクトとして表されます。イベント構造は、類似の最上位フィールドとサービス固有の詳細を含む CloudWatch イベントパターンに準拠しています。

最上位フィールドには、次の項目が含まれます (詳細については[event pattern] (イベントパターンを参照してください):

version

このフィールドは、常に 0 (ゼロ) に設定されます。

id

イベントの一意のトラッキング ID

detail-type (ディテールタイプ)

値は常にGameLift Queue Placement Event です。

ソース

値は常にaws.gamelift です。

アカウント

Amazon の管理に AWS 使用されているアカウント GameLift。

time

イベントのタイムスタンプ

region

プレイスメントリクエストが処理される AWS リージョン。これは、使用中のゲーム セッション キューが存在する リージョンです。

resources

プレイスメント リクエストを処理しているゲーム セッション キューの ARN 値。

PlacementFulfilled

プレイスメントリクエストは正常に実行されました。新しいゲームセッションが開始され、ゲームセッション配置リクエストにリストされた各プレイヤーに対して新しいプレイヤーセッションが作成されました。プレーヤーの接続情報が表示されます。

[Detail syntax:]（詳細構文:

placementId

ゲームセッション配置リクエストに割り当てられた一意の識別子

port

新しいゲームセッションのポート番号。

gameSessionArn

新しいゲームセッションの ARN 識別子

ipAddress

ゲームセッションの IP アドレス。

dnsName

新しいゲームセッションを実行しているインスタンスに割り当てられた DNS 識別子。値の形式は、ゲームセッションを実行しているインスタンスが TLS 対応かどうかによって異なります。TLS 対応フリートのゲームセッションに接続する場合、プレイヤーは IP アドレスではなく DNS 名を使用する必要があります。

TLS 対応フリート: `<unique identifier>.<region identifier>.amazongamelift.com`。

TLS 対応でないフリート: ec2-<unique identifier>.compute.amazonaws.com。

startTime

このリクエストがキューに配置された日時を示すタイムスタンプ。

endTime

このリクエストが完了した日時を示すタイムスタンプ。

gameSessionRegion

AWS ゲームセッションをホストしているフリートのリージョン。これは、 のリージョントークンに対応しますGameSessionArn。

placedPlayerSessions

ゲームセッション配置リクエストでプレイヤーごとに作成されたプレイヤーセッションのコレクション。

例


```
{
  "version": "0",
  "id": "1111aaaa-bb22-cc33-dd44-5555eeee66ff",
  "detail-type": "GameLift Queue Placement Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2021-03-01T15:50:52Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:gamelift:us-west-2:123456789012:gamesessionqueue/MegaFrogRace-NA"
  ],
  "detail": {
    "type": "PlacementFulfilled",
    "placementId": "9999ffff-88ee-77dd-66cc-5555bb44aa",
    "port": "6262",
    "gameSessionArn": "arn:aws:gamelift:us-west-2::gamesession/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa/4444dddd-55ee-66ff-77aa-8888bbbb99cc",
    "ipAddress": "98.987.98.987",
    "dnsName": "ec2-12-345-67-890.us-west-2.compute.amazonaws.com",
    "startTime": "2021-03-01T15:50:49.741Z",
    "endTime": "2021-03-01T15:50:52.084Z",
    "gameSessionRegion": "us-west-2",
    "placedPlayerSessions": [
```

```
      {
        "playerId": "player-1"
        "playerSessionId": "psess-123213123232412412312 3"
      }
    ]
  }
}
```

PlacementCancelled

プレイスメントリクエストは、 GameLift サービス への呼び出しでキャンセルされました[StopGameSessionPlacement](#)。

[Detail:](詳細:)

placementId

ゲームセッション配置リクエストに割り当てられた一意の識別子

startTime

このリクエストがキューに配置された日時を示すタイムスタンプ。

endTime

このリクエストがキャンセルされた日時を示すタイムスタンプ。

例

```
{
  "version": "0",
  "id": "1111aaaa-bb22-cc33-dd44-5555eeee66ff",
  "detail-type": "GameLift Queue Placement Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2021-03-01T15:50:52Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:gamelift:us-west-2:123456789012:gamesessionqueue/MegaFrogRace-NA"
  ],
  "detail": {

    "type": "PlacementCancelled",
```

```
    "placementId": "9999ffff-88ee-77dd-66cc-5555bb44aa",
    "startTime": "2021-03-01T15:50:49.741Z",
    "endTime": "2021-03-01T15:50:52.084Z"
  }
}
```

PlacementTimedOut

キューの時間制限が切れる前に、ゲームセッションの配置が正常に完了しませんでした。プレイスメントリクエストは、必要に応じて再送信できます。

[Detail:](詳細:)

placementId

ゲームセッション配置リクエストに割り当てられた一意の識別子

startTime

このリクエストがキューに配置された日時を示すタイムスタンプ。

endTime

このリクエストがキャンセルされた日時を示すタイムスタンプ。

例

```
{
  "version": "0",
  "id": "1111aaaa-bb22-cc33-dd44-5555eeee66ff",
  "detail-type": "GameLift Queue Placement Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2021-03-01T15:50:52Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:gamelift:us-west-2:123456789012:gamesessionqueue/MegaFrogRace-NA"
  ],
  "detail": {

    "type": "PlacementTimedOut",
    "placementId": "9999ffff-88ee-77dd-66cc-5555bb44aa",
    "startTime": "2021-03-01T15:50:49.741Z",
```

```
    "endTime": "2021-03-01T15:50:52.084Z"
  }
}
```

PlacementFailed

Amazon GameLift はゲームセッションリクエストを満たすことができませんでした。これは通常、予期しない内部エラーが原因で発生します。プレイスメントリクエストは、必要に応じて再送信できます。

[Detail:](詳細:)

placementId

ゲームセッション配置リクエストに割り当てられた一意の識別子

startTime

このリクエストがキューに配置された日時を示すタイムスタンプ。

endTime

このリクエストが失敗した日時を示すタイムスタンプ。

例

```
{
  "version": "0",
  "id": "39c978f3-ba46-3f7c-e787-55bfcca1bd31",
  "detail-type": "GameLift Queue Placement Event",
  "source": "aws.gamelift",
  "account": "252386620677",
  "time": "2021-03-01T15:50:52Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:gamelift:us-west-2:252386620677:gamesessionqueue/MegaFrogRace-NA"
  ],
  "detail": {

    "type": "PlacementFailed",
    "placementId": "e4a1119a-39af-45cf-a990-ef150fe0d453",
    "startTime": "2021-03-01T15:50:49.741Z",
    "endTime": "2021-03-01T15:50:52.084Z"
```

```
    }
}
```

Amazon GameLift の価格見積もりの生成

AWS Pricing Calculator を使用すると、Amazon GameLift の価格見積もりを作成できます。カレンダーを使用するのに AWS アカウント も AWS の深い知識も不要です。

AWS Pricing Calculator 計算ツールを使用すると、サービスコストに影響する決定事項を把握できるため、ゲームプロジェクトに対して Amazon GameLift のコストがどの程度かかるかを理解できます。Amazon GameLift をどのように使用する予定かがまだわからない場合は、デフォルト値を使用して見積もりを作成してください。本番環境での使用を計画する際、この計算ツールは想定されるシナリオをテストし、より正確な見積もりを出すのに役立ちます。

AWS Pricing Calculator を使用して、次の Amazon GameLift ホスティングオプションの見積もりを生成できます。

- Amazon GameLift ホスティングの見積もり
- Amazon GameLift スタンドアロン FlexMatch の見積もり

Amazon GameLift ホスティングの見積もり

このオプションでは、サーバーインスタンスの使用とデータ転送のコストを含む、Amazon GameLift マネージドサーバーでゲームをホストするためのコスト見積もりが表示されます。FlexMatch マッチメーキングは Amazon GameLift マネージドホスティングのコストに含まれています。

複数の AWS リージョンまたは複数のインスタンスタイプでゲームサーバーをホスティングしている、またはホストする予定がある場合は、リージョンとインスタンスタイプごとに見積もりを作成してください。

Amazon GameLift インスタンス

このセクションは、プレイヤーのゲームセッションをホストするのに必要なコンピューティングリソースの種類と数を見積もるのに役立ちます。Amazon GameLift は、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを使用してゲームサーバーを管理します。Amazon GameLift では、特定のインスタンスタイプとオペレーティングシステムを持つインスタンスのフリートをデプロイします。複数のフリートを保有している、または使用する予定の場合は、フリートごとに見積もりを作成してください。

開始するには、AWS Pricing Calculator の [Amazon GameLift を設定] ページを開きます。[説明] を追加し、[リージョン] を選択してから、[Amazon GameLift ホスティングの見積もり (インスタンス +

データ転送アウト)] を選択します。[Amazon GameLift インスタンス] に、以下のフィールドを入力します。

- [ピーク時の同時接続プレイヤー数 (ピーク CCU)]

  これは、ゲームサーバーに同時に接続できるプレーヤーの最大数です。このフィールドは、Amazon GameLift がピーク時のプレイヤー需要を満たすために必要なホスティングキャパシティを示します。選択した AWS リージョンのインスタンスを使用してホストする予定の 1 日のピーク時のプレイヤー数を入力します。

  例えば、一度に 1,000 人のプレイヤーがゲームに接続できるようにしたい場合は、デフォルト値の **1000** のままにします。

- [1 時間あたりの平均 CCU (1 日の最 CCU に対するパーセンテージ)]

  24 時間の 1 時間あたりのプレーヤーの平均数です。この値を使用して、Amazon GameLift がプレイヤーのために維持する必要のある持続的なホスティングキャパシティの量を見積もります。どのパーセンテージ値を使うべきかわからない場合は、デフォルト値の **50** パーセントのままにしてください。プレイヤーの需要が安定しているゲームでは、**70** パーセントの値を入力することをお勧めします。

  例えば、ゲームの 1 時間あたりの平均 CCU が 6,000 で、ピーク CCU が 10,000 の場合、**60** パーセントの値を入力します。

- [インスタンスあたりのゲームセッション]

  これは、各ゲームサーバーインスタンスが同時にホストできるゲームセッションの数です。この数に影響する要素としては、ゲームサーバーのリソース要件、各ゲームセッションでホストされるプレイヤーの数、プレイヤーが期待するパフォーマンスがあります。ゲームの同時ゲームセッション数がわかっている場合は、その値を入力します。または、デフォルト値の **20** のままにします。

- [ゲームセッションあたりのプレーヤー数]

  これは、ゲームデザインで定義されている、ゲームセッションに接続した平均プレイヤー数です。プレイヤー数が異なるゲームモードがある場合は、ゲーム全体のゲームセッションあたりの平均プレイヤー数を見積もります。デフォルト値は、「**8**」です。

- [インスタンスのアイドルバッファ %]

  これは、プレイヤーの需要の突然の急増に対応するために確保しておくべき未使用のホスティングキャパシティの割合です。バッファサイズは、フリート内のインスタンスの合計数の割合です。デフォルト値は **10** パーセントです。

例えば、アイドル状態のバッファが 20% の場合、アクティブなインスタンスが 100 個あるプレイヤーをサポートしているフリートは、20 個のアイドルインスタンスを維持します。

- [スポットインスタンス %]

  Amazon GameLift フリートは、オンデマンドインスタンスとスポットインスタンスを組み合わせて使用できます。オンデマンドインスタンスはより信頼性の高い可用性を提供しますが、スポットインスタンスはコスト効率の高い方法を提供します。コスト削減と可用性の両方を最適化するために、組み合わせて使用することをお勧めします。Amazon GameLift がスポットインスタンスをどのように使用するかについては、「オンデマンドインスタンスとスポットインスタンスの比較」を参照してください。

  このフィールドには、フリート内で維持する [スポットインスタンス] の割合 (パーセント) を入力します。スポットインスタンスの割合は 50 ~ 85% の間で設定することをお勧めします。デフォルト値は **50** パーセントです。

  例えば、100 個のインスタンスを含むフリートをデプロイして **40** パーセントを指定した場合、Amazon GameLift は 60 個のオンデマンドインスタンスと 40 個のスポットインスタンスを維持するようにします。

- インスタンスタイプ

  Amazon GameLift フリートでは、コンピューティング能力、メモリ、ストレージ、ネットワーク機能が異なるさまざまな Amazon EC2 インスタンスタイプを使用できます。Amazon GameLift フリートを設定するときは、ゲームのニーズに最適なインスタンスタイプを選択してください。Amazon GameLift でインスタンスタイプを選択する方法については、「Amazon GameLift 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。

  Amazon GameLift フリートで使用している、または使用する予定のインスタンスタイプがわかっている場合は、そのタイプを選択してください。どのタイプを選択すればよいか明確でない場合は、[c5.large] を選択することを検討してください。これは平均的なサイズと機能を備えた可用性の高いタイプです。

- オペレーティングシステム

  このフィールドでは、ゲームサーバーを実行するオペレーティングシステム (Linux または Windows) を指定します。デフォルト値は [Linux] です。

[データ転送アウト (DTO)]

このセクションは、ゲームクライアントとゲームサーバー間のトラフィックのコストを見積もるのに役立ちます。データ転送料金は、アウトバウンドトラフィックにのみ適用されます。インバウンドデータ転送にはコストはかかりません。

AWS Pricing Calculator の [Amazon GameLift の設定] ページで、[データ転送アウト (DTO)] を展開し、次のフィールドに値を入力します。

- [DTO 見積もりタイプ]

  ゲームのデータ転送を追跡する方法に応じて、次の 2 つの方法のいずれかで DTO を見積もることができます。

  - [1 か月あたり (GB)] – ゲームサーバーの月間トラフィックを追跡する場合は、このタイプを選択してください。
  - [プレイヤーあたり] – プレイヤーごとにデータ転送を追跡する場合は、このタイプを選択します。これはデフォルトのタイプです。

    次のフィールドでは、前のセクションで計算したプレーヤー時間数に基づいてプレーヤー 1 人あたりの DTO を見積もります。

- [DTO/月 (GB 単位)]

  [1 か月あたり (GB)] の DTO 見積もりタイプを選択した場合は、リージョンごとに、各インスタンスからの毎月の DTO 推定使用量を GB 単位で入力します。

- [DTO/プレイヤー]

  [プレイヤーあたり] の DTO 見積もりタイプを選択した場合は、ゲームのプレイヤー 1 人あたりの DTO 推定使用量を KB/秒単位で入力します。デフォルト値は、「**4**」です。

Amazon GameLift の価格見積もりの設定が完了したら、[見積もりに追加] を選択します。AWS Pricing Calculator での見積もりの作成と管理の詳細については、AWS Pricing Calculator ユーザーガイドの「見積もりの作成、サービスの設定、サービスの追加」を参照してください。

Amazon GameLift スタンドアロン FlexMatch の見積もり

このオプションでは、別のゲームサーバーソリューションでゲームをホストする際に、FlexMatch マッチメーキングをスタンドアロンサービスとして使用する場合のコストの見積もりが表示されま

す。これには、FleetIQ を使用した Amazon GameLift セルフマネージドホスティング、オンプレミスホスティング、ピアツーピア、またはクラウドコンピューティングプリミティブデータタイプが含まれます。スタンドアロン FlexMatch のコストは、使用するコンピューティング能力に基づきます。

異なる AWS リージョンに複数のマッチメーカーがある、またはそれらを配置する予定がある場合は、リージョンごとに見積もりを作成してください。

> **Note**
>
> Amazon GameLift FlexMatch は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (ソウル)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、欧州 (フランクフルト)、欧州 (アイルランド) のリージョンでご利用いただけます。

開始するには、AWS Pricing Calculator の [Amazon GameLift を設定] ページを開きます。[説明] を追加し、[リージョン] を選択してから、[Amazon GameLift スタンドアロン FlexMatch の見積もり] を選択します。[Amazon GameLift FlexMatch] で、以下のフィールドに入力します。

- [ピーク時の同時接続プレイヤー数 (ピーク CCU)]

  これは、ゲームサーバーに同時に接続し、マッチメーキングをリクエストできるプレーヤーの最大数です。選択したリージョンでげーうセッションにマッチする予定の 1 日のピーク時のプレイヤー数を入力します。

  例えば、一度に 1,000 人のプレイヤーをマッチしたい場合は、デフォルト値の **1000** のままにします。

- [1 時間あたりの平均 CCU (1 日の最 CCU に対するパーセンテージ)]

  24 時間の 1 時間あたりのプレーヤーの平均数です。この値は、マッチメーキングリクエストの量を見積もるのに役立ちます。どのパーセンテージ値を使うべきかわからない場合は、デフォルト値の **50** パーセントのままにしてください。プレイヤーの需要が安定しているゲームでは、**70** パーセントの値を入力することをお勧めします。

  例えば、ゲームの 1 時間あたりの平均 CCU が 6,000 で、ピーク CCU が 10,000 の場合、**60** パーセントの値を入力します。

- [マッチあたりのプレイヤー数]

これは、ゲームデザインで定義されている、ゲームセッションにマッチする平均プレイヤー数です。プレイヤー数が異なるゲームモードがある場合は、ゲーム全体のゲームセッションあたりの平均プレイヤー数を見積もります。デフォルト値は、「**8**」です。

- [ゲーム時間 (分)]

  これは、プレイヤーがゲームセッションの開始から終了まで継続する平均時間です。この値は、プレイヤーがどのくらいの頻度で新しいマッチを必要とするかを判断するのに役立ちます。プレイヤーの平均ゲーム時間を分単位で入力します。デフォルト値は、「**1**」です。

- [マッチメーキングルールの複雑さ]

  [マッチメーキングルールの複雑さ] とは、プレイヤーのマッチングに使用するルールの数と種類を意味します。ルールセットの複雑さによって、各マッチに必要なコンピューティング能力の量が決まります。

  - [低い複雑度] – マッチメーキングルールセットに含まれるルールが少なく、シンプルなルールタイプ (比較ルールなど) を使用し、少ない試行回数でマッチを成功させるルールがある場合は、このオプションを選択してください。
  - [高い複雑度] – マッチメーキングルールセットに複数のルールが含まれ、より複雑なルールタイプ (距離ルールやレイテンシールールなど) を使用し、制限の厳しいルールで失敗回数が増え、マッチングの試行回数も増える場合は、このオプションを選択します。

  ルールの複雑さと料金の詳細については、Amazon GameLift 料金ページの「Amazon GameLift FlexMatch」を参照してください。

Amazon GameLift FlexMatch の価格見積もりの設定が完了したら、[見積もりに追加] を選択します。AWS Pricing Calculator での見積もりの作成と管理の詳細については、AWS Pricing Calculator ユーザーガイドの「見積もりの作成、サービスの設定、サービスの追加」を参照してください。

クォータとサポートされているリージョン

AWS Amazon GameLift サービスクォータについては、「Amazon GameLift のクォータ」を参照してください。

AWS リソースのクォータ引き上げのリクエストについては、「AWS サービスクォータ」を参照してください。

Amazon GameLift をサポートする AWS リージョン の一覧については、Amazon GameLift Regionを参照してください。

Amazon GameLift リリースノート

Amazon GameLift リリースノートには、サービスに関連する新機能、更新、修正に関する詳細が記載されています。

SDK のバージョン

次の表に、SDK バージョン情報を含むすべての Amazon GameLift リリースを示します。ゲームサーバーとクライアントの統合に同等の SDK を使用する必要はありません。ただし、SDK のある以前のバージョンで、別の SDK の最新機能が完全にはサポートされていない場合はあります。

Amazon GameLift SDKs「」を参照してくださいAmazon での開発サポート GameLift。

最新の Amazon GameLift SDKs「Amazon GameLift SDKs」を参照してください。

現在のバージョン

以前のバージョン

サービスのリリース	AWS SDK	Server SDK				リアルタイムクライアント SDK
		Unity 用 C# プラグイン	C++	Unreal 用 C++	Go	
2023-12-14	1.11.225 以降	5.1.0	5.1.1	5.1.0	5.0.0	1.2.0
2023-11-02	1.11.193 以降	5.1.0	5.1.1	5.1.0	5.0.0	1.2.0
2023-09-28	1.11.144 以降	5.1.0	5.1.1	5.1.0	5.0.0	1.2.0
2023-08-17	1.11.144 以降	5.1.0	5.1.1	5.1.0	5.0.0	1.2.0
2023-07-27	1.11.111 以降	5.1.0	5.1.0	5.0.2	5.0.0	1.2.0

サービスのリリース	AWS SDK	Server SDK				リアルタイムクライアント SDK
		Unity 用 C# プラグイン	C++	Unreal 用 C++	Go	
2023-06-29	1.11.111 以降		5.0.4	5.0.2	5.0.0	1.2.0

サービスのリリース	AWS SDK	Server SDK				リアルタイムクライアント SDK
		Unity 用 C# プラグイン	C++	Unreal 用 C++	Go	
2023-06-15	1.11.87 以降		5.0.4	5.0.2	5.0.0	1.2.0
2023-05-25	1.11.87 以降		5.0.3	5.0.2	5.0.0	1.2.0
2023-04-20	1.11.63 以降		5.0.3	5.0.2	5.0.0	1.2.0
2023-04-13	1.10.21 以降		5.0.0	5.0.0	5.0.0	1.2.0
2023-02-09	1.10.21 以降		5.0.0	3.4.0	5.0.0	1.2.0
2023-01-31	1.10.21 以降	5.0.0	5.0.0	3.4.0	5.0.0	1.2.0

サービスのリリース	AWS SDK	Server SDK				リアルタイムクライアント SDK
		Unity 用 C# プラグイン	C++	Unreal 用 C++	Go	
2022-12-01	1.10.21 以降		5.0.0	3.4.0		1.2.0
2022-08-25	1.9.333 以降		3.4.2	3.4.0		1.2.0
2021-10-28	1.9.133 以降		3.4.2	3.4.0		1.2.0
2021-06-03	1.8.168 以降		3.4.2	3.4.0		1.2.0
2021-03-23	1.8.168 以降		3.4.1	3.3.3		1.1.0
2021-03-16	1.8.163 以降		3.4.1	3.3.3		1.1.0
2021-02-09	1.8.139 以降		3.4.1	3.3.3		1.1.0
2020-12-22	1.8.95 以降		3.4.1	3.3.3		1.1.0
2020-11-24	1.8.95 以降		3.4.1	3.3.2		1.1.0
2020-11-11	1.8.36 以降		3.4.1	3.3.2		1.1.0
2020-09-17	1.8.36 以降		3.4.1	3.3.2		1.1.0
2020-08-27	1.7.310 以降		3.4.0	3.3.1		1.1.0

サービスのリリース	AWS SDK	Server SDK				リアルタイムクライアント SDK
		Unity 用 C# プラグイン	C++	Unreal 用 C++	Go	
2020-04-16	1.7.310 以降		3.4.0	3.3.1		1.1.0
2020-04-02	1.7.310 以降		3.4.0			1.1.0
2019-12-19	1.7.249 以降		3.4.0			1.1.0
2019-11-14	1.7.210 以降		3.4.0			1.1.0
2019-10-24	1.7.210 以降		3.4.0			1.1.0
2019-09-03	1.7.175 以降		3.4.0			1.1.0
2019-07-09	1.7.140 以降		3.3.0			1.0.0
2019-04-25	1.7.91 以降		3.3.0			1.0.0
2019-03-07	1.7.65 以降		3.3.0			
2019-02-07	1.7.45 以降		3.3.0			
2018-12-14	1.6.20 以降		3.3.0			
2018-09-27	1.6.20 以降		3.2.1			

サービスのリリース	AWS SDK	Server SDK				リアルタイムクライアント SDK
		Unity 用 C# プラグイン	C++	Unreal 用 C++	Go	
2018-06-14	1.4.47 以降		3.2.1			
2018-05-10	1.4.47 以降		3.2.1			
2018-02-15	1.3.58 以降		3.2.1			
2018-02-08	1.3.52 以降		3.2.0			
2017-09-01	1.1.43 以降		3.1.7			
2017-08-16	1.1.31 以降		3.1.7			
2017-05-16	1.0.122 以降		3.1.5			
2017-04-11	1.0.103 以降		3.1.5			
2017-02-21	1.0.72 以降		3.1.5			
2016-11-18	1.0.31 以降		3.1.0			
2016-10-13	1.0.17 以降		3.1.0			
2016-09-01	0.14.9 以降		3.1.0			

サービスのリリース	AWS SDK	Server SDK				リアルタイムクライアント SDK
		Unity 用 C# プラグイン	C++	Unreal 用 C++	Go	
2016-08-04	0.12.16 以降		3.0.7			

リリースノート

次のリリースノートは時系列順です (更新が新しい順にリストされています)。Amazon GameLift は 2016 年に初めてリリースされました。ここに記載されているものより前の日付のリリースノートについては、「SDK のバージョン」でリリース日のリンクを参照してください。

2024 年 4 月 24 日: Amazon がコンテナフリート GameLift を起動

Amazon GameLift では、コンテナフリートのプレビューが提供されるようになりました。これにより、移植性、スケーラビリティ、耐障害性、俊敏性が向上します。

コンテナフリートでは、Amazon EC2 インスタンスは 1 つ以上のコンテナをホストします。これらのコンテナには、依存関係や設定など、必要なものとともにゲームサーバーが含まれます。依存関係の例としては、 SDKsやソフトウェアパッケージなどがあります。コンテナをプライベート Amazon Elastic Container Registry にアップロードすると、Amazon はフリートにコンテナ GameLift を入力します。

コンテナフリートで機能するには、ゲームサーバーが Linux で実行され、Server SDK 5.x と統合されている必要があります。コンテナフリートでは、ホスティングリソースを微調整して、CPU ユ

ニットやメモリなどのリソースの消費を最適化できます。リソースの使用を減らすために、コンテナで複数のゲームサーバーをホストすることもできます。

コンテナフリートでは、オンデマンドインスタンスタイプ、スケーリング (自動および手動)、キュー、マッチメーキングなど、他のタイプのフリートと同じ利点の多くが得られます。また、他のフリートタイプと同じメトリクスと、コンテナ用のいくつかの新しいメトリクスも取得できます。コンテナフリートを使用すると、以下のロケーションリージョンのプレイヤーにグローバルにアクセスできます。

- ap-northeast-1
- ap-northeast-2
- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2

さらに多くのリージョンとローカルゾーンに到達するには、マルチロケーションコンテナフリートを作成します。

詳細はこちら:

- Amazon GameLift コンテナによるホスティングの管理、Amazon GameLift デベロッパーガイド
- CreateContainerGroupDefinition、Amazon GameLift API リファレンス

2024 年 2 月 13 日: Amazon が SDKsの改善 GameLift を開始し、Unreal Engine 用 Amazon GameLift プラグインのインストールを簡素化

SDK バージョンを更新しました。

- Go Server SDK、バージョン 5.1.0
- C# Server SDK、バージョン 5.1.2
- C++ Server SDK、バージョン 5.1.2

以下の改善を行いました。

- ネットワークの中断時に自動再接続を追加することで、SDK の信頼性が向上しました。
- [Go] サーバーパラメータ`InitSDK()`の有無にかかわらず を呼び出すことができるようになりました。Amazon GameLift マネージド EC2 フリートで実行されるゲームサーバーは、環境変数から直接サーバーパラメータを読み取ります。Amazon GameLift Anywhere フリートのゲームサーバーは、サーバーパラメータ`InitSDK()`を使用して を呼び出す必要があります。

プラグインのバージョンを更新しました。

- Unreal Engine 用 Amazon GameLift プラグイン、バージョン 1.1.0
- Unity 用 Amazon GameLift プラグイン、バージョン 2.1.0
- Unreal 用 C++ Server SDK プラグイン、バージョン 5.1.1
- Unity 用 C# Server SDK プラグイン、バージョン 5.1.2

以下の改善を行いました。

- [Amazon GameLift plugin for Unreal Engine] インストール手順を更新し、パッケージを簡素化しました。このプラグインには、最新バージョンの C++ Server SDK for Unreal が含まれるようになりました。
- GameLift Server SDK の最新バージョンをサポートするようにプラグインをアップグレードしました。

詳細はこちら:

- Unreal Engine 用 Amazon GameLift プラグインとゲームの統合、Amazon GameLift デベロッパーガイド
- Amazon GameLift プラグインと SDK ダウンロード

2023 年 12 月 14 日: Amazon がアクティブなゲームセッションのゲームプロパティを更新する機能 GameLift を追加

ゲームセッションの作成時にゲームプロパティを設定し、指定したプロパティをゲームセッションで検索できるようになりました。アクティブなゲームセッションでこれらのプロパティを追加および更新できるようになりました。

例えば、プレイヤーがプレイするマップに投票します。ゲームクライアントは`UpdateGameSession`を呼び出して `GameProperty` の値を変更します`{"Key": "map",`

"Value":"jungle"}。その後、ゲームはゲームセッションでプレイヤーの新しいマップを実装します。

ゲーム管理者は、 SearchGameSessionsオペレーションを使用してゲームプロパティから有用なデータを取得することもできます。例えば、管理者は Statusの値ACTIVEとこのゲームプロパティを持つゲームセッションを一覧表示できます{"Key": "map", "Value":"desert"}。

詳細はこちら:

- the section called “Amazon GameLift をゲームクライアントに追加する”、Amazon GameLift デベロッパーガイド
- GameProperty、Amazon GameLift API リファレンス
- UpdateGameSession、Amazon GameLift API リファレンス
- SearchGameSessions、Amazon GameLift API リファレンス

2023 年 11 月 21 日: Amazon は、 を搭載した Terraform や Pulumi などの Infrastructure as Code ツールのサポート GameLift を開始 AWS Cloud Control API

Infrastructure as Code (IaC) ツールを使用して、Amazon GameLift リソーススタック全体を管理できるようになりました。これらのツールには AWS CloudFormation、、Terraform や Pulumi などのサードパーティーツールが含まれます。このサポートを追加することで、ゲームの構築に集中し、DevOps 戦略を活用してリソース管理、CI/CD、顧客へのデプロイを管理できるようになりました。

AWS Cloud Control API を使用して、すべての Amazon GameLift リソースタイプをプロビジョニングおよび設定できるようになりました。Amazon GameLift APIs または Amazon の AWS CloudFormation テンプレートを使用して、引き続き リソースを操作できます GameLift。

IaC で利用可能な Amazon GameLift リソースの詳細については、「Amazon GameLift リソースタイプリファレンス」の「Amazon GameLift リソースタイプリファレンス」を参照してください。 IaC

さらに、新しいフリートプロパティ を使用して、 AWS CloudFormation テンプレートまたは AWS Cloud Control API を使用してフリートを自動的にスケーリングできるようになりましたScalingPolicies。

Cloud Control API は、数百の AWS サービスと Terraform や Pulumi などの複数のサードパーティーツールにまたがるリソース (CRUDL) を作成、読み取り、更新、削除、一覧表示するための標準 APIs セットをデベロッパーに提供します。

詳細はこちら:

- AWS CloudFormation
- AWS クラウドコントロール API
- AWS CC Terraform プロバイダー
- プルミ

2023 年 11 月 16 日: Amazon が Unity のスタンドアロンプラグイン GameLift を更新

更新された SDK バージョン： Unity 用 Amazon GameLift プラグイン、バージョン 2.0.0

Unity 用 Amazon GameLift プラグインは、Amazon によるクラウドホスティングのために Unity ゲームを起動して実行する手順を合理化するツールとワークフローを提供します GameLift。Amazon GameLift は、ゲームデベロッパーがセッションベースのマルチプレイヤーゲーム専用のゲームサーバーを管理およびスケーリングできるようにするフルマネージドサービスです。

このバージョンでは、Unity 用プラグインが更新され、サーバー SDK バージョン 5.x や Amazon GameLift Anywhere でのローカルテストのサポートなど、最新の Amazon GameLift 機能が使用されます。プラグインは Unity バージョン Unity 2021.3 LTS および 2022.3 LTS と互換性があります。

プラグインの主な機能は次のとおりです。

- Unity エディタのガイド付き UI ワークフローでは、次のシナリオが使用できます。
  - ローカルワークステーションをホストとして使用 GameLift して、Amazon とのゲーム統合をテストします。このワークフローは、ローカルマシンの Amazon GameLift Anywhereフリートのセットアップ、ゲームサーバーとクライアントのインスタンスの起動、Amazon 経由でのゲームセッションのリクエスト GameLift、ゲームへの参加に役立ちます。
  - Amazon GameLift マネージド EC2 とサポート AWS リソースを使用して、統合されたゲームサーバー用のクラウドホスティングソリューションをデプロイします。このワークフローは、クラウドホスティング用にゲームを設定するのに役立ち、次の 3 つのデプロイシナリオを提供します。
    - ゲームサーバーを単一のフリートにデプロイします。
    - ゲームサーバーを複数の AWS リージョンの低コストのスポットフリートのセットにデプロイします。
    - FlexMatch マッチメーカーを使用してゲームサーバーをデプロイします。

- AWS アカウントユーザーにリンクするユーザープロファイルを設定し、デフォルトの AWS リージョンを設定する機能。複数のプロファイルを維持して、異なる AWS アカウント、アカウントユーザー、およびリージョンで機能させることができます。
- Amazon GameLift の統合とデプロイプロセスを合理化するのに役立つ次のような特別な利便性。
  - 各ホスティングソリューションには、一意のプレイヤー IDs とプレイヤー検証を提供する Amazon Cognito ユーザープールなどのサポート AWS リソースが含まれています。このソリューションには、ストレージ用の Amazon S3 バケット、Amazon SNS イベント通知、 AWS Lambda 関数、その他のリソースも含まれています。
  - Anywhere ワークフローの場合、プラグインは必要なサーバーパラメータ設定を自動化します。
  - Amazon EC2 ワークフローでは、各デプロイソリューションが Lambda 関数を使用する組み込みのクライアントバックエンドサービスを提供します。バックエンドサービスは、ゲームクライアントと Amazon GameLift サービスの間にあり、Amazon GameLift サービスへのすべての直接呼び出しを管理します。
- ゲームサーバーとゲームクライアントの統合を説明するためのシンプルなサンプルマルチプレイヤーゲームのアセットやコードなど、統合テストのコンテンツ。
- 詳細な統合ガイダンスとサンプルコードを含むプラグインドキュメント。

Anywhere および Amazon EC2 フリートを含むすべてのデプロイシナリオでは、 AWS CloudFormation テンプレートを使用してゲームのソリューションの AWS リソースを記述およびデプロイします。これらのテンプレートは Amazon GameLift プラグインのダウンロードに含まれています。そのまま使用することも、ゲームに合わせてカスタマイズすることもできます。

詳細はこちら:

- サーバー SDK 5.x 用 Unity ガイド用 Amazon GameLift プラグイン、Amazon GameLift デベロッパーガイド
- からプラグインをダウンロードする GitHub
- Amazon GameLift ホスティングについて
- Amazon GameLift フォーラム

2023 年 11 月 2 日: Amazon が共有認証情報のサポート GameLift を追加

更新された SDK バージョン： AWS SDK 1.11.193

新しい Amazon GameLift 共有認証情報機能を使用すると、マネージド EC2 フリートにデプロイされたアプリケーションは、他の AWS リソースとやり取りできます。この更新は、サーバー

SDK バージョン 5.x 以降と統合されたゲームサーバーバイナリとともにバンドルおよびデプロイするアプリケーションに影響します。(ゲームサーバー実行ファイルは、サーバー SDK 5.x `GetFleetRoleCredentials()` アクションを使用して認証情報をリクエストできるようになっています)。

例えば、Amazon CloudWatch エージェントを使用してゲームサーバービルドをデプロイして EC2 インスタンスのメトリクスやその他のデータを収集する場合、エージェントには CloudWatch リソースを操作するためのアクセス許可が必要です。これを行うには、まずリソースを使用するアクセス許可を持つ AWS Identity and Access Management IAM) ロールを設定し CloudWatch 、次に IAM ロールと共有認証情報を有効にしてフリートを設定する必要があります。Amazon がゲームサーバービルドを各 EC2 インスタンスに GameLift デプロイすると、共有認証情報ファイルが生成され、インスタンスに保存されます。インスタンス上のすべてのアプリケーションは、共有認証情報を使用できます。Amazon は、インスタンスの存続期間中、一時的な認証情報 GameLift を自動的に更新します。

以下の方法でマネージド EC2 フリートを作成すると、共有認証情報を有効にできます。

- Amazon GameLift コンソールのフリート作成ワークフロー。
- 新しいパラメータ `CreateFleet`を使用して Amazon GameLift サービス API オペレーションを呼び出す場合`InstanceRoleCredentialsProvider`。
- パラメータ `aws gamelift create-fleet`を使用して AWS CLI オペレーションを呼び出す場合`instance-role-credentials-provider`。

詳細はこちら:

- フリートの他の AWS リソースと通信する、Amazon GameLift デベロッパーガイド
- CreateFleet、 InstanceRoleCredentialsProvider、Amazon GameLift API リファレンス
- IAM サービスロールのセットアップ、Amazon GameLift デベロッパーガイド

2023 年 9 月 28 日: Amazon が Unreal Engine 用の新しいスタンドアロンプラグインを GameLift リリース

更新された SDK バージョン： Unreal Engine バージョン 1.0.0 用の Amazon GameLift プラグイン

Unreal Engine 用 Amazon GameLift プラグインは、クラウドホスティング GameLift 用に Amazon でゲームを起動して実行するステップを効率化するツールとワークフローを提供します。Amazon GameLift は、ゲームデベロッパーがセッションベースのマルチプレイヤーゲーム専用のゲームサー

バーを管理およびスケーリングできるようにするフルマネージドサービスです。このプラグインは UE バージョン 5.0、5.1、5.2 をサポートしています。主な特徴は以下のとおりです。

- Unreal Editor のガイド付き UI ワークフローは以下の手順をたどります。
  - ローカルワークステーションをホストとして使用 GameLift して、Amazon とのゲーム統合をテストします。このワークフローは、ローカルマシンの Amazon GameLift Anywhereフリートのセットアップ、ゲームサーバーとクライアントのインスタンスの起動、Amazon を介したゲームセッションのリクエスト GameLift、新しいゲームセッションの接続情報の取得に役立ちます。
  - 統合ゲームサーバーに Amazon EC2 クラウドホスティングソリューションをデプロイします。このワークフローは、クラウドホスティング用にゲームを設定するのに役立ち、1 つのフリートへのデプロイ、複数のリージョンのスポットフリートのセットへのデプロイ、 FlexMatch マッチメーカーによるフリートのセットへのデプロイの 3 つの異なるデプロイシナリオを提供します。各デプロイシナリオのソリューションには、Amazon GameLift リソースとサポート AWS リソースが含まれます。
- AWS アカウントユーザーにリンクし、デフォルトの AWS リージョンを定義するユーザープロファイルを設定する機能。複数のプロファイルを維持して、異なる AWS アカウント、アカウントユーザー、およびリージョンで機能させることができます。
- Amazon GameLift の統合とデプロイプロセスを合理化するのに役立つ次のような特別な利便性。
  - 各ホスティングソリューションには、一意のプレイヤー ID を提供する基本的な Amazon Cognito ユーザープール、ストレージ用の Amazon S3 バケット、Amazon SNS イベント通知、AWS Lambda 関数などのサポート AWS リソースが含まれています。 IDs
  - Anywhere ワークフローでは、プラグインはコマンドライン引数を使用して必要なサーバーパラメータ設定を自動化します。
  - Amazon EC2 ワークフローでは、各デプロイソリューションが Lambda 関数を使用する組み込みのクライアントバックエンドサービスを提供します。バックエンドサービスは、ゲームクライアントからリクエストを受け取り、Amazon GameLift サービスに渡します。
- スターターゲームマップおよび基本的なブループリントと UI 要素を含む 2 つのテストマップを含む、統合テスト用のコンテンツ。
- 詳細な統合ガイダンスとサンプルコードを含むプラグインドキュメント。

Anywhere および Amazon EC2 フリートを含むすべてのデプロイシナリオでは、 CloudFormation テンプレートを使用して AWS ソリューションを記述します。プラグインは、ゲームに Amazon GameLift リソースをデプロイするときにこれらのテンプレートを使用します。これらのテンプレートは Amazon GameLift プラグインのダウンロードに含まれ、編集可能です。そのまま使用することも、ゲームに合わせて変更することもできます。

詳細はこちら:

- Unreal Engine 用 Amazon GameLift プラグインとのゲームの統合、Amazon GameLift デベロッパーガイド
- からプラグインをダウンロードする GitHub
- Amazon GameLift ホスティングについて
- Amazon GameLift フォーラム

2023 年 8 月 17 日: Amazon GameLift が AWS Graviton プロセッサによるゲームサーバーホスティングを提供

更新された SDK バージョン： AWS SDK 1.11.144

Amazon GameLift では、 AWS Graviton プロセッサを搭載した EC2 インスタンスを使用して、クラウドでゲームをホストできるようになりました。Arm64-basedプロセッサ AWS を使用して によって設計された Graviton インスタンスは、EC2 を使用するクラウドワークロードに最適な価格のパフォーマンスを提供し、同等の x86 ベースのインスタンスよりも最大 40% 向上します。最新の Graviton3 プロセッサは、以前のバージョンに比べてコンピューティング性能が最大 25% 向上しています。

Amazon では GameLift、 AWS Graviton ファミリーの新しいインスタンスから選択できるようになりました。

- Graviton2 ベースのインスタンス: c6g、c6gn、r6g、m6g、g5g
- Graviton3 ベースのインスタンス: c7g、r7g、m7g

詳細はこちら:

- AWS Graviton プロセッサ : Graviton ベースの EC2 インスタンスの利点と実用性について説明します。
- Graviton の開始方法: Graviton ベースのインスタンスの概要と、オペレーティングシステム、言語、実行時間に応じてアプリケーションがどのように実行されるかに関するインサイトを取得します。

> **Note**
>
> Graviton Arm インスタンスには、Linux OS 上に構築された Amazon GameLift サーバーが必要です。C++ と C# には、サーバー SDK 5.1.1 以降が必要です。Go にはサーバー SDK 5.0 以降が必要です。これらのインスタンスは、Amazon Linux 2023 (AL2023) または Amazon Linux 2 (AL2) でのモノラルインストール out-of-the-box をサポートしていません。

2023 年 7 月 27 日: Amazon が Unity 開発のサポートを追加したサーバー SDK 5.1.0 を GameLift リリース

更新された SDK バージョン: C++ 用サーバー SDK、C#/Unity、Unreal 5.1.0

Amazon GameLift サーバー SDK の最新リリースでは、C++、C#、Unreal プラグインの更新と、Unity ゲームエンジンで使用する新しいプラグインが配信されます。ゲームデベロッパーは、Amazon GameLift サーバー SDK を Amazon でホストするためにデプロイするゲームサーバーに統合します GameLift。

最新のサーバー SDK バージョンには、カスタマーからの多数のリクエストを含む以下の更新が含まれています。

- 言語固有の SDK パッケージのダウンロード – 更新された Amazon GameLift ダウンロードサイトには、各言語の SDK パッケージが含まれています。現在のバージョンまたは以前のバージョンをダウンロードできます。
- Unity 用の新しい C# サーバー SDK プラグイン – Unity 用の新しいサーバー SDK パッケージには、Unity Editor のパッケージマネージャーを使用してインストールできる構築済みの C# ライブラリが含まれています (新しい「Unity 統合ガイド」を参照)。これらのライブラリには、 を通じて必要な依存関係が含まれています UnityNuGet。このプラグインは Windows および Mac OS 用の Unity 2020.3 LTS、2021.3 LTS、2022.3 LTS で使用できます。Unity の.NET Frameworkと.NET Standard (.NET Standard 2.1 と .NET 4.x) の.NET Standard プロファイルをサポートしています。
- C# 用の統合.NET ソリューション – C# 用サーバー SDK は、.NET Framework 4.6.2 (4.6.1 からアップグレード) と.NET 6.0 を単一のソリューションでサポートするようになりました。.NET 標準 2.1 は Unity で構築したライブラリで利用可能です。
- サーバー SDK 5.1.0 の更新
  - [C++、C#、Unreal] サーバーパラメータの有無にかかわらず `InitSDK()` を呼び出すことができるようになりました。Amazon GameLift マネージド EC2 フリートで実行されるゲームサーバーは、環境変数から直接サーバーパラメータを読み取ります。Amazon GameLift Anywhere フ

リートのゲームサーバーは、サーバーパラメータInitSDK()を使用して を呼び出す必要があります。

- [C++、C #、Unreal] サーバー SDK の呼び出しにより エラーメッセージが改善されました。
- [C++ SDK] サーバー SDK のビルド時間を短縮するため、ビルドフラグ -DRUN_CLANG_FORMAT はデフォルトで無効になっています。-DRUN_CLANG_FORMAT=1 で有効にできます。
- [C++ SDK] 標準ライブラリ (-DGAMELIFT_USE_STD=0) なしでライブラリをビルドすると、InitSDK() は std:: データ型を使用しなくなりました。

- サーバー SDK 5.x のドキュメントの拡大
  - C++、C#/Unity、および Unreal のサーバー SDK リファレンスガイドが更新され、すべてのデータ型の対象範囲が拡大されました。
    - C# と Unity 用 Amazon GameLift サーバー SDK 5.x リファレンス
    - C++ 用 Amazon GameLift サーバー SDK 5.x リファレンス
    - Amazon GameLift Unreal Engine サーバー SDK 5.x リファレンス
  - Unity プラグインと Unreal プラグイン用のサーバー SDK 5 統合ガイドの新バージョン
    - Unity プロジェクトに Amazon GameLift を統合する
    - Amazon GameLift を Unreal Engine プロジェクトに統合する
- ドキュメントのその他の更新
  - Amazon GameLift サービス API オペレーションのドキュメントを改訂GetComputeAccessしGetInstanceAccess、使用中の Amazon GameLift サーバー SDK バージョンに基づくリモートアクセス手順を明確にしました。
  - プレイスメントが「保留中」ステータスの場合にゲームセッション情報が一時的なものである方法をドキュメントGameSessionPlacement化するために、 の説明を改訂しました。

2023 年 7 月 13 日: Amazon がフリートハードウェアメトリクス GameLift を追加

Amazon GameLift マネージド EC2 フリートのハードウェアパフォーマンスメトリクスを追跡できるようになりました。メトリクスには、CPU 使用率、ネットワークトラフィック量、ディスクの読み取り/書き込みアクティビティに関する EC2 インスタンスのメトリクスが含まれます。Amazon の場合 GameLift、これらのメトリクスはフリートロケーション内のすべてのアクティブなインスタンスを記述します。これらのフリートハードウェアメトリクスは、 の Amazon CloudWatch ダッシュボードを使用して表示できます AWS Management Console。フリートの詳細で Amazon GameLift コンソールで表示することもできます。

詳細はこちら:

- Amazon CloudWatch で Amazon GameLift をモニタリングする (フリートのメトリクス)、Amazon GameLift デベロッパーガイド

2023 年 6 月 29 日: Amazon が Amazon Linux 2023 のサポート GameLift を開始

SDK バージョンの更新： AWS SDK 1.11.111

Amazon の GameLift お客様は、Amazon Linux 2023 オペレーティングシステムを使用してゲームサーバーをホストできるようになりました。AL2023 には、セキュリティなど、AL2 に比べていくつかの改善点があります。このオペレーティングシステムは、中国リージョン AWS リージョン を除くすべての で使用できます。

2023 年 12 月に Amazon Linux (AL1) のサポートが終了しても、カスタマーは新しい Linux オペレーティングシステムを使用でき、重要なセキュリティアップデートを引き続き受信することができます。Amazon Linux 2 のサポートは、2025 年まで続きます。

詳細はこちら:

- Amazon GameLift Linux Server に関するよくある質問
- Amazon Linux 2 と Amazon Linux 2023 の比較
- Amazon GameLift API リファレンスのリンク：
  - AWS SDK アクション `CreateBuild`
  - CLI コマンド `upload-build`
  - CLI コマンド `create-build`

2023 年 5 月 25 日: Amazon GameLift FleetIQ がドレインインスタンスでのゲームセッション配置を除外するフィルターを追加

更新された SDK バージョン： AWS SDK 1.11.87

ゲームホスティングに Amazon GameLift FleetIQ を使用する場合、現在ドレインしているインスタンスでのゲームセッションの配置を防ぐことができるようになりました。ドレイン中のインスタンスにはシャットダウンのフラグが立てられますが、他にホスティングリソースがない場合は、新しいゲームセッションをホストするように選択できます。この新機能を使えば、ドレインするインスタンスの使用を完全に除外できます。

この機能は、利用可能なゲームサーバーを検索するために `ClaimGameServer` を呼び出すときに使用します。新しい `FilterOption` パラメータを追加し、許可されるインスタンスステータスを [アクティブのみ] に設定します。それに応じて、Amazon GameLift FleetIQ は利用可能なゲームサーバーを検索して要求するときに、アクティブなインスタンスのみを調べます。

詳細はこちら:

- ClaimGameServer 「Amazon GameLift API リファレンス」の「」
- 「Amazon FleetIQ デベロッパーガイド」の「FleetIQ の仕組み GameLift FleetIQ」

2023 年 5 月 16 日: Amazon がフリートのコスト配分タグ付け GameLift をサポート

Amazon の GameLift お客様は、 AWS Billing コスト配分タグを使用してゲームホスティングコストを整理できるようになりました。個々の Amazon GameLift EC2 フリートリソースにコスト配分タグを割り当てることで、フリートが全体的なホスティングコストにどのように寄与しているかを追跡できます。

詳細はこちら:

- ゲームホスティングコストを管理する
- 「AWS Billing ユーザーガイド」の「AWS コスト配分タグの使用」

2023 年 4 月 20 日: Amazon が Windows Server 2016 のサポート GameLift を開始

更新された SDK バージョン： AWS SDK 1.11.63

Amazon の GameLift お客様は、Windows Server 2016 オペレーティングシステムを使用してゲームサーバーをホストできるようになりました。このオペレーティングシステムは、すべての で使用できます AWS リージョン。Microsoft が 2023 年 10 月に Windows Server 2012 のサポートを終了しても、カスタマーは新しい Windows オペレーティングシステムを使用でき、重要なセキュリティ更新プログラムを引き続き受け取ることができます。

本日より、Windows ランタイム環境を必要とする新規のカスタマーは、ホスティング用の新しいゲームサーバービルドを作成する際に Windows Server 2016 を指定する必要があります。既存のカスタマーは、引き続き Windows Server 2012 を使用して新しいビルドやフリートを作成できますが、2023 年 10 月 10 日の Microsoft のサポート終了日までに Windows Server 2016 への移行を完了する必要があります。

この更新では、以下がサービス変更されています。

- Amazon GameLift SDK または CLI コマンドを使用してゲームサーバービルドを作成する場合、オペレーティングシステムを明示的に設定する必要があります。デフォルト値はなくなりました。Windows Server 2016 にゲームサーバーをデプロイするには、`WINDOWS_2016` 値を使用します。
- Amazon GameLift コンソールを使用してゲームサーバービルドを作成する場合は、使用可能な値からオペレーティングシステムを選択する必要があります。Windows Server 2012 フリートがアクティブになっている既存のカスタマーの場合は、`WINDOWS_2012` または `WINDOWS_2016` を選択できます。

詳細はこちら:

- Amazon GameLift API リファレンスのリンク：
  - CLI コマンド `upload-build`
  - CLI コマンド `create-build`
  - AWS SDK アクション `CreateBuild`
- Windows 2012 の Amazon に関する GameLift よくある質問

2023 年 4 月 13 日: Amazon が Unreal 用のサーバー SDK 5.x GameLift を起動

更新された SDK バージョン: Unreal 用 Server SDK 5.0.0

Unreal Engine 用 Amazon GameLift 軽量プラグインの最新バージョンが Amazon GameLift サーバー SDK 5.x に基づくようになりました。Unreal Engine 環境と Amazon の統合を開始するには、次のリンク GameLift を参照してください。

詳細はこちら:

- Amazon GameLift を Unreal Engine プロジェクトに統合する
- Amazon GameLift をゲームサーバーに追加する
- C++ 用 Amazon GameLift サーバー SDK 5.x リファレンス

2023 年 3 月 14 日: Amazon が新しいコンソールエクスペリエンス GameLift を開始

新しい Amazon GameLift コンソールには、次の改善点が含まれています。

- ナビゲーションの改善 – 新しいナビゲーションペインにより、Amazon GameLift リソース間のナビゲーションが容易になります。

- Amazon GameLift ランディングページ – 新しいランディングページには、役立つドキュメントへのリンク、Amazon の概要の表示 GameLift、ドキュメント、よくある質問、および へのリンクによるサポートが用意されています AWS re:Post。
- Amazon CloudWatch メトリクスの改善 – Amazon GameLift メトリクスが Amazon GameLift コンソールと CloudWatchダッシュボードの両方で利用可能になりました。この更新には、パフォーマンス、使用率、プレイヤーセッションの新しいメトリクスも含まれています。

詳細はこちら:

- コンソールでのゲームデータの表示
- Amazon GameLift ホスティングリソースの管理
- FlexMatch マッチメーカーの構築

2023 年 2 月 14 日: Amazon は Amazon SNS トピックのサーバー側の暗号化をサポートする GameLift ようになりました

SNS トピックのサーバー側の暗号化 (SSE) は、保存中の機密データを暗号化します。SSE は AWS Key Management Service （ AWS KMS) キーを使用して SNS トピックの内容を保護します。

詳細はこちら:

- ゲームセッション配置のイベン通知を設定
- FlexMatch マッチメーキングイベント
- 保管時の暗号化

2023 年 2 月 9 日: Amazon GameLift サーバー SDK が C#10 で .NET 6 をサポート

更新された SDK バージョン: NET 6. 用 Server SDK 5.0.0 SDK の更新は必要ありません。

Unity リアルタイム開発プラットフォームを使用する場合は、.NET 4.6 で Amazon GameLift サーバー SDK 5.0.0 を引き続き使用します。Unity は .NET 6 をサポートしていません。

詳細はこちら:

- Amazon 入 GameLift 門ガイドで Amazon GameLift サーバー SDK の最新バージョンをダウンロードする
- C# と Unity 用 Amazon GameLift サーバー SDK 5.x リファレンス

2023 年 1 月 31 日: Amazon GameLift サーバー SDK が Go 言語をサポート

更新された SDK バージョン: Go 用 Server SDK 4.0.0

詳細はこちら:

- Amazon 入 GameLift 門ガイドで Amazon GameLift サーバー SDK の最新バージョンをダウンロードする
- Go 用 Amazon GameLift サーバー SDK リファレンス

2022 年 12 月 1 日: Amazon が Amazon GameLift Anywhere および Amazon GameLift Server SDK 5.0 GameLift を起動

更新された SDK バージョン： AWS SDK 1.10.21、C++ および C# 用の Server SDK 5.0.0

Amazon GameLift Anywhere は、ゲームサーバーリソースを使用して Amazon GameLift ゲームサーバーをホストします。Amazon GameLift Anywhere を使用して独自のコンピューティングリソースを Amazon GameLift マネージド EC2 コンピューティングと統合し、ゲームサーバーを複数のコンピューティングタイプに分散できます。Amazon GameLift Anywhere を使用して、 GameLift ビルドを Amazon にアップロードせずにゲームサーバーを繰り返しテストすることもできます。

ハイライト:

- 新しい Amazon GameLift Anywhere フリートとコンピューティングタイプ
- Amazon GameLift Anywhere コンピューティングリソース登録
- テストの反復サイクルの改善

Amazon GameLift Server SDK 5.0.0 では、既存のサーバー SDK と新しいリソースタイプであるコンピューティングが改善されています。Server SDK 5.0.0 は Amazon GameLift Anywhere と、ゲームサーバーホスティングのための独自のコンピューティングリソースの使用をサポートしています。

詳細はこちら:

- Amazon GameLift サーバー SDK リファレンス
- フリートの口ケーション
- Amazon GameLift コンピューティングリソースの選択
- Amazon GameLift Anywhere フリートを作成する

2022 年 8 月 25 日: Amazon が Local Zones のサポート GameLift を開始

SDK バージョンの更新： AWS SDK 1.9.333

Amazon GameLift が米国の 8 つのローカルゾーンで利用可能になりました。これにより、プレイヤーの近くにフリートをデプロイできます。ローカルゾーンをフリートに追加することで、ローカルゾーンですべてのマネージド Amazon GameLift 機能を使用できます。

Local Zones は、 AWS リソースとサービスを、大規模な人口、業界、情報技術 (IT) センターに近いクラウドのエッジに拡張します。つまり、1 桁ミリ秒単位のレイテンシーを必要とするアプリケーションを、エンドユーザーやオンプレミスのデータセンターの近くにデプロイできます。

詳細はこちら:

- ローカルゾーン
- フリートのロケーション
- Amazon GameLift マネージドフリートを作成する

2022 年 6 月 28 日: Amazon が新しいオプトインコンソールエクスペリエンス GameLift を開始

新しい Amazon GameLift コンソールには、次の改善点が含まれています。

- ナビゲーションの改善 – 新しいナビゲーションペインにより、Amazon GameLift リソース間のナビゲーションが容易になります。
- Amazon GameLift ランディングページ – 新しいランディングページには、役立つドキュメントへのリンク、Amazon の概要の表示 GameLift、ドキュメント、よくある質問、および へのリンクによるサポートが用意されています AWS re:Post。
- Amazon CloudWatch メトリクスの改善 – Amazon GameLift メトリクスが Amazon GameLift コンソールと CloudWatchダッシュボードの両方で利用可能になりました。この更新には、パフォーマンス、使用率、プレイヤーセッションの新しいメトリクスも含まれています。

詳細はこちら:

- コンソールでのゲームデータの表示
- Amazon GameLift ホスティングリソースの管理
- FlexMatch マッチメーカーの構築

2022 年 2 月 15 日: 複合ルールとその他の改善 FlexMatch を追加

FlexMatch ユーザーは、次の機能にアクセスできるようになりました。

- 複合ルール – 40 人以下のプレイヤーのマッチを対象とする複合マッチメイキングルールのサポートが追加されました。ロジカルステートメントを使用して複合ルールを作成してマッチを構成できるようになりました。ルールセットに複合ルールがない場合にマッチを形成するには、ルールセット内のすべてのルールが満たされている必要があります。複合ルールでは、`and`、`or`、`not`、`xor` の論理演算子を使用して適用するルールを選択できます。
- 柔軟なチーム選択 – マッチメイキングプロパティの式が更新され、利用可能な全チームのサブセットを選択できるようになりました。
- 文字列リストの拡大 – プレイヤー属性値の文字列リストに含まれる文字列の最大数を 10 から 100 に増加しました。

詳細はこちら:

- Amazon GameLift デ FlexMatch ベロッパーガイド:
  - FlexMatch ルールタイプ
  - FlexMatch プロパティ式
- AttributeValue: SL

2021 年 10 月 28 日: Amazon がアジアパシフィック (大阪) リージョンでマルチリージョンフリートのサポート GameLift を追加、Amazon GameLift FleetIQ が AWS Graviton2 プロセッサのサポートを追加

SDK バージョンの更新： AWS SDK 1.9.133

Amazon GameLift がアジアパシフィック (大阪) リージョンで利用可能になりました。ゲームデベロッパーは GameLift 、マルチリージョンフリートを使用して大阪にインスタンスをデプロイできるようになりました。

Arm ベースのプロセッサアーキテクチャに基づく Graviton2 ホストゲームサーバーを使用して、同等のインテルベースのコンピューティングオプションと比較して、低コストでパフォーマンスを向上させることができます。

ハイライト:

- Amazon GameLift がアジアパシフィック (大阪) リージョンで利用可能になりました。
- Amazon GameLift FleetIQ ゲームサーバーグループを設定して、Graviton2 インスタンスファミリー c6g、m6g、r6g を管理できるようになりました。

詳細はこちら:

- Amazon GameLift マルチリージョンフリート
- CreateGameServerGroup
- AWS グラビトンプロセッサ

2021 年 9 月 20 日: Amazon が Unity 用のプラグインを GameLift リリース

Unity バージョン 1.0.0 用の Amazon GameLift プラグインには、Amazon GameLift リソースへのアクセスと Unity ゲーム GameLift への Amazon の統合を容易にするライブラリとネイティブ UI が含まれています。Unity 用 Amazon GameLift プラグインを使用して Amazon GameLift APIs、一般的なゲームシナリオ用の AWS CloudFormation テンプレートをデプロイできます。プラグインには、サンプルシナリオで動作するサンプルゲームも含まれています。Amazon GameLift Local を使用して、ゲームクライアントとゲームサーバーの間で渡されたメッセージを表示し、一般的なゲームが Amazon とどのように相互作用するかを確認できます GameLift。

Unity 用プラグインは Unity 2019.4 LTS と 2020.3 LTS をサポートしています。

ハイライト:

- さまざまなシナリオでサンプルゲームを構築、実行、変更するか、独自のシナリオを作成します。
- 認証のみ、単一リージョンフリート、キューとカスタムマッチメーカーを備えたマルチリージョンフリート、キューとカスタムマッチメーカーを備えたスポットフリート、 など、一般的なゲームシナリオのサンプル AWS CloudFormation シナリオをデプロイします FlexMatch。

詳細はこちら:

- Unity 用 Amazon GameLift プラグインとのゲームの統合

2021 年 6 月 30 日: batchDistance ルール FlexMatch を追加

BatchDistance ルールタイプを使用すると、文字列または数値属性を指定でき、各セグメントに多くのメリットがもたらされます。

ハイライト:

- 大規模なマッチ (40人以上) では、スキルだけで均等にバランスをとる代わりに、スキル、モード、マップに基づいて同じバランスを取得できるようになりました。マッチの全員がスキルバンドに所属していることを確認し、リーグやプレイスタイルなどの複数の数値属性をバンド化し、マップやゲームモードなどの文字列属性に従ってグループ化します。時間の経過とともに拡張を作成することもできます。たとえば、プレイヤーが待っている時間が長いほど、より大きなスキルレベル範囲をマッチに入れるように拡張を作成できます。

  40 人未満のマッチでは、新しい簡略化されたルール式を使用できます。

2021 年 6 月 3 日: Amazon GameLift リアルタイムクライアント SDK とサーバー SDK の更新

更新された SDK バージョン: リアルタイムクライアント SDK 1.2.0、Unreal 用 サーバー SDK 3.4.0

この最新の SDK の更新により、RTS クライアント SDK を使用するモバイルアプリケーションに IL2CPP を統合し、Frameworks のベストプラクティスに従うことができるようになりました。Amazon GameLift Server SDK for Unreal バージョン 4.26 を構築できるようになりました。この更新には、Amazon Server SDK GameLift、Amazon Local、Unreal Engine プラグインの C++ および C# バージョンなど、Windows または Linux ゲーム GameLift サーバーと統合するコンポーネントが含まれています。

ハイライト:

- RTS クライアント SDK での IL2CPP のサポートとネイティブライブラリをフレームワークとして構築するサポートを追加したため、RTS クライアントを最新のモバイルデバイス用に構築できるようになりました。
- DescribePlayerSessions() を使用して、単一のプレイヤーセッション、ゲームセッション内のすべてのプレイヤーセッション、または単一のプレイヤー ID に関連付けられたすべてのプレイヤーセッションに関する情報を取得します。
- GetInstanceCertificate() を使用してフリートとそのインスタンスに関連付けられている PEM エンコードされた TLS 証明書のファイルの場所を取得します。

- Unreal バージョン 4.26 のサーバー SDK サポートを作成しました。
- 既存の C# SDK バージョン 4.0.2 は、Unity 2020.3 との互換性が確認されています。SDK の更新は必要ありませんでした。

詳細はこちら:

- Amazon GameLift デベロッパーガイド :
  - DescribePlayerSessions()
  - GetInstanceCertificate()

2021 年 3 月 23 日: Amazon がゲームセッションプレイスメントに通知 GameLift を追加

SDK バージョンの更新： AWS SDK 1.8.168

イベントを使用して、ゲームセッションキューのゲームセッション配置アクティビティをモニタリングできるようになりました。Amazon Simple Notification Service (Amazon SNS) トピックを作成してイベント通知を発行するか、 CloudWatch イベントを使用してイベント追跡を設定します。

ハイライト:

- キューごとに、すべてのイベントメッセージに含めるカスタムテキスト文字列を設定できます。
- Amazon SNS トピックを使用する場合、公開を特定のキューに制限する追加のアクセス条件を設定できます。

詳細はこちら:

- Amazon GameLift デベロッパーガイド :
  - ゲームセッション配置のイベン通知を設定 (新規)
  - ゲームセッションプレイスメントイベント (新規)
- API リファレンス (AWS SDK)
  - 新しいゲームセッションキューパラメータ `NotificationTarget`および `CustomEventData`: GameSessionQueue、CreateGameSessionQueue、 UpdateGameSessionQueue
- Amazon GameLift フォーラム

2021 年 3 月 16 日: Amazon がマルチリージョンフリート、6 つの新しいリージョン GameLift を追加

更新された SDK バージョン： AWS SDK 1.8.163

Amazon GameLift マネージドホスティングが 21 AWS リージョンで利用可能になりました。新しいリージョンはケープタウン (af-south-1)、バーレーン (me-south-1)、香港 (ap-east-1)、ミラノ (eu-south-1)、パリ (eu-west-3)、ストックホルム (eu-north-1)です。

新しい Amazon GameLift マルチロケーションフリート機能を使用して、Amazon が GameLiftサポートする 20 のリージョン (北京リージョンを除く) のいずれかまたはすべてでゲームサーバーをホストするように 1 つのフリートを設定できるようになりました。この機能は、Amazon GameLift ホスティングリソースをグローバルにセットアップして維持するために必要な作業を大幅に削減することを目的としています。マルチロケーションフリートは、 us-east-1 (バージニア北部) 、 us-west-2 (オレゴン) eu-central-1、 (フランクフルト) eu-west-1、 (アイルランド) 、 ap-southeast-2 (シドニー) 、 ap-northeast-1 (東京) 、 ap-northeast-2 (ソウル) の各 AWS リージョンで作成できます。他のすべてのリージョンでは、必要に応じて単一ロケーションフリートをセットアップし続けることができます。このリリースより前に作成されたすべてのフリートは、シングルロケーションフリートです。マルチロケーションフリートを使用しても、ホスティングコストには影響しません。Amazon GameLift の料金は、使用するインスタンスのタイプ、場所、ボリュームに基づきます。(詳細については、「Amazon GameLift の料金」を参照してください。) マルチロケーションフリート AWS CloudFormation のサポートは間もなく利用可能になります。

> **Note**
>
> 中国リージョンでは、マルチロケーションフリートは使用できません。中国リージョンに存在する Amazon GameLift リソースは、他の Amazon GameLift リージョンのリソースとやり取りしたり、それらのリソースで使用したりすることはできません。

ハイライト:

- マルチロケーションフリートでは、リモートロケーションのリストを明示的に追加します。Amazon は、ビルドとランタイムの設定を含む同じタイプと設定のインスタンスを、フリートのホームリージョンと追加されたすべてのロケーションに GameLift デプロイします。
- ロケーションごとに容量設定とスケーリングを個別に調整します。Auto Scaling ポリシーは、フリート全体に適用されますが、ロケーションごとにオンとオフを切り替えることができます。

- 特定のフリートロケーションで新しいゲームセッションを開始します。ゲームセッションキューまたはマッチメイキングを使用してゲームセッションを配置する場合、ロケーション、ホスティングコスト、プレイヤーのレイテンシーによって、新しいゲームセッションスタートのロケーションの優先順位が付けられるようになりました。
- Amazon GameLift コンソールでホスティングメトリクスを取得し、フリート内のすべてのロケーションについて集計するか、フリートのロケーションごとに分割します。

詳細はこちら:

- Amazon ゲームテックブログ
- API リファレンス (AWS SDK)
  - 新しいフリートロケーションオペレーション: CreateFleetLocations、DescribeFleetLocationAttributesDescribeFleetLocationCapacity、、DescribeFlee DeleteFleetLocations
  - フリートオペレーションを更新し、新しいマルチロケーションサポート: CreateFleet、UpdateFleetCapacity、DescribeEC2InstanceLimits、DescribeInstances、StopFleetAction StartFleetActions
  - ゲームセッション配置オペレーションを更新し、新しい優先度とフィルタリング機能: CreateGameSessionQueue、DescribeGameSessionQueues、 UpdateGameSessionQueue
  - 新しいロケーションサポートでゲームセッション作成オペレーションを更新: CreateGameSession、DescribeGameSessions、DescribeGameSessionDetails、SearchGameSessions
- Amazon GameLift デベロッパーガイド :
  - Amazon GameLift ホスティングロケーション (更新済み)
  - Amazon GameLift フリート設計ガイド (新規)

    Amazon GameLift ホスティングキャパシティのスケーリング (更新済み)
  - ゲームセッションキューの設計 (新規)
  - フリートの詳細の表示 (更新済み)
- Amazon GameLift フォーラム

2021 年 2 月 9 日: Amazon GameLift が AMD インスタンスのサポートをスタンドアロンで延長 FlexMatch

SDK バージョンの更新： AWS SDK 1.8.139

このリリースでは、以下が更新されています。

- Amazon GameLift FleetIQ ゲームサーバーグループは、AMD インスタンスファミリー C5a, M5aを管理するように設定できるようになりました。 R5a にリストされているように、サポートされている Amazon EC2 インスタンスタイプには GameServerGroup InstanceDefinition、以下が含まれるようになりました。
  - c5a.large、c5a.xlarge、c5a.2xlarge、c5a.4xlarge、c5a.8xlarge、c5a.12xlarge、c5a.16xlarge、c5a.24x
  - m5a.large、m5a.xlarge、m5a.2xlarge、m5a.4xlarge、m5a.8xlarge、m5a.12xlarge、m5a.16xlarge、m5
  - r5a.large、r5a.xlarge、r5a.2xlarge、r5a.4xlarge、r5a.8xlarge、r5a.12xlarge、r5a.16xlarge、r5a.24xlar

  注: FleetIQ の AMD インスタンスは、現在、中国 (北京) AWS リージョンでは使用できません。中国における「機能の可用性と実装の違い」を参照してください。
- Amazon GameLift マネージドゲームホスティングは、Sinnet が運営する中国 (北京) リージョンで AMD インスタンスをサポートするようになりました。新しい AMD インスタンスファミリーには、M5a と R5a が含まれます。フリート にリストされているように、サポートされている EC2 インスタンスタイプにはInstanceType以下が含まれるようになりました。
  - m5a.large、m5a.xlarge、m5a.2xlarge、m5a.4xlarge、m5a.8xlarge、m5a.12xlarge、m5a.16xlarge、m5
  - r5a.large、r5a.xlarge、r5a.2xlarge、r5a.4xlarge、r5a.8xlarge、r5a.12xlarge、r5a.16xlarge、r5a.24xlar
- Amazon GameLift FlexMatch は、Sinnet が運営する中国 (北京) リージョンでスタンドアロンのマッチメーキングソリューションとして使用できるようになりました。お客様は北京リージョンで FlexMatch マッチメーカーを作成し、 FlexMatchModeパラメータを STANDALONE に設定できます。Amazon GameLift マネージドホスティングまたは Amazon 以外のホスティングソリューション FlexMatchを使用する の詳細については、 GameLift「Amazon GameLift FlexMatch デベロッパーガイド」の「」を参照してください。
- Amazon のイベント通知を設定するときに GameLift FlexMatch、Amazon SNS FIFO トピックを通知ターゲットとして指定できるようになりました。詳細については、以下を参照してください。
  - MatchmakingConfiguration NotificationTarget、Amazon GameLift API リファレンス
  - FlexMatch イベント通知のセットアップ、Amazon GameLift FlexMatch デベロッパーガイド
  - Amazon SNS FIFO の紹介 – First-in-first-out Pub/Sub メッセージング、 AWS ニュースブログ

2020 年 12 月 22 日: Amazon GameLift サーバー SDK が Unreal Engine 4.25 と Unity 2020 をサポート

更新された SDK バージョン： Amazon GameLift Server SDK 4.0.2、Unreal プラグインバージョン 3.3.3

Amazon GameLift Server SDK の最新バージョンには、次のコンポーネントが含まれています。

- Unreal Engine 4.25 との互換性のために、更新された Unreal プラグインが更新されました。API は変更されませんでした。
- 既存の C# SDK バージョン 4.0.2 は、Unity 2020 との互換性が確認されています。SDK の更新は必要ありませんでした。

Amazon GameLift Server SDK の最新バージョンを Amazon 入 GameLift 門ガイド にダウンロードします。

2020 年 11 月 24 日: Amazon がどこでもホストされるゲームで利用可能 GameLift FlexMatch に

更新された SDK バージョン： AWS SDK 1.8.95

Amazon GameLift FlexMatch は、マルチプレイヤーゲーム用のカスタマイズ可能なマッチメーキングサービスです。Amazon GameLift マネージドホスティングのユーザー向けに当初設計されたは、、独自のオンプレミスコンピューティング、クラウドコンピューティングプリミティブタイプなど peer-to-peer、他のホスティングシステムを使用するゲームに統合 FlexMatch できるようになりました。Amazon EC2 でのゲームホスティングに Amazon GameLift FleetIQ を使用するゲームは、 とのマッチメーキングを実装できるようになりました FlexMatch。

FlexMatch は、マッチメーキングプロセスをカスタマイズするための幅広い緯度を提供する堅牢なマッチメーキングアルゴリズムとルール言語を提供し、主要なプレイヤーの特性と報告されたレイテンシーに基づいてプレイヤーが一緒にマッチングされるようにします。さらに、 は、プレイヤーパーティー、プレイヤーの承諾、マッチバックフィルなどの機能をサポートするマッチメーキングリクエストワークフロー FlexMatch を提供します。Amazon GameLift マネージドホスティングまたはリアルタイムサーバー FlexMatch で を使用すると、マッチメーカーは自動的に Amazon を使用してホスティングリソース GameLift を検索し、新しく形成された試合の新しいゲームセッションを開始します。をスタンドアロンサービス FlexMatch として使用する場合、マッチメーカーは試合結果をゲームに配信し、ホスティングソリューションを使用して新しいゲームセッションを開始できます。

の API オペレーション FlexMatch は、 AWS SDK と AWS Command Line Interface () に含まれている Amazon GameLift サービス API の一部ですAWS CLI。このリリースには、スタンドアロンのマッチメイキングをサポートする次の更新が含まれています。

- API リソース `MatchmakingConfiguration` には次の変更があります。
  - 新しいプロパティは、マッチメーカーが Amazon GameLift マネージドホスティングで使用されているか、スタンドアロンのマッチメーキングとして使用されているか`FlexMatchMode`を示します。
  - `FlexMatchMode` がスタンドアロンに設定されている場合、このプロパティ `GameSessionQueueArns` は必須ではありません。
  - これらのプロパティは、スタンドアロンのマッチメイキングでは使用されません。`AdditionalPlayerCount`、`BackfillMode`、`GameProperties`、`GameSessionData`。
- 自動バックフィル機能は、スタンドアロンのマッチメイキングでは使用できません。

2020 年 11 月 24 日: AMD インスタンスが Amazon で利用可能に GameLift

更新された SDK バージョン： AWS SDK 1.8.95

Amazon でサポートされている Amazon EC2 インスタンスタイプのリストに、C5a, M5aの 3 つの新しいインスタンスファミリーが含まれる GameLift ようになりました。 R5a これらのファミリは、最大 3.3 の周波数で動作する AMD EPYC プロセッサを搭載した AMD コンピューティング最適化インスタンスで構成されています。。AMD インスタンスは x86 互換です。Amazon で現在実行されているゲームは、変更を加えることなく AMD インスタンスタイプにデプロイ GameLift できます。新しいインスタンスは、米国東部 (バージニア北部およびオハイオ) 、米国西部 (オレゴンおよび北カリフォルニア) 、中部カナダ (モントリオール) 、南米 (サンパウロ) 、欧州中部 (フランクフルト) 、欧州西部 (ロンドンおよびアイルランド) 、アジアパシフィック南部 (ムンバイ) 、アジアパシフィック北東部 (ソウルおよび東京) 、およびアジアパシフィック南東部 (シンガポールおよびシドニー) の各 AWS リージョンで利用できます。

新しい AMD インスタンスには次のものが含まれます。

- c5a.large、c5a.xlarge、c5a.2xlarge、c5a.4xlarge、c5a.8xlarge、c5a.12xlarge、c5a.16xlarge、c5a.24xla
- m5a.large、m5a.xlarge、m5a.2xlarge、m5a.4xlarge、m5a.8xlarge、m5a.12xlarge、m5a.16xlarge、m5a
- r5a.large、r5a.xlarge、r5a.2xlarge、r5a.4xlarge、r5a.8xlarge、r5a.12xlarge、r5a.16xlarge、r5a.24xlarge

詳細はこちら:

- Amazon ゲームテックブログ
- Amazon GameLift インスタンスの料金
- AMD EPYC プロセッサを搭載した Amazon EC2 インスタンス
- Amazon GameLift フォーラム

2020 年 11 月 11 日: Amazon GameLift サーバー SDK のバージョン更新

更新された SDK バージョン： Amazon GameLift Server SDK 4.0.2

新しいサーバー SDK バージョン 4.0.2 では、API オペレーション `StartMatchBackfill()` に関する既知の問題が修正されています。このオペレーションは、マッチバックフィルリクエストに対する正しい応答を返すようになりました。

この問題はマッチバックフィルプロセスに影響せず、この機能の動作に変更はありません。この問題は、マッチバックフィルリクエストのログメッセージングとエラー処理に影響している可能性があります。

Amazon GameLift Server SDK の最新バージョンを Amazon 入 GameLift 門ガイド にダウンロードします。

2020 年 11 月 5 日: 新しい FlexMatch アルゴリズムのカスタマイズ

FlexMatch ユーザーは、マッチメーキングプロセスの次のデフォルトの動作を調整できるようになりました。これらのカスタマイズは、マッチメーキングルールセットで設定されます。Amazon GameLift SDKsに変更はありません。

- バックフィルチケットの優先順位: 受け入れ可能なマッチを検索するときに、マッチバックフィルチケットの優先順位を上げるか下げるかを選択できます。バックフィルチケットの優先順位付けは、自動バックフィル機能が有効になっている場合に便利です。アルゴリズムプロパティ `backfillPriority` を使用する。
- マッチの一貫性と効率性を最適化するための事前ソート: 評価のためにチケットをバッチ処理する前に、チケットプールを事前にソートするようにマッチメーカーを設定します。主要なプレイヤー属性に基づいてチケットを事前にソートすると、結果の試合ではそれらの属性がより類似しているプレイヤーを持つ傾向があります。また、マッチルールで使用されているのと同じ属性で事前ソートすることで、評価プロセスの効率を高めることもできます。`strategy` プロパティを「sorted」に設定したアルゴリズムプロパティ `sortByAttributes` を使用します。
- 拡張待ち時間のトリガー方法を調整する：不完全なマッチで最新（デフォルト）または最も古いチケットの経過時間に基づいて拡張をトリガーするかを選択します。最も古いチケットでトリガーす

ると、マッチをより早く完了させる傾向がありますが、最新のチケットでトリガーするとマッチ品質が高くなります。アルゴリズムプロパティ `expansionAgeSelection` を使用する。

2020 年 9 月 17 日: Amazon がサーバー SDK GameLift を更新

更新された SDK バージョン： Amazon GameLift Server SDK 4.0.1

新しいサーバー SDK には、次の更新が含まれます。

- C# API バージョン: 4.0.1
  - API オペレーション TerminateGameSession() はサポートされなくなりました。ProcessEnding() への呼び出しを置き換え、ゲームセッションとサーバープロセスの両方を終了します。
  - オペレーション GetInstanceCertificate() に関する既知の問題は修正されました。
  - これで、 オペレーションはデータ型 の値GetTerminationTime()を返します AwsDateTimeOutcome。
- C++ API バージョン 3.4.1
  - オペレーション TerminateGameSession() はサポートされなくなりました。これを ProcessEnding() への呼び出しを置き換え、ゲームセッションとサーバープロセスの両方を終了します。
- Unreal Engine プラグインバージョン 3.3.2
  - オペレーション TerminateGameSession() はサポートされなくなりました。これを ProcessEnding() への呼び出しを置き換え、ゲームセッションとサーバープロセスの両方を終了します。
  - コールバックオペレーション `OnUpdateGameSession` がマッチバックフィルをサポートする FProcessParameters に追加されました。

Amazon GameLift Server SDK の最新バージョンを Amazon 入 GameLift 門ガイド にダウンロードします。

2020 年 8 月 27 日: Amazon EC2 によるゲームホスティング用の Amazon GameLift FleetIQ (一般提供)

更新された SDK バージョン： AWS SDK 1.8.36

Amazon EC2 での低コストのクラウドベースのゲームホスティング用の Amazon GameLift FleetIQ ソリューションが一般公開されました。Amazon GameLift FleetIQ を使用すると、デベロッパーはゲームホスティングの実行可能性を最適化することでAmazon EC2 スポットインスタンスでゲームサーバーを直接ホストできます。ゲームデベロッパーは、Amazon GameLift FleetIQ を新しいゲームで使用したり、既存のゲームの容量を補足したりできます。このソリューションは、コンテナ、または AWS Shield や Amazon Elastic Container Service (Amazon ECS) などの他の AWS サービスの使用をサポートします。

この一般提供リリースには、Amazon GameLift FleetIQ ソリューションに対する以下の更新が含まれています。

- 新しい API オペレーションは、Amazon GameLift FleetIQ ゲームサーバーグループのすべてのアクティブなインスタンスのステータスを含む情報`DescribeGameServerInstances`を返します。
- 新しいバランシング戦略の `ON_DEMAND_ONLY` では、オンデマンドインスタンスのみを使用するようにゲームサーバーグループを設定します。ゲームサーバーグループのバランス戦略はいつでも更新でき、必要に応じてスポットインスタンスとオンデマンドインスタンスの使用を切り替えることができます。
- 次のプレビュー要素は、一般公開のために削除されました。
  - ゲームサーバーリソースのカスタムソートキーの使用。ゲームサーバーは、登録タイムスタンプに基づいてソートできます。
  - ゲームサーバーリソースのタグ付け。

2020 年 4 月 16 日: Amazon は Unity および Unreal Engine 用のサーバー SDK GameLift を更新します

更新された SDK バージョン： Amazon GameLift Server SDK 4.0.0、Amazon GameLift Local 1.0.5

Amazon GameLift Server SDK の最新バージョンには、以下の更新されたコンポーネントが含まれています。

- Unity 2019 用に更新された C# SDK バージョン 4.0.0
- Unreal Engine 4.22、4.23、4.24 用に Unreal プラグインバージョン 3.3.1 を更新
- Amazon GameLift Local バージョン 1.0.5 が更新され、C# サーバー SDK バージョン 4.0.0 を使用する統合がテストされました。

Amazon GameLift Server SDK の最新バージョンを Amazon 入 GameLift 門ガイド にダウンロードします。

2020 年 4 月 2 日: Amazon GameLift FleetIQ が EC2 でのゲームホスティングに利用可能 (パブリックプレビュー)

更新された SDK バージョン： AWS SDK 1.7.310

Amazon GameLift FleetIQ 機能は、ゲームホスティングで使用する低コストのスポットインスタンスの実行可能性を最適化します。この機能は、マネージド Amazon GameLift サービスではなく、ホスティングリソースを直接管理したいお客様向けに拡張されました。このソリューションは、コンテナ、または AWS Shield や Amazon Elastic Container Service (Amazon ECS) などの他の AWS サービスの使用をサポートします。

詳細はこちら:

Amazon GameLift FleetIQ に関するGameTech ブログ記事

2019 年 12 月 19 日: Amazon AWS リソースの GameLift リソース管理を改善

SDK バージョンの更新： AWS SDK 1.7.249

Amazon GameLift AWS リソースでリソース管理ツールを利用できるようになりました。特に、ビルド、スクリプト、フリート、ゲームセッションキュー、マッチメーキング設定、マッチメーキングルールセットなど、すべての主要な Amazon GameLift リソースに Amazon リソースネーム (ARN) 値が割り当てられます。リソース ARN は、すべての AWS リージョンで一意な一貫した識別子を提供します。リソース固有の AWS Identity and Access Management (IAM) アクセス許可ポリシーを作成するために使用できます。リソースに ARN と、リージョン固有ではない既存のリソース識別子が割り当てられるようになりました。

さらに、Amazon GameLift リソースでタグ付けがサポートされるようになりました。タグを使用してリソースを整理したり、IAM アクセス許可ポリシーを作成してリソースグループへのアクセスを管理したり、 AWS コスト内訳をカスタマイズしたりできます。Amazon GameLift リソースのタグを管理するときは、Amazon GameLift API アクション `TagResource()`、`UntagResource()`、および を使用します`ListTagsForResource()`。

詳細はこちら:

- TagResource 「Amazon GameLift API リファレンス」の「」
- AWS 全般のリファレンスの「AWS リソースのタグ付け 」

- 「AWS 全般のリファレンス」の「Amazon リソースネーム」

2019 年 11 月 14 日: 新しい AWS CloudFormation テンプレート、中国 (北京) リージョンでの更新

更新された SDK バージョン： AWS SDK 1.7.210

AWS CloudFormation Amazon 用の テンプレート GameLift

Amazon GameLift リソースは、 を通じて作成および管理できるようになりました AWS CloudFormation。既存の AWS CloudFormation ビルドテンプレートとフリートテンプレートが現在のリソースに合わせて更新され、新しいテンプレートがスクリプト、キュー、マッチメーキング設定、マッチメーキングルールセットで使用できるようになりました。 AWS CloudFormation テンプレートは、特に複数のリージョンにゲームをデプロイする場合、関連 AWS リソースのグループを管理するタスクを大幅に簡素化します。

詳細はこちら:

- AWS CloudFormation ユーザーガイドの Amazon GameLift リソースタイプのリファレンス
- AWS CloudFormation を使用したリソースの管理 「Amazon GameLift デベロッパーガイド」の「

AWS 用語集

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

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。