



ホスティングガイド

Amazon GameLift Servers



Amazon GameLift Servers: ホスティングガイド

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは Amazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していない他のすべての商標は、それぞれの所有者の所有物であり、Amazon と提携、接続、または後援されている場合とされていない場合があります。

Table of Contents

Amazon GameLift Servers とは	1
主な特徴	1
Amazon GameLift Servers の使用方法	4
ゲームホスティングオプション	5
マネージド EC2	5
マネージドコンテナ	6
Anywhere ホスティング	7
ハイブリッドホスティング	7
仕組み	8
コアコンポーネント	9
コンポーネント間の連携の仕組み	15
プレイヤーエクスペリエンス	17
世界中のプレイヤーに高速ゲームプレイを提供	17
プレイヤーがゲームにすばやく参加できるように支援する	18
ゲームセッションのプレイヤーをグループ化するクリエイティブな方法を構築する	18
ゲームプレイ全体で信頼性の高い接続を提供する	19
実際のデータに基づいてプレイヤーエクスペリエンスを向上させる	19
拡張ゲーム機能を他の AWS サービスに統合する	20
本番環境でのゲームホスティング	20
ホスティングのヘルスとパフォーマンスのモニタリング	21
ゲームサーバーの更新とパッチの管理	21
パフォーマンスとスケーリングの最適化	22
トラブルシューティングとライブオペレーション	22
ツールと統合	23
関連リソース	24
重要な用語	24
使用開始方法	32
開始する前に	32
進めたい作業内容を選択します	32
Amazon GameLift Servers をすぐに調べたい	32
カスタムホスティングソリューションを構築したい	33
推奨されるラーニングパス	33
AWS アカウントを設定する	34
にサインアップする AWS アカウント	35

管理アクセスを持つユーザーを作成する	36
Amazon GameLift Servers にユーザーの権限を設定する	37
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	38
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	40
IAM アクセス許可の例	41
IAM サービスロールをセットアップする	46
開発用ツールを取得する	50
ゲームサーバーの場合	50
ゲームクライアントの場合	54
ゲームバックエンドサービスの場合	54
リソース管理の場合	55
Amazon GameLift Servers/Realtime の場合	55
チュートリアル: Amazon GameLift Servers ラッパーを使用したクイックオンボーディング	56
前提条件	57
概要:	57
ステップ 1: ゲームサーバーラッパーを取得して構築する	58
ステップ 2: ゲームサーバービルドを準備する	58
ステップ 3: フリートのラッパーを設定する	59
ステップ 4: ゲームサーバービルドをアップロードする	61
ステップ 5: マネージド EC2 フリートを作成する	61
ステップ 6: ゲームセッションを作成して接続する	63
ステップ 7: フリートを管理およびモニタリングする	64
ステップ 8: ゲームサーバーをスケールする	64
一般的な問題のトラブルシューティング	66
次の手順	67
Amazon GameLift Servers の例	67
カスタムゲームサーバーの例	67
ゲームホスティングソリューションの構築	69
開発ロードマップ	70
マネージド EC2 開発ロードマップ	70
マネージドコンテナ開発ロードマップ	76
Anywhere 開発ロードマップ	83
ハイブリッドホスティングの開発ロードマップ	90
ホスト用にゲームを準備します。	98
ゲームエンジンプラグインで準備する	98
ゲームサーバーの統合	187

ゲームサーバービルドをパッケージ化	215
Amazon GameLift Servers のやり取り	229
バックエンドサービスを構築する	234
プレイヤーの認証	235
ゲームクライアント機能の統合	235
サーバーレスバックエンド	241
WebSocket ベースのバックエンド	243
ホスティングフリートをデプロイする	245
フリートの特性	245
ホスティングオプションを選択する	247
マネージド EC2 フリート	248
マネージドコンテナフリート	261
Anywhere フリート	282
ハイブリッドホスティングソリューションの構築	293
ゲームセッションの配置を設定する	296
キューの特性	297
ベストプラクティス	298
キューを作成する	299
イベント通知の設定	303
ゲームホスティングソリューションをカスタマイズする	307
ゲームサーバービルド	308
プレイヤーセッションとマッチメイキング	321
ゲームセッションの配置	323
ホスティングリソース	339
ゲームホスティングソリューションの管理	364
ホスティングリソースの管理	364
Amazon GameLift Serversコンソールの機能	364
コンソールダッシュボード	365
ビルド	366
フリート	370
Aliases	392
ゲームセッションキュー	395
ゲームのローンチの準備	398
使用するための準備	398
テストの準備	399
ローンチの準備	399

ローンチ後の計画	400
でゲームホスティングリソースを管理する CloudFormation	400
ベストプラクティス	401
CloudFormation スタックの使用	402
ビルドの更新	406
DDoS 保護	409
主な利点	409
プレイヤーゲートウェイの仕組み	409
トラフィックフロー	410
重要な概念	410
プレイヤーゲートウェイのモニタリング	412
IPv4 と IPv6 の互換性	412
フリートでプレイヤーゲートウェイを有効にする	413
前提条件	413
プレイヤーゲートウェイモード	413
IPv4 と IPv6 の互換性	414
プレイヤーゲートウェイをゲームに統合する	414
バックエンド統合	415
クライアント統合	417
統合をテストする	418
ベストプラクティス	419
テストとトラブルシューティング	420
反復型開発のための環境の設定	420
クラウドベースのテスト環境を構築する	422
ローカルテストの設定	426
ローカルテストの設定 (レガシー)	431
フリートの問題をデバッグする	439
フリート作成の問題	440
サーバープロセスの問題	442
フリート削除の問題	442
フリートインスタンスにリモート接続する	443
コンソールからのリモートアクセス	443
CLI AWS によるリモートアクセス	444
インスタンスに接続する (サーバー SDK 5)	447
インスタンスに接続する (サーバー SDK 4.x 以前)	448
リモートインスタンス上のファイルを表示する	451

スケーリング	453
コンソールでフリートの容量を管理するには	453
ホスティング容量制限の設定	454
容量制限を設定するには	454
フリートの容量を手動で設定する	456
自動スケーリングを停止する	456
フリートの容量を手動で設定するには	456
フリートキャパシティを自動スケーリングする	458
ターゲットベースの自動スケーリング	459
ルールベースの自動スケーリング	460
ゼロとの間でフリートのスケーリングを管理する	465
スケールイン動作	466
スケールアウト動作	466
ゼロとの間でスケールを設定する	466
ベストプラクティス	468
コンテナフリートのスケーリング	468
モニタリング	471
メトリクスの比較	472
ソース別のメトリクスの可用性	472
トピック	475
CloudWatch を使用して監視する	476
メトリクスのディメンション	477
フリートメトリクス	478
DDoS 保護 (プレイヤーゲートウェイ) メトリクス	495
キューメトリクス	497
FlexMatch メトリクス	501
FleetIQ のメトリクス	505
サーバーテレメトリメトリクスによるモニタリング	507
テレメトリメトリクスの利点	508
[開始する前に]	509
実装	510
使用可能なメトリクス	511
仕組み	516
ダッシュボードの整理と使用状況	517
一般的なモニタリングシナリオ	528
トラブルシューティングガイド	533

API コールのログ作成	535
CloudTrail での Amazon GameLift Servers 情報	536
Amazon GameLift Servers ログファイルエントリの理解	537
サーバーアクセスのログ記録	539
EC2 とコンテナのログ記録	540
カスタムサーバーのログ記録	542
セキュリティ	546
データ保護	547
保管中の暗号化	549
転送中の暗号化	549
ネットワーク間のトラフィックのプライバシー	550
アイデンティティ/アクセス管理	550
オーディエンス	551
アイデンティティを使用した認証	551
ポリシーを使用したアクセスの管理	552
Amazon GameLift Servers と IAM の連携方法	554
アイデンティティベースのポリシーの例	561
トラブルシューティング	566
AWS マネージドポリシー	568
Amazon GameLift Servers でのログ記録とモニタリング	570
コンプライアンス検証	571
耐障害性	571
インフラストラクチャセキュリティ	572
設定と脆弱性の分析	573
セキュリティに関するベストプラクティス	574
フリーランタイム環境を最新の状態に保つ	574
ポート設定を保護します。	576
追加のセキュリティリソース	577
料金	578
料金モデル	578
料金見積りを生成する	579
マネージドホスティングの見積もり	579
コスト最適化戦略	582
ベストプラクティス	583
リソースコストと使用率ツール	585
パフォーマンスのモニタリング	585

リファレンスガイド	587
サービス API (AWS SDK)	587
Amazon GameLift Servers ホスティングリソースの管理	588
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	592
サーバー SDK 5.x	593
サーバー SDK 5.x の更新	594
サーバー SDK 5.x への移行	596
C++ サーバー SDK	598
C# サーバー SDK	650
Go サーバー SDK	696
C++ (Unreal) サーバー SDK: アクション	726
サーバー SDK 4 以前	773
サーバー SDK for C++: アクション	774
サーバー SDK for C#: アクション	798
サーバー SDK for Unreal: アクション	819
ゲームセッション配置イベント	835
プレイメントイベント構文	835
PlacementFulfilled	836
PlacementCancelled	838
PlacementTimedOut	839
PlacementFailed	840
AMI バージョン	841
サービスロケーション	843
サポートされている AWS ロケーション	843
マネージドホスティングのロケーション	849
Amazon GameLift Servers Anywhere の場所	850
Amazon GameLift Servers FlexMatch の場所	851
Amazon GameLift Servers (中国)	851
サービスエンドポイントとクォータ	851
Amazon GameLift Servers API の制限	851
API 制限リファレンス	852
リソースレベルのスロットリング	861
Amazon GameLift Servers SDK API の制限	861
UDP ping ビーコン	865
UDP ping ビーコンの仕組み	865
UDP ping ビーコンの一般的なユースケース	865

ビーコンエンドポイントの取得	866
レイテンシー測定の実装	867
コードの例	868
リリースノートと SDK バージョン	876
SDK のバージョン	876
リリースノート	888
.....	cmxxxix

Amazon GameLift Servers とは

Amazon GameLift Servers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上に構築された Amazon GameLift Serversは、世界中のプレイヤーの需要に合わせてリソース使用量を動的にスケーリングしながら、高性能で信頼性の高いゲームサーバーを提供するのに役立ちます。

主な特徴

グローバルリーチと低レイテンシー

- 世界中の AWS リージョンとローカルゾーンにホスティングをデプロイして、ゲームを新しいプレイヤーに導入します。[Amazon GameLift Serversでサポートされているロケーション](#)の完全なリストを参照してください。
- プレイヤーがいる場所に関係なく、可能な限り低いレイテンシーと優れたプレイヤーエクスペリエンスを提供します。
- マルチロケーションプレイスメントキューを使用して、最適なプレイヤーエクスペリエンスでプレイヤーをゲームセッションに動的に配置します。

マネージドフリートの自動スケーリング

- サービスの品質を維持しながら、フリート容量を最適化してコスト効率を向上させます。
- ターゲットベースの自動スケーリングを使用してプレイヤーの需要に基づいて容量をスケールする
- 容量バッファを維持し、突然のプレイヤーの流入を処理します。

柔軟なホスティングオプション

詳細については、「[Amazon GameLift Servers ゲームホスティングオプション](#)」を参照してください。

- クラウドでのゲームホスティング、Amazon GameLift Servers によって管理
 - Amazon Elastic Compute Cloud (Amazon EC2) インフラストラクチャでゲームサーバーをホストします。コンピューティング最適化 (C ファミリー)、メモリ最適化 (R ファミリー)、コスト効率の高い ARM ベースの Graviton インスタンスなどのリソースタイプから選択します。

- Amazon GameLift Servers がゲームホスティングソリューションのあらゆる側面を管理する方法をカスタマイズします。これには、ゲームサーバーのデプロイ、ゲームセッションの開始、マッチングプレイヤーへの提供、ゲームへの参加などが含まれます。
- マルチロケーションフリートを使用して、グローバルゲームサーバーのデプロイを合理化します。
- [Amazon EC2 スポットインスタンスで低コストホスティングを活用します。](#)
- マネージド EC2 ホスティングを使用して Windows または Linux ゲームサーバーをデプロイします。
- マネージドコンテナホスティングを使用して、ゲームサーバーで Linux ベースのコンテナをデプロイします。
- Amazon GameLift Servers Anywhere でお客様が管理するゲームホスティング
 - Amazon GameLift Servers ゲームセッションの配置とセッション管理機能を活用します。
 - 独自のハードウェア、オンプレミスインフラストラクチャ、またはその他のクラウドプロバイダーでゲームサーバーのデプロイとスケーリングを自己管理します。
- ハイブリッドゲームホスティング
 - Amazon GameLift Servers マネージドフリートと Anywhere フリートの組み合わせを使用して既存のソリューションを操作するか、移行をサポートするゲームホスティングソリューションを作成します。

ゲームセッション管理

- Amazon GameLift Servers でゲームサーバープロセスを管理し、ゲームサーバーのステータスと可用性を追跡し、新しいゲームセッションをリクエストに応じて配置できます。
- ゲームセッション配置機能を使用すると、プレイヤーは可能な限り最高のプレイヤーエクスペリエンスを提供するゲームセッションを見つけて参加できます。Amazon GameLift Servers の意思決定を利用することも、ホスティングコスト、プレイヤーのレイテンシ、地理的な位置などの配置基準に合わせてカスタマイズすることもできます。
- ゲームセッションプレイスメントキューを作成して、新しいゲームセッションリクエストを効率的に処理します。機能を使用して、ゲームとプレイヤーの期待に合った配置品質と速度のバランスを設定します。
- オプションのプレイヤーセッション機能を活用して、ゲームセッションのプレイヤースロットを予約し、接続時にプレイヤーを検証し、スロットの可用性をモニタリングします。既存のゲームセッションを検索またはフィルタリングして、使用可能なスロットを見つけます。

FlexMatch での高度なマッチメイキング

- Amazon GameLift Servers ホスティングソリューションと完全に統合された堅牢なマッチメイキングを追加します。
- スキルレベルやキャラクターの選択などのプレイヤー属性や好みに基づいてカスタムマッチメイキングルールを構築し、必要に応じて複雑なロジックを使用します。
- 競争的または協調的なゲームプレイのためにチームベースのマッチングを設定します。チームやマッチコンポジション間でプレイヤーのバランスを取ります。
- マッチメイキングアルゴリズムを最適化して、マッチの品質と待機時間のバランスを取ります。
- 2~200 人の同時プレイヤーのマッチングをサポートします。
- マッチバックフィルをサポートして、使用可能なプレイヤースロットを埋め、ゲームセッションをいっぱいに保ちます。

包括的なモニタリング

- ゲームセッションとプレイヤーアクティビティの使用状況とパフォーマンス、マネージドフリートのホスティングパフォーマンスとヘルスに関するリアルタイムのメトリクスを取得します。
- アラートやダッシュボード用に Amazon CloudWatch と統合します。
- トラブルシューティングのためにサーバーログとイベント追跡にアクセスします。
- パフォーマンスと使用率のデータを分析して、スケーリング (マネージドフリート) またはリソース割り当て (セルフマネージドフリート) を最適化します。

カスタマイズと拡張性

- レイテンシー、コスト、場所の優先順位付けなど、ゲームセッション配置ロジックをカスタマイズします。
- マネージドホスティングの場合は、ゲームホスティングリソースを選択し、各インスタンスでゲームサーバーを起動して実行する手順を提供します。マネージドコンテナの場合、コンテナ間のリソース割り当てをファインチューニングします。
- 独自のマッチメイキングルールを定義します。
- 容量スケーリングのカスタム戦略を実装します。
- ゲームサーバーコードを追加して、データベースやコンテンツストレージツールなどの既存の AWS サービスと統合します。

- カスタム機能を使用してバックエンドサービスを作成し、プレイヤー参加リクエスト (プレイヤーパーティなど) の管理、認証の処理、カスタムロジックと永続性の追加を行います。

AWS エコシステムとの統合

- ゲーム状態の永続化には、Amazon DynamoDB、Amazon Simple Storage Service、Amazon Aurora DSQL などのサービスを使用します。
- Amazon Cognito (プレイヤー認証用) を導入します。
- Amazon Kinesis と Amazon S3 を使用してゲーム分析を処理します。
- Amazon Chime SDK で音声チャットを追加します。
- AWS Lambda と Amazon API Gateway を使用してカスタムゲーム機能を作成します。

Amazon GameLift Servers の使用方法

これらのツールは Amazon GameLift Servers で使用します。

AWS CLI

AWS Command Line Interface (AWS CLI) を使用して、 のサービス API を含む AWS SDK を呼び出します Amazon GameLift Servers。詳細については、「AWS Command Line Interface ユーザーガイド」の「[AWS CLIの使用開始](#)」を参照してください。

Amazon GameLift Servers コンソール

リソースの設定、ゲームサーバーのデプロイ管理、プレイヤーの使用状況やパフォーマンスメトリクスの追跡には、[AWS マネジメントコンソール for Amazon GameLift Servers](#) を使用します。Amazon GameLift Servers コンソールは、AWS CLIやプログラムによるリソース管理に代わる GUI です。

Amazon GameLift Servers SDK

Amazon GameLift Servers SDK には、ゲームクライアント、ゲームサーバー、ゲームサービスと Amazon GameLift Servers サービス間の通信を確立するために必要なライブラリが含まれています。詳細については、「[Amazon GameLift Servers開発用ツールを取得する](#)」を参照してください。

AWS CloudFormation

AWS CloudFormation を使用して、インフラストラクチャをコードとして扱うことで、ゲームホスティングソリューションの AWS リソースをモデル化、プロビジョニング、管理します。リ

ソースを記述するテンプレートを作成し、指定した場所にリソースを設定してデプロイするタスク CloudFormation を自動化します。

Amazon GameLift Servers ゲームホスティングオプション

Amazon GameLift Servers には、セッションベースのマルチプレイヤーゲーム用のゲームサーバーをホストするためのさまざまなオプションが用意されています。Amazon GameLift Servers が管理するクラウドベースのホスティングをセットアップするか、すでに持っているホスティングリソースを組み込むかにかかわらず、Amazon GameLift Servers を使用してプレイヤーに必要なホスティングソリューションを構築できます。ゲームホスティングソリューションの詳細については、「[Amazon GameLift Servers でのホスティングの仕組み](#)」を参照してください。

トピック

- [マネージド EC2](#)
- [マネージドコンテナ](#)
- [Anywhere ホスティング](#)
- [ハイブリッドホスティング](#)

マネージド EC2

マネージド EC2 ホスティングを使用して、本番稼働用ゲームサーバーを管理する作業を Amazon GameLift Servers にオフロードします。Amazon Elastic Compute Cloud (Amazon EC2) と 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の高いパフォーマンスと信頼性に依拠しながら、マルチプレイヤーゲームサーバーの最適化を活用します。

[Amazon GameLift Servers マネージド EC2 ホスティングソリューションの構築を開始する](#)

特性

- Amazon Linux または Windows Server オペレーティングシステム上で動作するマルチプレイヤーゲームサーバーをホストできます。
- Amazon GameLift Servers を使用して、カスタムゲームサーバーをホスティングリソース AWS クラウドにデプロイおよび管理します。ハードウェアタイプとデプロイ先を選択し、必要な場所のみ追加の詳細を設定します。
- ゲームセッションをホストするために、必要に応じてゲームサーバープロセスを自動的に維持するようにランタイムプロセスマネージャーを設定します。

- パフォーマンスの低いゲームサーバーを Amazon GameLift Servers が検出して解決できるように、カスタムヘルストラッキングを設定します。
- Amazon GameLift Servers パフォーマンスモニタリングを活用します。ハードウェアパフォーマンスやサーバープロセスのライフサイクルなどのメトリクスを追跡します。ゲームセッションログをダウンロードして保存することもできます。
- Amazon GameLift Servers を使用して、ゲームサーバープロセスと他の AWS リソース間のアクセスを管理します。
- 本番ホスティングの場合は、AWS CloudFormation テンプレートとを使用してゲームホスティングのデプロイと管理を自動化します AWS Cloud Development Kit (AWS CDK)。AWS CodePipelineなどの CI/CD (継続的インテグレーション/継続的デリバリー) ツールとサービスを活用できます。

マネージドコンテナ

Amazon GameLift Servers は、コンテナ化されたゲームサーバー用の完全なクラウドホスティングソリューションを提供するため、移植性、俊敏性、耐障害性など、コンテナの使用の主な利点を活用できます。マネージド EC2 ホスティングと同様に、マネージドコンテナホスティングは Amazon EC2 インスタンスにコンテナをデプロイして実行します。

[コンテナ化されたゲームサーバー用の Amazon GameLift Servers マネージドホスティングソリューションの開発を開始します。](#)

特性

- 軽量コンテナを使用してカスタムアーキテクチャを開発し、ゲームサーバーのビルド、依存関係、補助ソフトウェアを実行します。
- 提供された Docker ツールを使用して、Linux ベースのコンテナイメージを作成します。コンテナイメージを Amazon Elastic Container Registry (Amazon ECR) リポジトリに保存してデプロイします。
- Amazon GameLift Servers を使用して、カスタムゲームサーバーをホスティングリソース AWS クラウドにデプロイおよび管理します。ハードウェアタイプとデプロイ先を選択し、必要な場所のみ追加の詳細を設定します。
- ゲームサーバーバージョンをモデル化するためのツールを使用して、ホスティングフリートのライフサイクルを管理します。必要に応じて、ゲームサーバーの更新やその他の設定変更をデプロイします。

- コンテナ固有のパフォーマンスメトリクスを使用して、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状態をモニタリングします。
- 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 Amazon GameLift Servers。

Anywhere ホスティング

Anywhere ホスティングを使用すると、マッチメイキングや Amazon GameLift Servers ゲームセッションメトリクスなどのゲームセッション管理機能を活用して、任意の場所でゲームサーバーをホストできます。ゲームサーバーのデプロイ、ゲームサーバーのヘルスマニタリング、容量スケーリングを自己管理します。

Note

Anywhere フリートは、迅速で反復的なゲーム開発のテスト環境として特に役立ちます。Anywhere フリートを、自前のローカルワークステーション用またはクラウドベースのホスティングリソース用にセットアップできます。Anywhere でのテストの詳細については「[Amazon GameLift Servers Anywhereでローカルテストを設定する](#)」を参照してください。

ゲーム用の Amazon GameLift Servers Anywhere ホスティングソリューションの開発を開始します。

特性

- ゲームサーバーを独自のハードウェア、オンプレミスインフラストラクチャ、またはその他のクラウドプロバイダーにデプロイします。利用可能なハードウェアをプレイヤーの近く、どこでも使用します。
- Amazon GameLift Servers セッション管理機能を使用して、ゲームサーバーの可用性をモニタリングし、新しいゲームセッションを開始し、プレイヤーをゲームセッションスロットに参加させます。

ハイブリッドホスティング

本番ホスティングでは、ハイブリッドアプローチとして、提供および管理するホスティングリソース用の Anywhere フリートと、必要に応じて不足分を補う Amazon GameLift Servers マネージドフリートを選択できます。ハイブリッドソリューションは、同じプロセスを使用してゲームセッショ

ンを開始し、プレイヤーを接続してシームレスなプレイヤーエクスペリエンスを提供します。リソースタイプに関係なく、すべてのホスティングの使用状況メトリクスをモニタリングおよび収集できます。

ハイブリッドソリューションが適している一般的なシナリオは、次のとおりです。

- ホスティングソリューションを AWS クラウドに拡張する。オンプレミスハードウェアまたはその他のクラウドベースのホスティングなど、既存のホスティングソリューションの機能を、Amazon GameLift Servers マネージドホスティングを追加することで補完します。マネージドホスティングを使用すると、ホスティング容量を増やしたり、「バースト」容量を追加したりすることで、迅速にスケールアップし、必要なリソースに対してのみ支払うことができます。また、Amazon GameLift Servers サービスのグローバルなフットプリントを活用して、世界中のより多くのプレイヤーにリーチし、期待される低レイテンシーのマルチプレイヤーエクスペリエンスを提供できます。
- クラウドベースのホスティングへの移行に向けて準備する。AWS クラウドへの移行を検討または計画している場合、ハイブリッドホスティングソリューションは、必要なペースに合わせて段階的に移行を進めることができる有効な方法です。ハイブリッドアプローチを使用すると、今すぐ必要なソリューションを構築しながら、将来のニーズに備えることができます。
- Amazon GameLift Servers が提供されていない場所のプレイヤーのレイテンシーを短縮します。すでに Amazon GameLift Servers マネージドホスティングを使用している場合、特別な状況にあるプレイヤーをサポートする必要があることがあります。例えば、通常よりも遠隔地にいるプレイヤーにより確実にリーチしたり、特別なアクセスが必要なプレイヤーをサポートしたりする必要がある場合があります。カスタムホスティングロケーションを追加し、Amazon GameLift Servers Anywhere を使用して、それらのロケーションをマネージドホスティングリソースとあわせてサポートします。
-

[ゲーム用の Amazon GameLift Servers ハイブリッドホスティングソリューションの開発を開始します。](#)

Amazon GameLift Servers でのホスティングの仕組み

Amazon GameLift Servers は、セッションベースのマルチプレイヤーゲーム専用の低コストサーバー、インフラストラクチャのプロビジョニング、スケーリング、セッション管理を提供するサービスです。このサービスには、ゲーム用にカスタマイズしたり、独自のシステムと連携して使用したりできる柔軟なツールや機能が用意されています。Amazon GameLift Servers マネージドホスティング

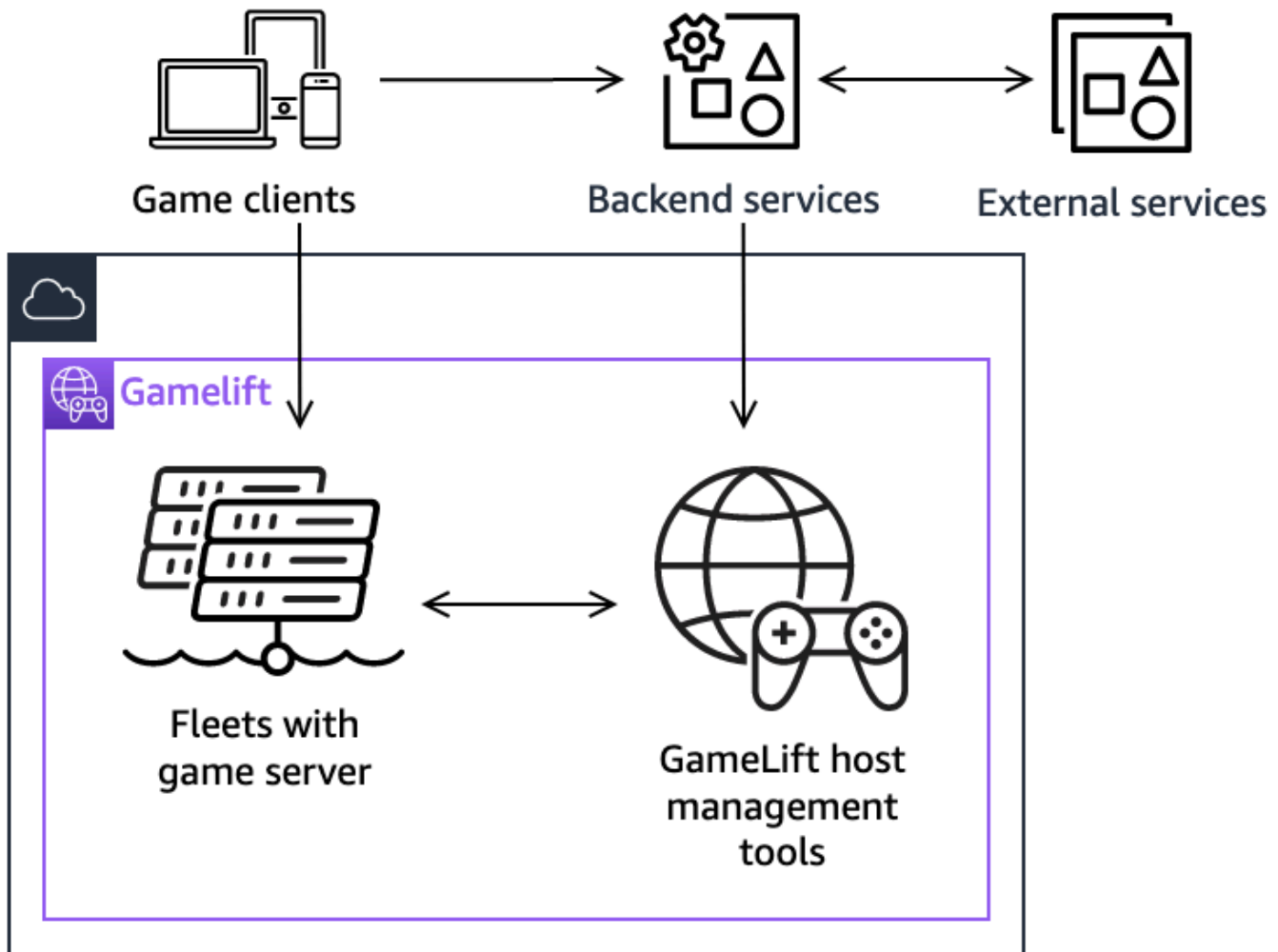
を利用すると、サーバーオペレーションを管理するのではなく、ゲームプレイエクスペリエンスの作成に集中できます。

このトピックでは、完全な Amazon GameLift Servers ホスティングソリューションのコアコンポーネントと、マルチプレイヤーゲームセッションをプレイヤーに提供するためにそれらがどのように連携するかについて説明します。

ゲームホスティングソリューションの主要コンポーネント

完全な Amazon GameLift Servers ホスティングソリューションは、マルチプレイヤーゲームエクスペリエンスを提供するために連携するいくつかの重要なコンポーネントで構成されています。これらのコンポーネントの中には、ユーザーが構築および管理するものもあれば、設定の選択に基づいて Amazon GameLift Servers によって提供および管理されるものもあります。これらのコンポーネントがどのように相互作用し、それぞれの役割を理解することは、効果的なゲームホスティングソリューションを開発するために不可欠です。コンポーネントには以下が含まれます。

- ゲームクライアントはプレイヤーデバイス上で実行されます。
- バックエンドサービスを使用すると、ゲームクライアントは Amazon GameLift Servers と通信してゲームセッション情報を取得し、ゲームに参加できます。
- ゲームサーバーソフトウェアはゲームホスティングリソースで実行され、プレイヤーグループのゲームセッションをホストします。
- プレイメントシステムはゲームセッションを開始し、プレイヤーをゲームにマッチングします。
- ゲームホスティングフリートは、1 つ以上の地理的場所にゲームサーバーを提供します。
- ゲームホスティング管理システムは、ゲームホスティングのステータスをモニタリングし、容量を管理します。



ゲームクライアント

ゲームクライアントはプレイヤーのデバイス上で動作するゲームのソフトウェアです。バックエンドサービスと通信してゲームセッションへのプレイヤー配置を開始し、ゲームサーバーに直接接続してゲームプレイに参加します。

ゲームホスティングの主な機能

- 結合リクエストをバックエンドサービスに送信します。関連するプレイヤーデータとゲームセッションデータを結合リクエストの一部として含めます。
- ゲームクライアントのレイテンシーデータを収集し、その情報を結合リクエストに含めます。
- バックエンドサービスからゲームセッション接続情報を受信し、それを使用してゲームサーバーに接続します。

- プレイヤーの検証、Amazon GameLift Servers FlexMatch のマッチ承認、セッションの中断や切断など、セッション接続シナリオを処理します。

構築者

ゲームクライアントソフトウェアにゲームホスティング機能を追加し、バックエンドサービスとの通信を設定します。

バックエンドサービス

バックエンドサービスは、ゲームクライアントと Amazon GameLift Servers サービスの間を仲介するレイヤーです。サービスとのすべての通信を制御して、ゲームセッション配置のリクエストを行い、ゲームセッションとプレイヤーセッション情報を取得します。バックエンドサービスの使用は、ゲームクライアントに代わってサービスとの安全な通信を維持し、ゲームクライアントと AWS 機密認証情報とアクセス許可を共有する必要がないようにするベストプラクティスです。

バックエンドサービスは、ゲームセッションのプレイヤーグループ化戦略を実装します。プレイヤーがゲームセッションに参加する方法に大きな柔軟性があります。プレイヤー用にゲームセッションブラウザを設定したり、プレイヤー定義のパーティをサポートしたりできます。または、受信したゲームセッションリクエストをプールし、(必要に応じて事前に分類して) プレイヤーグループを形成し、各グループごとにゲームセッションを開始することもできます。FlexMatch マッチメイキングを使用すると、プレイヤーマッチを形成し、既存のマッチをバックフィルできます。

ゲームホスティングの主な機能

- ゲームクライアントからの通信を認証します。
- Amazon GameLift Servers に新しいゲームセッションの開始、ゲームセッション情報の取得、既存のゲームセッションへの参加をリクエストします。必要に応じて、関連するゲームセッションとプレイヤーデータをリクエストに含めます。
- 必要に応じて、プレイヤーセッションのリクエストを作成して、ゲームセッションの可用性をより詳細にモニタリングします。プレイヤーセッションを使用して、ゲームセッションスロットを予約し、接続時にプレイヤーを検証し、プレイヤーの切断を追跡します。
- リクエストに応じて、ゲームセッション接続の詳細やその他の情報をゲームクライアントに返します。

構築者

ゲーム要件をサポートし、プレイヤーのゲーム参加方法を実装するために、あなたが管理する環境にバックエンドサービスを構築します。AWS SDK を統合して、Amazon GameLift Servers サービス API を呼び出します。必要に応じて、Amazon Simple Storage Service (Amazon S3) や Amazon Cognito などのサービス AWS を使用して AWS Lambda、でバックエンドサービスを構築します。

ゲームサーバー

ゲームサーバーは、ゲームの状態を管理し、プレイヤーのアクションを処理し、複数の接続されたプレイヤー間でゲームプレイを同期するカスタムサーバーソフトウェアです。ゲームサーバーは、Amazon GameLift Servers サービスとの通信を維持し、ゲームセッションホスティングを管理します。

ゲームホスティングの主な機能

- Amazon GameLift Servers と通信して、次を実行します。
 - セッションのホスト準備、プレイヤー受け入れ準備、ヘルスステータスなどの状態を報告します。
 - ゲームセッションの開始や終了などのサービスコールに応答します。
- ゲームセッションのライフサイクルを管理し、プロセスごとに 1 度に 1 つのゲームセッションをホストします。
- 他の AWS サービスと連携して機能を追加します。
- 必要に応じて、新しいプレイヤー接続を検証します。

構築者

ゲームサーバーソフトウェアを構築します。サーバー SDK for Amazon GameLift Servers を統合し、サービスとの接続を確立し、ゲームセッション管理をサポートする機能を追加します。

ゲームホスティングフリート

ホスティングフリートは、ゲームサーバーを実行するコンピューティングリソースのコレクションです。フリートリソースは複数の地理的ロケーションに分散でき、プレイヤーの所在地に関わらず低レイテンシーのゲームプレイを実現します。各フリートリソースは、Amazon GameLift Servers と直接通信する 1 つ以上のゲームサーバープロセスを実行します。各ゲームサーバープロセスは、一度に 1 つのゲームセッションをホストできます。

フリートの特性と機能は、フリートのホスティングタイプによって異なります。マネージドフリートは、リソースをデプロイ AWS クラウドし、によって管理されます Amazon GameLift

Servers。Anywhere フリートは、お客様が用意したコンピューティングリソースであり、Amazon GameLift Servers の外部で管理されます。

ゲームホスティングの主な機能

- ゲームサーバーホスティングリソースをプロビジョニングします。
- マネージドフリートは、幅広い構成設定でクラウドベースの Amazon EC2 インスタンスをデプロイします。フリート設定は、フリート内の各インスタンスのコンピューティング能力、インスタンスの物理的な場所、およびその他の詳細を決定します。マネージドフリートは、キャパシティスケールリングに応じて EC2 インスタンスを追加または削除します。
- Anywhere フリートデプロイはセルフマネージド型です。フリートは物理ハードウェアまたは他のクラウドリソースで構成され、必要に応じて設定できます。
- ランタイム環境とゲームサーバーソフトウェアをインストールします。
- マネージド Amazon EC2 Fleet インスタンスは、ゲームサーバービルドと、互換性のあるランタイム環境を備えた Amazon マシンイメージ (AMI) でデプロイされます。
- マネージドコンテナフリートインスタンスは、ゲームサーバービルドと、Docker ツールやその他の Amazon ECS と連携するためのコンポーネントを含むコンテナ最適化 AMI と共にデプロイされます。
- Anywhere フリートは、選択したゲームサーバーソフトウェアとオペレーティングシステムでデプロイされます。
- 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。事前設定されたランタイム手順を使用して、各フリートリソースでプロセスを開始および停止します。

構築者

これはフリートのホスティングタイプによって異なります。

- マネージド Amazon EC2 フリート: ゲームサーバービルドを にアップロードし Amazon GameLift Servers、フリートインスタンスへのデプロイ用に保存します。Amazon GameLift Serversはフリートの AWS クラウド インフラストラクチャを提供します。フリートを設定し、各フリートインスタンスでゲームサーバーを実行する方法を指示します。
- マネージドコンテナフリート: ゲームサーバーのビルドとランタイムの手順をコンテナイメージにパッケージ化し、デプロイのために Amazon ECR に保存します。Amazon GameLift Serversはフリートの AWS クラウド インフラストラクチャを提供します。コンテナアーキテクチャを提供し、コンテナをホストするようにフリートを設定します。

- Anywhere フリートでは、すべてのインフラストラクチャを自分でプロビジョニングし、サーバーソフトウェアのデプロイを管理します。Anywhere フリートを作成して、アクティブなゲームホスティングリソースをAmazon GameLift Servers に接続します。

ゲームセッション配置システム

ゲームセッション配置システムは、新しいゲームセッションをホストするために利用可能なゲームサーバーを特定します。システムは、ゲームサーバーの可用性に関するリアルタイム情報を使用して、最適な配置を判断します。

Amazon GameLift Servers では、ゲームセッション配置の主なメカニズムはキューです。ゲームセッションキューは、設定可能なアルゴリズムを使用して、可能な限り最良の結果となるようゲームセッションを配置します。ホスティングコストが最も低いことや、プレイヤーのレイテンシーが最も低いことなどの要因に基づいて配置に優先順位を付け、複数の地理的ロケーションをまたいで探索するようにキューを設定できます。キューの代わりに、ゲームセッションをホストする特定のフリートを指定することもできます。

ゲームホスティングの主な機能

- バックエンドサービスから受信したゲームセッション配置リクエストを処理します。
- ホスティングリソースの可用性に関するリアルタイム情報に基づいて配置を行います。
- プレイヤーレイテンシーデータやその他のデータを使用して配置オプションに優先順位を付けます (キューのみ)。
- ゲームサーバープロセスに新しいゲームセッションを開始するプロンプトを表示します。
- 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なら、ゲームセッション接続情報を更新します。
- 必要に応じて FlexMatch を構成し、プレイヤーマッチを作成して、マッチのゲームセッション配置をリクエストできます。

構築者

Amazon GameLift Servers はプレイメントシステムを提供します。バックエンドサービスがプレイメントリクエストを行う方法、およびオプションでゲームセッションキューと FlexMatch マッチメーカーを設定することで、プレイメント動作を設定します。Amazon GameLift Servers コンソール、AWS SDK、または AWS CLI を使用して、ゲームのキューとマッチメーカーを作成および設定します。

ゲームホスティング管理システム

ゲームホスティング管理システムは、ゲームホスティングソリューションのあらゆる側面を調整およびモニタリングする運用上のバックボーンです。このシステムは、Amazon GameLift Servers の配置とスケーリング機能を可能にするインテリジェンスと自動化を提供します。

ゲームホスティングの主な機能

- ゲームセッションの配置と自動容量スケーリングをサポートするために、すべてのフリートでゲームサーバープロセス、ゲームセッション、プレイヤーセッションのリアルタイムのステータスと可用性を追跡します。
- フリートのヘルスとパフォーマンスをモニタリングします。
- ゲームホスティングアクティビティメトリクスを収集して分析します。
- プレイヤーの需要とフリート使用率に基づく自動スケーリングなど、キャパシティスケーリングツールを提供します。
- ゲームサーバーソフトウェアとランタイム手順の更新を管理します。

構築者

- マネージドフリート: Amazon GameLift Servers は、可用性の追跡、ゲームホスティングアクティビティとハードウェアパフォーマンスに関するメトリクス、キャパシティスケーリングツールを提供します。利用可能な AWS ツールを使用して、ゲームサーバーソフトウェアの更新の管理、ランタイム手順の変更、メトリクスの使用のカスタマイズ (モニタリング用の CloudWatch の設定など) を行います。カスタムスケーリングポリシーを設定し、必要に応じて変更します。
- Anywhere フリート: Amazon GameLift Servers は、可用性の追跡とゲームホスティングアクティビティメトリクスを提供します。ゲームサーバーソフトウェアの更新やランタイム手順など、フリート設定の変更を管理します。フリートのパフォーマンスをモニタリングし、フリート容量のスケーリングを管理するシステムを作成します。

コンポーネント間の連携の仕組み

ゲームホスティングコンピューティングがデプロイされたとき

- ゲームサーバーソフトウェアのインストール: コンピューティングはランタイム環境とゲームサーバービルドでインストールされます。
- ゲームサーバーの起動: ゲームサーバーの実行可能ファイルのインスタンスが少なくとも 1 つ起動され、オプションの起動パラメータがコンピューティングに表示されます。

- ゲームサーバーを Amazon GameLift Servers に接続する: スタートアップアクションの一環として、ゲームサーバープロセスはサーバー SDK を呼び出してサービスへの接続を初期化します。
- 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ことを報告します。ゲームサーバープロセスはスタートアップアクションを完了すると、サーバー SDK を呼び出して準備状況を報告します。設定に基づいてヘルスステータスのレポートが開始されます。
- Amazon GameLift Servers が可用性を追跡する: サービスは、ゲームセッション配置に対するゲームサーバープロセスの可用性を記録します。また、フリート、インスタンス、ゲームセッションのメトリクスを更新して、使用状況と容量を追跡します。
- ゲームサーバーはゲームセッションの割り当てを待機: ゲームサーバープロセスは、プロンプトがゲームセッションを開始するのを待つ間、アイドル状態を維持します。

プレイヤーがゲームに参加したい場合

- プレイヤーがゲームを開始する: プレイヤーはゲームクライアントを起動し、バックエンドサービスで認証して、ゲームへの参加をリクエストします。
- バックエンドサービスはゲームセッションをリクエスト: バックエンドサービスは Amazon GameLift Servers を呼び出して、プレイヤーのグループ化戦略に基づいて適切なゲームセッションを検索または作成します。リクエストには、ゲームセッションで使用するプレイヤーまたはゲームデータが含まれる場合があります。
- Amazon GameLift Servers がゲームセッションを配置: 新しいゲームセッションを開始すると、プレイメントシステムはセッションをホストする最適な場所を特定し、利用可能なゲームサーバープロセスを選択します。サービスは、選択したプロセスに新しいゲームセッションを開始するように促し、プレイヤーまたはゲームデータを渡します。FlexMatch マッチメイキングが使用されている場合、マッチメーカーは最初にマッチを作成し、次にマッチの配置をリクエストします。
- ゲームサーバーがゲームセッションを開始: ゲームサーバーはゲームセッションを開始するステップを開始します。完了すると、プレイヤー接続を受け入れる準備ができたことを Amazon GameLift Servers に報告します。
- ゲームクライアントに配信される接続情報: ゲームサーバーがステータスを更新すると、Amazon GameLift Servers はゲームセッションの接続情報を提供します。バックエンドサービスはこの情報を受け取り、ゲームクライアントに配信します。
- プレイヤーがゲームセッションに接続: ゲームクライアントは接続情報を使用してゲームサーバープロセスに直接接続し、ゲームセッションに参加します。
- Amazon GameLift Servers は、ゲームセッションのステータスをモニタリング: ゲームサーバープロセスは、進行中のゲームセッションの可用性を追跡するために、ヘルスステータス、オプションのプレイヤー接続ステータス、ゲームセッションステータスを報告します。

- ゲームサーバープロセスがシャットダウン: ゲームサーバープロセスはゲームセッションを終了し、ステータスを報告し、それ自体をシャットダウンします。

Amazon GameLift Servers とプレイヤーエクスペリエンス

プレイヤーは、高速、公平、信頼性の高いマルチプレイヤーゲームエクスペリエンスを期待しています。Amazon GameLift Servers は、プレイヤーがマルチプレイヤーゲームと接続してエンゲージする方法を直接強化するツールをゲーム開発者に提供することで、これらのエクスペリエンスを提供します。このトピックでは、プレイヤーの主な利点である Amazon GameLift Servers の機能について説明します。

世界中のプレイヤーに高速ゲームプレイを提供

プレイヤーは、最小限の遅延で近くのサーバーに接続したいと考えています。Amazon GameLift Servers は、インテリジェントな配置とグローバルインフラストラクチャを通じて最適な接続エクスペリエンスを実現します。

- プレイヤーはグローバルに一貫したパフォーマンスを体験 - プレイヤーとゲームサーバー間の距離を最小限に抑えるために、グローバルに利用可能な AWS リージョンとローカルゾーンでゲームをホストします (「」を参照[Amazon GameLift Servers サービスロケーション](#))。これらの場所のさまざまな Amazon EC2 インスタンスタイプから選択して、ゲームワークロードをホストするための最適なコンピューティングリソースを見つけることができます。
- プレイヤーは、利用可能な最適なサーバーに接続 - Amazon GameLift Servers は、ゲームクライアントネットワークデータを収集し、複数のロケーションで利用可能なサーバーを検索することで、可能な限りレイテンシーの低いゲームセッションにプレイヤーを配置します。UDP ping ビーコンを使用して、正確なレイテンシーデータを収集します。接続品質とプレイヤーが許容できる待機時間のバランスを取るように、レイテンシポリシーを設定します。
- さまざまなリージョンのプレイヤーと一緒にプレイできる - ゲームセッション配置は、許容可能なレイテンシーを維持しながら、リージョンのプレイヤープールのプレイヤーとゲームセッションのバランスを取ることができます。入力のないリージョンのプレイヤーは、接続品質を犠牲にすることなくマッチングを見つけることができます。

プレイヤーがゲームにすばやく参加できるように支援する

プレイヤーは、需要に関係なくゲームにすばやくアクセスしたいと考えています。Amazon GameLift Servers には、ピーク時やクワイエット時にプレイヤーがゲームを見つけるのに役立つさまざまな容量スケーリングツールが用意されています。

- プレイヤーは利用可能なサーバーを簡単に見つけることができる - 自動スケーリングツールを使用して、必要なときにホスティングリソースの準備を整えることができます。ターゲット追跡は、ゲームのプレイヤー需要パターンを予測し、プロアクティブにスケーリングするために、容量の柔軟なバッファを管理します。
- プレイヤーは、特別なイベント中にスムーズなパフォーマンスを体験 - 自動スケーリングと手動スケーリング技術を組み合わせて、ゲームの更新、マーケティングキャンペーン、トーナメントなどの計画されたイベントの後の突然の流入を処理します。
- プレイヤーはサーバーが起動するのを待つ必要ない - すべてのゲームホスティングリソースで、プレイヤーが参加するとすぐに新しいセッションをホストできるようにゲームサーバーが事前ウォーミングされます。

ゲームセッションのプレイヤーをグループ化するクリエイティブな方法を構築する

プレイヤーは公平でバランスの取れた方法で他のユーザーと競争またはコラボレーションしたいと考えています。Amazon GameLift Servers には、プレイヤーをゲームセッションにグループ化するためのオプションが多数用意されています。

- プレイヤーは、プレイヤーエクスペリエンスを最適化するためにグループ化される - さまざまな機能を使用して、ゲームに最適なプレイヤーのグループ化を管理できます。スキルレベル、レイテンシー、その他の属性に基づいてプレイヤーが到着したときに、ゲームセッションまたはグループプレイヤーリクエストを検索または参照できるようにします。プレイヤーのグループ化は自分で管理することも、カスタマイズ可能な FlexMatch マッチルールを使用してマッチメイキングを実装することもできます。
- プレイヤーはアクティブな試合に参加できます - プレイヤーグループを構築して新しいゲームセッションを開始するか、プレイヤーセッションを使用して既存のセッションで使用可能なスロットを追跡し、新しいプレイヤーを追加します。FlexMatchでは、適切にマッチしたプレイヤーでセッションを自動的にバックフィルします。
- 友だちはチームとして一緒にプレイできます - プレイヤーパーティーのサポートにより、友だちのグループが一緒にプレイできます。ゲームセッションでは、残りのスロットを他のプレイヤーで埋

めたり、アクセスを制限したりできます。FlexMatchを使用すると、友だちはチームを編成できます。

- プレイヤーはゲームセッションにカスタムデータを取り込むことができます - ゲームセッションとプレイヤーセッションのリクエストにゲームとプレイヤーの情報を含めます。情報はゲームセッションで使用するためにゲームサーバーに渡されます。
- プレイヤーは完全マッチングを無期限に待機しません - FlexMatch を使用すると、プレイヤーのフラストレーションが過度にキューイングされないように、時間の経過とともにマッチング要件を緩和できます。一致品質の最も重要な側面を優先する段階的緩和ルールを作成します。
- プレイヤーはマッチングの承諾または拒否を選択できます - FlexMatch では、マッチング案をプレイヤーに承諾または拒否させるワークフローをカスタマイズできます。

ゲームプレイ全体で信頼性の高い接続を提供する

プレイヤーは、中断や切断なしで一貫したゲームプレイを期待します。Amazon GameLift Servers は包括的なセッション管理と保護を提供します。

- プレイヤーは クラウドで AWS 高品質のパフォーマンスを享受 - によって管理されるホスティングによりAmazon GameLift Servers、プレイヤーは AWS コンピューティングインフラストラクチャで動作する高速で信頼性の高いゲームサーバーを利用できます。
- 接続時にプレイヤーを検証する - ゲームのプレイヤーセッションを有効にすると、ゲームセッションにプレイヤースロット Amazon GameLift Servers を予約し、接続時にプレイヤーを検証できます。
- プレイヤーはゲームの進行状況を再接続して維持できます。Amazon GameLift Servers は、サーバー側とクライアント側の両方の中断に対する再接続をサポートします。
- プレイヤーのゲームは終了から保護 - オプションのゲームセッション保護機能により、スケーリングイベントやその他の中断中にアクティブなセッションが終了するのを防ぎます。

実際のデータに基づいてプレイヤーエクスペリエンスを向上させる

プレイヤーは、ゲームが一貫して動作し、時間の経過とともに改善されることを期待しています。Amazon GameLift Servers は、プレイヤーエクスペリエンスの最適化に役立つ包括的なモニタリングと分析を提供します。

- プレイヤーの中断が少ない - フリートのパフォーマンス、ゲームセッション、プレイヤーアクティビティをリアルタイムでモニタリングすることで、ゲームプレイに大きな影響を与える前に問題を特定できます。ゲームセッションの正常なシャットダウンと移行をカスタマイズします。

- プレイヤーはプロアクティブサーバーヘルスマモニタリングの恩恵を受ける - 継続的なサーバーヘルスマモニタリング Amazon GameLift Servers を提供し、異常なゲームサーバーを自動的に置き換えて、ハードウェアまたはソフトウェアの障害の影響を最小限に抑えます。さまざまなゲーム要件をサポートするようにヘルスチェックパラメータを設定します。
- プレイヤーはゲームプレイでデータ駆動型の改善を得られる - ゲームセッションログと詳細な分析を活用して、プレイヤーの動作とサーバーパフォーマンスのパターンを明らかにします。ゲーム固有のイベントのカスタムログ記録サポートを追加します。

拡張ゲーム機能を他の AWS サービスに統合する

プレイヤーは、音声チャット、安全な認証、永続的な進行状況などの統合機能を必要とします。は、他の AWS サービスと統合して、包括的なゲーム体験を提供Amazon GameLift Serversできます。

- プレイヤーが認証を合理化 - Amazon Cognito 統合は、ゲームセッション全体で安全なプレイヤー ID を維持しながら、さまざまな ID プロバイダーと認証方法をサポートし、プレイヤーを返すためのログインプロセスを合理化します。
- プレイヤーはセッション間でゲームの進行状況を維持できる - Amazon DynamoDB 統合を使用して、プレイヤーの進行状況、インベントリ、永続データを高性能アクセスで保存できるため、クロスセッションの継続性をサポートしながらゲームプレイへの影響を最小限に抑えることができます。
- プレイヤーは分析インサイトの恩恵を受ける - Amazon Kinesis と Amazon Simple Storage Service (Amazon S3) の統合は、ゲーム分析を処理してプレイヤーの動作と好みに関するインサイトを収集し、開発者がプレイヤーパターンの変化に適応し、ゲームを新鮮で魅力的な状態に保つのに役立つリアルタイム分析を可能にします。

Amazon GameLift Servers を使用した本番環境でのゲームホスティングの管理

Amazon GameLift Servers でゲームを起動したら、最適なパフォーマンス、信頼性、プレイヤーエクスペリエンスを確保するために、ゲームホスティングインフラストラクチャを管理する必要があります。このトピックでは、本番環境でゲームホスティングのライフサイクルを管理するための主要な機能とツールについて説明します。

ホスティングのヘルスとパフォーマンスのモニタリング

Amazon GameLift Servers には、ゲームホスティングインフラストラクチャの状態とパフォーマンスを追跡するのに役立つ包括的なモニタリングツールが用意されています。

- パフォーマンスメトリクス – インスタンス使用率、プレイヤーセッション、ゲームセッション配置など、マネージドフリートの主要なパフォーマンス指標をモニタリングします。これらのメトリクスを Amazon GameLift Servers または Amazon CloudWatch で追跡します。ここでは、メトリクスがしきい値を超えたときに通知を受け取るようにアラームを設定することもできます。
- フリートメトリクスとイベント – アクティブなサーバープロセス、利用可能なゲームセッション、アクティブなプレイヤーセッションなどのフリート固有のメトリクスを追跡して、最適な容量を確保します。
- ゲームセッション配置メトリクスとイベント – メトリクスとイベントを追跡して、ゲームセッション配置システムの状態とパフォーマンスをモニタリングします。
- ゲームセッションログ – ゲームセッションログにアクセスして分析し、問題を特定し、プレイヤーの動作パターンを理解します。
- Realtime スクリプトログ – リアルタイムサーバーを使用するゲームの場合、詳細なログを使用してスクリプトの実行とパフォーマンスをモニタリングします。
- AWS ヘルスダッシュボード – Amazon GameLift Serversデプロイに影響する可能性のある AWS サービスのヘルス状態について常に把握します。

これらのモニタリングツールには、Amazon GameLift Serversコンソール、AWS CLI、または AWS SDK for Amazon GameLift Servers を使用してカスタムダッシュボードとモニタリングソリューションを作成できます。

ゲームサーバーの更新とパッチの管理

ゲームサーバーを最新の状態に保つことは、セキュリティ、パフォーマンス、新機能の追加に不可欠です。Amazon GameLift Servers には、更新を管理するためのいくつかのアプローチが用意されています。

- ビルド管理 – ゲームサーバービルドの複数のバージョンをアップロードおよび管理します。各ビルドはバージョニングされ、異なるフリートにデプロイできます。
- フリートの置き換え – AMI バージョンとゲームサーバービルドの更新を使用して新しいフリートを作成し、ゲームセッション配置キューとエイリアスを使用して古いフリートから新しいフリートにトラフィックを徐々に移行します。

- スクリプトの更新 - リアルタイムサーバーの場合は、新しいスクリプトバージョンをアップロードして、フリート全体を置き換えずにサーバースクリプトを更新します。
- コンテナの更新 - コンテナベースのデプロイでは、コンテナイメージと定義を更新します。ゲームサーバーの新しいバージョンを既存のフリートにデプロイします。フリートを置き換えて AMI バージョンを更新します。
- 自動デプロイ - AWS CodePipeline と AWS CodeDeploy を使用して、自動ゲームサーバー更新用の CI/CD パイプラインを作成します。

更新を計画するときは、ブルー/グリーンデプロイ戦略を使用してアクティブなプレイヤーの中断を最小限に抑え、問題が検出された場合は迅速なロールバックを可能にすることを検討してください。

パフォーマンスとスケーリングの最適化

プレイヤーベースが進化するにつれて、最適なパフォーマンスとコスト効率を維持するためにホスティング設定を調整する必要があります。

- 自動スケーリング - プレイヤー数やゲームセッション使用率などのメトリクスに基づいてフリートスケーリングポリシーを設定し、容量を自動的に調整します。
- リージョンデプロイの調整 - プレイヤーの属性とレイテンシー要件に基づいて、マルチリージョンデプロイからリージョンを追加または削除します。
- キュー管理 - ゲームセッション配置キュー設定を最適化して、プレイヤーエクスペリエンスとホスティングコストのバランスを取ります。
- インスタンスタイプの選択 - パフォーマンスメトリクスを分析して、ゲームサーバーの要件に最も費用対効果の高いインスタンスタイプを決定します。
- スポットインスタンスの使用 - 重要ではないワークロードのスポットインスタンスを活用してコストを削減し、オンデマンドインスタンスに適切なフォールバック戦略を適用します。
- FlexMatch チューニング - 実際のプレイヤーデータとフィードバックに基づいてマッチメイキングルールとアルゴリズムを調整します。

CloudWatch メトリクスとコストレポートを定期的に確認し、最適化の機会を特定し、Amazon GameLift Servers コンソールまたは API を使用して変更を実装します。

トラブルシューティングとライブオペレーション

効果的なトラブルシューティングとライブオペレーション管理は、ポジティブなプレイヤーエクスペリエンスを維持するために不可欠です。

- フリートイベント通知 - スケーリングアクティビティ、インスタンスの終了、ゲームセッション配置の失敗などのフリートイベントに対して Amazon Simple Notification Service 通知を設定します。
- ゲームセッション配置のデバッグ - 詳細な配置ログを使用して、ゲームセッション配置の問題を特定して解決します。
- サーバードプロセスのヘルスチェック - サーバードプロセスのヘルスをモニタリングし、ヘルスチェックに失敗したプロセスを自動的に置き換えます。
- リモートアクセス - フリートインスタンスに接続し、AWS Systems Manager Session Manager を使用して直接トラブルシューティングを行います。
- エイリアス管理 - エイリアスを使用して、クライアント設定を変更することなく、問題のあるフリートからプレイヤートラフィックをすばやくリダイレクトします。
- バックアップと復旧 - 重要なゲームデータと設定の定期的なバックアップを実装して、障害からの迅速な復旧を可能にします。

一般的な問題の明確なインシデント対応手順とランブックを確立し、ダウンタイムとプレイヤーへの影響を最小限に抑えます。

ツールと統合

Amazon GameLift Servers は、さまざまな AWS サービスやサードパーティーのツールと統合して、本稼働管理機能を強化します。

- AWS CloudFormation - GameLift リソースを、一貫したデプロイのためのコードとしてのインフラストラクチャとして定義および管理します。
- AWS Lambda - サーバーレス関数を作成して、日常的な管理タスクを自動化し、イベントに対応します。
- Amazon EventBridge - Amazon GameLift Servers 環境の変化に自動的に対応するイベント駆動型アーキテクチャを構築します。
- AWS SDK 統合 - AWS SDKs を使用して、特定のニーズに合わせてカスタマイズされたカスタム管理ツールとダッシュボードを構築します。
- サードパーティーのモニタリング - CloudWatch メトリクスのエクスポートを使用して、サードパーティーのモニタリングおよび分析プラットフォームと統合します。

これらの統合を活用して、チームのワークフローと要件に合った包括的な管理ソリューションを作成します。

関連リソース

- [Amazon CloudWatch で Amazon GameLift をモニタリングする](#)
- [Amazon GameLift フリートの更新](#)
- [Amazon GameLift によるゲームホスティング容量のスケーリング](#)
- [Amazon GameLift Server SDK リファレンス](#)

Amazon GameLift Servers の重要な用語

このトピックでは、Amazon GameLift Servers ドキュメントとサービスで使用する主要な技術用語の定義について説明します。

アクティブなゲームセッション

プレイヤーを現在実行し、受け入れまたはホストしているゲームセッション。フリートの使用率と容量計画をモニタリングするためのメトリクスとして使用されます。「[Amazon CloudWatch で Amazon GameLift Servers を監視する](#)」を参照してください。

エイリアス

ゲームクライアントとフリートリソースの間に抽象化レイヤーを提供するフリートのわかりやすい識別子。エイリアスを使用すると、ゲームクライアントコードを変更せずにフリート設定を更新できます。「[エイリアスを使用して Amazon GameLift Servers フリートの指定を抽象化する](#)」を参照してください。

AMI (Amazon マシンイメージ)

マネージドフリートの Amazon EC2 インスタンスを作成するために使用される事前設定された仮想マシンイメージ。ゲームサーバーの実行に必要なオペレーティングシステム、ランタイムライブラリ、Amazon GameLift Servers エージェントソフトウェアが含まれています。「[Amazon GameLift Servers マネージド EC2 フリート](#)」を参照してください。

Anywhere フリート

Amazon GameLift Servers の外部で管理される、お客様が用意したコンピューティングリソースを使用するフリートのタイプ。Anywhere フリートを使用すると、既存のインフラストラクチャまたはハイブリッドクラウドソリューションを Amazon GameLift Servers ゲームセッション管理と統合できます。「[Amazon GameLift Servers Anywhere フリート](#)」を参照してください。

認証トークン

Anywhere コンピューティングリソースで実行されているゲームサーバープロセスを認証する、Amazon GameLift Servers によって生成されたセキュリティ認証情報。サーバープロセスがAmazon GameLift Serversサービスと通信するために必要です。「[Amazon GameLift Servers Anywhere フリート](#)」を参照してください。

バックエンドサービス

ゲームセッション配置リクエストの通信を制御し、ゲームセッション情報を取得するゲームクライアントと Amazon GameLift Servers サービス間の調整レイヤー。ホスティングリソースを悪意のある攻撃から保護するための安全な仲介として機能します。「[Amazon GameLift Servers でのホスティングの仕組み](#)」を参照してください。

ゲームサーバーの構築

Amazon GameLift Servers ホスティングリソースでゲームサーバーを実行するために必要なすべてのファイルを含む、ゲームサーバーソフトウェアのパッケージバージョン。ゲームサーバーの実行可能ファイル、依存関係、アセットが含まれます。「[デプロイ用にゲームサーバービルドをパッケージ化する](#)」を参照してください。

コンピューティング

コンテナフリートでは、フリートインスタンス上のゲームサーバーコンテナグループのコピーを表します。各コンピューティングは、一度に1つのゲームセッションをホストできます。「[Amazon GameLift Servers におけるコンテナの仕組み](#)」を参照してください。

コンテナフリート

従来のビルドではなくコンテナイメージを使用してコンテナ化されたゲームサーバーを実行するマネージドフリートの一種。特定のランタイム環境と依存関係を持つゲームサーバーをデプロイするための柔軟性を提供します。「[Amazon GameLift Servers マネージドコンテナフリート](#)」を参照してください。

コンテナグループ

ゲームサーバー機能を提供するために連携するコンテナのコレクション。コンテナタスクまたはポッドと同様に、コンテナフリート内でのコンテナの動作、依存関係、リソース共有を定義します。「[Amazon GameLift Servers におけるコンテナの仕組み](#)」を参照してください。

目的地

ゲームセッションキューでは、ゲームセッション配置リクエストを受信できるフリートまたはエイリアス。キューには、優先順位とレイテンシーポリシーが異なる複数の送信先を設定できます。「[ゲームセッションキューをカスタマイズする](#)」を参照してください。

Fleet

ゲームサーバーを実行するコンピューティングリソースのコレクション。フリートリソースは複数の地理的場所に分散でき、各リソースは 1 つ以上のゲームサーバープロセスを実行します。

「[Amazon GameLift Servers のホスティングフリートをデプロイする](#)」を参照してください。

フリートのロケーション

フリートインスタンスがデプロイされる特定の AWS リージョンまたはローカルゾーン。マルチロケーションフリートは、レイテンシーを短縮し、可用性を向上させるために複数のロケーションにまたがることができます。「[Amazon GameLift Servers のホスティングフリートをデプロイする](#)」を参照してください。

FlexMatch

Amazon GameLift Servers 設定可能なルールに基づいてプレイヤーマッチングを自動的に形成し、完了したマッチングのゲームセッション配置をリクエストするマッチメイキングサービス。スキルベースのマッチング、チームバランシング、マッチバックフィルをサポートします。

「[GameLift FlexMatch とは](#)」を参照してください

ゲームクライアント

プレイヤーのデバイスで実行されるゲームソフトウェア。バックエンドサービスと通信してゲームセッションへのプレイヤー配置を開始し、ゲームプレイのためにゲームサーバーに直接接続します。「[Amazon GameLift Servers でのホスティングの仕組み](#)」を参照してください。

ゲームのプロパティ

ゲームセッションに関連付けられたカスタムデータのキーと値のペア。設定情報、ゲームルール、またはその他のメタデータをゲームサーバーやクライアントに渡すために使用されます。API リファレンスの「[GameProperty](#)」を参照してください。

ゲームサーバー

ゲーム状態を管理し、プレイヤーのアクションを処理し、複数の接続されたプレイヤー間でゲームプレイを同期するカスタムサーバーソフトウェア。Amazon GameLift Servers Server SDK と統合して、サービスと通信します。「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。

ゲームサーバーのアクティベーション

ホスティングリソースでゲームサーバープロセスを開始および初期化するプロセス。ゲームサーバーの実行可能ファイルのロード、Amazon GameLift Servers との通信の確立、ゲームセッションのホストの準備が含まれます。「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。

ゲームサーバーコンテナ

ゲームサーバーアプリケーションを実行するゲームサーバーコンテナグループ内のコンテナ。コンテナグループのライフサイクルに不可欠なものとして自動的に指定されます。「[Amazon GameLift Servers におけるコンテナの仕組み](#)」を参照してください。

ゲームサーバーコンテナグループ

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しているコンテナを管理するコンテナグループの一種。コンテナフリートがゲームセッションをホストするために必要で、フリートインスタンス間でレプリケートできます。「[Amazon GameLift Servers におけるコンテナの仕組み](#)」を参照してください。

ゲームサーバープロセス

ホスティングリソース上のゲームサーバー実行可能ファイルの、実行中のインスタンス。各プロセスは一度に 1 つのゲームセッションをホストでき、Amazon GameLift Servers と直接通信してステータスを報告し、指示を受け取ります。「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。

ゲームセッション

ゲームサーバープロセスで実行されているマルチプレイヤーゲームのインスタンス。複数のプレイヤーが参加して参加できる単一のゲームプレイエクスペリエンスを表します。API リファレンスの「[Amazon GameLift Servers でのホスティングの仕組み](#)」および「[GameSession](#)」を参照してください。

ゲームセッションデータ

ゲームサーバーとクライアントで使用されるプレイヤーデータ、ゲーム状態、セッションプロパティ、その他のランタイム情報など、ゲームセッションに関連する情報。API リファレンスの「[GameSession](#)」を参照してください。

ゲームセッションの配置

新しいゲームセッションをホストするために利用可能なゲームサーバーを見つけるプロセス。ゲームサーバーの可用性に関するリアルタイムの情報を使用して、コストやレイテンシーなどの要因に基づいて最適な配置を決定します。API リファレンスの「[Amazon GameLift Servers ゲームクライアント機能の統合する](#)」および「[StartGameSessionPlacement](#)」を参照してください。

ゲームセッション保護

スケーリングイベントまたはフリートの更新中にアクティブなゲームセッションをホストしているゲームサーバープロセスを、Amazon GameLift Servers が終了できないようにする設定。進行

中のゲームが中断されないようにします。「[Amazon GameLift Servers フリートの容量を手動で設定する](#)」を参照してください。

ゲームセッションキュー

設定可能なアルゴリズムを使用してゲームセッションを配置し、最適な結果を得るための配置メカニズム。複数の地理的ロケーションで、ホスティングコストが最も低く、プレイヤーのレイテンシーが最も低いなどの要因に基づいて配置を優先できます。「[ゲームセッションの配置を設定する](#)」を参照してください。

アイドル状態のインスタンス

実行中だが、現在アクティブなゲームセッションをホストしていないフリートインスタンス。フリートの効率とスケーリングの決定をモニタリングするためのメトリクスとして使用されます。「[Amazon CloudWatch で Amazon GameLift Servers を監視する](#)」を参照してください。

インバウンドアクセス許可

フリートインスタンスへの接続に使用できる IP アドレスとポート範囲を定義するネットワークアクセスルール。ゲームサーバーへのアクセスを制御し、安全な接続を確保します。マネージド EC2 とマネージドコンテナフリートの両方に、インバウンドアクセス許可の設定が必要です。

「[IpPermission](#)」および「[ネットワーク接続を構成する](#)」を参照してください。

レイテンシーポリシー

プレイヤー接続の許容レイテンシーしきい値を定義するゲームセッションキューのルール。許容可能なネットワークパフォーマンスでプレイヤーをゲームセッションに配置するために使用されます。「[ゲームセッションキューをカスタマイズする](#)」を参照してください。

マネージドフリート

Amazon GameLift Servers が AWS クラウドでコンピューティングリソースをデプロイして管理するフリートのタイプ。自動スケーリング、ヘルスモニタリング、インフラストラクチャ管理を提供します。「[Amazon GameLift Servers マネージド EC2 フリート](#)」を参照してください。

マッチメイキング

スキルレベル、レイテンシー、その他のゲーム固有の要因などの基準に基づいて、マルチプレイヤーゲームプレイ用にプレイヤーをグループ化するプロセス。Amazon GameLift Servers では、主に FlexMatch によって処理されます。「[GameLift FlexMatch とは](#)」を参照してください

メトリクスグループ

モニタリングと自動スケーリングの決定に使用される関連するパフォーマンスメトリクスのコレクション。アクティブなゲームセッション、使用可能なゲームサーバー、プレイヤー使用率など

のメトリクスをグループ化します。「[Amazon GameLift Servers でのフリート容量の自動スケーリング](#)」を参照してください。

インスタンスごとのコンテナグループ

各フリートインスタンスで追加のソフトウェアを実行するオプションのコンテナグループタイプ。バックグラウンドサービスやモニタリングツールに役立ち、インスタンスごとにデプロイされるコピーは 1 つだけです。「[Amazon GameLift Servers におけるコンテナの仕組み](#)」を参照してください。

プレイヤーセッション

ゲームセッションのプレイヤーの接続スロット。ゲームセッションスロットの予約、接続時のプレイヤーの検証、プレイヤーの接続ステータスと切断の追跡に使用されます。API リファレンスの「[Amazon GameLift Servers ゲームクライアント機能の統合する](#)」および「[PlayerSession](#)」を参照してください。

優先設定

ゲームセッション配置の送信先を選択するときに、ゲームセッションキューの優先順位をカスタマイズする設定。個々のプレイメントリクエストの優先度を変更する優先度オーバーライドリストとは対照的に、キュー全体の動作を定義します。「[ゲームセッション配置に優先順位を付ける](#)」を参照してください。

優先度オーバーライドリスト

ゲームセッション配置リクエストに含めることができるロケーションの代替優先度リスト。他のリクエストに影響を与えることなく、その 1 つのリクエストのみのロケーションに対してキューが設定した優先順位を効果的に置き換えます。「[ゲームセッション配置に優先順位を付ける](#)」を参照してください。

プロセスマネージャー

コンテナまたはインスタンス内の複数のゲームサーバープロセスのライフサイクルを管理するコンポーネント。個々のゲームサーバープロセスの開始、停止、モニタリングを処理します。「[マネージド Amazon GameLift Servers でのゲームサーバーランタイム設定の最適化](#)」を参照してください。

リアルタイムスクリプト

Amazon GameLift Servers リアルタイムサーバーのゲームロジックを定義する JavaScript コード。カスタムゲームサーバーを必要とせずに、プレイヤー接続、メッセージルーティング、ゲーム状態管理を処理します。「[Amazon GameLift リアルタイムサーバーとは](#)」を参照してください。

ルーティング戦略

エイリアスがトラフィックをフリートに送信する方法を決定する設定。定義されたルールに基づいて、特定のフリートにルーティングしたり、複数のフリートにトラフィックを分散したりできます。「[エイリアスを使用して Amazon GameLift Servers フリートの指定を抽象化する](#)」を参照してください。

ランタイム設定

同時プロセスの数、起動パラメータ、リソース割り当てなど、フリートインスタンスでゲームサーバープロセスを実行する方法を定義する設定。「[マネージド Amazon GameLift Servers でのゲームサーバーランタイム設定の最適化](#)」を参照してください。

スケーリング

プレイヤーの需要に基づいてホスティングリソースを追加または削除することで、フリートの容量を調整するプロセス。自動 (使用率メトリクスに基づく) または手動 (管理者制御) を使用できます。「[Amazon GameLift Servers でのフリート容量の自動スケーリング](#)」を参照してください。

スケーリングポリシー

フリート容量を自動的に調整するタイミングと方法を定義するルール。ターゲットベース (特定の使用率を維持) またはルールベース (メトリクスのしきい値に基づいてスケーリング) にすることができます。「[Amazon GameLift Servers でのフリート容量の自動スケーリング](#)」を参照してください。

サーバー SDK

ゲームサーバーを Amazon GameLift Servers と統合するために必要なツールとライブラリを提供する Software Development Kit。セッション管理、ヘルスレポート、ステータス更新のために、ゲームサーバーとサービス間の通信を有効にします。「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。

セッションベースのマルチプレイヤー

ゲームプレイが個別の時間制限付きセッション内で発生するマルチプレイヤーゲームアーキテクチャ。各セッションには開始と終了が定義されており、必要に応じてプレイヤーが参加および退出します。「[Amazon GameLift Servers でのホスティングの仕組み](#)」を参照してください。

スポットインスタンス

スペア AWS 容量を低コストで使用する Amazon EC2 インスタンスのタイプ。スポットインスタンスは、が容量を戻す AWS 必要がある場合に中断できるため、耐障害性のあるゲームワークロードに適しています。「[スポットインスタンスのキューの設計](#)」を参照してください。

TLS 証明書

ゲームクライアントとサーバー間の暗号化された通信を有効にするために使用されるデジタル証明書。安全なデータ転送を確保するために TLS 対応フリートに必要です。「[Amazon GameLift Servers マネージド EC2 フリートを作成する](#)」を参照してください。

Amazon GameLift Servers の開始方法

これらの入門リソースを活用して、Amazon GameLift Servers サービスの詳細とセッションベースのマルチプレイヤーゲーム用のカスタムホスティングソリューションの開発を開始する方法について説明します。

開始する前に

- で使用する を作成する AWS アカウント (または既存のものを指定する)Amazon GameLift Servers。
- Amazon GameLift Servers および関連 AWS サービスのアクセス許可を持つユーザーを設定します。
- 作業 AWS リージョン する を選択します。開発するには、最寄りのリージョンを選択します。リージョンはいつでも変更できます。

[AWS ユーザーアカウントを設定する](#)

進めたい作業内容を選択します

Amazon GameLift Servers をすぐに調べたい

最適な用途: 学習、概念実証の作成、ラピッドプロトタイピング

クイック スタート オプション:

- [Amazon GameLift Servers のゲームサーバーラッパー – チュートリアル: Amazon GameLift Servers ラッパーを使用したクイックオンボーディング](#)を参照してください。このツールは、Amazon GameLift Servers でゲームサーバーをホストしてゲームセッションを実行する最も迅速かつ簡単な方法です。ゲームコードを変更する必要はありません。ゲームサーバーラッパーは、基本的なゲームセッション管理機能と効率的なゲームサーバーのデプロイを提供します。独自のゲームプロジェクトまたはサンプルプロジェクトを使って Amazon GameLift Servers を実際に評価する際に最適です。カスタムゲームホスティングソリューションを構築する準備ができたなら、サーバー SDK for Amazon GameLift Servers と完全に統合したカスタム開発オプションの 1 つに切り替えます。ゲームでカスタムホスティングソリューションが不要な場合は、ゲームサーバーラッパーを使い続けて、ゲームサーバーを本番環境にデプロイしホストできます。

- [Amazon GameLift Servers Unreal Engine または Unity 用 プラグイン](#) – プラグインは、GUI ワークフローとサンプルアセットを提供し、最初のステップをガイドし、基本的なホスティングソリューションでゲームサーバーをデプロイします。プラグインを使用して、セルフマネージド型の Anywhere フリートでホスティングを設定するか、クラウドベースのマネージド EC2 フリートまたはコンテナフリートをデプロイします。カスタムホスティングソリューションを開発する準備ができたなら、プラグインで構築したソリューションを基盤として拡張できます。
- [Amazon GameLift Servers マネージドコンテナ用のスターターキット](#) – このキットは、ゲームサーバーの統合、ゲームサーバーコンテナイメージの準備、ホスティング用のコンテナフリートのデプロイのタスクを合理化します。統合のために、キットはゲームサーバーに必須のゲームセッション管理機能を追加します。このキットでは、事前設定されたテンプレートを使用して、ゲームサーバーのコンテナフリートと自動デプロイパイプラインを構築します。完全なゲームセッション管理機能を追加する準備ができたなら、カスタム開発ロードマップのいずれかに従ってサーバー SDK for Amazon GameLift Serversを統合します。

カスタムホスティングソリューションを構築したい

最適: カスタムゲームサーバー用の本番稼働レベルのソリューションを作成する

[Amazon GameLift Servers ゲームホスティングオプション](#) ページを確認し、ソリューションのオプションのいずれかを選択します。これらの開発ロードマップのいずれかに従って、ゲーム用のフル機能のカスタムホスティングソリューションの構築を開始します。ロードマップは、ホスティングソリューションの各コンポーネントを作成、テスト、カスタマイズする方法に関する詳細なガイダンスを提供します。

- [Amazon GameLift Servers マネージド EC2 でホスティングするための開発ロードマップ](#)
- [Amazon GameLift Servers マネージドコンテナでホスティングするための開発ロードマップ](#)
- [Amazon GameLift Servers Anywhere でホスティングするための開発ロードマップ](#)
- [Amazon GameLift Serversを利用するハイブリッドホスティングの開発ロードマップ](#)

推奨されるラーニングパス

1. 小さく開始します。ゲームサーバーラッパーまたは Unreal 用プラグインを試してください。
2. コンセプトを理解します。 [Amazon GameLift Servers でのホスティングの仕組み](#) ページを確認してください。
3. アーキテクチャを選択します。ゲームの要件に基づいてホスティングモデルを選択します。

4. ビルドとテストを行います。適切な開発ロードマップに従います。各コンポーネントの基本バージョンを構築し、反復処理とカスタマイズを行います。
5. スケーリングと最適化。本番稼働レベルの使用状況に合わせてソリューションを最適化します。マッチメイキングなどの高度な機能を追加します。

AWS ユーザーアカウントを設定する

Tip

これらのトピックを使用して、これらのタスクに関するヘルプを取得します。

- で使用する AWS アカウント 新しい を取得しますAmazon GameLift Servers。
- Amazon GameLift Servers リソースを操作するアクセス許可を持つユーザーまたはグループを作成します。
- セキュリティ認証情報を設定する (Unreal と Unity の AWS CLI ツールとAmazon GameLift Serversプラグインを使用するにはこれらが必要です)
-

すべての AWS サービスと同様に、Amazon GameLift Servers サービスとツール AWS アカウントを使用するには が必要です。AWS アカウント には 2 つの主な機能があります。(1) アカウントで作成したすべての AWS リソースのコンテナを提供します。(2) ユーザー認証の設定やユーザーアクセス許可の制御など、AWS リソースのセキュリティを管理できます。の作成に料金はかかりません AWS アカウント。

AWS アカウントAmazon GameLift Serversの有無にかかわらず探索する

以下の目的で AWS アカウントは必要ありません。

- [AWS for Games](#) でゲームエクスペリエンスを構築、実行、成長するための AWS ツールをご覧ください。 [ブログ](#)をお読みになり、 [ソリューションのゲームライブラリ](#)を参照してください。
- Amazon GameLift Servers の詳細については、「[製品概要](#)、[FAQ](#)、[リソース](#)」を参照してください。製品に関する質問に対する回答を見つけるよう AWSに尋ねます。(次のプロンプトを試してください: 「マルチプレイヤーゲームをホストするための低コストのオプションを探しています」。)
- 詳細については、ホスティングとマッチメイキングのデベロッパーガイドやサービス API リファレンスガイドなど、Amazon GameLift Servers が機能する仕組みを[技術ドキュメント](#)で確認してください。

- [Amazon GameLift Servers 料金](#)とコスト最適化の手法に関する情報をご覧ください。[料金計算ツールを試して](#)、同時プレイヤー使用量 (CCU) のピークに基づいてホスティングコストがどのように計算されるかを確認してください。
- ダウンロードを取得し、Amazon GameLift Servers SDK、プラグイン、ツールキットのコードリポジトリを確認します。「[Amazon GameLift Servers の開始方法](#)」を参照してください。(これらを使用するには AWS アカウント が必要です)。

以下を行うには、AWS アカウントが必要です。

- Unreal と Unity の Amazon GameLift Servers プラグインを使用してオンボーディングワークフローに従うか、ゲームサーバーラッパーを使用します。
- AWS マネジメントコンソールを使用して AWS リソースを作成および管理します。
- AWS Command Line Interfaceを使用して AWS リソースを作成および管理します。
- Amazon GameLift Servers 技術ドキュメントで Amazon Q を使用して、回答、ガイダンス、推奨事項を見つけます。

トピック

- [にサインアップする AWS アカウント](#)
- [管理アクセスを持つユーザーを作成する](#)
- [Amazon GameLift Servers にユーザーの権限を設定する](#)
- [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](#)
- [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](#)
- [Amazon GameLift Servers 向け IAM アクセス許可の例](#)
- [Amazon GameLift Servers 用に IAM サービスロールをセットアップする](#)

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、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

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。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、[ルートユーザーアクセスが必要なタスク](#)の実行にはルートユーザーのみを使用するようにしてください。

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 [マイアカウント] 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、日常的なタスクにルートユーザーを使用しないように AWS アカウントのルートユーザー、 を保護し AWS IAM アイデンティティセンター、 を有効にして管理ユーザーを作成します。

を保護する 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 マネジメントコンソール](#)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 \(コンソール\) の仮想 MFA デバイス](#)を有効にする」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 アイデンティティセンター ユーザーガイド」の「[AWS IAM アイデンティティセンターの有効化](#)」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、「AWS IAM アイデンティティセンター ユーザーガイド」の[「デフォルトを使用してユーザーアクセスを設定する IAM アイデンティティセンターディレクトリ」](#)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「アクセスポータルにサインインする」](#)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 アイデンティティセンター ユーザーガイド」の[「アクセス許可セットを作成する」](#)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 アイデンティティセンター ユーザーガイド」の[「グループを追加する」](#)を参照してください。

Amazon GameLift Servers にユーザーの権限を設定する

Amazon GameLift Servers 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大します。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 Servers 向け 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。

アクセスを提供するには、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 アイデンティティセンター:

アクセス許可セットを作成します。「AWS IAM アイデンティティセンター ユーザーガイド」の「[アクセス許可セットを作成する](#)」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 \(フェデレーション\) 用のロールを作成する](#) 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 ユーザーのロールの作成](#) 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します。IAM ユーザーガイド の [ユーザー \(コンソール\) へのアクセス許可の追加](#) の指示に従います。

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 外部とやり取りする場合は、プログラムによるアクセスが必要です AWS マネジメントコンソール。プログラムによるアクセスを許可する方法は、 が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ムによる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。

プログラムによるアクセス権を必要とするユーザー	目的	方法
IAM	(推奨) コンソール認証情報を一時的な認証情報として使用して AWS CLI、AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、AWS Command Line Interface「ユーザーガイド」の AWS

プログラムによるアクセス権を必要とするユーザー	目的	方法
		「ローカル開発のためのログイン」を参照してください。 • AWS SDKs 「SDK およびツールリファレンスガイド」のAWS 「ローカル開発用のログイン」を参照してください。AWS SDKs
ワークフォースアイデンティティ (IAM アイデンティティセンター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 CLI、AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、 「AWS Command Line Interface ユーザーガイド」の「使用する AWS CLI ように AWS IAM アイデンティティセンターを設定する」を参照してください。 • AWS SDKs、ツール、API については、AWS APIs 「SDK およびツールリファレンスガイド」の「IAM アイデンティティセンター認証」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 CLI、AWS SDKs、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。

プログラムによるアクセス権を必要とするユーザー	目的	方法
IAM	(非推奨) 長期認証情報を使用して、AWS CLI、AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • 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「長期認証情報を使用した認証」を参照してください。AWS SDKs • API AWS APIs 「IAM 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 Amazon GameLift Serversを使用して と通信するためにバックエンドサービスを使用します。(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たりします。これらのサービスには、Amazon GameLift Servers の サービス API 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。

ではAmazon GameLift Servers、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 GameLift Servers 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。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 AWS ([IAM Roles Anywhere](#)) の外部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、Amazon GameLift Servers との通信時にバックエンドサービスが保存して使用する必要がある長期認証情報をプレイヤーユーザーに提供します。

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 Servers 向け IAM アクセス許可の例

これらの例の構文を使用して、Amazon GameLift Servers リソースへのアクセスを必要とするユーザーの 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 GameLift Servers にユーザーの権限を設定する](#)」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。

スタンドアロンソリューション Amazon GameLift Servers FleetIQ として 使用している場合は、「[AWS アカウント の のセットアップ Amazon GameLift Servers FleetIQ](#)」を参照してください。

管理アクセス許可の例

これらの例では、ホスティング管理者または開発者に、Amazon GameLift Servers ゲームホスティングリソースを管理するためのターゲットを絞ったアクセス権を付与します。

Example Amazon GameLift Servers のリソースへのフルアクセス許可の構文

以下の例では、すべての Amazon GameLift Servers リソースへのフルアクセスを拡張します。

JSON

```
{
  "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 Servers 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 Servers 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。

JSON

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

Example Amazon ECR のコンテナイメージにアクセスするための Amazon GameLift Servers リソースの構文

次の例では、マネージドコンテナフリートを操作するときに Amazon GameLift Servers ユーザーが必要とする Amazon Elastic Container Registry (Amazon ECR) アクションへのアクセスを拡張します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "ecr:DescribeImages",
      "ecr:BatchGetImage",
      "ecr:GetDownloadUrlForLayer"
    ],
    "Resource": "*"
  }
}
```

Example Amazon GameLift Servers 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 Servers 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 Servers に渡せるようにします。「[Amazon GameLift Servers 用に IAM サービスロールをセットアップする](#)」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 Servers 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 Servers 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](#)」を参照してください。

JSON

```
{
  "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 Servers 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、マッチメイ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 Servers API へのアクセスを拡張します。

JSON

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

Example マッチメイキングアクセス許可の構文

次の例では、FlexMatch マッチメイキングアクティビティを管理する Amazon GameLift Servers APIs へのアクセスを拡張します。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 Servers でホストされているゲームのゲームセッション配置を開始します。FlexMatch の詳細については、「[Amazon GameLift Servers FlexMatch とは](#)」を参照してください。

JSON

```
{
  "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 Servers 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。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
```

```
"Sid": "PlayerPermissionsForManualGameSessions",
"Effect": "Allow",
"Action": [
    "gamelift:CreateGameSession",
    "gamelift:DescribeGameSessions",
    "gamelift:SearchGameSessions",
    "gamelift:CreatePlayerSession",
    "gamelift:CreatePlayerSessions",
    "gamelift:DescribePlayerSessions"
],
"Resource": "*"
}
```

Amazon GameLift Servers 用に IAM サービスロールをセットアップする

一部の Amazon GameLift Servers 機能では、所有している他の 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。これを行うには、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、AWS アイデンティティができることとできないことを決定するアクセス許可ポリシーを持つアイデンティティであるという点で、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られ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。その代わりに、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 Servers マネージドフリートで利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 Servers FleetIQ を使用して Amazon Elastic Compute Cloud (Amazon EC2) インスタンスのゲームホスティングを最適化する場合は、[「AWS アカウントのセットアップ Amazon GameLift Servers FleetIQ」](#) を参照してください。

以下の手順では、カスタムアクセス許可ポリシー、および Amazon GameLift Servers にロールの引き受け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。

Amazon GameLift Servers マネージド EC2 フリートの IAM サービスロールを作成する

ステップ 1: 許可ポリシーを作成する。

このページの手順と例を使用して、使用している Amazon GameLift Servers フリートのタイプに応じたカスタムアクセス許可ポリシーを作成します。

JSON ポリシーエディタでポリシーを作成するには

1. にサインイン AWS マネジメントコンソール し、<https://console.aws.amazon.com/iam/> で IAM コンソールを開きます。
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、[管理ポリシーによるこそ] 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、または一般警告をすべて解決してから、[次へ] を選択します。

 Note

いつでも [ビジュアル] 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、[ビジュアル] エディタで [次へ] に変更または選択した場合、IAM はポリシーを再構成してビジュアル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) でポリシーを作成または編集するときに AWS マネジメントコンソール、テンプレートで利用できる JSON または YAML ポリシー CloudFormation 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、次に [CloudFormation テンプレートを生成] を選択します。詳細については CloudFormation、「AWS CloudFormation ユーザーガイド」の「[AWS Identity and Access Management リソースタイプのリファレンス](#)」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わったら、[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) タグをキーと 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 でのタグの使用の詳細については、IAM ユーザーガイドの[AWS Identity and Access Management 「リソースのタグ」](#)を参照してください。
11. [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 GameLift Servers が推測できるロールを作成する。

IAM ロールを作成するには

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。

JSON

```
{
  "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。

Amazon GameLift Servers マネージドコンテナの IAM ロールを作成する

Amazon GameLift Servers マネージドコンテナを使用している場合は、コンテナフリートで使用する IAM サービスロールを作成する必要があります。このロールは、コンテナフリートリソースを管理し、ユーザーに代わってアクションを実行Amazon GameLift Serversするために必要な制限付きアクセス許可を付与します。

コンテナフリートの IAM ロールを作成するには

1. にサインイン AWS マネジメントコンソール し、<https://console.aws.amazon.com/iam/> で IAM コンソールを開きます。
2. 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。
3. 信頼されたエンティティの選択ページで、AWS サービスを選択し、ユースケース「GameLift」を選択します。[次へ] を選択します。
4. [アクセス許可の追加] ページで、マネージドポリシー GameLiftContainerFleetPolicy を選択します。[次へ] を選択します。このポリシーの詳細については、「[AWS の 管理ポリシー Amazon GameLift Servers](#)」を参照してください。
5. [名前、確認、および作成] ページで、ロール名を入力して、[ロールの作成] をクリックします。

アクセス許可ポリシーの構文

- Amazon GameLift Servers がサービスロールを引き受けるためのアクセス許可

JSON

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

- デフォルトで有効になっていない AWS リージョンへのアクセス許可

JSON

```
{
  "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 GameLift Servers 開発用ツールを取得する

Amazon GameLift Servers には、ゲーム用のゲームホスティングソリューションの構築に役立つ一連の SDKs やその他のツールが用意されています。SDK を使用すると、ゲームサーバー、ゲームクライアント、バックエンドサービスに機能が追加され、Amazon GameLift Servers サービスと連携できるようになります。Amazon GameLift Servers SDK のバージョンと互換性に関する最新情報については、[Amazon GameLift Servers リリースノート](#) を参照してください。

ゲームサーバーの場合

64 ビットゲームサーバーをサーバー SDK for Amazon GameLift Servers と統合して構築します。ゲームサーバーは、サーバー SDK を使用して、ゲームセッションの開始、更新、停止など、ゲームセッション管理のために Amazon GameLift Servers サービスと通信します。サーバー SDK をゲームプロジェクトに統合する方法については、「[Amazon GameLift Servers でホストするゲームを準備します](#)」を参照してください。

開発サポート

- 開発 OS
 - Server
 - Linux
- プログラミング言語

[Amazon GameLift Servers SDK を取得します](#)。バージョン固有の情報とインストール手順については、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](#)
 - [サーバー SDK リファレンス](#)
 - [統合方法](#)
- [C# サーバー SDK](#) (.NET 4、.NET 6、.NET 8 のサポートはバージョンによって異なります。
「[SDK のバージョン](#)」を参照してください)
 - [サーバー SDK リファレンス](#)
 - [統合方法](#)
- [Go サーバー SDK](#)
 - [サーバー SDK リファレンス](#)
 - [統合方法](#)
- ゲームエンジンのサポート

のフル機能のプラグインAmazon GameLift Serversには、UI ワークフローとサンプルアセット、AWS および SDK とサーバー SDK の組み込みバージョンが含まれています。ワークフローでは、マネージド EC2 フリート、マネージドコンテナフリート、またはセルフマネージド Anywhere フリートでホスティングするためのゲームサーバーを設定およびデプロイする方法を説明します。ガイド付きワークフローが必要ない場合は、同じ GitHub リポジトリからゲームエンジン用サーバー SDK のみを取得できます。

プラグインがサポートしていない別のゲームエンジンまたは開発環境を使用している場合は、プログラミング言語用サーバー SDK を取得し、ゲームプロジェクトに追加します。

バージョン固有の情報とインストール手順については、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。

- [Unreal Engine 用プラグイン](#) – Unreal Engine バージョン 5.0、5.1、5.2、5.3、5.4、5.5、5.6 で使用するよう構築されています。Unreal のサポートについては、バージョン固有の readme ファイルを確認してください。
- [Unreal Engine 用プラグイン](#)
- [C++ \(Unreal\) サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)
- [Unity 用プラグイン](#) – Unity Editor 6.0、2022.3、または 2021.3 の LTS バージョンで使用するよう設計されています。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 のサポートについては、バージョン固有の readmes を確認してください。
- [Unity 用プラグインガイド](#)
- [??? C# サーバー SDK リファレンス](#)
- [サーバー SDK for Unreal](#)
 - [サーバー SDK リファレンス](#)
 - [Amazon GameLift Servers を Unreal Engine プロジェクトに統合する](#)
- [サーバー SDK for Unity](#)
 - [サーバー SDK リファレンス](#)
 - [Amazon GameLift Servers を Unity プロジェクトに統合する](#)

ランタイムのサポート

マネージドホスティングソリューションの場合は、次のいずれかの Amazon マシンイメージ (AMIs) で実行するようにゲームサーバーを構築します。詳細については [Amazon GameLift Servers AMI バージョン](#) の Amazon GameLift Servers を参照してください。

- [Windows Server 2016](#)
- [Amazon Linux 2023](#)
- [Amazon Linux 2](#)

Note

Amazon Linux 2 (AL2) は、2026 年 6 月 30 日にサポートを終了します。詳細については、「[Amazon Linux 2 に関する FAQ](#)」を参照してください。AL2 でホストされ、Amazon GameLift Servers サーバー SDK 4.x. を使用するゲームサーバーの場合、まずゲームサー

バービルドをサーバー SDK 5.x に更新し、AL2023 インスタンスにデプロイします。「[サーバー SDK 5.x への移行 Amazon GameLift Servers](#)」を参照してください。

その他のツール

[Amazon GameLift Servers のゲームサーバーラッパー](#)

このツールは、一連の基本的なゲームセッション管理機能を使用してホスティング用のゲームサーバーをデプロイするのに役立ちます。このツールを使用すると、ゲームコードを変更したり、サーバー SDK for Amazon GameLift Servers を統合したりする必要はありません。ゲームサーバーラッパーを使用してゲームサーバーをパッケージ化し、3 つの Amazon GameLift Servers ホスティングソリューション (Anywhere、マネージド EC2、またはマネージドコンテナ) のいずれかを使用してゲームホスティング用にデプロイします。このツールはゲームサーバーのカスタマイズをサポートしていないため、このツールは独自のゲームまたはサンプルゲームによる早期評価またはプロトタイプ作成に最適です。ゲームにカスタム機能が必要ない場合は、本番ホスティング用のゲームサーバーラッパーを使用してゲームサーバーをデプロイできます。

[Amazon GameLift Servers ツールキット](#)

Amazon GameLift Servers Toolkit は、開発者が一般的なシナリオや問題に対応できるように開発したスクリプトやその他のツールのコレクションです。ツールキットのマテリアルには、スクリプト、サンプルコード、リードムが含まれます。

- [Containers starter kit](#) – このツールを使用して、Amazon GameLift Servers マネージドコンテナでホストするためのゲームサーバービルドを設定するタスクを合理化します。このキットは、重要なゲームセッション管理機能をゲームサーバーに統合し、事前設定されたテンプレートを使用してコンテナフリートを作成し、ゲームサーバービルドの自動デプロイパイプラインを設定します。デプロイ後、Amazon GameLift Servers コンソールと API ツールを使用して、フリートのパフォーマンスのモニタリング、ゲームセッションの管理、メトリクスの分析を行うことができます。このキットは、ビルドの自動化には、ストレージには Amazon Simple Storage Service、インフラストラクチャのデプロイ AWS CloudFormation にはと AWS CodeBuild 統合されています。
- [高速ビルド更新ツール](#) – このツールを使用して、マネージド EC2 フリートにデプロイ済みのゲームサーバービルドを変更します。このツールは、変更のたびに新しい EC2 フリートを設定および作成することなく、ゲームビルドファイルをすばやく交換できるように構築されています。個々のインスタンスまたはフリート内のすべてのインスタンスを更新できます。オプションを使用すると、特定のビルドファイルまたはビルド全体を置き換えたり、更新後にゲームサーバーを再起動する方法を管理したりできます。

ゲームクライアントの場合

ゲームクライアントをクライアント SDK for と統合しますAmazon GameLift Servers。クライアント SDK は、ゲームクライアントがプレイヤーゲートウェイや UDP ping ビーコンなどのAmazon GameLift Servers機能を使用するためのユーティリティを提供します。

クライアント SDK を取得する

- [Client SDK for C++](#) – エンドポイント選択アルゴリズムが組み込まれたプレイヤーゲートウェイと UDP ping ビーコンユーティリティ、および統合を示すサンプルアプリケーションが含まれています。C++17 とスレッディングライブラリが必要です。
- [Unreal Engine 用クライアント SDK](#) – プレイヤーゲートウェイ (エンジンの変更の有無にかかわらず) と、エンドポイント選択アルゴリズムが組み込まれた UDP ping ビーコンユーティリティが含まれます。ソースビルドの Unreal Engine 5.1 以降が必要です。アクセスするには、[GitHub の Epic Games 組織](#)にメンバーシップが必要です。

統合ガイドについては、各 GitHub リポジトリのドキュメントを参照してください。

ゲームバックエンドサービスの場合

ゲーム用の 64 ビットバックエンドサービスを作成し、 のサービス API を含む AWS SDK の機能と統合しますAmazon GameLift Servers。バックエンドサービスを使用して、ゲームセッションの開始や検索、プレイヤーのゲームへの参加など、Amazon GameLift Serversサービスとのやり取りを処理します。

[AWS SDK を取得](#)

で AWS SDK を使用する方法の詳細についてはAmazon GameLift Servers、次のリソースを参照してください。

- [Amazon GameLift Servers API リファレンス](#)
- ゲームクライアント機能の統合
 - [カスタムゲームサーバービルドによるゲームのクライアント側のサービス統合](#)
 - [Realtimeサーバーとのゲームのクライアント側のサービス統合](#)
- [バックエンドサービスを設計する](#)

Amazon GameLift Servers リソース管理の場合

次のツールを使用して、Amazon GameLift Servers マネージドホスティングリソースを作成、更新、モニタリングします。

- [AWS マネジメントコンソール](#) – AWS コンソールは、を含むすべての個々の AWS サービスコンソールへの一元化されたアクセスを提供するウェブベースのアプリケーションです Amazon GameLift Servers。コンソールを使用して AWS アカウントを作成またはサインインし、Amazon GameLift Servers コンソールを開いてゲームホスティングリソースを操作します。ホスティングフリートやその他のリソースの設定とデプロイ、使用状況とパフォーマンスのメトリクスの表示、ダッシュボードでのリソースの追跡、その他多くのタスクを行うことができます。[[Amazon GameLift Servers コンソール](#)] に移動します。
- [Service API for Amazon GameLift Servers](#) – この API は、すべての Amazon GameLift Servers リソースへのプログラムによるアクセスを提供します。これは AWS SDK の一部であり、最も一般的なプログラミング言語で使えるようにダウンロードできます。 [AWS SDK を取得します](#)。
- [AWS コマンドラインインターフェイス \(CLI\)](#) – AWS CLI では、コマンドラインシェルを使用して AWS サービスとやり取りできます。このツールを使用すると、AWS サービスのパブリック APIs と、サービスで利用できるカスタマイズされたコマンドに直接アクセスできます。 [CLI AWS を取得します](#)。
- [AWS CloudFormation](#) for Amazon GameLift Servers – この CloudFormation サービスは、インフラストラクチャのデプロイと管理を合理化するための AWS リソースのモデル化とセットアップに役立ちます。テンプレートを作成してホスティングソリューションの Amazon GameLift Servers リソース CloudFormation を記述し、テンプレートを使用して追加のリソースを構築したり、設定を更新したりできます。 [Amazon GameLift Servers リソースタイプのリファレンス](#)を表示します。

Amazon GameLift Servers/Realtime の場合

マルチプレイヤーゲームをホストするために Realtime 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 Realtime サーバーに接続できるようにするには、Amazon GameLift Servers Realtime クライアント SDK を使用します。開始するには、[Realtime クライアント SDK](#) をダウンロードします。設定情報については、「[Amazon GameLift Servers Realtime のゲームクライアントの統合](#)」を参照してください。

SDK サポート

Realtime クライアント SDK には、以下の言語のソースが含まれています。

- C# (.NET)

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。
- ゲームエンジン - Unity、C# ライブラリをサポートするエンジン。

ゲームサーバーオペレーティングシステム

Realtime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 2](#)

Note

AL2 のサポートはまもなく終了します。詳細については、「[Amazon Linux 2 に関する FAQ](#)」を参照してください。

- [Amazon Linux 2023](#)

チュートリアル: Amazon GameLift Servers ラッパーを使用したクイックオンボーディング

Amazon GameLift Servers のオンボーディングチュートリアルへようこそ。このチュートリアルでは、クラウドベースのコンピューティングリソースのフリートでホストされるようにゲームサーバーを迅速にデプロイします。このチュートリアルを使用すると、Amazon GameLift Servers 用サーバー SDK をゲームコードに統合する作業をスキップし、代わりに最小限の機能でゲームをデプロイして Amazon GameLift Servers サービスと通信し、ゲームセッションを実行できます。基本的なホスティングソリューションを設定し、それを使用して、自動スケーリングやマッチメイキングなどの幅広い機能を体験します。また、ライブデモの一部としてゲームのプロトタイプをホストしたり、テストしたりするのも最適です。

このオンボーディング方法の主な利点:

- 高速ホスティングのためにゲームサーバーをすばやくデプロイします。

- ゲームコードの変更はゼロで、変更は必要ありません。
- ゲームエンジンに関係なく、このメソッドを任意のゲーム実行可能ファイルで使います。
- ゲームセッションアクティビティのモニタリングやホスティングの状態など、すべての Amazon GameLift Servers 管理ツールについて説明します。

Note

ラッパーは、評価と基本的な本番環境での使用を目的としています。詳細なプレイヤーセッション管理などの高度な機能には、完全なサーバー SDK 統合が必要です。

前提条件

開始する前に、以下を確認してください。

- 適切なアクセス許可 AWS アカウント を持つ
- AWS CLI インストール済み
- Go 1.18+
- マルチプレイヤーゲームサーバー実行可能ファイル
- Make (Linux/Mac)
- アクティブなアカウントでインストールされた Git

概要:

このチュートリアルでは、次の作業を行います。

1. ラッパーの取得と構築
2. ゲームビルドを準備する
3. ラッパーを設定する
4. サンプルゲームサーバービルドをアップロードします。
5. マネージド EC2 フリートを作成する
6. ゲームセッションを作成して接続する
7. ゲームサーバーのモニタリングと管理

8. ゲームサーバーのスケーリング

ステップ 1: ゲームサーバーラッパーを取得して構築する

次のコマンドを使用して、ゲームサーバーのラッパーソースを取得し、ラッパーを構築します。これらのコマンドは SSH を使用しますが、Github リポジトリに直接移動することもできます。

Server

```
> git clone git@github.com:amazon-gamelift/amazon-gamelift-servers-game-server-wrapper.git
> cd amazon-gamelift-servers-game-server-wrapper
> powershell -file .\build.ps1
```

Mac と Linux

```
$ git clone git@github.com:amazon-gamelift/amazon-gamelift-servers-game-server-wrapper.git
$ cd amazon-gamelift-servers-game-server-wrapper
$ make
```

ビルドが成功すると、「out」ディレクトリが amazon-gamelift-servers-game-server-wrapper に追加されます。このディレクトリには、サポートされているホスティングフリートオプションごとに 1 つずつ、ビルドアーティファクトのセットを含む 3 つのフォルダがあります。このチュートリアルでは、マネージド EC2 フリートにデプロイするため、gamelift-servers-managed-ec2 フォルダを使用します。

ステップ 2: ゲームサーバービルドを準備する

このステップでは、Amazon GameLift Servers にアップロードするゲームサーバービルドファイルを準備します。

ディレクトリを作成します。

次に、ローカルマシンにゲームディレクトリを準備します。このディレクトリには、Amazon GameLift Servers でゲームサーバーを実行するため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。これには、ゲームサーバーラッパー、ゲームサーバービルド、およびラッパーをゲームサーバーと連携させる `config.yaml` が含まれます。

以下のステップを使用します。

1. マネージド EC2 フリート。ゲームサーバーのラッパー出力フォルダで、マネージド EC2 フリートにデプロイするためのビルドアーティファクトを見つけます。ビルドは、`out\linux\amd64\gamelift-servers-managed-ec2` に示すように アウトディレクトリに書き込まれます。
2. ゲームサーバーの実行可能ファイルと、`gamelift-servers-managed-ec2` フォルダの実行に必要なすべての関連ファイルをコピーします。必要に応じて、ネストされたディレクトリを持つことができます。

ディレクトリ構造の例は次のようになります。

```
gamelift-servers-managed-ec2
#-- config.yaml
#-- amazon-gamelift-servers-game-server-wrapper
#-- MyGame
# #-- server-executable.exe
# #-- my-game-settings
# # .....
```

ステップ 3: フリーットのラッパーを設定する

Amazon GameLift Servers はフリーットのコンピューティングインスタンスのライフサイクルを管理し、サーバービルドがインストールされた新しいインスタンスをスピンアップし、必要に応じてインスタンスをリサイクルします。サービスは、各インスタンス上で実行される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。マネージド EC2 フリートは、複数の場所にインスタンスを配置して、どこにいてもプレイヤーをサポートできます。

`config.yaml` ファイルを編集して、ログ記録、ポート設定、サーバー初期化用のラッパーを設定します。

1. ログ記録設定を構成します。ゲームサーバーラッパーは、ゲームサーバープロセスごとにログメッセージを生成します。デフォルトでは、ログレベルは最大の詳細度でデバッグするように設定されています。これはセットアップとトラブルシューティングに非常に役立ち、ログメッセー

ジの詳細度を決定します。この場合、最も詳細です。オプションには、デバッグ、情報、警告、エラー (最も詳細でない) などがあります。

2. ゲームサーバーログディレクトリへのパスを指定します。ゲームサーバーログのデフォルトパスは `./game-server-logs` です。このディレクトリには、ゲームサーバーによって生成されたすべてのログが含まれ、各インスタンスにこのディレクトリが存在します。ログは、[イベント] タブからアクセスされる Amazon GameLift Servers に自動的にアップロードされます。詳細については、「トラブルシューティング」セクションを参照してください。
3. ネットワークポート設定を定義します。ゲームポートを任意の値に設定します。このチュートリアルでは、1 つのポートのみを指定します。これは、インスタンスごとに 1 つの同時ゲームサーバープロセスのみを実行するフリートを作成するためです。一度に複数のプロセスを実行する場合は、同時プロセスごとに十分なポートを設定する必要があります。デフォルト値は設定ファイルに示すように 37016 ですが、一般的に Linux ビルドを使用するフリートではポート 22 と 1026-60000 を使用します。Windows ビルドを使用するフリートの場合は、ポート 1026-60000 を使用します。
4. ゲームサーバー実行可能ファイルへのパスを設定します。 `./MyGame/my-server-executable.exe` では、実際の名前と場所を使用して、ゲームサーバー実行ファイルへのパスをカスタマイズします。これは、ゲームサーバーを起動するためのエントリポイントです。
5. ゲームサーバーの引数を設定します。少なくとも、`-port` 引数を指定し、前に定義したのと同じゲームポート値を使用します。「pos」値 0 は、これが最初の引数であることを示します。必要に応じて他の引数を追加します。これらの引数は起動時にゲームサーバーに渡されるため、ランタイム動作を設定できます。
 - a. 引数: `--port`
 - b. 値: `"{{.port number here}}"`
 - c. 位置: 0 (リストの最初の引数)

設定例:

```
log-config:
  wrapper-log-level: debug
game-server-logs-dir: ./game-server-logs
ports:
  gamePort: 37016
game-server-details:
  executable-file-path: ./MyGame/my-server-executable
game-server-args:
```

```
- arg: "--port"  
  val: "{{.gameport}}"  
  pos: 0
```

ステップ 4: ゲームサーバービルドをアップロードする

これで、ゲームサーバービルドに必要なすべての要素 (ゲームサーバーラッパー、config.yaml、ゲームサーバーファイル) が完了し、ホスティングのためにゲームビルドを Amazon GameLift Servers にアップロードする準備が整いました。ゲームビルドをアップロードする最も簡単な方法は、次の例 upload-build に示すように AWS CLI コマンドを使用することです。

Windows でのゲームビルドのアップロード:

```
aws gamelift upload-build \  
  --name gamelift-test-2025-03-11-1 \  
  --build-version gamelift-test-2025-03-11-1 \  
  --build-root out/windows/amd64/gamelift-servers-managed-ec2 \  
  --operating-system WINDOWS_2022 \  
  --server-sdk-version 5.3.0 \  
  --region us-west-2
```

Note

Mac および Linux ビルドの場合は、--operating-system AMAZON_LINUX_2023 を使用します。

ビルドを作成するときは、API レスポンスからビルド ID を記録して、フリートの作成に使用します。

ステップ 5: マネージド EC2 フリートを作成する

次のステップでは、最小限のフリート設定について説明します。これにより、できるだけ早く起動して実行できます。

フリートを作成するには:

1. にログイン AWS マネジメントコンソール し、 に移動します Amazon GameLift Servers。
2. コンソールウィンドウの上部にあるメニューバーで、ビルドがどのリージョンにあるかを確認します。フリートが同じリージョンに存在する必要があるため、メモしておきます。そうしないと、ビルドを検索または選択できなくなります。
3. マネージド EC2 セクションのナビゲーションペインで、[ビルド] を選択します。
4. 以前にアップロードしたビルドを選択して、ビルドの詳細ページを表示します。
5. フリートセクションで [フリートを作成] を選択すると、マネージド EC2 フリートの定義の詳細ページが表示され、フリートのステータスを追跡したり、[イベント] タブでフリート作成イベントを表示したりできます。
6. 名前と説明を入力し、[次へ] を選択します。
7. インスタンスの詳細の定義ページには、ビルドのリージョンがデフォルトで表示されます。追加するリージョンを選択します。
8. [フリートタイプ] で [オンデマンド] を選択します。
9. [インスタンスタイプ] で [c5.large] を選択し、[次へ] を選択します。
- 10[ランタイム設定] では、アップロードされたゲームビルドはラッパーを使用するため、代わりにラッパー実行可能ファイルを指す必要があります。Windows サーバーの場合、これは「C:\game\amazon-gamelift-servers-game-server-wrapper.exe」になります。Linux ゲームサーバーの場合、これは「/local/game/amazon-gamelift-servers-game-server-wrapper」です。

例: LaunchPath": "/local/game/amazon-gamelift-servers-game-server-wrapper", "ConcurrentExecutions": 1, "Parameters": "-port 37016

また、config.yaml で設定されたポートとランタイム設定起動パラメータを許可する範囲でゲームポート値を設定します。config.yaml のポートは、フリートのランタイム設定で指定されたポートと一致する必要はありませんが、設定で別のポートが指定されている場合、その値は config.yaml の値よりも優先されます。ランタイム設定に入力された起動パラメータは、config.yaml の内容も上書きします。

- 11[確認と作成]ページで、すべての設定を再確認し、[送信] を選択してフリートを作成します。フリートのステータスは、ゲームサーバーをホストするために容量をスピンアップすると変化し、すぐにアクティブのステータスが表示されます。アクティベーションが完了し、フリートがデプロイされると、サービスはゲームセッションリクエストを受け取る準備ができてラッパーを起動します。

ステップ 6: ゲームセッションを作成して接続する

フリートのステータスがアクティブの場合、ゲームサーバーの準備が整い、ゲームセッションのホストを待っていることを意味します。ゲームセッションを開始するには、Amazon GameLift Servers サービスにゲームセッションリクエストを行います。ここでは、AWS CLI を使用してこのリクエストを行います。

Note

を使用してゲームセッションを作成する AWS CLI ことは、プロセスをテストして理解するのに役立ちます。ある時点で、ゲームのマッチメイキングまたはゲームセッションプレイスメントシステムの一部として、へのプログラムによる AWS SDK 呼び出しをゲームバックエンドサービスに追加します。

以下を使用してゲームセッションを作成します。

```
aws gamelift create-game-session \  
--fleet-id <FLEET_ID> \  
--game-properties '[{"Key": "exampleProperty", "Value": "exampleValue"}]' \  
--maximum-player-session-count 3 \  
--region us-west-2
```

カスタマイズされたゲームプロパティをサーバー実行可能ファイルに渡すこともできます。詳細については、README のゲームサーバー引数を参照してください。create-game-session 呼び出しを受信すると、Amazon GameLift Servers はラッパーに通知してゲームサーバー実行可能ファイルを起動し、ゲームセッションを開始します。config.yaml の内容はゲームサーバーの設定に影響し、コンソールで設定される起動パラメータによってゲームセッション自体の設定が決まります。

ゲームプロパティを追加するための形式例:

```
defaultArgs:  
  - arg: "--port"  
    val: "${.GamePort}"  
    pos: 0  
  - arg: "--ipAddress"  
    val: "${.IpAddress}"
```

```
pos: 1
- arg: "--gameSessionId"
  val: "{.GameSessionId}"
pos: 2
```

ステップ 7: フリートを管理およびモニタリングする

ゲームサーバーフリートが設定され、ゲームセッションが開始されたので、Amazon GameLift Servers コンソールから管理およびモニタリングできます。これを実現する最善の方法は、フリートの詳細ページから、フリートの詳細を編集したり、[スケーリング] タブでフリートのスケーリングと容量を変更したりできます。ゲームサーバーのスケーリングについては、次のセクションを参照してください。

[メトリクス] タブを選択すると、アクティビティメトリクスとハードウェアメトリクスのモニタリングを示すグラフが表示されます。メトリクスグラフの詳細については、フリートの ID の横にある[情報] リンクを選択します。さらに、[メトリクス] タブからゲームサーバーを注意深くモニタリングできますが、CloudWatch ダッシュボードでこれらのメトリクスにアラームを追加することもできます。

コンソールから CloudWatch ダッシュボードに移動するには:

1. 検索バーに「CloudWatch」と入力し、検索結果リストから選択して CloudWatch 概要を表示します。
2. 下にスクロールして [GameLift ダッシュボードを表示] を選択すると、フリートとゲームセッションの主要なメトリクスに基づくグラフが表示されます。

ステップ 8: ゲームサーバーをスケールする

次のステップでは、自動スケーリングを設定します。自動スケーリングでは、ゲームサーバーの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調整します。自動スケーリングでは、ゲームセッションを開始するプレイヤーが増えるとインスタンスを追加し、プレイヤーが減ると未使用のインスタンスを解放できます。自動スケーリングは、スムーズで高速なプレイヤー体験を提供しながら、ホスティングリソースとコストを最小限に抑える効果的な方法です。

ゲームの起動を準備するときは、フリートの自動スケーリングを設定する必要があります。自動スケーリングは、スムーズで高速なプレイヤー体験を提供しながら、ホスティングリソースとコストを最小限に抑える効果的な方法です。

フリートの容量を手動で設定するには

フリートの容量を手動で設定するには

1. フリートの詳細ページの [スケーリング] タブに移動します。
2. 場所を選択し、[編集] を選択します。
3. [目的のインスタンス] 値を変更し、現在の値を超えてスケールするように [最小] サイズと [最大] サイズ設定を変更して、[確定] を選択します。

Note

最大インスタンス設定をストップギャップとして使用し、過剰なスケーリングとコストを防ぎます。

ターゲットベースの自動スケーリングを使用するには

ターゲットベースの自動スケーリングを使用するには

ターゲットベースの自動スケーリング (ターゲット追跡) は、フリートスケーリングを利用可能なゲームセッションの割合にリンクします。プレイヤーがゲームプレイを急増し、利用可能なゲームセッションが減少すると、フリートに自動的にインスタンスを追加することで応答します。

1. [ターゲットベースの自動スケーリングポリシー] で、[ポリシーを追加] を選択し、設定した利用可能なゲームセッションの割合のしきい値に達したときにフリートの容量が自動的に変更されるように設定します。バッファを大きくすると、急増をより適切に処理し、新しいプレイヤーがゲームにすばやく参加できますが、ホスティングコストが高くなる可能性もあります。
2. [確定] を選択して変更を受け入れます。

ルールベースの自動スケーリングにより、スケーリングを他のフリートメトリクスにリンクし、カスタムしきい値とスケーリングレスポンスを設定する機能など、よりきめ細かい制御が可能になります。強力なオプションを提供しますが、カスタムルールの動作を理解するには、CLI の使用と十分なテストが必要です。このチュートリアルでは、最初にターゲットベースのアプローチを設定することに重点を置いています。

一般的な問題のトラブルシューティング

ゲームサーバーとゲームセッションの実行中に発生する可能性がある一般的な問題は次のとおりです。サーバーまたはゲームセッションのいずれかが正しく動作していない場合、最初のステップはログを確認することです。ログには、新規デプロイや本番環境でのゲームで、以下に示す問題のいずれかが明らかになることがあります。

ログには多くの場合、次の内容が表示されます。

- ゲームサーバープロセスは起動できません。これは wrapper 設定のエラーである可能性があります。ファイルに正しい起動パス、起動パラメータ、および引数が設定されていることを確認します。
- ゲームサーバービルドを実行できません。これはゲームコードのエラーである可能性があります。
- プレイヤーはゲームセッションに接続できません。これは、おそらくポート設定エラーです。
- 遅延または接続の低速化。スケーリングポリシーとしきい値を確認します。
- 接続がありません。フリートのポートルールと設定を確認します。

Amazon GameLift Servers フリートのイベントログを表示するには

Amazon GameLift Servers フリートのイベントログを表示するには

1. Amazon GameLift Servers コンソールを開きます。
2. フリートの詳細ページで [イベント] タブを選択し、ログをダウンロードします。[メトリクス] タブから、ゲームサーバーのヘルス、ゲームセッションのアクティベーション、アクティビティ、およびハードウェアメトリクスをモニタリングすることもできます。

ゲームセッションログを表示するには

ゲームセッションログを表示するには

1. コンソールからフリートを開き、[ゲームセッション] タブを開きます。
2. リストからゲームセッション ID を選択すると、概要ページが表示されます。
3. [ログをダウンロード] を選択して、ログファイルをローカルにダウンロードします。

CLI でゲームセッションログを表示するには、GetGameSessionLogURL API を使用します。Amazon GameLift Servers はログを 14 日間自動的に保存します。

フリートの Amazon CloudWatch logs を設定することもできます。これにより、追加のログ記録機能と他の AWS モニタリングサービスとの統合が提供されます。

CloudWatch を介したリアルタイムログアクセスまたは延長保持期間の場合:

1. Amazon GameLift Servers コンソールダッシュボードの上部の検索バーに「CloudWatch」と入力し、結果のドロップダウンから選択します。
2. CloudWatch ロググループに移動し、特定のセッションを検索します。最も簡単な方法は、[すべて検索] をクリックし、gameSessionId または clientId を使用してフィルタリングすることです。

次の手順

- [マルチロケーションフリートを作成して、追加のロケーションにホスティングを追加します](#)
- [ゲームセッションキューを追加して、複数の場所にわたるゲームセッションの配置を最適化する](#)
- [ゲームのマッチメーカーとルールセットを作成して FlexMatch マッチメイキングを試す](#)
- [プレイヤーが参加リクエストを行い、ゲームセッションに直接接続できるように、ゲームクライアントとバックエンドサービスコンポーネントの機能の実装を開始します](#)
- [準備ができれば、完全に統合されたソリューションに移行します。](#)

Amazon GameLift Servers の例

カスタムゲームサーバーの管理に Amazon GameLift Servers の使用を検討している場合、または Amazon GameLift Servers や Realtime を活用したい場合は、サービス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 Servers 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。Amazon GameLift Servers と Realtime の例では、Realtime サーバーを使用してゲームをホスティングできるように準備する方法を示します。

カスタムゲームサーバーの例

この例では、サンプルゲームサーバーをホスティング用に Amazon GameLift Servers が管理する EC2 フリートへデプロイする手順を示します。サンプルゲームクライアントを使用して、ライブゲームセッションに接続します。コンソールや AWS CLI などの Amazon GameLift Servers ツールを使用して、フリートのホスティングパフォーマンスと使用状況を監視する方法を体験できます。

この例では、次の手順を具体的に説明します。

- サンプルゲームサーバービルドをアップロードします。
- ゲームサーバービルドを実行するフリートを作成します。
- サンプルゲームクライアントを取得し、それを使用してゲームサーバーに接続し、ゲームセッションに参加します。
- フリートとゲームセッションのメトリクスを確認します。

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します。Amazon GameLift Servers コンソールを使用して、ホスティングリソースの表示、メトリクスの追跡、フリートのホスティング容量をスケーリングするオプションを検討します。

使用を開始するには、[Amazon GameLift Servers コンソール](#)にサインインします。左側のナビゲーションで、[リソース]、[サンプルゲームを試す] の順に選択します。

Amazon GameLift Servers を使用したゲームホスティングソリューションの構築

このセクションでは、設計のベストプラクティスなど、Amazon GameLift Servers ゲームホスティングソリューションのすべてのコンポーネントを構築する方法に関するガイダンスを提供します。ゼロから開始する場合も、既存のゲームを移行する場合も、Amazon GameLift Servers を使用してゲームホスティングインフラストラクチャを設計、実装、最適化するのに役立つ包括的な情報が得られます。

ホスティングソリューションコンポーネント

ホスティングソリューションには、次のコンポーネントのバージョンが必要です。これらのコンポーネントとその連携方法の詳細については、[Amazon GameLift Servers でのホスティングの仕組み](#) を参照してください。

- ゲームクライアントはプレイヤーデバイス上で実行されます。
- バックエンドサービスは、ゲームクライアントと Amazon GameLift Servers と間の通信を調整します。
- ゲームサーバーソフトウェアは、マルチプレイヤーのゲームプレイを管理します。
- プレイメントシステムはプレイヤーをゲームサーバーにマッチングします。
- ゲームホスティングフリートは、地理的な場所にコンピューティングリソースを提供します。
- ゲームホスティング管理システムは、ゲームホスティングのステータスをモニタリングし、容量を管理します。

トピック

- [開発ロードマップ](#)
- [Amazon GameLift Serversでホストするゲームを準備します](#)
- [Amazon GameLift Serversのバックエンドサービスを構築する](#)
- [Amazon GameLift Servers のホスティングフリートをデプロイする](#)
- [ゲームセッションの配置を設定する](#)
- [ゲームホスティングソリューションをカスタマイズする](#)

開発ロードマップ

開発ロードマップは、Amazon GameLift Servers でさまざまなタイプのゲームホスティングを駆逐するための、構造化され、詳細な手順を示したガイダンスを提供します。各ロードマップは特定のホスティングアプローチに合わせて調整され、初期設定からデプロイ、テストまで、重要なタスクを順を追って説明します。

ホスティング要件と技術的な制約に最も適したロードマップを選択します。また、複数のロードマップに従ってアプローチを組み合わせて、特定のニーズを満たすハイブリッドソリューションを作成することもできます。

トピック

- [Amazon GameLift Servers マネージド EC2 でホスティングするための開発ロードマップ](#)
- [Amazon GameLift Servers マネージドコンテナでホスティングするための開発ロードマップ](#)
- [Amazon GameLift Servers Anywhere でホスティングするための開発ロードマップ](#)
- [Amazon GameLift Serversを利用するハイブリッドホスティングの開発ロードマップ](#)

Amazon GameLift Servers マネージド EC2 でホスティングするための開発ロードマップ

このロードマップでは、マルチプレイヤーゲーム用の Amazon GameLift Servers マネージド EC2 ホスティングソリューションを開発する方法について説明します。Amazon GameLift Servers には、いくつかのゲームホスティングオプションが用意されています。これらのオプションの詳細については、「[Amazon GameLift Servers ゲームホスティングオプション](#)」を参照してください。

Amazon GameLift Servers マネージドホスティングでは、ゲームサーバーは、設定に基づいて Amazon GameLift Servers 所有および運用する AWS クラウドベース仮想コンピューティングリソースでホストされます。マルチプレイヤーゲームホスティング用にさらに最適化された Amazon Elastic Compute Cloud (Amazon EC2) インスタンスのセキュリティ、信頼性、グローバル可用性が得られます。Amazon GameLift Servers は、自動サーバーデプロイ、ライフサイクル処理、容量の自動スケーリングなどのツールを使用してホスティング管理を合理化します。

Amazon GameLift Servers マネージドソリューションは、以下のコンポーネントで構成されています。

- サーバー SDK for Amazon GameLift Servers と統合された 1 つのゲームサーバービルド。すべてのフリートにデプロイします。

- AWS SDK と統合されたゲームクライアントとバックエンドサービス。Amazon GameLift Servers サービスとやり取りしてゲームセッションをリクエストします。
- すべてのフリートで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配置する Amazon GameLift Servers キュー。
- (オプション) マルチプレイヤーマッチを作成し、マッチのゲームセッションを立ち上げる FlexMatch マッチメーカー。
- マルチプレイヤーゲームホスティング用に最適化された Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを使用する 1 つまたは複数の Amazon GameLift Servers マネージドフリート。

このロードマップは、Amazon GameLift Servers マネージド EC2 ホスティングを使用してマルチプレイヤーゲームを稼働させるための効率的な道筋を示しています。必要なコンポーネントが揃ったら、ゲーム開発を繰り返し、ホスティングソリューションをカスタマイズできます。ローンチが近づいたら、本番稼働レベルの使用に備えてホスティングソリューションを準備する方法について、「[Amazon GameLift Servers ホスティングによる起動の準備](#)」を参照してください。

i Unreal Engine と Unity 用の Amazon GameLift Servers プラグインでジャンプスタートするデプロイを高速化するには、Unreal Engine と Unity の [Amazon GameLift Servers プラグイン](#)を試してください。最小限のセットアップでゲームサーバーをすばやくデプロイするためのガイド付き UI ワークフローを提供するため、ゲームコンポーネントを実際に試すことができます。その後、この基盤をもとに、ゲーム用のカスタムホスティングソリューションを作成できます。詳細については、[Amazon GameLift Servers プラグインを使用して Unreal または Unity ゲームを準備する](#)を参照してください。

ステップ 1: Amazon GameLift Servers を使用するためのゲームサーバーを準備する

ゲームサーバーに機能を追加して、ホスティング用にデプロイされたときに Amazon GameLift Servers サービスと通信できるようにします。

- ゲームプロジェクト用に SDK for Amazon GameLift Servers (バージョン 5.x) を取得します。サーバー SDK は、C++、C#、Go で利用可能です。[Amazon GameLift Servers サーバー SDK をダウンロードします](#)。
- ゲームサーバーコードを変更して、サーバーの SDK 機能を追加します。ガイダンスについては、「[Amazon GameLift Serversでホストするゲームを準備します](#)」を参照してください。少なくとも、以下を実行します。

- Amazon GameLift Servers SDK を初期化するコードを追加し、Amazon GameLift Servers サービスとの WebSocket 接続を確立します。サーバー SDK アクション `InitSdk()` を使用します。
- サーバード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 Servers サービスに報告するコードを追加します。サーバー SDK アクション `ProcessReady()` を使用します。
- 必要なコールバック関数 `OnProcessTerminate()`、および `OnStartGameSession()` を実装します。これらの関数を使用することで、ゲームサーバードプロセスは Amazon GameLift Servers サービスとの接続を維持し、Amazon GameLift Servers からの指示によりゲームセッションを開始し、ゲームサーバードプロセスを終了するプロンプトに応答できます。
- サーバードプロセスがゲームセッションを終了することを Amazon GameLift Servers サービスに報告するコードを追加します。サーバー SDK アクション `ProcessEnding()` を使用します。
- ゲームサーバービルドをパッケージ化します。ビルドファイル、依存関係、サポートソフトウェアを含むインストールスクリプトを作成します。「[ゲームビルドファイルをパッケージ化する](#)」を参照してください。Amazon Simple Storage Service (Amazon S3) バケットを使用して、ゲームビルドのバージョンを保存することをお勧めします。
- ゲームサーバーの統合をテストします。このタスクでは、「[Amazon GameLift Servers Anywhere でローカルテストを設定する](#)」で説明されているように、ローカルワークステーションで Amazon GameLift Servers Anywhere フリートをセットアップすることをお勧めします。このステップでは、ゲームサーバービルドをテストデバイスに手動でインストールし、サーバードプロセスを開始します。AWS CLI を使用して新しいゲームセッションをリクエストし、Amazon GameLift Servers サービスがサーバードプロセスにゲームセッションの開始を正常に指示することを確認します。

ステップ 2: ホストされているゲームセッションに参加するためのゲームクライアントを準備する

ゲームクライアントがゲームセッションへの参加をリクエストし、接続情報を取得して、ホストされているゲームセッションに直接接続する方法を作成します。最も一般的なアプローチは、ゲームクライアントと Amazon GameLift Servers サービス間の仲介役として機能するバックエンドサービス機能をセットアップすることです。このアプローチにより、ホスティングリソースを保護し、プレイヤーがゲームセッションにどのように配置されるかをより詳細に制御することができます。

- ホスティング用のバックエンドサービス機能を構築します。バックエンドサービスは Amazon GameLift Servers サービスと通信し、ゲームクライアントに接続情報を送信します。この機能には、ゲームセッションの開始、ゲームへのプレイヤーの配置、ゲームセッション情報の取得が含まれます。ガイダンスについては、「[Amazon GameLift Servers でホストするゲームを準備します](#)」を参照してください。少なくとも、以下を実行します。

- AWS SDK for Amazon GameLift Serversを取得し、バックエンドサービスプロジェクトに追加します。「[クライアントサービス用の Amazon GameLift Servers SDK リソース](#)」を参照してください。
- Amazon GameLift Servers クライアントを初期化するコードを追加し、キー設定を保存します。「[Amazon GameLift Servers APIをセットアップします。](#)」を参照してください。
- AWS SDK アクションを呼び出しCreateGameSession()、ゲームクライアントにゲームセッション接続情報を提供する機能を追加します。「[特定のフリートにゲームセッションを作成する](#)」を参照してください。

CreateGameSession() の呼び出しは、新しいゲームセッションをリクエストする際に便利な出発点となります。ゲームセッション配置システムが整ったら (ステップ 3 を参照)、このコードを StartGameSessionPlacement() への呼び出しに置き換えます (StartMatchmaking() を使用している場合は FlexMatch)。

バックエンドサービスを設計する方法については、「[Amazon GameLift Serversのバックエンドサービスを構築する](#)」を参照してください。

- プレイヤーがホストされているゲームセッションに参加できるようにする機能をゲームクライアントに追加します。ゲームクライアントは、Amazon GameLift Servers に直接リクエストするのではなく、バックエンドサービスにリクエストを行います。バックエンドサービスがゲームセッション接続情報を提供すると、ゲームクライアントはゲームセッションに直接接続してゲームをプレイします。
- ゲームクライアントの統合をテストします。テストでは、ローカルワークステーションで同じ Amazon GameLift Servers Anywhere フリートを使用できます。

高速反復型開発や複数人チームでの作業には、[クラウドベースのテスト環境](#)を立ち上げることをお勧めします。この Amazon GameLift Servers Toolkit ソリューションは、Amazon GameLift Servers マネージドフリートの動作を模倣しますが、ゲームサーバーのビルドを最小限のターンアラウンドタイムで更新できます。

ステップ 3: ゲームセッション配置をセットアップする

Amazon GameLift Servers が新しいゲームセッションのリクエストをどのように処理し、ホスト可能なゲームサーバーをどのように特定するかをカスタマイズできます。Amazon GameLift Servers は、すべてのフリート内のゲームサーバーの可用性を自動的に追跡します。ゲームクライアントがゲームセッションへの参加リクエストを送信すると、Amazon GameLift Servers は最小レイテンシー、コスト、可用性などの定義された一連の優先順位に基づいて「可能な限り最良の」配置を探します。

- 利用可能なゲームサーバーで新しいゲームセッションを配置するためのゲームセッションキューを作成します。キューは、ゲームセッション配置の主要なメカニズムです。ガイダンスについては、「[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。
- 少なくとも、Anywhere フリートをキュー内の送信先として追加します。その他のすべての設定はオプションのカスタマイズです。
- バックエンドサービスコードで、**CreateGameSession()** 呼び出しを **StartGameSessionPlacement()** に変換します。「[マルチフリートキューにゲームセッションを作成する](#)」を参照してください。
- ゲームセッションに参加できる状態になったら、ゲームクライアントに通知するメカニズムを作成します。開発中は、DescribeGameSessionPlacement を呼び出してゲームセッションのステータスをポーリングできます。ただし、大量の処理にキューを使用する前に、イベント通知を有効にする必要があります。「[ゲームセッション配置のイベント通知を設定](#)」を参照してください。
- (オプション) FlexMatch マッチメイキングコンポーネントを追加します。ガイダンスについては、「[Amazon GameLift Servers FlexMatch デベロッパーガイド](#)」を参照してください。

ステップ 4: マネージドクラウドベースのフリートを作成する

これまで、セルフマネージド型の Anywhere フリートを使用してゲームコンポーネントをテストして反復処理し、ゲームセッションの配置をファインチューニングしてきました。ソリューションの最後の部分では、本番稼働システムに必要なタイプのホスティングリソースを設定します。本番稼働に向けた計画と設定を開始するには、Amazon GameLift Servers マネージドフリートの使用に移行します。

- ゲームサーバーを構築し、そのビルドを Amazon GameLift Servers にアップロードします。ビルドファイル、依存関係、サポートソフトウェアを含むインストールスクリプトを作成します。「[Amazon GameLift Servers のゲームサーバービルドを作成する](#)」を参照してください。コンソールまたは CLI Amazon GameLift Servers を使用して、ビルドを AWS にアップロードできます。

ビルドをアップロードする前に、フリート AWS リージョン を作成するものを決定します。ビルドを同じリージョンにアップロードする必要があります。フリートのロケーションの選び方については、「[地理的場所](#)」を参照してください。

- マネージド EC2 フリートを作成します。フリートを作成すると、Amazon GameLift Servers はすぐにホスティング用にゲームサーバービルドのデプロイを開始します。マネージドフリートの多くの側面を設定できます。ガイダンスについては、「[Amazon GameLift Servers マネージド EC2 フリートを作成する](#)」を参照してください。少なくとも、以下を実行します。
- フリートに名前を付け、デプロイするアップロード済みのゲームビルドを指定します。

- フリートのオンデマンドインスタンスを選択し、フリートのロケーションで利用可能なインスタンスタイプを選択します。スポットフリートは有用なオプションですが、追加の設計と設定が必要です。
- フリートのランタイム設定を作成します。少なくとも、ゲームサーバー実行可能ファイルの起動パスを指定します。
- インバウンドトラフィックがゲームサーバーにアクセスできるように、ポート設定を指定します。
- マネージドフリートをキューに追加します。ゲームセッションキューで、Anywhere フリートをマネージドフリートで置き換えます。
- マネージドフリートでゲームホスティングをテストします。この時点で、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から、接続情報を取得して、ゲームセッションに正常に接続するまでのホスティングサイクル全体をテストできるはずです。

ステップ 5: マネージドフリートをカスタマイズする

ゲームのローンチに備えて、マネージドホスティングリソースをファインチューニングする必要があります。考慮すべき決定事項には、次のようなものがあります。

- コスト削減のためにスポットフリートの追加を検討します。「[スポットフリートによるゲームホスティングコストの削減](#)」を参照してください。
- ゲームサーバーが他の AWS リソースと通信する必要がある場合は、IAM ロールを設定してアクセスを管理します。「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」を参照してください。
- ゲームサーバーを地理的にどこに配置するかを決定します。マネージドフリートにリモートロケーションを追加します。「[ホスティングリソースのカスタマイズ](#)」を参照してください。
- インスタンスのタイプとサイズを選択し、複数のサーバープロセスを実行するようにランタイムを設定することで、フリートのパフォーマンスを最適化します。「[マネージド Amazon GameLift Servers でのゲームサーバーランタイム設定の最適化](#)」を参照してください。
- 優先順位付け設定のカスタマイズなど、マネージドフリートのゲームセッション配置オプションを試してみます。「[ゲームセッションキューをカスタマイズする](#)」を参照してください。
- 予想されるプレイヤー需要に応えるための自動容量スケーリングを設定します。「[Amazon GameLift Servers を使用したゲームホスティング容量のスケーリング](#)」を参照してください。
- 他の にスタンバイフリートを設定し AWS リージョン、必要に応じてフェイルオーバーを処理するようにキューと自動スケーリングを変更します。

- 分析やログ記録など、ホスティングのオブザーバビリティツールをセットアップします。
「[Amazon GameLift Servers のモニタリング](#)」を参照してください。
- [Infrastructure as Code \(IaC\)](#) を使用してデプロイを自動化します。「[を使用してAmazon GameLift Serversホスティングリソースを管理する CloudFormation](#)」を参照してください。

Amazon GameLift Servers では、デプロイ固有の設定に AWS CloudFormation テンプレートを使用できます。を使用してAmazon GameLift Serversリソース AWS Cloud Development Kit (AWS CDK) を定義することもできます。の詳細については AWS CDK、[AWS Cloud Development Kit \(AWS CDK\) デベロッパーガイド](#)を参照してください。

CloudFormation スタックのデプロイを管理するには、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) ツールや などのサービスを使用することをお勧めします AWS CodePipeline。こうしたツールにより、ゲームサーバーバイナリが構築されるたびに、自動的に、または承認によりデプロイできます。CI/CD ツールまたはサービスを使用すると、新しいゲームサーバーバージョンのリソースのデプロイは次のようになります。

- ゲームサーバーバイナリをビルドしてテストします。
- バイナリを Amazon GameLift Servers にアップロードします。
- 新しいビルドを使用して新しいフリートをデプロイします。
- ゲームセッションキューに新しいフリートを追加し、以前のビルドバージョンのフリートを削除する。
- 前のビルドのフリートがアクティブなゲームセッションをホストしなくなったら、それらのフリートの CloudFormation スタックを削除します。

Amazon GameLift Servers マネージドコンテナでホスティングするための開発ロードマップ

このロードマップでは、コンテナ化されたゲームサーバー用の Amazon GameLift Servers マネージドホスティングソリューションを開発する方法を説明します。マネージドコンテナは、Amazon GameLift Servers が提供するホスティングソリューションの 1 つにすぎません。ホスティングオプションの詳細については、「[Amazon GameLift Servers ゲームホスティングオプション](#)」を参照してください。

Amazon GameLift Servers を使用したマネージドコンテナソリューションには、次のコンポーネントがあります。

- ゲームサーバービルドを含むコンテナイメージを、Amazon Elastic Container Registry (Amazon ECR) のプライベートリポジトリにアップロードします。ゲームサーバービルドは、サーバー SDK for Amazon GameLift Servers と統合され、Linux で動作するように構築されています。
- ゲームクライアントに代わって Amazon GameLift Servers サービスとやり取りするバックエンドサービス。バックエンドサービスは、AWS SDK の一部である Amazon GameLift Servers 用サービス API の機能を使用します。
- 新しい Amazon GameLift Servers ゲームセッションのリクエストを処理し、すべてのフリートで利用可能なサーバーを検索し、ゲームセッションを開始するようにゲームサーバーを促すゲームセッションキュー、またはその他の配置メカニズム。
- (オプション) マルチプレイヤーマッチを作成し、マッチのゲームセッションを立ち上げる FlexMatch マッチメーカー。
- Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを使用し、マルチプレイヤーゲームホスティング向けに最適化された 1 つ以上のコンテナフリート。

このロードマップは、コンテナ化されたゲームサーバーを Amazon GameLift Servers マネージドコンテナで正常に稼働させるための効率的な手順を示します。必要なコンポーネントが揃ったら、ゲーム開発を繰り返し、ホスティングソリューションをカスタマイズできます。ローンチが近づいたら、本番稼働レベルの使用に備えてホスティングソリューションを準備する方法について、「[Amazon GameLift Servers ホスティングによる起動の準備](#)」を参照してください。

i マネージドコンテナ用のこれらのツールを使用して、オンボーディングを高速化します。

- [コンテナスターターキット](#)は、統合とフリートのセットアップを合理化します。ゲームサーバーに重要なゲームセッション管理機能を追加し、事前設定されたテンプレートを 사용하여、ゲームサーバー用のコンテナフリートと自動デプロイパイプラインを構築します。デプロイ後、Amazon GameLift Servers コンソールと API ツールを使用して、フリートのパフォーマンスのモニタリング、ゲームセッションの管理、メトリクスの分析を行います。
- Unreal Engine および Unity の開発者は、[Amazon GameLift Servers プラグイン](#)を使用してゲームサーバーを統合し、ゲームエンジンの開発環境内からコンテナフリートを構築できます。プラグインのガイド付きワークフローは、マネージドコンテナを使用したクラウドベースのホスティングによる高速でシンプルなソリューションの作成に役立ちます。この基盤をもとに、ゲーム用のカスタムホスティングソリューションを作成できます。

ステップ 1: Amazon GameLift Servers を使用するためのゲームサーバーを準備する

ゲームサーバーに機能を追加して、ホスティング用にデプロイされたときに Amazon GameLift Servers サービスと通信できるようにします。

- ゲームプロジェクト用に SDK for Amazon GameLift Servers (バージョン 5.2 以上) を取得します。サーバー SDK は、C++、C#、Go で利用可能です。[サーバー SDK for Amazon GameLift Servers をダウンロードします](#)。サーバー SDK は、C++、C#、Go で利用可能です。
- ゲームサーバーコードを変更して、サーバーの SDK 機能を追加します。ガイダンスについては、「[Amazon GameLift Servers でホストするゲームを準備します](#)」を参照してください。少なくとも、以下を実行します。
 - Amazon GameLift Servers SDK を初期化するコードを追加し、Amazon GameLift Servers サービスとの WebSocket 接続を確立します。サーバー SDK アクション `InitSdk()` を使用します。
 - サーバード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 Servers サービスに報告するコードを追加します。サーバー SDK アクション `ProcessReady()` を使用します。
 - 必要なコールバック関数 `OnStartGameSession()`、および `OnProcessTerminate()` を実装します。これらの関数を使用することで、ゲームサーバードロセスは Amazon GameLift Servers サービスとの接続を維持し、Amazon GameLift Servers からの指示によりゲームセッションを開始し、ゲームサーバードロセスを終了するプロンプトに応答できます。
 - サーバードロセスがゲームセッションを終了することを Amazon GameLift Servers サービスに報告するコードを追加します。サーバー SDK アクション `ProcessEnding()` を使用します。
- ゲームサーバービルドをパッケージ化します。Linux で実行するようにゲームサーバーを構築します。ゲームサーバーの実行に必要なビルドやその他のファイルを準備します。Windows で開発している場合、このステップでは、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する場合があります。ゲームサーバーの構築をテストしたり、コンテナイメージを構築およびテストしたりするには、Linux 環境が必要です。
- ゲームサーバーの統合をテストします。統合されたゲームサーバーが Amazon GameLift Servers サービスに接続し、プロンプトに応答できることを確認します。「[Amazon GameLift Servers Anywhere でローカルテストを設定する](#)」で説明されているように、ローカルワークステーションをテストホストとしてシンプルな Amazon GameLift Servers Anywhere フリートを設定することをお勧めします。ゲームサーバービルドをテストホストにインストールし、サーバードロセスを開始します。AWS CLI を使用して新しいゲームセッションをリクエストし、Amazon GameLift Servers サービスがサーバードロセスにゲームセッションの開始を正常に促すことを確認します。

ステップ 2: ゲームサーバーコンテナイメージを作成する

ゲームサーバーを正常に統合したら、ゲームサーバー実行可能ファイルでコンテナイメージを作成します。Amazon GameLift Servers で使用するため、Amazon Elastic Container Registry (Amazon ECR) リポジトリに保存します。詳細な手順については、「[Amazon GameLift Servers 用のコンテナイメージを構築する](#)」を参照してください。

- ゲームサーバーコンテナ (Amazon GameLift Servers 提供) の Dockerfile テンプレートを取得します。ゲームサーバーのビルドファイルの ファイルを変更します。
- ゲームサーバーコンテナイメージを構築します。Linux 環境で作業するには、Docker ツールを使用してイメージを作成します。
- コンテナイメージを Amazon ECR 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 AWS リージョン 同じ と を使用して AWS アカウント、Amazon ECR にパブリックリポジトリまたはプライベートリポジトリを作成します。コンテナイメージをそこにプッシュします。
- 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 (オプション)。クラウドホスト型のコンテナフリートにデプロイする前に、コンテナイメージをローカルでテストすることもできます。テストには、既存の Amazon GameLift Servers Anywhere フリートをローカルワークステーションで使用できます。ゲームサーバーコンテナをインストールして実行し、次を確認します。
(1) Amazon GameLift Servers サービスがサーバープロセスにゲームセッション開始を正常に促すこと、(2) ゲームクライアントがゲームセッションに接続できること。

ステップ 3: クラウドクラウドベースのコンテナフリートを作成する

ここまでは、セルフマネージド Anywhere フリートを使用して、ゲームコンポーネントをテストして反復処理してきました。Amazon GameLift Servers 用に統合されたゲームサーバービルドが動作している場合は、本番環境に必要なクラウドベースの Amazon GameLift Servers マネージドコンテナフリートホスティングリソースのセットアップを開始できます。

- コンテナグループ定義の作成。コンテナグループ定義は、フリートのコンテナアーキテクチャを記述します。 と は、デプロイするコンテナイメージを識別します。「[コンテナフリートAmazon GameLift Servers用のコンテナグループ定義を作成する](#)」を参照してください。コンテナイメージが保存され AWS リージョン ているのと同じにコンテナグループ定義を作成します。フリートのロケーションの選び方については、「[地理的場所](#)」を参照してください。少なくとも、以下を実行します。
 - ゲームサーバーコンテナグループ定義を作成します。
 - ゲームサーバービルドでコンテナイメージを含むコンテナ定義を追加します。
 - コンテナのゲームサーバープロセスのポート範囲を設定します。

- マネージドコンテナフリートを作成します。フリートを作成すると、Amazon GameLift Servers はすぐにホスティング用にゲームサーバービルドのデプロイを開始します。マネージドフリートの多くの側面を設定できます。ガイダンスについては、「[Amazon GameLift Servers マネージドコンテナフリートを作成する](#)」を参照してください。少なくとも、以下を実行します。
- コンテナフリートの AWS Identity and Access Management (IAM) サービスロールを設定します。「[Amazon GameLift Servers 用に IAM サービスロールをセットアップする](#)」を参照してください。
- フリートインスタンスにデプロイするゲームサーバーコンテナグループ定義を指定します。
- 他のすべてのパラメータで利用できるデフォルト値を使用します。Amazon GameLift Servers は、最適な設定のためにいくつかのパラメータを計算します。
- コンテナフリートをキューに追加します。ゲームセッションキューで、Anywhere テストフリートをマネージドコンテナフリートで置き換えます。
- コンテナフリートでゲームホスティングをテストします。この時点で、ソリューション全体をテストできるはずですが、ゲームクライアントを起動し、バックエンドサービスを介してゲームセッションをリクエストします。接続情報を取得し、コンテナフリートのゲームセッションに接続します。
- フリートデプロイで反復処理します。コンテナグループ定義とフリート設定を更新し、既存のフリートに更新をデプロイできます。

ステップ 4: ホストされたゲームセッションに参加するゲームクライアントを準備する

ゲームクライアントがゲームセッションへの参加をリクエストし、接続情報を取得して、ホストされているゲームセッションに直接接続する方法を作成します。最も一般的なアプローチは、ゲームクライアントと Amazon GameLift Servers サービス間の仲介役として機能するバックエンドサービス機能をセットアップすることです。このアプローチにより、ホスティングリソースを保護し、プレイヤーがゲームセッションにどのように配置されるかをより詳細に制御することができます。

- ホスティング用のバックエンドサービス機能を構築します。バックエンドサービスは Amazon GameLift Servers サービスと通信し、ゲームクライアントに接続情報を送信します。この機能には、ゲームセッションの開始、ゲームへのプレイヤーの配置、ゲームセッション情報の取得が含まれます。ガイダンスについては、「[Amazon GameLift Serversでホストするゲームを準備します](#)」を参照してください。少なくとも、以下を実行します。
- AWS SDK for を取得しAmazon GameLift Servers、バックエンドサービスプロジェクトに追加します。「[クライアントサービス用の Amazon GameLift Servers SDK リソース](#)」を参照してください。

- Amazon GameLift Servers クライアントを初期化するコードを追加し、キー設定を保存します。
「[Amazon GameLift Servers APIをセットアップします。](#)」を参照してください。
- AWS SDK アクションを呼び出しCreateGameSession()、ゲームクライアントにゲームセッション接続情報を提供する機能を追加します。「[特定のフリートにゲームセッションを作成する](#)」を参照してください。

CreateGameSession() の呼び出しは、新しいゲームセッションをリクエストする際に便利な出発点となります。ゲームセッション配置システムが整ったら (ステップ 3 を参照)、このコードを StartGameSessionPlacement() への呼び出しに置き換えます (StartMatchmaking() を使用している場合は FlexMatch)。

バックエンドサービスを設計する方法については、「[Amazon GameLift Serversのバックエンドサービスを構築する](#)」を参照してください。

- プレイヤーがホストされているゲームセッションに参加できるようにする機能をゲームクライアントに追加します。ゲームクライアントは、Amazon GameLift Servers に直接リクエストするのではなく、バックエンドサービスにリクエストを行います。バックエンドサービスがゲームセッション接続情報を提供すると、ゲームクライアントはゲームセッションに直接接続してゲームをプレイします。
- ゲームクライアントの統合をテストします。テストには、既存の Amazon GameLift Servers Anywhere フリートをローカルワークステーションで使用できます。新しいバックエンドサービスを使用して新しいゲームセッションをリクエストし、(1) Amazon GameLift Servers サービスがサーバープロセスにゲームセッションの開始を正常に促し、(2) ゲームクライアントがゲームセッションに接続できることを確認します。

ステップ 5: ゲームセッション配置を設定する

Amazon GameLift Servers が新しいゲームセッションのリクエストをどのように処理し、ホスト可能なゲームサーバーをどのように特定するかをカスタマイズできます。Amazon GameLift Servers は、すべてのフリート内のゲームサーバーの可用性を自動的に追跡します。ゲームクライアントがゲームセッションへの参加リクエストを送信すると、Amazon GameLift Servers は最小レイテンシー、コスト、可用性などの定義された一連の優先順位に基づいて「可能な限り最良の」配置を探します。

- 利用可能なゲームサーバーで新しいゲームセッションを配置するためのゲームセッションキューを作成します。キューは、ゲームセッション配置の主要なメカニズムです。ガイダンスについては、「[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。
- 少なくとも、Anywhere フリートをキュー内の送信先として追加します。その他のすべての設定はオプションのカスタマイズです。

- バックエンドサービスコードで、**CreateGameSession()** 呼び出しを **StartGameSessionPlacement()** に変換します。「[マルチフリートキューにゲームセッションを作成する](#)」を参照してください。
- ゲームセッションに参加できる状態になったら、ゲームクライアントに通知するメカニズムを作成します。開発中は、DescribeGameSessionPlacement を呼び出してゲームセッションのステータスをポーリングできます。ただし、大量の処理にキューを使用する前に、イベント通知を有効にする必要があります。「[ゲームセッション配置のイベント通知を設定](#)」を参照してください。
- FlexMatch マッチメイキングを追加します (オプション)。マッチメイキングルールセットを構築し、ゲームセッションキューを操作するマッチメイキング設定を作成します。マッチメイキングシステムの設定に関するガイダンスについては、「[Amazon GameLift Servers FlexMatch デベロッパーガイド](#)」を参照してください。
- プレイメントシステムをテストします。テストには、既存の Amazon GameLift Servers Anywhere フリートをローカルワークステーションで使用できます。バックエンドサービスを使用して新しいゲームセッションをリクエストし、Amazon GameLift Servers サービスがサーバープロセスにゲームセッションの開始を正常に指示することを確認します。

ステップ 6: マネージドコンテナソリューションをカスタマイズする

ゲームのローンチに備えて、マネージドホスティングリソースをファインチューニングする必要があります。考慮すべき決定事項には、次のようなものがあります。

- コンテナフリートの設定を最適化します。「[Amazon GameLift Servers コンテナフリートをカスタマイズする](#)」を参照してください。
- コスト削減のためにスポットフリートの追加を検討します。「[スポットフリートによるゲームホスティングコストの削減](#)」を参照してください。
- ゲームサーバーが他の AWS リソースと通信する必要がある場合は、IAM ロールを設定してアクセスを管理します。「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」を参照してください。
- ゲームサーバーを地理的にどこに配置するかを決定します。マネージドフリートにリモートロケーションを追加します。「[ホスティングリソースのカスタマイズ](#)」を参照してください。
- 優先順位付け設定のカスタマイズなど、マネージドフリートのゲームセッション配置オプションを試してみます。「[ゲームセッションキューをカスタマイズする](#)」を参照してください。
- 予想されるプレイヤー需要に応えるための自動容量スケーリングを設定します。「[Amazon GameLift Servers を使用したゲームホスティング容量のスケーリング](#)」を参照してください。

- 他の フリートを作成し AWS リージョン、必要に応じてフェイルオーバーを処理するようにキューと自動スケーリングを変更します。
- 分析やログ記録など、ホスティングのオブザーバビリティツールをセットアップします。[Amazon GameLift Servers のモニタリング](#)を参照してください。
- [Infrastructure as Code \(IaC\)](#) を使用して、フリートのデプロイを自動化します。「[を使用して Amazon GameLift Servers ホスティングリソースを管理する CloudFormation](#)」を参照してください。

Amazon GameLift Servers では、デプロイ固有の設定に AWS CloudFormation テンプレートを使用できます。を使用して Amazon GameLift Servers リソース AWS Cloud Development Kit (AWS CDK) を定義することもできます。の詳細については AWS CDK、[AWS Cloud Development Kit \(AWS CDK\) デベロッパーガイド](#)を参照してください。

CloudFormation スタックのデプロイを管理するには、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 などのサービスを使用することをお勧めします AWS CodePipeline。こうしたツールにより、ゲームサーバーバイナリが構築されるたびに、自動的に、または承認によりデプロイできます。CI/CD ツールまたはサービスを使用すると、新しいゲームサーバーバージョンのリソースのデプロイは次のようになります。

- ゲームサーバーバイナリをビルドしてテストします。
- バイナリを Amazon GameLift Servers にアップロードします。
- 新しいビルドを使用して新しいフリートをデプロイします。
- ゲームセッションキューに新しいフリートを追加し、以前のビルドバージョンのフリートを削除する。
- 以前のビルドのフリートがアクティブなゲームセッションをホストしなくなったら、それらのフリートの CloudFormation スタックを削除します。

Amazon GameLift Servers Anywhere でホスティングするための開発ロードマップ

このロードマップでは、自前のリソース (オンプレミスのハードウェアまたは仮想マシン) を使用して、マルチプレイヤーゲーム用のホスティングソリューションを開発する方法について説明します。Amazon GameLift Servers は複数のゲームホスティングオプションを提供しています。これらのオプションの詳細については「[Amazon GameLift Servers ゲームホスティングオプション](#)」を参照してください。

Amazon GameLift Servers Anywhere ホスティングを使用すると、ゲームサーバーは、ユーザーが提供および管理するコンピューティングリソースでホスティングされます。Anywhere フリートは、必要な設定で作成し、プレイヤーの所在地に合わせて地理的に配置することができます。また、Amazon GameLift Servers は Anywhere フリート向けに次の機能を提供します。

- 設定に基づいてゲームセッションの配置プロセスを処理します。この処理には、以下が含まれます。
 - Anywhere フリート全体のゲームサーバーの可用性を追跡します。
 - ゲームクライアントサービスからのゲームリクエストを処理し、ゲームリクエストを利用可能なサーバーとマッチングさせます。
 - Anywhere フリートのゲームサーバーにゲームセッションの開始を指示します。
 - 接続の詳細をゲームクライアントに返します。
- セッション配置プロセスのパフォーマンスメトリクスとゲームセッションおよびプレイヤーの使用状況メトリクスを収集します。
- FlexMatch マッチメイキング機能をすべてサポートしているため、マッチメーカーを構築し、ゲームセッション配置システムと統合できます。
- Anywhere フリートでキーホスト管理タスクを処理する Amazon GameLift Servers エージェントを提供します。
- Amazon GameLift Servers マネージドフリートとの組み合わせをサポートし、柔軟なハイブリッドソリューションを実現します。

Amazon GameLift Servers Anywhere ソリューションは、次のコンポーネントで構成されています。

- サーバー SDK for Amazon GameLift Servers と統合された 1 つのゲームサーバービルド。すべてのフリートにデプロイします。
- AWS SDK と統合されたゲームクライアントとバックエンドサービス。Amazon GameLift Servers サービスとやり取りしてゲームセッションをリクエストします。
- すべてのフリートで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配置する Amazon GameLift Servers キュー。
- (オプション) マルチプレイヤーマッチを作成し、マッチのゲームセッションを立ち上げる FlexMatch マッチメーカー。
- オンプレミスまたはその他のホスティングリソースを持つ 1 つまたは複数の Amazon GameLift Servers Anywhere フリート。既存の設定管理およびデプロイツールで管理されます。(オプションで 使用できます) AWS Systems Manager。

このロードマップは、Amazon GameLift Servers Anywhere ホスティングを使用してマルチプレイヤーゲームを稼働させるための効率的な道筋を示しています。必要なコンポーネントが揃ったら、ゲーム開発を繰り返し、ホスティングソリューションをカスタマイズできます。ローンチが近づいたら、本番稼働レベルの使用に備えてホスティングソリューションを準備する方法について、「[Amazon GameLift Servers ホスティングによる起動の準備](#)」を参照してください。

- ① Unreal Engine と Unity 用の Amazon GameLift Servers プラグインでジャンプスタートするデプロイを高速化するには、Unreal Engine と Unity の [Amazon GameLift Servers プラグイン](#)を試してください。最小限のセットアップでゲームサーバーをすばやくデプロイするためのガイド付き UI ワークフローを提供するため、ゲームコンポーネントを実際に試すことができます。その後、この基盤をもとに、ゲーム用のカスタムホスティングソリューションを作成できます。詳細については、[Amazon GameLift Servers プラグインを使用して Unreal または Unity ゲームを準備する](#)を参照してください。

ステップ 1: Amazon GameLift Servers を使用するためのゲームサーバーを準備する

ゲームサーバーに機能を追加して、ホスティング用にデプロイされたときに Amazon GameLift Servers サービスと通信できるようにします。

- ゲームプロジェクト用に SDK for Amazon GameLift Servers (バージョン 5.x) を取得します。サーバー SDK は、C++、C#、Go で利用可能です。[Amazon GameLift Servers サーバー SDK をダウンロードします](#)。
- ゲームサーバーコードを変更して、サーバーの SDK 機能を追加します。ガイダンスについては、「[Amazon GameLift Servers でホストするゲームを準備します](#)」を参照してください。少なくとも、以下を実行します。
 - Amazon GameLift Servers SDK を初期化するコードを追加し、Amazon GameLift Servers サービスとの WebSocket 接続を確立します。サーバー SDK アクション `InitSdk()` を使用し、Anywhere フリートに必要なサーバーパラメータを含めます。
 - 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ことを Amazon GameLift Servers サービスに報告するコードを追加します。サーバー SDK アクション `ProcessReady()` を使用します。
 - 必要なコールバック関数 `OnProcessTerminate()`、および `OnStartGameSession()` を実装します。これらの関数を使用することで、ゲームサーバープロセスは Amazon GameLift Servers サービスとの接続を維持し、Amazon GameLift Servers からの指示によりゲームセッションを開始し、ゲームサーバープロセスを終了するプロンプトに応答できます。

- サーバードプロセスがゲームセッションを終了することを Amazon GameLift Servers サービスに報告するコードを追加します。サーバー SDK アクション `ProcessEnding()` を使用します。
- ゲームサーバービルドをパッケージ化します。ビルドファイル、依存関係、サポートソフトウェアを含むインストールスクリプトを作成します。「[ゲームビルドファイルをパッケージ化する](#)」を参照してください。Amazon Simple Storage Service (Amazon S3) バケットを使用して、ゲームビルドのバージョンを保存することをお勧めします。
- ゲームサーバーの統合をテストします。このタスクでは、「[Amazon GameLift Servers Anywhere でローカルテストを設定する](#)」で説明されているように、ローカルワークステーションで Amazon GameLift Servers Anywhere フリートをセットアップすることをお勧めします。このステップでは、ゲームサーバービルドをテストデバイスに手動でインストールし、サーバードプロセスを開始します。AWS CLI を使用して新しいゲームセッションをリクエストし、Amazon GameLift Servers サービスがサーバードプロセスにゲームセッションの開始を正常に指示することを確認します。

ステップ 2: ホストされているゲームセッションに参加するためのゲームクライアントを準備する

ゲームクライアントがゲームセッションへの参加をリクエストし、接続情報を取得して、ホストされているゲームセッションに直接接続する方法を作成します。最も一般的なアプローチは、ゲームクライアントと Amazon GameLift Servers サービス間の仲介役として機能するバックエンドサービス機能をセットアップすることです。このアプローチにより、ホスティングリソースを保護し、プレイヤーがゲームセッションにどのように配置されるかをより詳細に制御することができます。

- ホスティング用のバックエンドサービス機能を構築します。バックエンドサービスは Amazon GameLift Servers サービスと通信し、ゲームクライアントに接続情報を送信します。この機能には、ゲームセッションの開始、ゲームへのプレイヤーの配置、ゲームセッション情報の取得が含まれます。ガイダンスについては、「[Amazon GameLift Servers でホストするゲームを準備します](#)」を参照してください。少なくとも、以下を実行します。
- AWS SDK for Amazon GameLift Servers を取得し、バックエンドサービスプロジェクトに追加します。「[クライアントサービス用の Amazon GameLift Servers SDK リソース](#)」を参照してください。
- Amazon GameLift Servers クライアントを初期化するコードを追加し、キー設定を保存します。「[Amazon GameLift Servers API をセットアップします。](#)」を参照してください。
- AWS SDK アクションを呼び出し `CreateGameSession()`、ゲームクライアントにゲームセッション接続情報を提供する機能を追加します。「[特定のフリートにゲームセッションを作成する](#)」を参照してください。

CreateGameSession() の呼び出しは、新しいゲームセッションをリクエストする際に便利な出発点となります。ゲームセッション配置システムが整ったら (ステップ 3 を参照)、このコードを StartGameSessionPlacement() への呼び出しに置き換えます (StartMatchmaking() を使用している場合は FlexMatch)。

バックエンドサービスを設計する方法については、「[Amazon GameLift Serversのバックエンドサービスを構築する](#)」を参照してください。

- プレイヤーがホストされているゲームセッションに参加できるようにする機能をゲームクライアントに追加します。ゲームクライアントは、Amazon GameLift Servers に直接リクエストするのではなく、バックエンドサービスにリクエストを行います。バックエンドサービスがゲームセッション接続情報を提供すると、ゲームクライアントはゲームセッションに直接接続してゲームをプレイします。
- ゲームクライアントの統合をテストします。テストでは、ローカルワークステーションで同じ Amazon GameLift Servers Anywhere フリートを使用できます。

ステップ 3: ゲームセッション配置をセットアップする

Amazon GameLift Servers が新しいゲームセッションのリクエストをどのように処理し、ホスト可能なゲームサーバーをどのように特定するかをカスタマイズできます。Amazon GameLift Servers は、すべてのフリート内のゲームサーバーの可用性を自動的に追跡します。ゲームクライアントがゲームセッションへの参加リクエストを送信すると、Amazon GameLift Servers は最小レイテンシー、コスト、可用性などの定義された一連の優先順位に基づいて「可能な限り最良の」配置を探します。

- 利用可能なゲームサーバーで新しいゲームセッションを配置するためのゲームセッションキューを作成します。キューは、ゲームセッション配置の主要なメカニズムです。ガイダンスについては、「[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。
- 少なくとも、Anywhere フリートをキュー内の送信先として追加します。その他のすべての設定はオプションです。
- バックエンドサービスコードで、**CreateGameSession()** 呼び出しを **StartGameSessionPlacement()** に変換します。「[マルチフリートキューにゲームセッションを作成する](#)」を参照してください。
- ゲームセッションに参加できる状態になったら、ゲームクライアントに通知するメカニズムを作成します。開発中は、DescribeGameSessionPlacement を呼び出してゲームセッションのステータスをポーリングできます。ただし、大量の処理にキューを使用する前に、イベント通知を有効にする必要があります。「[ゲームセッション配置のイベント通知を設定](#)」を参照してください。

- (オプション) FlexMatch マッチメイキングコンポーネントを追加します。ガイドンスについては、「[Amazon GameLift Servers FlexMatch デベロッパーガイド](#)」を参照してください。

ステップ 4: Amazon GameLift Servers エージェントを使用して Anywhere フリートをセットアップする

ここまでは、Anywhere フリートのコンピューティングとして登録されたローカルデバイスを使用して、ゲームコンポーネントのテストと反復作業を行ってきました。次のステップでは、本番稼働システムに必要なフリートのタイプを設定します。これらのリソースでは、Amazon GameLift Servers エージェントを使用して、コンピューティング上で直接行われるいくつかの主要なホスト管理タスクを管理します。詳細については、[Amazon GameLift Servers エージェントの操作](#)を参照してください。

- Amazon GameLift Servers エージェントを取得し、ゲームサーバーのインストールパッケージに追加します。[Amazon GameLift Servers エージェント GitHub リポジトリ](#)にあるエージェントソースコードを取得してビルドします。ビルドした実行可能 JAR ファイルをゲームビルド実行可能ファイルと同じディレクトリに配置します。
- 必要に応じて、エージェントのスタートアップスクリプトを変更します。コンピューティングの実行が開始されるとすぐに、エージェント実行可能ファイルが起動することを確認します。ホスティングコンピューティングにエージェントをインストールして実行する方法については、エージェントのリポジトリにある readme ファイルを参照してください。起動コマンドには、少なくとも、Anywhere フリート ID と AWS リージョン、カスタムロケーション、コンピューティング名を指定するオプションを含める必要があります。

エージェントは、以下のタスクを自動的に処理するため、これらのタスクをスクリプトで処理している場合は、削除できます。

- `RegisterCompute()` を呼び出して、コンピューティングを Anywhere フリートに追加する。
- `GetComputeAuthToken()` を呼び出して、ゲームサーバーが Amazon GameLift Servers サービスに接続する際に認証を行う。エージェントは認証トークンの取得と更新を管理します。このトークンは、コンピューティングで実行されているすべてのゲームサーバープロセスで使用できます。
- 一連のランタイム手順に基づいて、コンピューティングで新しいサーバープロセスを開始する。
- Anywhere フリートでコンピューティングのランタイム設定を作成します。Amazon GameLift Servers コンソールまたは CLI AWS を使用して、フリートのランタイム手順を作成または変更できます。エージェントはこの手順を実行し、定期的に Amazon GameLift Servers サービスから更新をリクエストします。

- 必要に応じて、ゲームセッションキューを設定または変更します。Amazon GameLift Servers エージェントでデプロイされた Anywhere フリートを使用するための新しいキューを作成します (または既存のキューを更新します)。
- Anywhere フリートとのエージェント統合をテストします。エージェントがランタイム設定に基づいてサーバープロセスを適切に開始していることを確認します。

ステップ 5: Anywhere フリートをカスタマイズする

ゲームのローンチに備えて、マネージドホスティングリソースをファインチューニングする必要があります。考慮すべき決定事項には、次のようなものがあります。

- ゲームサーバーソフトウェアのインストールと実行など、必要に応じて、コンピューティングを開始およびシャットダウンするプロセスを自動化します。コンピューティングのリサイクルは、コンピューティングを定期的に更新するために役立ちます。また、必要のないときはコンピューティングをシャットダウンすることで、コストを削減できます。
- ゲームサーバーが他の AWS リソースと通信する必要がある場合は、IAM ロールを設定してアクセスを管理します。「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」を参照してください。
- ゲームサーバーを地理的にどこに配置するかを決定します。マネージドフリートにリモートロケーションを追加します。「[ホスティングリソースのカスタマイズ](#)」を参照してください。
- コンピューティングリソース設定を選択してフリートのパフォーマンスを最適化し、コンピューティングごとに最適な数のサーバープロセスを実行するようにランタイム手順を設定します。
- 優先順位付け設定のカスタマイズなど、マネージドフリートのゲームセッション配置オプションを試してみます。「[ゲームセッションキューをカスタマイズする](#)」を参照してください。
- 予想されるプレイヤー需要に応えるために、手動または自動で容量をスケーリングするメカニズムを構築します。ゲームセッションをホストするために使えるコンピューティングの数を増減させる要因として、どのようなものが考えられるか検討してください。
- 必要に応じて、他のリソースへのフェイルオーバーを設計して実装します。
- 分析やログ記録など、ホスティングのオブザーバビリティツールをセットアップします。「[Amazon GameLift Servers のモニタリング](#)」を参照してください。

Amazon GameLift Serversを利用するハイブリッドホスティングの開発ロードマップ

このロードマップでは、マルチプレイヤーゲームのホスティングソリューションを開発する方法について説明します。Amazon GameLift Servers には複数のゲームホスティングオプションがあります。これらの詳細については、[Amazon GameLift Servers ゲームホスティングオプション](#) を参照してください。

ハイブリッドソリューションでは、Amazon GameLift Servers が管理するクラウドベースのリソースと、ユーザーが管理するホスティングリソースを組み合わせで使用します。ハイブリッドホスティングの詳細については、「[Hybrid game server hosting with Amazon GameLift Servers Anywhere](#)」の記事を参照してください。Amazon GameLift Servers を使用すると、共通のコンポーネントとプロセスを利用したハイブリッドソリューションをセットアップでき、グローバルフリートを一元管理しながら、さまざまな種類のリソース間で負荷を容易に移動できます。

ハイブリッドアーキテクチャは、次のコンポーネントで構成されます。

- サーバー SDK for Amazon GameLift Servers と統合された 1 つのゲームサーバービルド。すべてのフリートにデプロイします。
- AWS SDK と統合された 1 つのゲームクライアントとバックエンドサービス。Amazon GameLift Servers サービスとやり取りし、ゲームセッションをリクエストします。
- 利用可能なゲームサーバーで新しいゲームセッションを配置し、すべてのフリート間で負荷を分散するための共有 Amazon GameLift Servers キュー。
- (オプション) Anywhere フリートにデプロイされるプロセスマネージャーエージェント。すべてのフリートのコンピューティング全体でサーバープロセス管理タスクを簡素化します。
- (オプション) マルチプレイヤーマッチを作成し、マッチのゲームセッションを立ち上げる FlexMatch マッチメーカー。
- マルチプレイヤーゲームホスティング用に最適化された Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを使用する 1 つまたは複数の Amazon GameLift Servers マネージドフリート。
- 構成管理やデプロイツールを含む、既存のオンプレミスまたは他のホスティングリソースを使用する、1 つ以上の Amazon GameLift Servers Anywhere フリート。(AWS Systems Managerを使用することもできます)。

このロードマップは、Amazon GameLift Servers を使用したハイブリッドホスティングソリューションでマルチプレイヤーゲームを稼働させるための効率的な道筋を示しています。必要なコンポーネントが揃ったら、ゲーム開発を繰り返し、ホスティングソリューションをカスタマイズできます。ロー

ンチが近づいたら、本番稼働レベルの使用に備えてホスティングソリューションを準備する方法について、「[Amazon GameLift Servers ホスティングによる起動の準備](#)」を参照してください。

Amazon GameLift Servers プラグインでジャンプスタートする

Unreal Engine または Unity でプロジェクトを開発している場合は、ゲームをホスティングするための設定を Amazon GameLift Servers プラグインで行うことができます。プラグインを使用すると、Amazon GameLift Servers SDK をゲームプロジェクトに追加し、ガイド付きワークフローを使用して Anywhere フリートと Amazon GameLift Servers マネージドフリートの両方を使用したハイブリッドホスティングソリューションの簡易版を構築できます。その後、これらの基本要素を基に構築し、必要に応じてカスタマイズできます。

ステップ 1: Amazon GameLift Servers を使用するためのゲームサーバーを準備する

ゲームサーバーに機能を追加して、ホスティング用にデプロイされたときに Amazon GameLift Servers サービスと通信できるようにします。ゲームサーバーが Amazon GameLift Servers マネージドフリートで実行されている場合でも、Anywhere フリートで実行されている場合でも、同じ機能が必要です。

- ゲームプロジェクト用に SDK for Amazon GameLift Servers (バージョン 5.x) を取得します。サーバー SDK は、C++、C#、Go で利用可能です。[サーバー SDK for Amazon GameLift Servers をダウンロードします](#)。
- ゲームサーバーコードを変更して、サーバーの SDK 機能を追加します。ガイダンスについては、「[Amazon GameLift Serversでホストするゲームを準備します](#)」を参照してください。少なくとも、以下を実行します。
 - Amazon GameLift Servers SDK を初期化するコードを追加し、Amazon GameLift Servers サービスとの WebSocket 接続を確立します。サーバー SDK アクション `InitSdk()` を使用します。Anywhere フリートコンピューティングで実行する際にサーバーパラメータを指定するコードを含めます。
 - サーバード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 Servers サービスに報告するコードを追加します。サーバー SDK アクション `ProcessReady()` を使用します。
 - 必要なコールバック関数 `OnProcessTerminate()`、および `OnStartGameSession()` を実装します。これらの関数を使用することで、ゲームサーバードプロセスは Amazon GameLift Servers サービスとの接続を維持し、Amazon GameLift Servers からの指示によりゲームセッションを開始し、ゲームサーバードプロセスを終了するプロンプトに応答できます。

- サーバードプロセスがゲームセッションを終了することを Amazon GameLift Servers サービスに報告するコードを追加します。サーバー SDK アクション `ProcessEnding()` を使用します。
- ゲームサーバービルドをパッケージ化します。ビルドファイル、依存関係、サポートソフトウェアを含むインストールスクリプトを作成します。「[ゲームビルドファイルをパッケージ化する](#)」を参照してください。Amazon Simple Storage Service (Amazon S3) バケットを使用して、ゲームビルドのバージョンを保存することをお勧めします。
- ゲームサーバーの統合をテストします。このタスクでは、「[Amazon GameLift Servers Anywhere でローカルテストを設定する](#)」で説明されているように、ローカルワークステーションで Amazon GameLift Servers フリートをセットアップすることをお勧めします。このステップでは、ゲームサーバービルドをテストデバイスに手動でインストールし、サーバードプロセスを開始します。AWS CLI を使用して新しいゲームセッションをリクエストし、Amazon GameLift Servers サービスがサーバードプロセスにゲームセッションの開始を正常に指示することを確認します。

ステップ 2: ホストされているゲームセッションに参加するためのゲームクライアントを準備する

ゲームクライアントがゲームセッションへの参加をリクエストし、接続情報を取得して、ホストされているゲームセッションに直接接続する方法を作成します。最も一般的なアプローチは、ゲームクライアントと Amazon GameLift Servers サービス間の仲介役として機能するバックエンドサービス機能をセットアップすることです。このアプローチにより、ホスティングリソースを保護し、プレイヤーがゲームセッションにどのように配置されるかをより詳細に制御することができます。

- ホスティング用のバックエンドサービス機能を構築します。バックエンドサービスは Amazon GameLift Servers サービスと通信し、ゲームクライアントに接続情報を送信します。この機能には、ゲームセッションの開始、ゲームへのプレイヤーの配置、ゲームセッション情報の取得が含まれます。ガイダンスについては、「[Amazon GameLift Servers でホストするゲームを準備します](#)」を参照してください。少なくとも、以下を実行します。
- AWS SDK for Amazon GameLift Servers を取得し、バックエンドサービスプロジェクトに追加します。「[クライアントサービス用の Amazon GameLift Servers SDK リソース](#)」を参照してください。
- Amazon GameLift Servers クライアントを初期化するコードを追加し、キー設定を保存します。「[Amazon GameLift Servers API をセットアップします。](#)」を参照してください。
- AWS SDK アクションを呼び出し `CreateGameSession()`、ゲームクライアントにゲームセッション接続情報を提供する機能を追加します。「[特定のフリートにゲームセッションを作成する](#)」を参照してください。

CreateGameSession() の呼び出しは、新しいゲームセッションをリクエストする際に便利な出発点となります。ゲームセッション配置システムが整ったら (ステップ 3 を参照)、このコードを StartGameSessionPlacement() への呼び出しに置き換えます (StartMatchmaking() を使用している場合は FlexMatch)。

バックエンドサービスを設計する方法については、「[Amazon GameLift Serversのバックエンドサービスを構築する](#)」を参照してください。

- プレイヤーがホストされているゲームセッションに参加できるようにする機能をゲームクライアントに追加します。ゲームクライアントは、Amazon GameLift Servers に直接リクエストするのではなく、バックエンドサービスにリクエストを行います。バックエンドサービスがゲームセッション接続情報を提供すると、ゲームクライアントはゲームセッションに直接接続してゲームをプレイします。
- ゲームクライアントの統合をテストします。テストでは、ローカルワークステーションで同じ Amazon GameLift Servers Anywhere フリートを使用できます。

開発段階で Amazon GameLift Servers マネージドフリートでのゲームビルドの動作をテストする場合は、[クラウドベース](#)のテスト環境も用意することをお勧めします。この Amazon GameLift Servers Toolkit ソリューションは、マネージドフリートの動作を模倣しますが、ゲームサーバーのビルドを最小限のターンアラウンドタイムで更新できます。

ステップ 3: ゲームセッション配置をセットアップする

Amazon GameLift Servers が新しいゲームセッションのリクエストをどのように処理し、ホスト可能なゲームサーバーをどのように特定するかをカスタマイズできます。Amazon GameLift Servers は、すべてのフリート内のゲームサーバーの可用性を自動的に追跡します。ゲームクライアントがゲームセッションへの参加リクエストを送信すると、Amazon GameLift Servers は最小レイテンシー、コスト、可用性などの定義された一連の優先順位に基づいて「可能な限り最良の」配置を探します。

- 利用可能なゲームサーバーで新しいゲームセッションを配置するためのゲームセッションキューを作成します。キューは、ゲームセッション配置の主要なメカニズムです。ガイダンスについては、「[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。
- 少なくとも、Anywhere フリートをキュー内の送信先として追加します。その他のすべての設定はオプションのカスタマイズです。
- バックエンドサービスコードで、**CreateGameSession()** 呼び出しを **StartGameSessionPlacement()** に変換します。「[マルチフリートキューにゲームセッションを作成する](#)」を参照してください。

- ゲームセッションに参加できる状態になったら、ゲームクライアントに通知するメカニズムを作成します。開発中は、DescribeGameSessionPlacement を呼び出してゲームセッションのステータスをポーリングできます。ただし、大量の処理にキューを使用する前に、イベント通知を有効にする必要があります。「[ゲームセッション配置のイベント通知を設定](#)」を参照してください。
- (オプション) FlexMatch マッチメイキングコンポーネントを追加します。ガイダンスについては、「[Amazon GameLift Servers FlexMatch デベロッパーガイド](#)」を参照してください。

ステップ 4: Amazon GameLift Servers エージェントを使用して Anywhere フリートをセットアップする

ここまでは、Anywhere フリートのコンピューティングとして登録されたローカルデバイスを使用して、ゲームコンポーネントのテストと反復作業を行ってきました。次のステップでは、本番稼働システムに必要なフリートのタイプを設定します。Anywhere フリートから開始し、コンピューティング上で直接行われるいくつかの主要なホスト管理タスクを管理するための Amazon GameLift Servers エージェントを追加します。詳細については、[Amazon GameLift Servers エージェントの操作](#)を参照してください。

- Amazon GameLift Servers エージェントを取得し、ゲームサーバーのインストールパッケージに追加します。[Amazon GameLift Servers エージェント GitHub リポジトリ](#)にあるエージェントソースコードを取得してビルドします。ビルドした実行可能 JAR ファイルをゲームビルド実行可能ファイルと同じディレクトリに配置します。
- 必要に応じて、エージェントのスタートアップスクリプトを変更します。コンピューティングの実行が開始されるとすぐに、エージェント実行可能ファイルが起動することを確認します。ホスティングコンピューティングにエージェントをインストールして実行する方法については、エージェントのリポジトリにある readme ファイルを参照してください。起動コマンドには、少なくとも Anywhere フリート ID と、AWS リージョンカスタムロケーション、およびコンピューティング名を指定するオプションが含まれている必要があります。

エージェントは、以下のタスクを自動的に処理するため、これらのタスクをスクリプトで処理している場合は、削除できます。

- RegisterCompute() を呼び出して、コンピューティングを Anywhere フリートに追加する。
- GetComputeAuthToken() を呼び出して、ゲームサーバーが Amazon GameLift Servers サービスに接続する際に認証を行う。エージェントは認証トークンの取得と更新を管理します。このトークンは、コンピューティングで実行されているすべてのゲームサーバープロセスで使用できます。
- 一連のランタイム手順に基づいて、コンピューティングで新しいサーバープロセスを開始する。

- Anywhere フリートでコンピューティングのランタイム設定を作成します。少なくとも、ゲームサーバー実行可能ファイルの起動パスを指定します。Amazon GameLift Servers コンソールまたは CLI AWS を使用して、フリートのランタイム手順を作成または変更できます。エージェントはこの手順を実行し、定期的に Amazon GameLift Servers サービスから更新をリクエストします。
- 必要に応じて、ゲームセッションキューを設定または変更します。新しいキューを作成するか、既存のキューを更新し、Amazon GameLift Servers エージェントでデプロイされた Anywhere フリートの送信先を指定します。
- Anywhere フリートとのエージェント統合をテストします。エージェントがランタイム設定に基づいてサーバープロセスを適切に開始していることを確認します。

ステップ 5: マネージドクラウドベースのフリートを作成する

Anywhere フリートを補完する Amazon GameLift Servers マネージド EC2 フリートを作成します。開発を高速化するためにステップ 2 でクラウドベースのテスト環境を立ち上げた場合は、ゲーム開発とテストのほとんどを完了した後にマネージドフリートを作成する計画を立てます。自動容量スケールリングなどの追加設定を設定およびテストするには、フルマネージドフリートが必要です。

- ゲームサーバーを構築し、そのビルドを Amazon GameLift Servers にアップロードします。ビルドファイル、依存関係、サポートソフトウェアを含むインストールスクリプトを作成します。Anywhere フリートとマネージドフリートの両方で同じビルドソフトウェアを使用できます。「[Amazon GameLift Servers のゲームサーバービルドを作成する](#)」を参照してください。コンソールまたは CLI Amazon GameLift Servers を使用して、ビルドを AWS にアップロードできます。

ビルドをアップロードする前に、どの AWS リージョン にマネージドフリートを作成するかを決めます。ビルドを同じリージョンにアップロードする必要があります。フリートのロケーションの選び方については、「[地理的場所](#)」を参照してください。

- マネージド EC2 フリートを作成します。Amazon GameLift Servers コンソールまたは CLI AWS を使用して、マネージドフリートを作成できます。フリートを作成すると、Amazon GameLift Servers はすぐにホスティング用にゲームサーバービルドのデプロイを開始します。マネージドフリートの多くの側面を設定できます。ガイダンスについては、「[Amazon GameLift Servers マネージド EC2 フリートを作成する](#)」を参照してください。少なくとも、以下を実行します。
- フリートに名前を付け、デプロイするアップロード済みのゲームビルドを指定します。
- フリートのオンデマンドインスタンスを選択し、フリートのロケーションで利用可能なインスタンスタイプを選択します。スポットフリートは有用なオプションですが、追加の設計と設定が必要です。

- Anywhere フリートで使ったのと同様の設定でランタイム設定を作成します。少なくとも、ゲームサーバー実行可能ファイルの起動パスを指定します。
- インバウンドトラフィックがゲームサーバーにアクセスできるように、ポート設定を指定します。
- マネージドフリートを共有ゲームセッションキューに追加します。ステップ 4 のキューを更新して、Amazon GameLift Servers エージェントでデプロイされたマネージドフリートと Anywhere フリートの両方の送信先を含めます。
- マネージドフリートでゲームホスティングをテストします。この時点で、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から、接続情報を取得して、ゲームセッションに正常に接続するまでのホスティングサイクル全体をテストできるはずです。

ステップ 6: フリートをカスタマイズする

ゲームのローンチに備えて、ホスティングソリューションをファインチューニングする必要があります。考慮すべき決定事項には、次のようなものがあります。

- Anywhere フリートの場合、ゲームサーバーソフトウェアのインストールと実行など、必要に応じてコンピューティングを開始およびシャットダウンするプロセスを自動化します。コンピューティングのリサイクルは、コンピューティングを定期的に更新するために役立ちます。また、必要のないときはコンピューティングをシャットダウンすることで、コストを削減できます。
- ゲームサーバーが他の AWS リソースと通信する必要がある場合は、IAM ロールを設定してアクセスを管理します。「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」を参照してください。
- ゲームサーバーを地理的にどこに配置するかを決定します。マネージドフリートにリモートロケーションを追加します。「[ホスティングリソースのカスタマイズ](#)」を参照してください。
- マネージドフリートの場合は、コスト削減のためにスポットフリートの使用を検討します。「[スポットフリートによるゲームホスティングコストの削減](#)」を参照してください。
- コンピューティングリソース設定を選択してフリートのパフォーマンスを最適化し、コンピューティングごとに最適な数のサーバープロセスを実行するようにランタイム手順を設定します。Anywhere フリートとマネージドフリートの両方でこれを行います。「[マネージド Amazon GameLift Servers でのゲームサーバーランタイム設定の最適化](#)」を参照してください。
- 優先順位付け設定のカスタマイズなど、マネージドフリートのゲームセッション配置オプションを試してみます。「[ゲームセッションキューをカスタマイズする](#)」を参照してください。

- マネージドフリートの場合、予想されるプレイヤー需要に応えるための自動容量スケーリングを設定します。「[Amazon GameLift Servers を使用したゲームホスティング容量のスケーリング](#)」を参照してください。
- Anywhere フリートの場合、予想されるプレイヤー需要に応えるために、手動または自動で容量をスケーリングするメカニズムを構築します。
- 必要に応じて、他のリソースへのフェイルオーバーを設計して実装します。他の AWS リージョンにスタンバイフリートを用意し、必要に応じてフェイルオーバーを処理できるようにキューと自動スケーリングを変更します。
- 分析やログ記録など、ホスティングのオブザーバビリティツールをセットアップします。「[Amazon GameLift Servers のモニタリング](#)」を参照してください。すべてのホスティングリソースの分析を集約するメトリクスグループを作成します。
- [Infrastructure as Code \(IaC\)](#) を使用してデプロイを自動化します。「[を使用してAmazon GameLift Serversホスティングリソースを管理する CloudFormation](#)」を参照してください。

Amazon GameLift Servers では、デプロイ固有の設定に AWS CloudFormation テンプレートを使用できます。を使用してAmazon GameLift Serversリソース AWS Cloud Development Kit (AWS CDK) を定義することもできます。の詳細については AWS CDK、「[AWS Cloud Development Kit \(AWS CDK\) デベロッパーガイド](#)」を参照してください。

CloudFormation スタックのデプロイを管理するには、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) ツールや などのサービスを使用することをお勧めします AWS CodePipeline。こうしたツールにより、ゲームサーバーバイナリが構築されるたびに、自動的に、または承認によりデプロイできます。CI/CD ツールまたはサービスを使用すると、新しいゲームサーバーバージョンのリソースのデプロイは次のようになります。

- ゲームサーバーバイナリをビルドしてテストします。
- バイナリを Amazon GameLift Servers にアップロードします。
- 新しいビルドを使用して新しいフリートをデプロイします。
- ゲームセッションキューに新しいフリートを追加し、以前のビルドバージョンのフリートを削除する。
- 以前のビルドのフリートがアクティブなゲームセッションをホスティングしなくなった場合は、そうしたフリートの CloudFormation スタックを削除します。

Amazon GameLift Serversでホストするゲームを準備します

Amazon GameLift Serversでホスティングするためのゲームをデプロイする前に、ゲームサーバーとゲームクライアントコンポーネントの両方を準備する必要があります。これには、ゲームサーバーをAmazon GameLift Servers Server SDK と統合し、プレイヤー認証とゲームセッション管理を処理するバックエンドサービスを構築することが含まれます。

Server SDK 統合により、ゲームサーバーはAmazon GameLift Serversサービスと通信し、ヘルスステータスを報告し、ゲームセッションを管理できます。バックエンドサービスは、ゲームクライアントとAmazon GameLift Serversサービスの間の仲介として機能し、プレイヤーのリクエストを処理し、ゲームセッションの配置を調整します。

トピック

- [Amazon GameLift Servers プラグインを使用して Unreal または Unity ゲームを準備する](#)
- [Amazon GameLift Servers でゲームサーバーの統合](#)
- [デプロイ用にゲームサーバービルドをパッケージ化する](#)
- [Amazon GameLift Servers とゲームクライアント / サーバーのやり取り](#)

Amazon GameLift Servers プラグインを使用して Unreal または Unity ゲームを準備する

Amazon GameLift Servers プラグインは、Unreal または Unity ゲームエンジンへのフル機能のアドオンです。ここでは、Amazon GameLift Servers でゲームをホスティングするための基本的な手順について説明します。プラグインのツールセットとワークフローを使用すると、ゲームエンジン開発環境内でゲームサーバーをホスティングする準備、テスト用のローカルマシンでのホスティングのセットアップ、シンプルなバックエンドサービスの作成、ゲームサーバーを マネージドクラウドベースのホスティングにデプロイすることができます。

プラグインを使用すると Amazon GameLift Servers での作業が体験でき、ゲームホスティングソリューションをすぐに起動して実行できます。サンプルゲームアセットまたは独自のゲームプロジェクトを使用して作業をすることができます。プラグインは、シンプルな作業ソリューションを迅速に構築できるように、いくつかの手順を自動化します。プラグインのガイド付きワークフローを完了すると、Amazon GameLift Servers を介してゲームクライアントをライブでホストされたゲームセッションに接続できます。プラグインを使用してシンプルなホスティングソリューションを作成したら、ゲームのニーズに合わせてソリューションをカスタマイズできます。

プラグインは、次のゲームエンジンで使用できます。

- Unreal Engine
- Unity

プラグインには、ゲームエンジンごとに以下のコンポーネントが含まれています。

- ゲームエンジンエディタ用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から Amazon GameLift Servers 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 Servers サービス API 用ライブラリ。
- Amazon GameLift Servers サーバー SDK (バージョン 5) 用ライブラリ。
- サーバー統合のテストに使用するサンプルアセット。
- ゲームサーバーソリューションを定義する CloudFormation テンプレート形式の編集可能な設定。

トピック

- [プラグインワークフロー](#)
- [Unreal Engine 用 Amazon GameLift Servers プラグイン](#)
- [Unity 用 Amazon GameLift Servers プラグイン \(サーバー SDK 5.x\)](#)
- [サーバー SDK 4 向け Unity 用 Amazon GameLift Servers プラグイン](#)

プラグインワークフロー

次の手順では、Amazon GameLift Servers でゲームプロジェクトを準備してデプロイするための一般的なパスについて説明します。これらの手順を完了するには、ゲームエンジンエディタとゲームコードを使って作業します。

1. AWS アカウントユーザーにリンクし、 を使用するアクセス許可を持つ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 Amazon GameLift Servers。
2. プラグインがホスティングソリューションで使用する関連 AWS リソースを設定します (「ブートストラップ」と呼ばれます)。
3. サーバーコードをプロジェクトに追加して、実行中のゲームサーバーと Amazon GameLift Servers サービス間の通信を確立します。
4. ゲームクライアントが Amazon GameLift Servers にリクエストを送信して新しいゲームセッションを開始し、接続できるようにするクライアントコードをプロジェクトに追加します。

5. Anywhere ワークフローを使用して、Anywhere コンピューティングとしてローカルワークステーションに設定し、ゲームサーバーをホストします。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。
6. マネージド EC2 ワークフローを使用して、ゲームサーバーを Amazon GameLift Servers にアップロードし、シンプルかつ完全な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。プラグインを使用してローカルでゲームクライアントを起動し、ゲームセッションを要求したら、それ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場合、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](#)の対象となる場合があります。

Unreal Engine 用 Amazon GameLift Servers プラグイン

このプラグインは、Amazon GameLift Servers C++ サーバー SDK とツールを UE エディタに追加します。ガイド付き UI ワークフローを使用して、サーバー SDK 機能をゲームプロジェクトに統合し、ゲームサーバー用の Amazon GameLift Servers ホスティングソリューションをデプロイします。

プラグインを使用すると、基本的なホスティング環境を構築し、必要に応じて最適化やカスタマイズができます。ローカルワークステーションをホストとして、Amazon GameLift Servers Anywhere フリートをセットアップします。マネージド EC2 またはマネージドコンテナフリートによるクラウドホスティングの場合は、ゲームセッションリクエストとクライアント接続を管理するための完全なソリューションでゲームサーバーをデプロイします。

トピック

- [Unreal ゲームプロジェクト用プラグインをインストールします。](#)
- [次のステップ: ゲームホスティングソリューションをカスタマイズする](#)
- [Unreal 用プラグイン: AWS ユーザープロファイルを設定する](#)
- [Unreal 用プラグイン: ゲームコードを統合](#)
- [Unreal 用プラグイン: Amazon GameLift Servers Anywhere でゲームをローカルでホストする](#)
- [Unreal 用プラグイン: ゲームをマネージド EC2 フリートにデプロイする](#)
- [Unreal 用プラグイン: ゲームをマネージドコンテナフリートにデプロイする](#)

Unreal ゲームプロジェクト用プラグインをインストールします。

[GitHub から Unreal Engine 用 Amazon GameLift Servers プラグインを取得する](#)

ゲームプロジェクトの Unreal Editor にプラグインをインストールする方法については、GitHub リポジトリの readme を参照してください。

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から Amazon GameLift Servers 機能にアクセスできるようになります。
- Amazon GameLift Servers サービス API の C++ ライブラリ。クライアント側のバックエンドサービスで API 機能を使用して、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、ゲームセッション情報を送受信できるようにします。
- Amazon GameLift Servers サーバー SDK (バージョン 5) 用の Unreal ライブラリ。ゲームサーバーコードでサーバー SDK を使用して、ホストされたゲームサーバープロセスと Amazon GameLift Servers サービス間の通信を管理します。
- 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ときに使用する CloudFormation テンプレート形式の編集可能な設定。

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用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、ゲームに合わせて必要に応じてカスタマイズします。

次のステップ: ゲームホスティングソリューションをカスタマイズする

プラグインのガイド付きワークフローを使用すると、Amazon GameLift Servers ホスティングソリューションを迅速に起動して実行できます。プラグインを使用すると、ソリューションの各コンポーネントの基本的なバージョンを設定できます。

準備ができれば、各コンポーネントをカスタマイズし、ゲームの起動に備えてソリューションをファインチューニングすることで、この基本的なソリューションを構築できます。以下のオプションを検討してください。

- フリートとフリート設定を変更します。「[ホスティングリソースのカスタマイズ](#)」を参照してください。

- ゲームセッションキューの設定をカスタマイズします。「[ゲームセッションキューをカスタマイズする](#)」を参照してください。
- ゲームサーバーとゲームクライアントに機能を追加します。「[Amazon GameLift Servers でゲームサーバーの統合](#)」および「[Amazon GameLift Servers ゲームクライアント機能の統合する](#)」を参照してください。
- バックエンドサービスをカスタマイズします。「[Amazon GameLift Serversのバックエンドサービスを構築する](#)」を参照してください。
- 予想されるプレイヤー需要に応えるための自動容量スケーリングを設定します。「[Amazon GameLift Servers を使用したゲームホスティング容量のスケーリング](#)」を参照してください。
- 分析やログ記録など、ホスティングのオブザーバビリティツールをセットアップします。「[Amazon GameLift Servers のモニタリング](#)」を参照してください。
- [Infrastructure as Code \(IaC\)](#) を使用してデプロイを自動化します。マネージドソリューション用のプラグインのガイド付きワークフローでは、AWS CloudFormation テンプレートを使用します。必要に応じてロールをカスタマイズできます。「[を使用してAmazon GameLift Serversホスティングリソースを管理する CloudFormation](#)」を参照してください。

トピック

- [Unreal 用プラグイン: AWS ユーザープロファイルを設定する](#)
- [Unreal 用プラグイン: ゲームコードを統合](#)
- [Unreal 用プラグイン: Amazon GameLift Servers Anywhere でゲームをローカルでホストする](#)
- [Unreal 用プラグイン: ゲームをマネージド EC2 フリートにデプロイする](#)
- [Unreal 用プラグイン: ゲームをマネージドコンテナフリートにデプロイする](#)

Unreal 用プラグイン: AWS ユーザープロファイルを設定する

プラグインをインストールしたら、有効な AWS アカウントでユーザープロファイルを設定します。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 つだけです。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。各ワークフローページには、現在選択されているプロファイルが表示されます。

複数のデプロイを管理することで、異なるホスティングデプロイ間で切り替えることができます。例えば、同じ AWS アカウントを使用しても、異なる AWS リージョンにデプロイするプロファイルを設定できます。または、異なる AWS アカウントまたはユーザーとアクセス許可セットを使用して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、プロファイルが既に設定されている場合、Amazon GameLift Servers 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します。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。

アカウントユーザーで必要な AWS リソースをセットアップするには、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。

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 Servers メニューを選択し、[AWS アクセス認証情報] を選択します。このアクションにより、Amazon GameLift Servers プラグインで [ユーザープロファイルの設定] ページが開きます。
2. ボタンを使用して、新しい AWS アカウントを作成するか、既に持っている AWS アカウントのユーザープロファイルを設定します。
3. ユーザープロファイルがまだない場合は、プロファイルの詳細を入力し、新しいプロファイルを作成するように求められます。以下の情報を指定します。
 - AWS アカウント。新しい AWS アカウントを作成する場合は、へのリンクを使用してプロンプト AWS マネジメントコンソール に従います。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 Servers およびその他の必要な AWS サービスを使用するアクセス許可を持つ AWS ユーザー。長期認証情報を使用してアクセス Amazon GameLift Servers 許可と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[AWS ユーザーアカウントを設定する](#)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 と AWS シークレットキーで構成されます。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 region。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リージョンのリストから AWS リージョン](#)を選択します。
4. プラグインが既存のプロファイルを検出すると、使用可能なプロファイルのリストが表示されます。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。

ユーザープロファイルのブートストラップ

Amazon GameLift Servers 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、その他の依存関係の保存に使用されます。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 リソースの作成が含まれており、コストが発生する可能性があります。

プロファイルをブートストラップするには

1. [AWS アクセス認証情報] ページで、使用するユーザープロファイルのブートストラップステータスを確認します。プロファイルのブートストラップステータスが「非アクティブ」で、S3 バケットがリストされていない場合は、プロファイルをブートストラップする必要があります。
2. 使用するプロファイルを選択し、[ブートストラッププロファイル] を選択します。
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。

Unreal 用プラグイン: ゲームコードを統合

ゲームサーバーをフリートにデプロイする前に、ゲームコードに一連の更新を行い、Amazon GameLift Servers サービスで使用するゲームコンポーネントをパッケージ化する必要があります。

このトピックでは、最小限の統合を実行する手順について説明します。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。

- [ビルドターゲットとモジュールルールを設定する](#)
- [ゲームサーバーコードを更新する](#)
- [クライアントゲームマップを統合する](#)
- [ゲームコンポーネントをパッケージ化する](#)

ビルドターゲットとモジュールルールを設定する

ゲームプロジェクトファイルを変更して、Amazon GameLift Servers で使用するビルドコンポーネントを適切に生成します。

クライアントとサーバーのビルドターゲットを追加するには:

1. ゲームプロジェクトのコードファイルを開き、ファイル `.../Games/[your application name]Source/[your application name]Target.cs` ファイルを見つけます。例: `.../Source/GameLiftUnrealAppTarget.cs`。(Visual Studio を使用する場合は、プロジェクトの `.sln` ファイルを開きます。)
 2. このファイルをコピーして、Source/ ディレクトリに 2 つの新しいターゲットファイルを作成します。
- クライアントターゲット - 新しいファイルの名前を `[your application name]Client.Target.cs` に変更します。次のサンプルコードに示すように、コンテンツを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。

```
using UnrealBuildTool;
using System.Collections.Generic;

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

- サーバーターゲット - 新しいファイルの名前を `[your application name]Server.Target.cs` に変更します。次のサンプルコードに示すように、コンテンツを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。

```
using UnrealBuildTool;
using System.Collections.Generic;

public class GameLiftUnrealAppServerTarget : TargetRules
{
    public GameLiftUnrealAppServerTarget ( TargetInfo Target ) : base
    ( Target )
    {
        Type = TargetType.Server;
    }
}
```

```
        DefaultBuildSettings = BuildSettingsVersion.V2;  
        IncludeOrderVersion = EngineIncludeOrderVersion.Unreal5_1;  
        ExtraModuleNames.Add( "GameLiftUnrealApp");  
    }  
}
```

3. プロジェクトファイルを再生成します。Visual Studio を使用している場合は、ゲームプロジェクトの .uproject ファイルを右クリックし、[Visual Studio プロジェクトファイルの生成] を選択します。

ゲームプロジェクトモジュールルールを更新するには:

ゲームプロジェクトのモジュールルールを更新して、プラグインに依存します。

1. ゲームプロジェクトのコードファイルを開き、ファイル .../Games/[*your application name*]Source/[*your application name*].Build.cs ファイルを見つけます。例えば、.../Source/GameLiftUnrealApp.Build.cs などです。(Visual Studio を使用する場合は、プロジェクトの .sln ファイルを開きます。)
2. 次のサンプルコードに示すように、ModuleRules クラスを見つけて更新します。

```
using UnrealBuildTool;  
  
public class GameLiftUnrealApp : ModuleRules  
{  
    public GameLiftUnrealApp ( ReadOnlyTargetRules Target ) : base ( Target )  
    {  
        PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;  
        PublicDependencyModuleNames.AddRange( new string[] { "Core",  
            "CoreUObject", "Engine", "InputCore", "HeadMountedDisplay",  
            "EnhancedInput" });  
        // Add the following section  
        if (Target.Type == TargetType.Server)  
        {  
            PublicDependencyModuleNames.Add("GameLiftServerSDK");  
        }  
        else  
        {  
            PublicDefinitions.Add("WITH_GAMELIFT=0");  
        }  
        bEnableExceptions = true;  
    }  
}
```

```
}
```

3. 新しいターゲットファイルを作成し、モジュールルールを変更したら、ゲームプロジェクトを再構築します。

ゲームサーバーコードを更新する

ゲームサーバーコードを更新して、ゲームサーバープロセスと Amazon GameLift Servers サービス間の通信を有効にします。ゲームサーバーは、新しいゲームセッションを開始および停止するなど、Amazon GameLift Servers からのリクエストに応答できる必要があります。

Amazon GameLift Servers のサーバーコードを追加するには

1. Code Editor 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。

gameMode.h コードの例

```
// 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 "GameLiftUnrealAppGameMode.generated.h"

struct FProcessParameters;

DECLARE_LOG_CATEGORY_EXTERN(GameServerLog, Log, All);

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

```
GENERATED_BODY()

public:
    AGameLiftUnrealAppGameMode();

protected:
    virtual void BeginPlay() override;

private:
    void InitGameLift();

private:
    TSharedPtr<FProcessParameters> ProcessParameters;
};
```

- 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。

以下のコード例は、Amazon GameLift Servers 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。

- Amazon GameLift Servers API クライアントを初期化します。Amazon GameLift Servers Anywhere フリートには、サーバーパラメータを指定した InitSDK() コールが必要です。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、onHealthCheck など、Amazon GameLift Servers サービスからのリクエストに応答する必須の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、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 Servers サービスに通知します。

ゲームサーバーコードの例

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```



```
#include "GameLiftUnrealAppGameMode.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);

AGameLiftUnrealAppGameMode::AGameLiftUnrealAppGameMode() :
    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 AGameLiftUnrealAppGameMode..."));
}

void AGameLiftUnrealAppGameMode::BeginPlay()
{
    Super::BeginPlay();

#if WITH_GAMELIFT
    InitGameLift();
#endif
}

void AGameLiftUnrealAppGameMode::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.
FString TimeString = FString::FromInt(std::time(nullptr));
FString ProcessId = "ProcessId_" + TimeString;
ServerParametersForAnywhere.m_processId = TCHAR_TO_UTF8(*ProcessId);
}

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);
}

FString glAnywhereAwsRegion = "";
if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereAwsRegion="),
glAnywhereAwsRegion))
{
    ServerParametersForAnywhere.m_awsRegion =
TCHAR_TO_UTF8(*glAnywhereAwsRegion);
}

FString glAnywhereAccessKey = "";
if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereAccessKey="),
glAnywhereAccessKey))
{
    ServerParametersForAnywhere.m_accessKey =
TCHAR_TO_UTF8(*glAnywhereAccessKey);
}

FString glAnywhereSecretKey = "";
if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereSecretKey="),
glAnywhereSecretKey))
{
    ServerParametersForAnywhere.m_secretKey =
TCHAR_TO_UTF8(*glAnywhereSecretKey);
}
```

```

    }

    FString glAnywhereSessionToken = "";
    if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereSessionToken="),
glAnywhereSessionToken))
    {
        ServerParametersForAnywhere.m_sessionToken =
TCHAR_TO_UTF8(*glAnywhereSessionToken);
    }

    UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_YELLOW);
    UE_LOG(GameServerLog, Log, TEXT(">>>> WebSocket 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, Log, TEXT(">>>> Aws Region: %s"),
*ServerParametersForAnywhere.m_awsRegion);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Access Key: %s"),
*ServerParametersForAnywhere.m_accessKey);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Secret Key: %s"),
*ServerParametersForAnywhere.m_secretKey);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Session Token: %s"),
*ServerParametersForAnywhere.m_sessionToken);
    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, Amazon GameLift Servers 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. Amazon GameLift Servers 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 Amazon GameLift Servers we are indeed going to
shutdown.
ProcessParameters->OnTerminate.BindLambda([=]()
{
    UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
    // First call ProcessEnding()
    FGameLiftGenericOutcome processEndingOutcome = GameLiftSdkModule-
>ProcessEnding();
    // Then call Destroy() to free the SDK from memory
    FGameLiftGenericOutcome destroyOutcome = GameLiftSdkModule->Destroy();
    // Exit the process with success or failure
    if (processEndingOutcome.IsSuccess() && destroyOutcome.IsSuccess()) {
        UE_LOG(GameServerLog, Log, TEXT("Server process ending successfully"));
    }
});

```

```

    }
    else {
        if (!processEndingOutcome.IsSuccess()) {
            const FGameLiftError& error = processEndingOutcome.GetError();
            UE_LOG(GameServerLog, Error, TEXT("ProcessEnding() failed. Error:
%s"),
                error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
                *error.m_errorMessage);
        }
        if (!destroyOutcome.IsSuccess()) {
            const FGameLiftError& error = destroyOutcome.GetError();
            UE_LOG(GameServerLog, Error, TEXT("Destroy() failed. Error: %s"),
                error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
                *error.m_errorMessage);
        }
    }
});

//This is the HealthCheck callback.
//Amazon GameLift Servers 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. Amazon GameLift
Servers 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 = FString::Atoi(*Value);
            }
        }
    }

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

    //The game server calls ProcessReady() to tell Amazon GameLift Servers 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えます。プラグインのデプロイワークフローをテストしたり、ゲーム用のカスタムバックエンドサービスを作成したりするためのガイドとして使えます。

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。
- クライアントの機能には、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()`)、ゲームセッションへ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)。

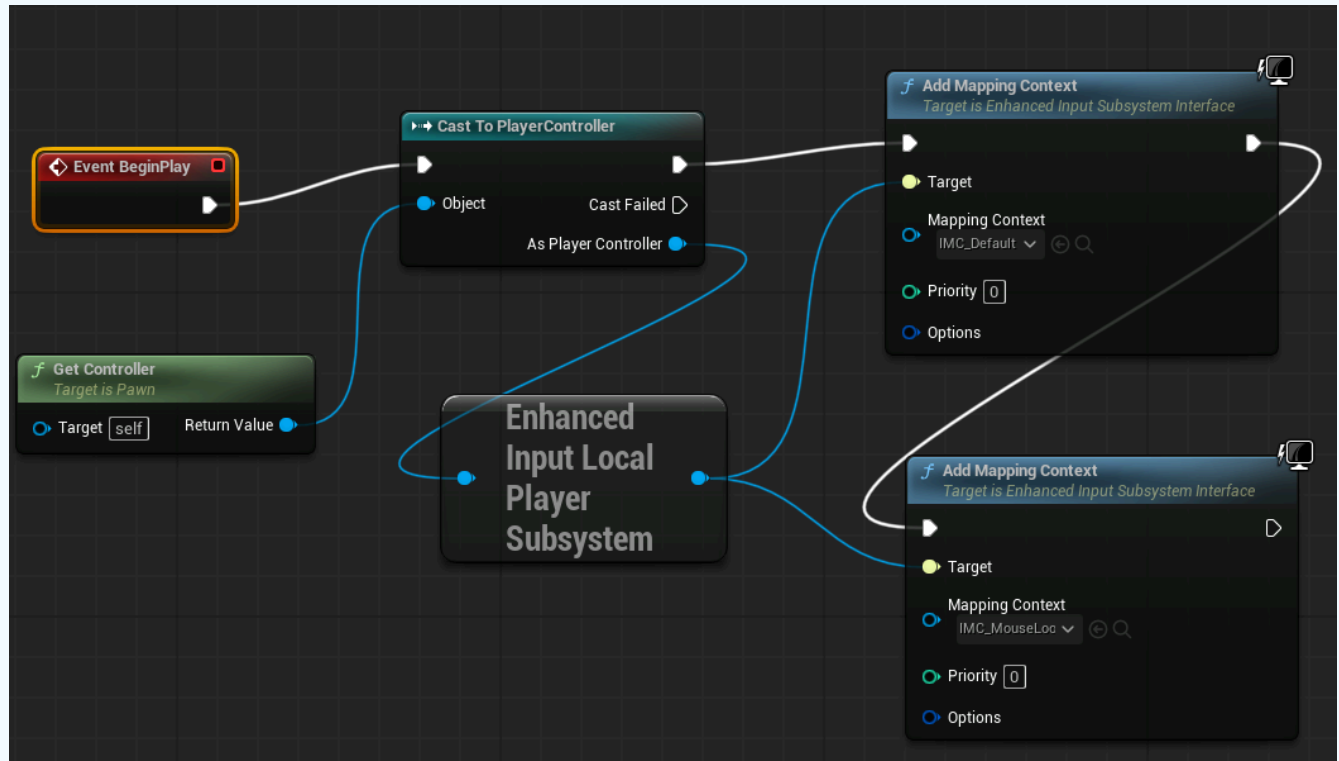
スタートアップゲームマップを使用するには

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. [エディタのスタートアップマップ] では、ドロップダウンリストで [StartupMap] を選択します。... > Unreal Projects/[project-name]/Plugins/Amazon GameLift Servers Plugin Content/Maps にあるファイルを検索しなければなら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で [StartupMap] を選択します。
4. サーバーデフォルトマップでは、Unreal Engine 5.6 以降では「Lv1_ThirdPerson」、以前のバージョンでは「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 のオーバーライド] を「なし」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ビルドできるようになります。

Note

Unreal Engine 5.6 以降では、ゲームサーバーに接続した後にキャラクターを移動できない場合は、BP_ThirdPersonCharacter ブループリントを更新して、以下IMC_MouseLookに示すように IMC_Defaultおよびの入カマッピングコンテキストを追加します。



ゲームコンポーネントをパッケージ化する

ゲームサーバーとゲームクライアントビルドをパッケージ化する

1. 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。
2. Unreal Engine 5.6 以降を使用している場合は、編集、プロジェクト設定、パッケージングを参照してください。プロジェクトコンテンツディレクトリですべてクックを見つけて有効にします。
3. ゲームサーバーとゲームクライアントビルドをパッケージ化します。
 - 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。

Unreal 用プラグイン: Amazon GameLift Servers Anywhere でゲームをローカルでホストする

Anywhere フリートをを使用して、ローカルワークステーションをゲームサーバーのホストとして設定します。これを使用して、クラウドベースのマネージドフリートにデプロイする前にゲームサーバーの統合をテストできます。また、反復ゲーム開発中のローカルテストにも役立ちます。

Amazon GameLift Servers 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 Servers メニューを選択し、[Anywhere でホストする] 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。作成する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。

ユーザープロファイルを設定する

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。プロファイルはまだ作成していない場合、または新しいプロファイルを作成したい場合は、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] ではない場合、[ブートストラッププロファイル] を選択して、ステータスが [アクティブ] になるまで待機します。

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、Amazon GameLift Servers と連携するようにゲームサーバーとゲームクライアントビルドを準備します。ゲームコードをまだ統合していない場合は、「[Unreal 用プラグイン: ゲームコードを統合](#)」を参照してください。ローカルワークステーションのゲーム実行可能ファイルへのパスを入力します。

- **ゲームサーバー:** ゲームサーバーをサーバー SDK for Amazon GameLift Servers と統合し、ゲームサーバービルドをパッケージ化します。手順については、「[Unreal 用プラグイン: ゲームコードを統合](#)」を参照してください。Amazon GameLift Servers サービスとの通信を確立し、プロンプトに応答して新しいゲームセッションを開始し、ゲームクライアント接続を受け入れるには、ゲームサーバーをサーバー SDK と統合する必要があります。
- **ゲームクライアント:** 少なくとも、IP アドレスとポート情報を使用してゲームサーバーに接続できるゲームクライアントが必要です。ゲームクライアントコンポーネントを用にまだ設定していない場合は Amazon GameLift Servers、AWS CLI ツールを使用して新しいゲームセッションを手動でリクエストし、接続情報を取得し、その情報を使用してゲームクライアントを接続できます。

ある時点で、新しいゲームセッションリクエストを Amazon GameLift Servers サービスに送信し、接続情報をゲームクライアントに中継するためのバックエンドサービスが必要です。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 Servers 機能をゲームプロジェクトに追加します。カスタムソリューションの構築については、「[Amazon GameLift Servers ゲームクライアント機能の統合する](#)」を参照してください。

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] 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 GameLift Servers はフリートのステータスがアクティブであることを確認し、フリート ID を表示します。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。

Anywhere コンピューティングとしてワークステーションを登録する

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。このフィールドはデフォルトでマシンのパブリック IP アドレスになります。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 GameLift Servers はコンピューティングに接続できることを確認し、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、ゲーム実行ファイルが Amazon GameLift Servers サービスとの通信を初期化する際に必要となる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、Amazon GameLift Servers 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。

上記のコード例では、[API リクエストに AWS Signature Version 4 \(SigV4\)](#) を使用することもできます。SigV4 は、API リクエストに認証情報を追加するための AWS 署名プロトコルです。

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 GameLift Servers 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タスクはすべて完了しています。

ホストされたゲームをプレイする

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ときに Amazon GameLift Servers 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を介して Amazon GameLift Servers に送信されます。これに応答して、Amazon GameLift Servers 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、新しいゲームセッションを開始します。ゲームセッションでプレイヤーを受け入れる準備が整うと、Amazon GameLift Servers 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を提供します。

Unreal 用プラグイン: ゲームをマネージド EC2 フリートにデプロイする

このワークフローでは、プラグインを使用して、Amazon GameLift Servers 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。統合されたゲームサーバービルドを Amazon GameLift Servers サービスにアップロードしてデプロイします。ゲームコードをまだ統合していない場合は、「[Unreal 用プラグイン: ゲームコードを統合](#)」を参照してください。このワークフローが完了すると、動作するゲームクライアントができ、クラウド内のゲームサーバーに接続できます。

Amazon GameLift Servers 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 Servers メニューを選択し、[マネージド EC2 でホストする] 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。作成する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

ロー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。

ユーザープロフィールを設定する

1. 使用可能なプロフィールのドロップダウンリストからプロフィールを選択します。まだプロフィールがない場合、または新しいプロフィールを作成する場合は、Amazon GameLift メニューに移動し、**Set AWS User Profiles** 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]** ではない場合、**[ブートストラッププロフィール]** を選択して、ステータスが **[アクティブ]** になるまで待機します。

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、Amazon GameLift Servers C++ サーバー SDK for Unreal と連携するようにゲームサーバーとゲームクライアントビルドを準備します。ゲームコードをまだ統合しておらず、ゲームクライアントとサーバー実行可能ファイルを構築していない場合は、「[Unreal 用プラグイン: ゲームコードを統合](#)」を参照してください。ローカルワークステーションのゲーム実行可能ファイルへのパスを入力します。

ワークフローの子の手順では、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は、プラグインに説明とソースコードへのリンクが表示されます。クライアントおよびサーバーコンポーネントを構築するときは、ソース構築バージョンを使用する必要があります。

サーバー SDK と統合されたゲームサーバーを構築したら、次のタスクを実行して、ホスティングのために Amazon GameLift Servers にアップロードする準備をします。

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 (Windows)

Unreal Editor がデフォルトでサーバービルドファイルを保存する **WindowsServer** フォルダに、次の内容を追加します。

1. サーバービルドのインストールスクリプトを **WindowsServer** フォルダのルートにコピーします。インストールスクリプトは、プラグインのダウンロードに含まれています。ファイル `[project-name]/Plugins/Resources/CloudFormation/extra_server_resources/install.bat` を探します。Amazon GameLift Servers はこのファイルを使用して、ホスティングコンピューティングにサーバービルドをインストールします。
2. **VC_redist.x64.exe** ファイルを **WindowsServer** フォルダにコピーします。Unreal Engine バージョン 5.6 以降を使用している場合は、このステップをスキップできます。このファイル

は Visual Studio のインストールに含まれています。一般的に C:/Program Files (x86)/Microsoft Visual Studio/2019/Professional/VC/Redist/MSVC/v142 にあります。

3. OpenSSL ライブラリファイルをゲームサーバービルドに追加します。ゲームサーバーがサーバー SDK 5.3 以降と統合されている場合は、このステップをスキップできます。このバージョンは、Unreal バージョン 3.0 以降の Amazon GameLift Servers プラグインに含まれています。

OpenSSL ライブラリを手動で見つけて、<YourGame>/Binaries/Win64 のゲームビルドパッケージディレクトリにコピーします。Unreal Engine 5 バージョンで使用するのと同じ OpenSSL バージョンを使用する必要があります。間違った OpenSSL ライブラリでデプロイされたゲームビルドは、Amazon GameLift Servers サービスと通信できません。

ゲームエンジンソースで OpenSSL ライブラリを探します。場所は開発環境によって異なります。

Windows の場合:

- [ENGINE_ROOT_DIR]\Engine\Extras\ThirdPartyNotUE\libimobiledevice\x64\libssl-1_1-x64.dll
- [ENGINE_ROOT_DIR]\Engine\Extras\ThirdPartyNotUE\libimobiledevice\x64\libcrypto-1_1-x64.dll

Linux の場合:

- Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libssl.so.1.1
- Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libcrypto.so.1.1

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 (Linux)

Linux 用に構築されたゲームサーバーを準備する詳細な手順については、「[Amazon Linux での Unreal Engine 5 向けにサーバー SDK for Amazon GameLift Servers を構築する](#)」を参照してください。

1. ビルドファイルを整理する作業ディレクトリを指定します。作業ディレクトリの構造は、各ホスティングコンピューティングにそのままデプロイされます。Linux で構築されたゲームサーバーとすべての依存ファイルを追加します。
2. 作業ディレクトリのルートにサーバービルドインストールスクリプトを作成します。必要に応じて、install.sh ファイルを作成し、ゲームサーバービルドを適切にインストールするために

必要なコマンドを追加します。Amazon GameLift Servers 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。

3. OpenSSL ライブラリファイルをゲームサーバービルドに追加します。ゲームサーバーがサーバー SDK 5.3 以降と統合されている場合は、このステップをスキップできます。

ライブラリを手動で見つけてコピーします。Unreal Engine 5 バージョンで使用するのと同じ OpenSSL バージョンを使用する必要があります。間違った OpenSSL ライブラリでデプロイされたゲームビルドは、Amazon GameLift Servers サービスと通信できません。

- a. ゲームエンジンソースで OpenSSL ライブラリを探します。場所は開発環境によって異なります。

Windows の場合:

- [ENGINE_ROOT_DIR]\Engine\Extras\ThirdPartyNotUE\libimobiledevice\x64\libssl-1_1-x64.dll
- [ENGINE_ROOT_DIR]\Engine\Extras\ThirdPartyNotUE\libimobiledevice\x64\libcrypto-1_1-x64.dll

Linux の場合:

- Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libssl.so.1.1
 - Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libcrypto.so.1.1
- b. OpenSSL ライブラリを見つけたら、<YourGame>/Binaries/Linux のゲームビルドパッケージディレクトリにコピーします。

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: ゲームサーバーを、アクティブなプロファイルのデフォルト AWS リージョンにある単一のホスティングリソースのフリートにデプロイします。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、プレイヤー対戦を作成するためのルールを含む FlexMatch マッチメーカーをセットアップします。このシナリオでは、マルチフリート、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て、耐久性のある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向けのマッチメーカーコンポーネントの設計をすでに開始している場合に役立ちます。このシナリオでは、このソリューションのため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 のマッチメイキング設定と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イ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 つの AWS 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イ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 SNS トピック + 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、以下にアップロードするためのゲームについて説明します AWS。

- 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付けてくま。AWS はこの名前を使ってアップロードされて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のフォルダを指す必要があります。次の口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。このプロセスは、AWS がサーバービルドを検証し、ホスティングリソースをプロビジョニングし、ゲームサーバーをインストールし、サーバープロセスを起動し、ゲームセッションをホスティングする準備が整うまで数分かかる場合があります。

デプロイを開始するには、[CloudFormation をデプロイ] を選択します。ゲームホスティングの状況は、こちらで追跡できます。詳細については、の AWS AWS マネジメントコンソールにサインインし、イベント通知を表示できます。プラグインのアクティブな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 GameLift Servers でホストされている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タスクはすべて完了しています。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、ソリューションは少なくとも 2 つのクライアントがゲームセッションプレイメントのキューに入るのを待ってから、プレイヤーがサーバーマップに入ることができます。

Unreal 用プラグイン: ゲームをマネージドコンテナフリートにデプロイする

このガイド付きプラグインワークフローを使用して、ゲームサーバーのコンテナイメージを作成し、コンテナベースのホスティングソリューションにデプロイします。ゲームコードをまだ統合していない場合は、「[Unreal 用プラグイン: ゲームコードを統合](#)」を参照してください。このワークフローを正常に完了すると、コンテナ化されたゲームサーバーがクラウドで実行され、プラグインを使用してゲームクライアントを起動し、ゲームセッションに接続してゲームをプレイできます。

開始する前に

このワークフローでは、次のタスクを完了していることを前提としています。

- ゲームサーバーコードを Amazon GameLift Servers サーバー SDK と統合します。ホストされたゲームサーバーは、新しいゲームセッションの開始リクエストに応答し、ゲームセッションのステータスをレポートできるように、Amazon GameLift Servers サービスと通信する必要があります。このタスクをまだ完了していない場合は、まずプラグインワークフロー [Anywhere でホスト] を実行することをお勧めします。ゲームサーバーコードの準備に関するガイダンスについては、「[ゲームサーバーコードを更新する](#)」を参照してください。マネージドコンテナフリートの場合、ゲームサーバーを サーバー SDK バージョン 5.2 以上 と統合する必要があります。

Note

スタートアップゲームマップを使用している場合、このタスクはすでに完了しています。

- Linux で実行するようにゲームサーバー実行可能ファイルをパッケージ化します。Windows で開発し、C++ サーバー SDK バージョン 5.2.x 以前を統合する場合は、[Unreal クロスコンパイルツール](#)

[キット](#)を使用する必要があります。または、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することもできます。

- ゲームサーバービルドでデプロイするファイルを収集します。ローカルマシン上に 作業ディレクトリを作成し、コンテナイメージに組み込むファイルを整理します。これには、ゲームの依存関係、コンテナ起動時にゲームサーバーやその他のプロセスを起動するためのスクリプトなどが含まれます。
- ゲームサーバービルドの OpenSSL ライブラリファイルを追加します。ゲームサーバーがサーバー SDK 5.3 以降と統合されている場合は、このステップをスキップできます。

OpenSSL ライブラリを手動で見つけて、<YourGame>/Binaries/Win64 のゲームビルドパッケージディレクトリにコピーします。Unreal Engine 5 バージョンで使用するのと同じ OpenSSL バージョンを使用する必要があります。間違った OpenSSL ライブラリでデプロイされたゲームビルドは、Amazon GameLift Servers サービスと通信できません。

OpenSSL ライブラリを検索するには、ゲームエンジンソースを確認します。場所は開発環境によって異なります。

Windows の場合:

- [ENGINE_ROOT_DIR]\Engine\Extras\ThirdPartyNotUE\libimobiledevice\x64\libssl-1_1-x64.dll
- [ENGINE_ROOT_DIR]\Engine\Extras\ThirdPartyNotUE\libimobiledevice\x64\libcrypto-1_1-x64.dll

Linux の場合:

- Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libssl.so.1.1
- Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libcrypto.so.1.1
- ゲームクライアントコードを Amazon GameLift Servers と統合します。このタスクを完了する方法の一つの方法は、プラグインに付属している、既に統合済みのサンプルアセットを追加することです。ゲームクライアントコードの準備に関するガイダンスについては、「[クライアントゲームマップを統合する](#)」を参照してください。
- ローカルマシンに Docker をインストールしてセットアップします。プラグインにコンテナイメージの作成と、ECR リポジトリへのプッシュを行わせたい場合、このツールをインストールしておく必要があります。または、これらのタスクを手動で実行し、既存のコンテナイメージを使用するようにプラグインへ指示することもできます。コンテナイメージを手動で構築する方法の詳細については、「[Amazon GameLift Servers 用のコンテナイメージを構築する](#)」を参照してください。

Amazon GameLift Servers マネージドコンテナ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 Serversメニューを選択し、マネージドコンテナを選択します。この操作により、プラグインページ [マネージドコンテナでホスト] が開き、ゲームサーバービルドを基にしたコンテナイメージの作成、コンテナフリートへのデプロイ、ゲームの起動までをステップごとに案内するワークフローが表示されます。

ステップ 0: プロファイルを設定する

このセクションには、現在選択されているユーザープロフィールが表示されます。現在のユーザープロフィールが、このワークフローで使用するものであることを確認します。このワークフローで作成するすべてのリソースは、プロフィールの AWS アカウントに関連付けられ、プロフィールのデフォルト AWS リージョンに配置されます。プロフィー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。

次の場合は、選択中のユーザープロフィールを変更する必要がある場合があります。

- 現在、プロフィールが選択されていない場合。
- 別のプロフィールを選択する場合、または新しいプロフィールを作成する場合。
- 選択したプロフィールのブートストラップステータスが「非アクティブ」の場合、そのプロフィールをブートストラップする必要があります。

選択するユーザープロフィールを設定または変更するには

- Amazon GameLift Servers メニューで、Open AWS Access Credentials を選択します。

ステップ 1: コンテナの準備状況进行评估する

ゲームサーバーをコンテナフリートにデプロイする前に、これをコンテナイメージにパッケージ化し、Amazon ECR リポジトリに保存する必要があります。これらのタスクは、プラグインが自動的に処理することも、手動で実行することもできます。このステップでは、コンテナイメージおよび ECR リポジトリの状態に関する情報を入力します。

評価用の質問に回答し、プラグインが実行すべき手順を指示します。

- 新しいコンテナイメージを作成します。このオプションを選択すると、次のステップでゲームサーバービルドディレクトリの場所と実行可能ファイルの入力を求められます。プラグインは (Amazon GameLift Servers が提供する) Dockerfile テンプレートを使用し、ゲーム用に自動的に設

定します。テンプレートは [Amazon GameLift Servers 用のコンテナイメージを構築する](#) で使用できます。このオプションを選択した後、プラグインが新しいイメージを保存する場所を指定します。

- 新しい Amazon ECR リポジトリを作成し、コンテナイメージをプッシュします。プラグインは、AWS アカウントと選択したユーザープロファイル AWS リージョン のデフォルトを使用してプライベート ECR リポジトリを作成します。
- Amazon ECR リポジトリにコンテナイメージをプッシュします。このオプションを選択すると、次のステップでリストから既存の Amazon ECR リポジトリを選択するよう求められます。リストには、AWS アカウントのすべての Amazon ECR リポジトリと、選択したユーザープロファイル AWS リージョン のデフォルトが含まれます。パブリックリポジトリまたはプライベートリポジトリを選択できます。
- 既存のコンテナイメージを使用します。イメージを手動で構築した場合は、Amazon GameLift Servers が提供する Dockerfile テンプレートを使用することをお勧めします。このテンプレートは [Amazon GameLift Servers 用のコンテナイメージを構築する](#) で入手できます。このオプションを選択したら、イメージの場所を指定します。
- ローカルに保存された Docker 生成イメージ。このオプションを選択すると、プラグインは新しい Amazon ECR プライベートリポジトリを作成し、ローカルのイメージファイルをプッシュします。次のステップでは、プラグインがイメージファイルを見つけるために使用するイメージ ID の入力を求められます。
- Amazon ECR リポジトリに既に保存されているコンテナイメージ。このオプションを選択すると、次のステップで、リストから既存の Amazon ECR リポジトリとイメージを選択するよう求められます。リストには、AWS アカウントのすべての Amazon ECR リポジトリと、選択したユーザープロファイル AWS リージョン のデフォルトが含まれます。パブリックリポジトリまたはプライベートリポジトリを選択できます。

ステップ 2: イメージデプロイを設定する

このステップでは、コンテナイメージをコンテナフリートにデプロイするために、プラグインが必要とする情報を提供します。このステップでは、以下の情報をリクエストします。

- ステップ 1 で選択した内容に基づき、ゲームサーバービルド、コンテナイメージ、または Amazon ECR リポジトリの場所。
- マネージドコンテナのデプロイで使用するシナリオ。
- クライアント設定の出力パス。AWS 設定を含むクライアントビルドのフォルダを選択します。次のロケーションを探してください: `[client-build]/[project-name]/Content/CloudFormation`。

- オプションのデプロイ設定。このセクションには、プラグインがデフォルトで使用する設定が含まれます。または、デフォルト値を変更できます。
- ゲーム名は、デフォルトでゲームプロジェクトの名前に設定されます。プラグインが作成するすべての AWS リソースは、ゲーム名の値を参照します。
- ポート範囲、メモリ制限、vCPU 制限は、コンテナフリートの設定です。これらの値のカスタマイズの詳細については、接続ポート範囲については[ネットワーク接続を構成する](#)、リソース制限については「[リソース制限の設定](#)」を参照してください。
- コンテナイメージタグは、Amazon ECR でコンテナイメージを分類するために使用されます。デフォルト値は unreal-gamelift-plugin です。
- Amazon ECR リポジトリの名前。このフィールドを編集してカスタム名を提案できるのは、プラグインが ECR リポジトリを作成している場合のみです。デフォルト値は unreal-game-lift-plugin-ecr-repository です。

デプロイシナリオのオプション

単一リージョンコンテナフリート

このシナリオでは、ゲームサーバーを 1 つのコンテナフリートにデプロイします。これは、AWS とのサーバー統合とコンテナ設定をテストするための出発点として最適です。次のリソースをデプロイします。

- Amazon GameLift Servers コンテナグループ定義では、コンテナイメージをコンテナフリートにデプロイして実行する方法について説明します。
- Amazon GameLift Servers ゲームサーバーコンテナがインストールされ実行されている、エイリアス付きのコンテナフリート (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。
- API ゲートウェイへの過剰なプレイヤー呼び出しをスロットリングするための Web アクセスコントロールリスト (ACL)。
- ゲームセッションのリクエストやゲームへの参加など、ゲームクライアントに代わって Amazon GameLift Servers サービスにリクエストを行うバックエンドサービス。
 - プレイヤーがゲームセッション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 を使用した単一リージョンコンテナフリート

このシナリオでは、ゲームサーバーをコンテナフリートにデプロイし、ゲームセッションの配置を設定し、FlexMatch マッチメイキングを設定します。このシナリオは、ホスティングソリューション用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ため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。

- Amazon GameLift Servers コンテナフリートでコンテナイメージをデプロイして実行する方法を説明するコンテナグループ定義。
- Amazon GameLift Servers ゲームサーバーコンテナがインストールされ実行されている、エイリアス付きのコンテナフリート (オンデマンド)。
- FlexMatch のマッチメイキング設定と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イキングルールセット。
- (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 Amazon GameLift Servers 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。
- API ゲートウェイへの過剰なプレイヤー呼び出しをスロットリングするための Web アクセスコントロールリスト (ACL)。
- ゲームセッションのリクエストやゲームへの参加など、ゲームクライアントに代わって Amazon GameLift Servers サービスにリクエストを行うバックエンドサービス。
 - プレイヤーがゲームセッション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 `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。
- DynamoDB テーブルには、プレイヤーのマッチメイ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。
- `GameSessionQueue` イベントを処理する SNS トピック + Lambda 関数。

コンテナフリートをデプロイする

フリート設定が完了したら、[コンテナフリートをデプロイ] ボタンを選択してデプロイを開始します。このプロセスには、プラグインがコンテナイメージを作成して ECR にプッシュし、コンテナフ

フリートのホスティングリソースをプロビジョニングし、選択したホスティングソリューションシナリオのフリートやその他の AWS リソースをデプロイするまでに数分かかることがあります。

デプロイを開始すると、各ステップの進行状況を追跡できます。設定によっては、次の手順が含まれる場合があります。

- コンテナイメージの設定
- Amazon ECR リポジトリの作成
- イメージの構築と Amazon ECR へのプッシュ
- コンテナグループ定義の作成
- コンテナフリートの作成

デプロイの詳細については、[AWS マネジメントコンソールで表示] を選択します。コンテナフリートがアクティブステータスに達すると、フリートはゲームセッションをホストする準備ができているサーバープロセスで実行しているコンテナをアクティブに実行します。

デプロイが完了すると、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る準備ができている作業コンテナフリートがあります。

進行中のデプロイを停止することはできません。デプロイが不正な状態になった場合、または失敗した場合は、[デプロイをリセット] オプションを使用して最初からやり直すことができます。

クライアントの起動

この時点で、Amazon GameLift Servers でホストされている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タスクはすべて完了しています。ゲームをプレイするには、[クライアントを起動] を選択してゲームクライアントのローカル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 2 つ目のインスタンスを開き、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できます。
- FlexMatch シナリオをデプロイした場合、ホスティングソリューションは少なくとも 2 つのゲームクライアントがマッチメイキングリクエストを行うのを待ちます。少なくとも 2 つのインスタンスのゲームクライアントを、1 人のプレイヤーとともに開きます。2 人のプレイヤーがマッチングされ、マッチングのゲームセッションに参加するように求められます。

コンテナフリートの作成

マネージドコンテナホスティングソリューションを正常にデプロイした場合は、[デプロイを更新] 機能を使用できます。このオプションを使用すると、1 つのフリートを作成することなく、デプロイされたコンテナフリートの設定を更新できます。

デプロイを更新する際には、別のゲームサーバービルドでコンテナイメージをデプロイしたり、Amazon ECR リポジトリを変更したり、別のデプロイシナリオを選択したり、オプションの設定をカスタマイズしたりできます。

変更をデプロイする準備ができたなら、[更新] を選択します。デプロイの更新に必要な時間は、完全なデプロイに似ています。デプロイの詳細については、AWS マネジメントコンソールで表示 を選択します。

デプロイされたリソースをクリーンアップします。

ベストプラクティスとして、不要になったマネージドコンテナソリューションの AWS リソースをすぐにクリーンアップします。これらのリソースを削除しないと、引き続きコストが発生する可能性があります。

リソースを削除するには、次の作業を行います。

- マネージドコンテナリソーススタック。このスタックのリソースは、選択したデプロイシナリオによって異なります。スタック全体を削除するには、CloudFormation コンソールを使用します。Amazon GameLift Servers プラグインから生成されたスタックは、命名規則 `GameLiftPluginForUnreal-{GameName}-Containers` を使用します。スタックの削除プロセスが完了するまで待ってから、プラグインで新しいマネージドコンテナのデプロイを開始します。詳細については、「[CloudFormation コンソールからスタックを削除する](#)」を参照してください。
- Amazon ECR リポジトリ。プラグインを使用してコンテナイメージのリポジトリを作成した場合は、不要になったリポジトリを削除できます。マネージドコンテナのデプロイをリセットする前にリポジトリを削除する必要はありません。デプロイを更新またはリセットすると、別のリポジトリを使用するように指示されない限り、プラグインは同じリポジトリを自動的に使用します。詳細については、「[Amazon ECR のプライベートリポジトリの削除](#)」を参照してください。

Unity 用 Amazon GameLift Servers プラグイン (サーバー SDK 5.x)

このプラグインは、Amazon GameLift Servers C# サーバー SDK とツールを Unity エディタに追加します。ガイド付き UI ワークフローを使用して、サーバー SDK 機能をゲームプロジェクトに統合

し、ゲームサーバー用の Amazon GameLift Servers ホスティングソリューションをデプロイします。

プラグインを使用すると、基本的なホスティング環境を構築し、必要に応じて最適化やカスタマイズができます。ローカルワークステーションをホストとして、Amazon GameLift Servers Anywhere フリートをセットアップします。マネージド EC2 またはマネージドコンテナフリートによるクラウドホスティングの場合は、ゲームセッションリクエストとクライアント接続を管理するための完全なソリューションでゲームサーバーをデプロイします。

トピック

- [Unity ゲームプロジェクトのプラグインをインストールする](#)
- [Unity 用プラグイン: AWS ユーザープロファイルを設定する](#)
- [Unity 用プラグイン: Amazon GameLift Servers Anywhere を使用してローカルテストを設定する](#)
- [Unity 用プラグイン: ゲームをマネージド EC2 フリートにデプロイする](#)
- [Unity 用プラグイン: ゲームをマネージド コンテナフリートにデプロイする](#)

Unity ゲームプロジェクトのプラグインをインストールする

[GitHub から Unity 用の Amazon GameLift Servers プラグインを取得する](#)

ゲームプロジェクトのプラグインをインストールする方法については、GitHub リポジトリの [readme](#) を参照してください。

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項目から Amazon GameLift Servers 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 Servers サービス API 用の C# ライブラリ。
- Amazon GameLift Servers サーバー SDK (バージョン 5.x) 用の C# ライブラリ。
- サンプルゲームコンテンツを利用すると、ビルド済みのマルチプレイヤーゲームがなくても、Amazon GameLift Servers を試すことができます。
- プラグインがホスティングのためにゲームサーバーをクラウドにデプロイするときに使用する CloudFormation テンプレートとして提供されるソリューション設定。

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用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、ゲームに合わせて必要に応じてカスタマイズします。

Unity 用プラグイン: AWS ユーザープロファイルを設定する

プラグインをインストールしたら、有効な AWS アカウントでユーザープロファイルを設定します。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 つだけです。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。各ワークフローページには、現在選択されているプロファイルが表示されます。

複数のデプロイを管理することで、異なるホスティングデプロイ間で切り替えることができます。たとえば、同じ AWS アカウントを使用しても、異なる AWS リージョンにデプロイするプロファイルを設定できます。または、異なる AWS アカウントまたはユーザーとアクセス許可セットを使用して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、プロファイルが既に設定されている場合、Amazon GameLift Servers 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します。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。

アカウントユーザーで必要な AWS リソースをセットアップするには、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。

AWS プロファイルを管理するには

1. Unity メインツールバーで Amazon GameLift Servers メニューを選択し、AWS アクセス認証情報を選択します。このアクションにより、Amazon GameLift Servers プラグインで [ユーザープロファイルの設定] ページが開きます。
2. ボタンを使用して、新しい AWS アカウントを作成するか、既に持っている AWS アカウントのユーザープロファイルを設定します。
3. ユーザープロファイルがまだない場合は、プロファイルの詳細を入力し、新しいプロファイルを作成するように求められます。以下の情報を指定します。
 - AWS アカウント。新しい AWS アカウントを作成する場合は、[このリンク](#)を使用してプロンプト AWS マネジメントコンソール に従います。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 Servers およびその他の必要な AWS サービスを使用するアクセス許可を持つ AWS ユーザー。長期認証情報を使用してアクセス Amazon GameLift Servers 許可とプログ

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[AWS ユーザーアカウントを設定する](#)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 と AWS シークレットキーで構成されます。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 region。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場所です。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リージョンのリストから AWS リージョン](#)を選択します。
4. プラグインが既存のプロファイルを検出すると、使用可能なプロファイルのリストが表示されます。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。

ユーザープロファイルのブートストラップ

Amazon GameLift Servers 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、その他の依存関係の保存に使用されます。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 リソースの作成が含まれており、コストが発生する可能性があります。

プロファイルをブートストラップするには

1. [AWS アクセス認証情報] ページで、使用するユーザープロファイルのブートストラップステータスを確認します。プロファイルのブートストラップステータスが「非アクティブ」で、S3 バケットがリストされていない場合は、プロファイルをブートストラップする必要があります。
2. 使用するプロファイルを選択し、[ブートストラッププロファイル] を選択します。
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。

Unity 用プラグイン: Amazon GameLift Servers Anywhere を使用してローカルテストを設定する

このワークフローでは、Amazon GameLift Servers 機能用のクライアントコードとサーバーコードを追加し、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の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ビルドします。

Amazon GameLift Servers 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 Servers を選択し、続いて [Anywhere でホスト] を選択します。この操作により、@Anywhere フリートでゲームを設定するために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し、ビルドし、起動するための 5 ステップ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。作成する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。まだプロファイルがない場合、または新しいプロファイルを作成する場合は、Amazon GameLift Servers 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] ではない場合、[ブートストラッププロファイル] を選択して、ステータスが [アクティブ] になるまで待機します。

ゲームコードを C# サーバー SDK と統合する

Note

サンプルゲームをインポートした場合は、この手順をスキップできます。サンプルゲームアセットには、必要なサーバーコードとクライアントコードがすでに設定されています。

ワークフローのこの手順では、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、ゲームセッション開始のプロンプトの受信、接続情報の提供、ステータスの報告を行うために、Amazon GameLift Servers サービスと通信する必要があります。
- ゲームクライアントは、ゲームセッションの情報、ゲームセッションへの参加または開始情報、ゲームに参加するための接続情報を取得する必要があります。

サーバーコードを統合する

カスタムシーンがある独自のゲームプロジェクトを使用している場合は、指定のサンプルコードを使用して、必要なサーバーコードをゲームプロジェクトに追加します。

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](https://github.com/aws/amazon-gamelift-plugin-unity) に移動し、パス Samples~/SampleGame/Assets/Scripts/Server を開きます。
3. GameLiftServer.cs ファイルを見つけて、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 Servers C# サーバー SDK (バージョン 5) を使用する、以下の最小限の必須要素が含まれています。

- Amazon GameLift Servers APIクライアントを初期化します。Amazon GameLift Servers Anywhere フリートには、サーバーパラメータを指定した InitSDK() コールが必要です。これらの設定は、プラグインでできるように自動的に設定されます。
- Amazon GameLift Servers サービスからのリクエストに応答するため、OnStartGameSession、OnProcessTerminate、onHealthCheck を含む必須の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、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整ったことを Amazon GameLift Servers 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、以下のリソースを参照してください。

- [サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)
- [C# サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)

クライアントコードを統合する

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 Servers (AWS SDK 内の) のサービ

ス API 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 フリートでのローカルテスト用クライアントを構築する場合、Amazon GameLift Servers サービスへの直接呼び出しを追加できます。クラウドホスティング用のゲームを開発する場合、または本番ホスティングに Anywhere フリートを使用する場合、ゲームクライアントと Amazon GameLift Servers サービス間のすべての通信を処理するクライアント側バックエンドサービスを作成する必要があります。

Amazon GameLift Servers をクライアントコードに統合するには、次のリソースを参考にしてください。

- 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に関する制御を提供します。
- GitHub リポジトリ `aws/amazon-gamelift-plugin-unity`、`Samples~/SampleGame/Assets/Scripts/Client/GameLiftClient.cs` にあるサンプルゲーム統合を表示します。
- Unity ゲームクライアントに Amazon GameLift Servers を追加する手順に従います。

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以下の情報が必要です。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 フリートの一意的識別子。
- `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 Amazon GameLift Servers へのアクセスを許可するローカルマシンの AWS 認証情報プロファイル。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 Servers 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] 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 GameLift Servers はフリートのステータスがアクティブであることを確認し、フリート ID を表示します。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を登録] を選択します。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。

プラグインは、IP アドレスが localhost (127.0.0.1) に設定されているローカルワークステーションを登録します。この設定は、同じマシンでゲームクライアントとサーバーを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 GameLift Servers はコンピューティングに接続できることを確認し、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ビルドし、それ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の設定を行います。このステップでは、プラグインにプロンプトして、ゲームプロジェクトの GameLiftClientSettings アセットを更新します。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 Servers サービスに接続する際に必要となる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]、[Amazon GameLift]、[クライアント設定] の順に選択します。プロジェクトで GameLiftClientSettings の複数のコピーを作成すると、プラグインはこれを自動検出し、プラグインが更新するアセットについて通知します。
 - b. [ゲームを起動] で、[クライアントの設定: Anywhere 設定を適用] 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ビルドして実行します。
 - 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] の順に選択し、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。エディタで、[サーバー: サーバーを起動] を選択します。このタスクでは、ライブサーバーが起動されます。Unity エディタが開いている限り、クライアントはこのサーバーに接続できます。
4. ゲームセッションを開始するかゲームセッションに参加します。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。参加方法は、機能をクライアントに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、次の特徴があります。

- プレイヤーログインコンポーネント。Anywhere フリーッ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します。
- ゲーム参加のシンプルな UI。クライアントがゲームに参加しようとする、クライアントは利用可能なプレイヤースロットがあるアクティブなゲームセッションを自動的に検索します。ゲームセッションが利用できない場合、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がゲームセッションを開始し、残りのクライアントは既存のゲームセッションに自動的に参加します。

- 4 つのプレイヤースロットがあるゲームセッション。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できます。

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]、[ビルド] の順に選択し、プラットフォームを専有サーバーとビルドに切り替えます。
2. Anywhere フリート ID と AWS リージョンを使用して AWS CLI コマンド [get-compute-auth-token](#) 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、「プロファイルの設定」に AWS リージョンが表示されます。

```
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]
```

Unity 用プラグイン: ゲームをマネージド EC2 フリートにデプロイする

このワークフローでは、プラグインを使用して Amazon GameLift Servers 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上でゲームをホスティングできるよう準備します。Amazon GameLift Servers の機能用のクライアントコードとサーバーゲームコードを追加し、サーバービルドを Amazon GameLift Servers サービスにアップロードしてホスティングします。このワークフローが完了すると、クラウドで実行されるゲームサーバーとゲームサーバーに接続されるゲームクライアントが完成します。

Amazon GameLift Servers 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 Servers を選択し、「マネージド EC2 フリートでホスト」を選択します。このワークフローに、ゲームコンポーネントを統合、構築、デプロイそして起動する 6 つの手順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。作成する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。まだプロファイルがない場合、または新しいプロファイルを作成する場合は、Amazon GameLift Servers 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] ではない場合、[ブートストラッププロファイル] を選択して、ステータスが [アクティブ] になるまで待機します。

Amazon GameLift Servers とゲームを統合します。

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、ゲームセッション開始のプロンプトの受信、接続情報の提供、ステータスの報告を行うために、Amazon GameLift Servers サービスと通信する必要があります。
- ゲームクライアントは、ゲームセッションの情報、ゲームセッションへの参加または開始情報、ゲームに参加するための接続情報を取得する必要があります。

Note

サンプルゲームをインポートした場合は、この手順をスキップできます。サンプルゲームアセットには、必要なサーバーコードとクライアントコードがすでに設定されています。

サーバーコードを統合する

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、必要なサーバーコードをゲームプロジェクトに追加します。テスト用のゲームプロジェクトを Anywhere フリートと統合した場合、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](https://github.com/aws/amazon-gamelift-plugin-unity) に移動し、パス Samples~/SampleGame/Assets/Scripts/Server を開きます。
3. GameLiftServer.cs ファイルを見つけて、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 Servers C# サーバー SDK (バージョン 5) を使用する、以下の最小限の必須要素が含まれています。

- Amazon GameLift Servers API クライアントを初期化します。Amazon GameLift Servers Anywhere フリートには、サーバーパラメータを指定した InitSDK() コールが必要です。これらの設定は、プラグインでできるように自動的に設定されます。
- Amazon GameLift Servers サービスからのリクエストに応答するため、OnStartGameSession、OnProcessTerminate、onHealthCheck を含む必須の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、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整ったことを Amazon GameLift Servers 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、以下のリソースを参照してください。

- [サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)
- [C# サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)

クライアントコードを統合する

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では、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介して Amazon GameLift Servers 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 フリートでホスティングするための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事前構築済みの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 ID 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、接続情報を受信します。
- マッチメイ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てください。

- クライアントを GitHub リポジトリ [aws/amazon-gamelift-plugin-unity](https://github.com/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に関する制御を提供します。
- 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](https://github.com/aws/amazon-gamelift-plugin-unity)、`Samples~/SampleGame/Assets/Scripts/Client/GameLiftClient.cs` に移動します。
- [Amazon GameLift Servers ゲームクライアント機能の統合する](#)。

(Unity 6.3+ のみ) install.sh スクリプトを統合する

Linux での Unity 6.3 には glibc 2.35 が必要です。Amazon Linux 2023 には glibc 2.34 が含まれています。ソースからの glibc 2.35 の構築と、glibc 2.35 を使用するように Unity バイナリを設定する処理を行う install.sh スクリプトが用意されています。

1. GitHub リポジトリ [aws/amazon-gamelift-plugin-unity](https://github.com/aws/amazon-gamelift-plugin-unity) に移動し、パス `ExtraServerResources` を開きます。
2. ファイルを見つけ `install.sh`、サーバービルドを含むフォルダにコピーします。
3. 既存の `install.sh` がある場合は `install.sh`、両方のスクリプトの内容を 1 つの `install.sh` にマージします。

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: ゲームサーバーを、アクティブなプロファイルのデフォルト AWS リージョンにある単一のホスティングリソースのフリートにデプロイします。このシナリオは、と

サーバーの統合 AWS とサーバービルド設定をテストするための出発点として最適です。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、プレイヤー対戦を作成するためのルールを含む FlexMatch マッチメーカーをセットアップします。このシナリオでは、マルチフリート、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て、耐久性のある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向けのマッチメーカーコンポーネントの設計をすでに開始している場合に役立ちます。このシナリオでは、このソリューションのため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 のマッチメイキング設定と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イ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 つの AWS 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イ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 SNS トピック + Lambda 関数。

ゲームパラメータを設定する

このステップでは、アップロード先のゲームについて説明します AWS。

- 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 Servers は AWS コンソールでリソースを一覧表示するときに、この名前を (フリート ID とともに) 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、にアップロードAmazon GameLift Servers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: サーバービルドがある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。このプロセスは、AWS がサーバービルドを検証し、ホスティングリソースをプロビジョニングし、ゲームサーバーをインストールし、サーバープロセスを起動し、ゲームセッションをホスティングする準備が整うまで数分かかる場合があります。

デプロイを開始するには、[CloudFormation をデプロイ] を選択します。ゲームホスティングの状況は、こちらで追跡できます。詳細については、の AWS AWS マネジメントコンソールにサインインし、イベント通知を表示できます。プラグインのアクティブな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。クライアントをビルドして起動し、ゲームセッションに接続して参加します。

1. ゲームクライアントの設定を行います。このステップでは、プラグインにプロンプトして、ゲームプロジェクトの `GameLiftClientSettings` アセットを更新します。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 Servers サービスに接続する際に必要となる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]、[Amazon GameLift]、[クライアント設定] の順に選択します。プロジェクトで `GameLiftClientSettings` の複数のコピーを作成すると、プラグインはこれを自動検出し、プラグインが更新するアセットについて通知します。
 - b. [ゲームを起動] で、[クライアントの設定: マネージド EC2 設定を適用] を選択します。このアクションは、先ほど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ビルドします。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] の順に選択し、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開始し、各インスタンスの 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、次の特徴があります。

- プレイヤーログインコンポーネント。Anywhere フリー트의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します。
- ゲーム参加のシンプルな UI。クライアントがゲームに参加しようとする、クライアントは利用可能なプレイヤースロットがあるアクティブなゲームセッションを自動的に検索します。ゲームセッションが利用できない場合、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がゲームセッションを開始し、残りのクライアントは既存のゲームセッションに自動的に参加します。

- 4つのプレイヤースロットがあるゲームセッション。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できます。

Unity 用プラグイン: ゲームをマネージド コンテナフリートにデプロイする

このガイド付きプラグインワークフローを使用して、ゲームサーバーのコンテナイメージを作成し、コンテナベースのホスティングソリューションにデプロイします。このワークフローを正常に完了すると、コンテナ化されたゲームサーバーがクラウドで実行され、プラグインを使用してゲームクライアントを起動し、ゲームセッションに接続してゲームをプレイできます。

開始する前に

このワークフローでは、次のタスクを完了していることを前提としています。

- ゲームサーバーコードを Amazon GameLift Servers サーバー SDK と統合します。ホストされているゲームサーバーは、Amazon GameLift Servers サービスと通信する必要があります。これにより、新しいゲームセッション開始のリクエストに応答し、ステータスを報告できます。このタスクをまだ完了していない場合は、まずプラグインワークフロー [Anywhere でホスト] を実行することをお勧めします。ゲームサーバーコードの準備に関するガイダンスについては、「[サーバーコードを統合する](#)」を参照してください。マネージドコンテナフリートの場合、ゲームサーバーをサーバー SDK バージョン 5.2 以上 と統合する必要があります。

Note

サンプルゲームをインポートした場合、このタスクはすでに完了しています。

- Linux で実行するようにゲームサーバー実行可能ファイルをパッケージ化します。
- ゲームサーバービルドでデプロイするファイルを収集します。ローカルマシン上に作業ディレクトリを作成し、コンテナイメージに組み込むファイルを整理します。これには、ゲームの依存関係、コンテナ起動時にゲームサーバーやその他のプロセスを起動するためのスクリプトなどが含まれます。
- ゲームクライアントコードを Amazon GameLift Servers と統合します。このタスクを完了する方法は、プラグインに付属している、既に統合済みのサンプルアセットを追加することです。ゲームクライアントコードの準備に関するガイダンスについては、「[クライアントコードを統合する](#)」を参照してください。

- ローカルマシンに Docker をインストールしてセットアップします。プラグインにコンテナイメージの作成と、ECR リポジトリへのプッシュを行わせたい場合、このツールをインストールしておく必要があります。または、これらのタスクを手動で実行し、既存のコンテナイメージを使用するようにプラグインへ指示することもできます。コンテナイメージを手動で構築する方法の詳細については、「[Amazon GameLift Servers 用のコンテナイメージを構築する](#)」を参照してください。

Amazon GameLift Servers マネージドコンテナワークフローを開始するには:

- Unity エディタのメインツールバーで、Amazon GameLift Servers メニューを選択し、[マネージドコンテナ] を選択します。この操作により、プラグインページ [マネージドコンテナでホスト] が開き、ゲームサーバービルドを基にしたコンテナイメージの作成、コンテナフリートへのデプロイ、ゲームの起動までをステップごとに案内するワークフローが表示されます。

ステップ 0: プロファイルを設定する

このセクションには、現在選択されているユーザープロフィールが表示されます。現在のユーザープロフィールが、このワークフローで使用するものであることを確認します。このワークフローで作成するすべてのリソースは、プロフィールの AWS アカウントに関連付けられ、プロフィールのデフォルト AWS リージョンに配置されます。プロフィー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。

次の場合は、選択中のユーザープロフィールを変更する必要がある場合があります。

- 現在、プロフィールが選択されていない場合。
- 別のプロフィールを選択する場合、または新しいプロフィールを作成する場合。
- 選択したプロフィールのブートストラップステータスが「非アクティブ」の場合、そのプロフィールをブートストラップする必要があります。

選択するユーザープロフィールを設定または変更するには

- Amazon GameLift Servers メニューで、Open AWS Access Credentials を選択します。

ステップ 1: コンテナの準備状況进行评估する

ゲームサーバーをコンテナフリートにデプロイする前に、これをコンテナイメージにパッケージ化し、Amazon ECR リポジトリに保存する必要があります。これらのタスクは、プラグインが自動的

に処理することも、手動で実行することもできます。このステップでは、コンテナイメージおよび ECR リポジトリの状態に関する情報を入力します。

評価用の質問に回答し、プラグインが実行すべき手順を指示します。

- 新しいコンテナイメージを作成します。このオプションを選択すると、次のステップでゲームサーバービルドディレクトリの場所と実行可能ファイルの入力を求められます。プラグインは (Amazon GameLift Servers が提供する) Dockerfile テンプレートを使用し、ゲーム用に自動的に設定します。テンプレートは [Amazon GameLift Servers 用のコンテナイメージを構築する](#) で使用できます。このオプションを選択した後、プラグインが新しいイメージを保存する場所を指定します。
- 新しい Amazon ECR リポジトリを作成し、コンテナイメージをプッシュします。プラグインは、AWS アカウントと選択したユーザープロファイル AWS リージョン のデフォルトを使用してプライベート ECR リポジトリを作成します。
- Amazon ECR リポジトリにコンテナイメージをプッシュします。このオプションを選択すると、次のステップでリストから既存の Amazon ECR リポジトリを選択するよう求められます。リストには、AWS アカウントのすべての Amazon ECR リポジトリと、選択したユーザープロファイル AWS リージョン のデフォルトが含まれます。パブリックリポジトリまたはプライベートリポジトリを選択できます。
- 既存のコンテナイメージを使用します。イメージを手動で構築した場合は、Amazon GameLift Servers が提供する Dockerfile テンプレートを使用することをお勧めします。このテンプレートは [Amazon GameLift Servers 用のコンテナイメージを構築する](#) で入手できます。このオプションを選択したら、イメージの場所を指定します。
- ローカルに保存された Docker 生成イメージ。このオプションを選択すると、プラグインは新しい Amazon ECR プライベートリポジトリを作成し、ローカルのイメージファイルをプッシュします。次のステップでは、プラグインがイメージファイルを見つけるために使用するイメージ ID の入力を求められます。
- Amazon ECR リポジトリに既に保存されているコンテナイメージ。このオプションを選択すると、次のステップで、リストから既存の Amazon ECR リポジトリとイメージを選択するよう求められます。リストには、AWS アカウントのすべての Amazon ECR リポジトリと、選択したユーザープロファイル AWS リージョン のデフォルトが含まれます。パブリックリポジトリまたはプライベートリポジトリを選択できます。
- Linux での Unity 6.3 には glibc 2.35 が必要です。Amazon Linux 2023 には glibc 2.34 が含まれています。ソースからの glibc 2.35 の構築と、glibc 2.35 を使用するように Unity バイナリを設定する処理を行う Dockerfile テンプレートを以下に示します。

Unity 6.3 ゲームサーバーコンテナイメージの Dockerfile テンプレート

このテンプレートには、ゲームサーバーコンテナが Amazon GameLift Servers 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。ゲームサーバーに必要な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.
FROM public.ecr.aws/amazonlinux/amazonlinux

# Game build directory
# -----
# Add your game build directory in the 'GAME_BUILD_DIRECTORY' env variable below.
#
# Game executable
# -----
# Add the relative path to your executable in the 'GAME_EXECUTABLE' env variable
below.
# The game build provided over here needs to be integrated with gamelift server sdk.
# This template assumes that the executable path is relative to the game build
directory.
#
# Launch parameters
# -----
# Add any launch parameters to pass into your executable in the 'LAUNCH_PARAMS' env
variable below.
#
# Default directory
# -----
# The value provided in 'HOME_DIR' below will be where the game executable and logs
exist.
#
ARG GAME_BUILD_DIRECTORY
ARG GAME_EXECUTABLE
ARG LAUNCH_PARAMS

ENV GAME_BUILD_DIRECTORY=$GAME_BUILD_DIRECTORY \
    GAME_EXECUTABLE=$GAME_EXECUTABLE \
    LAUNCH_PARAMS=$LAUNCH_PARAMS \
    HOME_DIR="/local/game"
```

```

# install dependencies as necessary
RUN yum install -y shadow-utils bison wget gcc make patchelf tar gzip && \
    yum clean all && rm -fr /var/cache

RUN mkdir -p $HOME_DIR
COPY .$GAME_BUILD_DIRECTORY/ $HOME_DIR

# Change directory to home
WORKDIR $HOME_DIR

# Build glibc 2.35 and patch Unity binaries
RUN GLIBC_VERSION="2.35" && \
    INSTALL_PREFIX="/opt/glibc-`${GLIBC_VERSION}`" && \
    ARCH=$(uname -m) && \
    if [ "$ARCH" = "x86_64" ]; then \
        INTERPRETER="`${INSTALL_PREFIX`}/lib/ld-linux-x86-64.so.2"; \
        MONO_ARCH="x86_64"; \
    elif [ "$ARCH" = "aarch64" ]; then \
        INTERPRETER="`${INSTALL_PREFIX`}/lib/ld-linux-aarch64.so.1"; \
        MONO_ARCH="aarch64"; \
    else \
        echo "ERROR: Unsupported architecture: $ARCH"; exit 1; \
    fi && \
    cd /tmp && \
    wget -q "https://ftp.gnu.org/gnu/glibc/glibc-`${GLIBC_VERSION`}.tar.gz" && \
    tar -xzf "glibc-`${GLIBC_VERSION`}.tar.gz" && \
    cd "glibc-`${GLIBC_VERSION`}" && mkdir glibc-build && cd glibc-build && \
    touch /etc/ld.so.conf && \
    ../configure --prefix="`${INSTALL_PREFIX`}" && \
    make -s all && make -s install && \
    cd / && rm -rf /tmp/glibc-* && \
    GLIBC_LIB="`${INSTALL_PREFIX`}/lib" && \
    EXECUTABLE_NAME=$(echo "$GAME_EXECUTABLE" | sed 's/\.[^.]*/') && \
    DATA_DIR="$HOME_DIR/`${EXECUTABLE_NAME`}_Data" && \
    patchelf --set-interpreter "$INTERPRETER" "$HOME_DIR/`${GAME_EXECUTABLE`}" && \
    patchelf --set-rpath "$GLIBC_LIB:$HOME_DIR:/lib64:$DATA_DIR/MonoBleedingEdge/`
$MONO_ARCH`" "$HOME_DIR/`${GAME_EXECUTABLE`}" && \
    patchelf --set-rpath "$GLIBC_LIB:/lib64" "$HOME_DIR/UnityPlayer.so" && \
    MONO_LIB="$DATA_DIR/MonoBleedingEdge/`${MONO_ARCH`}/libmonobdwgc-2.0.so" && \
    if [ -f "`${MONO_LIB`}" ]; then patchelf --set-rpath "$GLIBC_LIB:/lib64" "`${MONO_LIB`";
fi && \
    GAME_ASSEMBLY="$HOME_DIR/GameAssembly.so" && \

```

```
if [ -f "$GAME_ASSEMBLY" ]; then patchelf --set-rpath "$GLIBC_LIB:/lib64"
"$GAME_ASSEMBLY"; fi

RUN useradd -m gamescale && \
    echo "gamescale ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers && \
    chown -R gamescale:gamescale $HOME_DIR

# Add permissions to game build
RUN chmod +x ./$GAME_EXECUTABLE

USER gamescale

# Check directory before starting the container
RUN ls -lhrt .

# Check path before starting the container
RUN echo $PATH

# Start the game build
ENTRYPOINT ["/bin/sh", "-c", "./$GAME_EXECUTABLE $LAUNCH_PARAMS"]
```

ステップ 2: イメージデプロイを設定する

このステップでは、コンテナイメージをコンテナフリートにデプロイするために、プラグインが必要とする情報を提供します。このステップでは、以下の情報をリクエストします。

- ステップ 1 で選択した内容に基づき、ゲームサーバービルド、コンテナイメージ、または Amazon ECR リポジトリの場所。
- マネージドコンテナのデプロイで使用するシナリオ。
- オプションのデプロイ設定。このセクションには、プラグインがデフォルトで使用する設定が含まれます。または、デフォルト値を変更できます。
 - ゲーム名は、デフォルトでゲームプロジェクトの名前に設定されます。プラグインが作成するすべての AWS リソースは、ゲーム名の値を参照します。
 - ポート範囲、メモリ制限、vCPU 制限は、コンテナフリートの設定です。これらの値のカスタマイズの詳細については、接続ポート範囲については[ネットワーク接続を構成する](#)、リソース制限については「[リソース制限の設定](#)」を参照してください。
 - コンテナイメージタグは、Amazon ECR でコンテナイメージを分類するために使用されます。デフォルト値は `unity-gamelift-plugin` です。

デプロイシナリオのオプション

単一リージョンコンテナフリート

このシナリオでは、ゲームサーバーを 1 つのコンテナフリートにデプロイします。これは、AWS とのサーバー統合とコンテナ設定をテストするための出発点として最適です。次のリソースをデプロイします。

- Amazon GameLift Servers コンテナグループ定義では、コンテナイメージをコンテナフリートにデプロイして実行する方法について説明します。
- Amazon GameLift Servers ゲームサーバーコンテナがインストールされ実行されている、エイリアス付きのコンテナフリート (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。
- API ゲートウェイへの過剰なプレイヤー呼び出しをスロットリングするための Web アクセスコントロールリスト (ACL)。
- ゲームセッションのリクエストやゲームへの参加など、ゲームクライアントに代わって Amazon GameLift Servers サービスにリクエストを行うバックエンドサービス。
 - プレイヤーがゲームセッション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 を使用した単一リージョンコンテナフリート

このシナリオでは、ゲームサーバーをコンテナフリートにデプロイし、ゲームセッションの配置を設定し、FlexMatch マッチメイキングを設定します。このシナリオは、ホスティングソリューション用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ため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。

- Amazon GameLift Servers コンテナフリートでコンテナイメージをデプロイして実行する方法を説明するコンテナグループ定義。
- Amazon GameLift Servers ゲームサーバーコンテナがインストールされ実行されている、エイリアス付きのコンテナフリート (オンデマンド)。

- FlexMatch のマッチメイキング設定と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イキングルールセット。
- (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 Amazon GameLift Servers 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。
- API ゲートウェイへの過剰なプレイヤー呼び出しをスロットリングするための Web アクセスコントロールリスト (ACL)。
- ゲームセッションのリクエストやゲームへの参加など、ゲームクライアントに代わって Amazon GameLift Servers サービスにリクエストを行うバックエンドサービス。
 - プレイヤーがゲームセッション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 `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。
- DynamoDB テーブルには、プレイヤーのマッチメイ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。
- GameSessionQueue イベントを処理する SNS トピック + Lambda 関数。

コンテナフリートをデプロイする

フリート設定が完了したら、[コンテナフリートをデプロイ] ボタンを選択してデプロイを開始します。このプロセスには、プラグインがコンテナイメージを作成して ECR にプッシュし、コンテナフリートのホスティングリソースをプロビジョニングし、選択したホスティングソリューションシナリオのフリートやその他の AWS リソースをデプロイするまでに数分かかることがあります。

デプロイを開始すると、各ステップの進行状況を追跡できます。設定によっては、次の手順が含まれる場合があります。

- コンテナイメージの設定
- Amazon ECR リポジトリの作成
- イメージの構築と Amazon ECR へのプッシュ
- コンテナグループ定義の作成
- コンテナフリートの作成

デプロイの詳細については、[AWS マネジメントコンソールで表示] を選択します。コンテナフリートがアクティブステータスに達すると、フリートはゲームセッションをホストする準備ができているサーバープロセスで実行しているコンテナをアクティブに実行します。

デプロイが完了すると、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る準備ができている作業コンテナフリートがあります。

進行中のデプロイを停止することはできません。デプロイが不正な状態になった場合、または失敗した場合は、[デプロイをリセット] オプションを使用して最初からやり直すことができます。

クライアントの起動

この時点で、Amazon GameLift Servers でホストされている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タスクはすべて完了しています。ゲームをプレイするには、[クライアントを起動] を選択してゲームクライアントのローカル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 2 つ目のインスタンスを開き、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できます。
- FlexMatch シナリオをデプロイした場合、ホスティングソリューションは少なくとも 2 つのゲームクライアントがマッチメイキングリクエストを行うのを待ちます。少なくとも 2 つのインスタンスのゲームクライアントを、1 人のプレイヤーとともに開きます。2 人のプレイヤーがマッチングされ、マッチングのゲームセッションに参加するように求められます。

コンテナフリートの作成

マネージドコンテナホスティングソリューションを正常にデプロイした場合は、[デプロイを更新] 機能を使用できます。このオプションを使用すると、1 つのフリートを作成することなく、デプロイされたコンテナフリートの設定を更新できます。

デプロイを更新する際には、別のゲームサーバービルドでコンテナイメージをデプロイしたり、Amazon ECR リポジトリを変更したり、別のデプロイシナリオを選択したり、オプションの設定をカスタマイズしたりできます。

変更をデプロイする準備ができたら、[更新] を選択します。デプロイの更新に必要な時間は、完全なデプロイに似ています。デプロイの詳細については、AWS マネジメントコンソールで表示 を選択します。

デプロイされたリソースをクリーンアップします。

ベストプラクティスとして、不要になったマネージドコンテナソリューションの AWS リソースをすぐにクリーンアップします。これらのリソースを削除しないと、引き続きコストが発生する可能性があります。

リソースを削除するには、次の作業を行います。

- マネージドコンテナリソーススタック。このスタックのリソースは、選択したデプロイシナリオによって異なります。スタック全体を削除するには、CloudFormation コンソールを使用します。Amazon GameLift Servers プラグインから生成されたスタックは、命名規則 `GameLiftPluginForUnity-{GameName}-Containers` を使用します。スタックの削除プロセスが完了するまで待ってから、プラグインで新しいマネージドコンテナのデプロイを開始します。詳細については、「[CloudFormation コンソールからスタックを削除する](#)」を参照してください。
- Amazon ECR リポジトリ。プラグインを使用してコンテナイメージのリポジトリを作成した場合は、不要になったリポジトリを削除できます。マネージドコンテナのデプロイをリセットする前にリポジトリを削除する必要はありません。デプロイを更新またはリセットすると、別のリポジトリを使用するように指示されない限り、プラグインは同じリポジトリを自動的に使用します。詳細については、「[Amazon ECR のプライベートリポジトリの削除](#)」を参照してください。

サーバー SDK 4 向け Unity 用 Amazon GameLift Servers プラグイン

Note

このトピックでは、Unity 用 Amazon GameLift Servers プラグインの前のバージョンについて説明します。バージョン 1.x では、サーバー SDK for Amazon GameLift Servers 4.x 以前が使用されます。サーバー SDK 5.x を使用し、Amazon GameLift Servers Anywhere やマネージドコンテナホスティングなどの新しい機能をサポートする最新のプラグインバージョンのドキュメントについては、「[Unity 用 Amazon GameLift Servers プラグイン \(サーバー SDK 5.x\)](#)」を参照してください。

Amazon GameLift Servers には、マルチプレイヤーゲームサーバーを Amazon GameLift Servers で実行するための準備を整えるツールが用意されています。Unity 用 Amazon GameLift Servers プラグインを使用すると、Amazon GameLift Servers と Unity ゲームプロジェクトの統合や、クラウドホスティング用の Amazon GameLift Servers リソースのデプロイが簡単になります。Unity 用プラグインを使用して、Amazon GameLift Servers API にアクセスし、一般的なゲームシナリオで 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、GitHub の [Amazon GameLift Servers Unity サンプル](#)を試すことができます。

トピック

- [Amazon GameLift Servers と Unity ゲームサーバープロジェクトを統合する](#)
- [Amazon GameLift Servers と Unity ゲームクライアントプロジェクトを統合する](#)
- [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 Amazon GameLift Serversでゲームを統合する](#)
- [サンプルゲームをインポートして実行する](#)

Amazon GameLift Servers と Unity ゲームサーバープロジェクトを統合する

Note

このトピックでは、Unity 用 Amazon GameLift Servers プラグインの前のバージョンについて説明します。バージョン 1.x では、サーバー SDK for Amazon GameLift Servers 4.x 以前が使用されます。サーバー SDK 5.x を使用し、Amazon GameLift Servers Anywhere やマネージドコンテナホスティングなどの新しい機能をサポートする最新のプラグインバージョンのドキュメントについては、「[Unity 用 Amazon GameLift Servers プラグイン \(サーバー SDK 5.x\)](#)」を参照してください。

このトピックでは、Amazon GameLift Servers 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。ゲームサーバーは、ステータス、ゲームセッションの開始および停止 (プロンプトが表示された場合)、およびその他のタスクの実行について Amazon GameLift Servers に通知可能である必要があります。詳細については、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」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 GameLift Servers 用に IAM サービスロールをセットアップする](#)
- [プラグインをインストールしてセットアップする](#)

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 Servers プラグインについて説明します。

Amazon GameLift Servers との通信をセットアップし、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()、OnGameSessionUpdate()、OnProcessTerminate()、OnHealthCheck など、Amazon GameLift Servers 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。コールバックの提供には数分かかる Amazon GameLift Servers 場合があります。
3. Amazon GameLift Servers はサーバープロセスのステータスを ACTIVE に更新します。
4. Amazon GameLift Servers は定期的に onHealthCheck を呼び出します。

次のコード例は、Amazon GameLift Servers を使用して簡単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。

```
//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 Servers 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 GameLift Servers はこのメソッドを呼び出して、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、「[C# サーバー SDK for Amazon GameLift Servers 4.x -- アクション](#)」を参照してください。

以下のコード例では、Amazon GameLift Servers でゲームセッションを開始する方法を示しています。

```
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();
}
```

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 Servers プラグインについて説明します。

ゲームセッションの終了を Amazon GameLift Servers 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. `onProcessTerminate` という名前の関数をセットアップし、Amazon GameLift Servers からのリクエストを受信して `ProcessEnding()` を呼び出します。
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 GameLift Servers にアップロードする

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 Servers プラグインについて説明します。

ゲームサーバーを Amazon GameLift Servers と統合したら、ビルドファイルをフリーにアップロードして、Amazon GameLift Servers がゲームホスティング用にデプロイできるようにします。Amazon GameLift Servers にサーバーをアップロードする方法については、「[Amazon GameLift Servers のゲームサーバービルドを作成する](#)」を参照してください。

Amazon GameLift Servers と Unity ゲームクライアントプロジェクトを統合する

Note

このトピックでは、Unity 用 Amazon GameLift Servers プラグインの前のバージョンについて説明します。バージョン 1.x では、サーバー SDK for Amazon GameLift Servers 4.x 以前が使用されます。サーバー SDK 5.x を使用し、Amazon GameLift Servers Anywhere やマネージドコンテナホスティングなどの新しい機能をサポートする最新のプラグインバージョンのドキュメントについては、「[Unity 用 Amazon GameLift Servers プラグイン \(サーバー SDK 5.x\)](#)」を参照してください。

このトピックは、バックエンドサービスを通じて Amazon GameLift Servers がホストするゲームセッションに接続するゲームクライアントをセットアップするのに役立ちます。Amazon GameLift Servers API を使用して、マッチメイキングの開始、ゲームセッションプレイメントのリクエストなどを行います。

Amazon GameLift Servers サービスと通信できるようにするコードをバックエンドサービスプロジェクトに追加します。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、「」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します。
- Amazon GameLift Servers サービス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イヤーを追加する。
- 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](#)
- [ゲームクライアントを初期化する](#)
- [特定のフリートにゲームセッションを作成する](#)
- [プレイヤーをゲームセッションに追加する](#)
- [ゲームセッションからプレイヤーを削除する](#)

前提条件

Amazon GameLift Servers クライアントとのゲームサーバー通信をセットアップする前に、次のタスクを実行します。

- [AWS ユーザーアカウントを設定する](#)
- [プラグインをインストールしてセットアップする](#)
- [Amazon GameLift Servers と Unity ゲームサーバープロジェクトを統合する](#)
- [Amazon GameLift Servers のホスティングフリートをデプロイする](#)

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 Servers 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。このコードを起動時に実行する。Amazon GameLift Servers の他の機能にも必要です。

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、「[Amazon GameLift Servers APIをセットアップします。](#)」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 Serversクライアントを設定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 Servers 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 Amazon GameLift Servers 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 Servers 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 Servers が新しいゲームセッションを作成して `GameSession` を返したら、そのゲーム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 Servers プラグインについて説明します。

Amazon GameLift Servers 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、そのゲーム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ズ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ズロ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 Servers 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 Servers 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。それに応じて、Amazon GameLift Servers がプレイヤースロットを利用可能に変更することで、Amazon GameLift Servers は新しいプレイヤーに割り当てられるようになります。

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 Servers プラグインについて説明します。

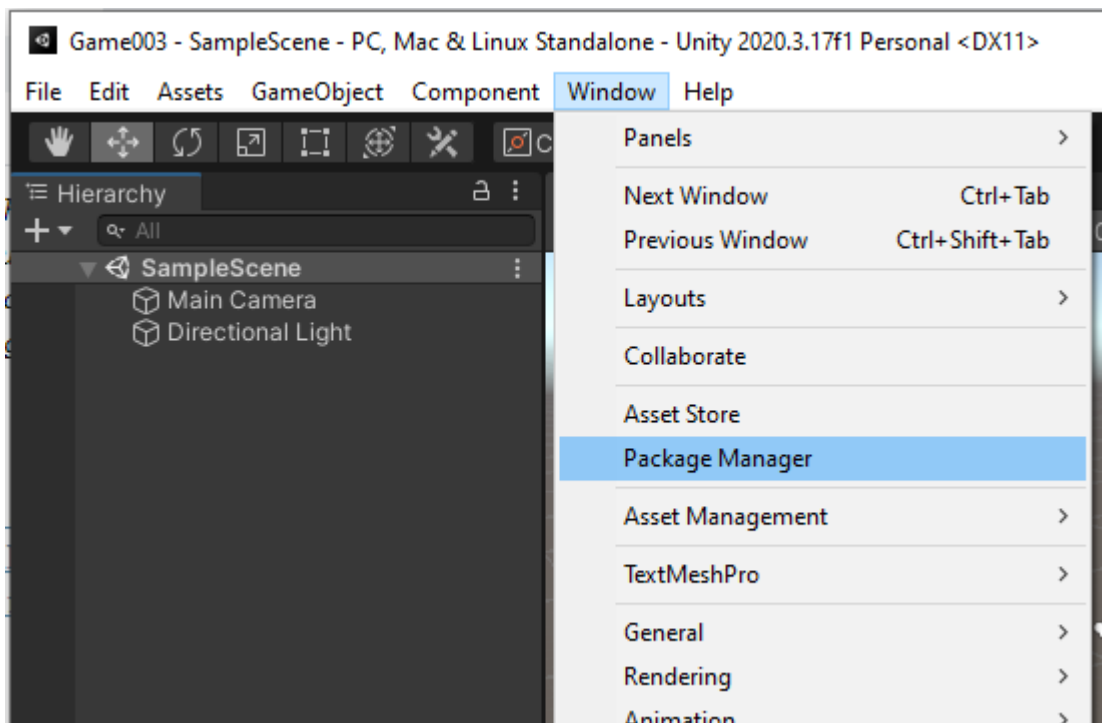
このセクションでは、Unity 用 Amazon GameLift Servers プラグインのバージョン 1.0.0 をダウンロード、インストール、セットアップする方法について説明します。

前提条件

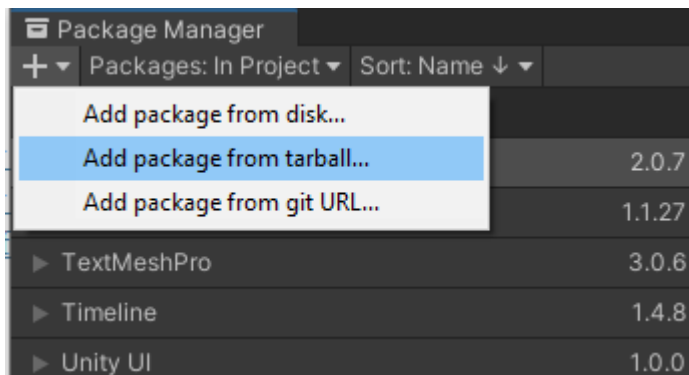
- Windows 2019.4 LTS 用 Unity、Windows 2020.3 LTS 用、または MacOS 用 Unity
- Java の現在のバージョン
- .NET 4.x の現在のバージョン

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

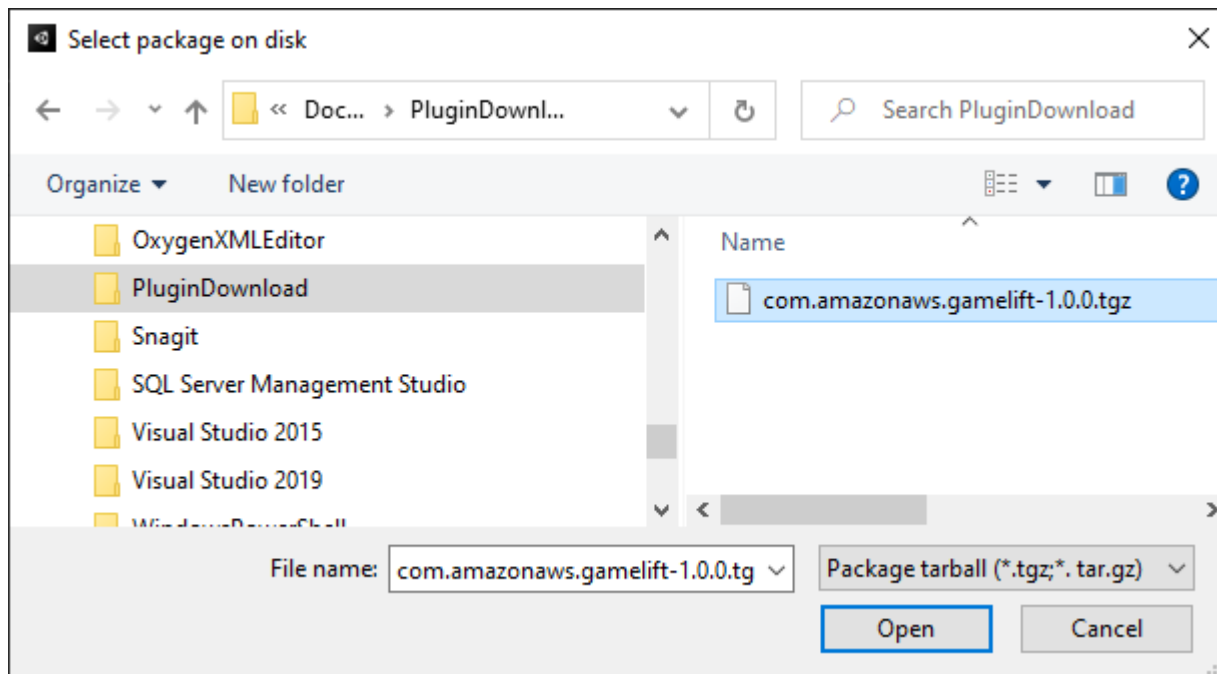
1. Unity 用の Amazon GameLift Servers プラグインをダウンロードします。最新バージョンは、[\[Unity リポジトリ用 Amazon GameLift Servers プラグイン\]](#) ページにあります。[\[最新リリース\]](#) で [アセット] を選択し、com.amazonaws.gamelift-version.tgz ファイルをダウンロードします。
2. Unity を起動し、プロジェクトを選択します。
3. 上部のナビゲーションバーの [ウィンドウ] で、[パッケージマネージャー] を選択します。



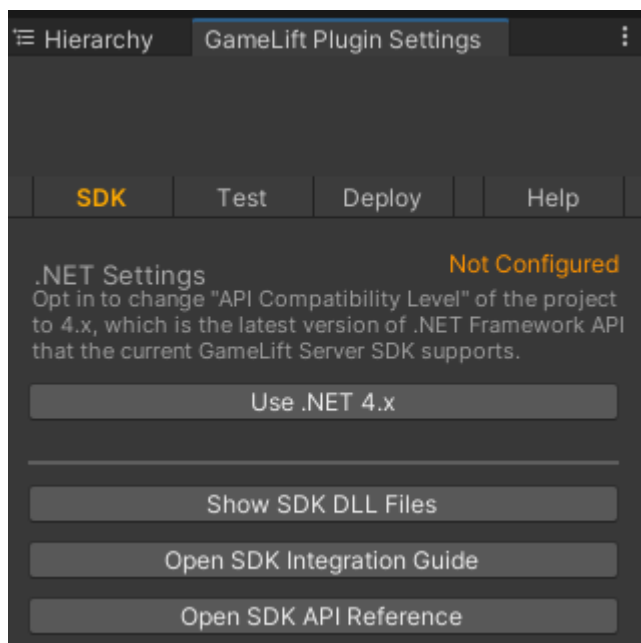
4. [パッケージマネージャー] タブで [+] を選択し、[ターボールからパッケージを追加...] を選択します。



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 Servers が 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 Servers プラグインの設定] タブが自動的に開きます。



7. SDK ペインで、[.NET 4.x を使用] 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 Servers プラグインについて説明します。

Amazon GameLift Servers Local を使用して、ローカルデバイスで Amazon GameLift Servers を実行します。Amazon GameLift Servers 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 Servers Local をダウンロード] 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 Servers プラグインの設定] ウィンドウで、[Amazon GameLift Servers ローカルパス] ウィンドウを選択して解凍したフォルダに移動し、GameLiftLocal.jar ファイルを選択して [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. [ローカルテスト] ウィンドウで、[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. [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 Servers プラグインについて説明します。

シナリオでは、CloudFormation テンプレートを使用して、ゲーム用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 Servers 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 Servers サービスの IAM ロールが必要になります。Amazon GameLift Servers のロールの作成方法の詳細については、[AWS ユーザーアカウントを設定する](#)を参照してください。

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 Servers
- Amazon S3
- CloudFormation
- API ゲートウェイ
- 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 Servers プラグインについて説明します。

Unity 用 Amazon GameLift Servers 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。
- ゲームを開始し、ゲーム接続情報を表示する Amazon API Gateway REST endpoint-backed AWS Lambda ハンドラー。

単一リージョンフリート

このシナリオでは、単一の Amazon GameLift Servers 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。
- フリートで開いているプレイヤースロットを持つ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 Servers 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してマッチを形成します。次のリソースを作成します。

- Amazon GameLift Servers がメッセージをパブリッシュする 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ト通知を設定](#)」を参照してください。
- プレイ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 Servers キューとカスタムマッチメーカーを使用してマッチを作成し、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、「[ホスティングリソースのカスタマイズ](#)」を参照してください。
- サーバーの可用性を高く、コストを低く抑えるための Amazon GameLift Servers 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をカスタマイズする](#)」を参照してください。

FlexMatch

このシナリオでは、マネージドマッチメイキングサービスである FlexMatch を使用してゲームプレイヤー同士をマッチングします。FlexMatch の詳細については、「[Amazon GameLift Servers FlexMatch とは](#)」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イキングチケットを作成する Lambda 関数。
- FlexMatch マッチイベントをリッスンする別の Lambda 関数。

で不要な料金が発生しないようにするには AWS アカウント、各シナリオで作成されたリソースの使用が終了したら削除します。対応する 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 Servers プラグインについて説明します。

Unity 用 Amazon GameLift Servers プラグインでは、シナリオをデプロイするための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 Servers 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 AWS リージョンを選択するは、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、新しい 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 Servers プラグインについて説明します。

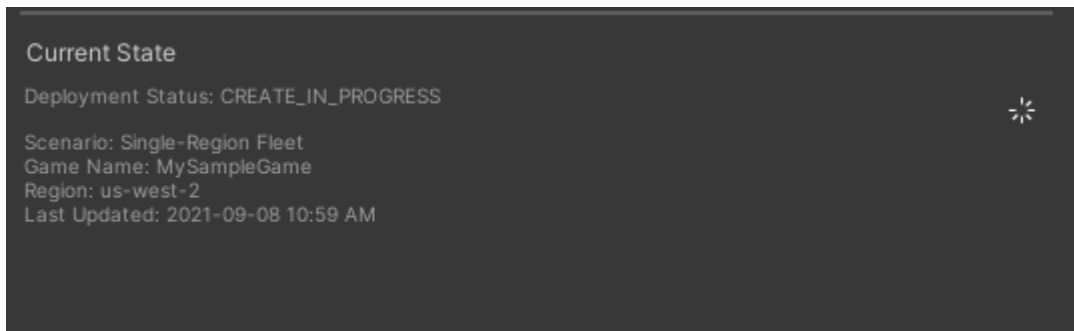
シナリオを使用して Amazon GameLift Servers でゲームをテストできます。各シナリオでは、CloudFormation テンプレートを使用して、必要なリソースを含むスタックを作成します。ほとんどのシナリオでは、ゲームサーバーの実行可能ファイルとビルドパスが必要です。シナリオがデプロイされると、Amazon GameLift Servers はデプロイの一環としてブートストラップのロケーションにゲームアセットをコピーします。

シナリオをデプロイするには、AWS 認証情報と 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。

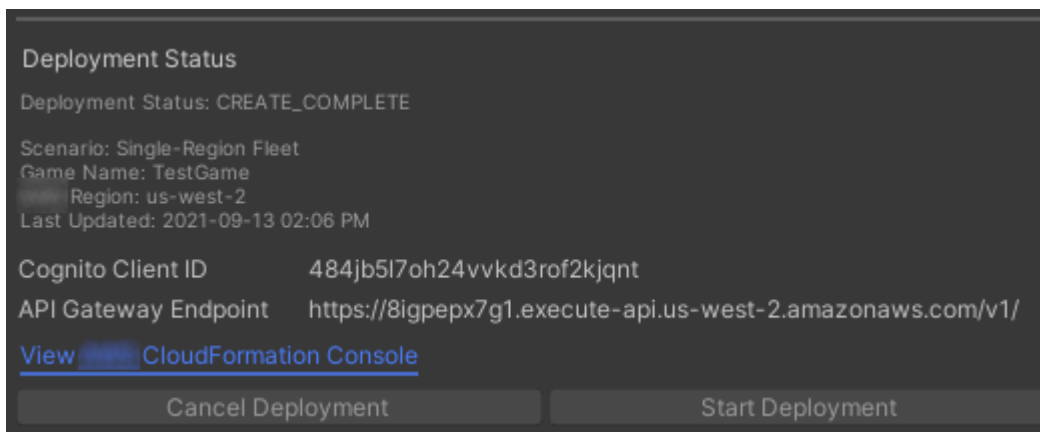
シナリオをデプロイするには

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 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数分かかる場合があります。



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 [Cognito クライアント ID] を入力します。シナリオのデプロイ AWS リージョン に使用したものと同一を選択します。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 Servers 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 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 Deployment のAmazon GameLift Servers 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 CloudFormation コンソールを開きます。
2. CloudFormation コンソールで、スタックを選択し、デプロイ時に指定され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。ガシナリオで使用されるスタック CloudFormation を削除すると、そのステータスは に変わります ROLLBACK_COMPLETE。

Unity Amazon GameLift Serversでゲームを統合する

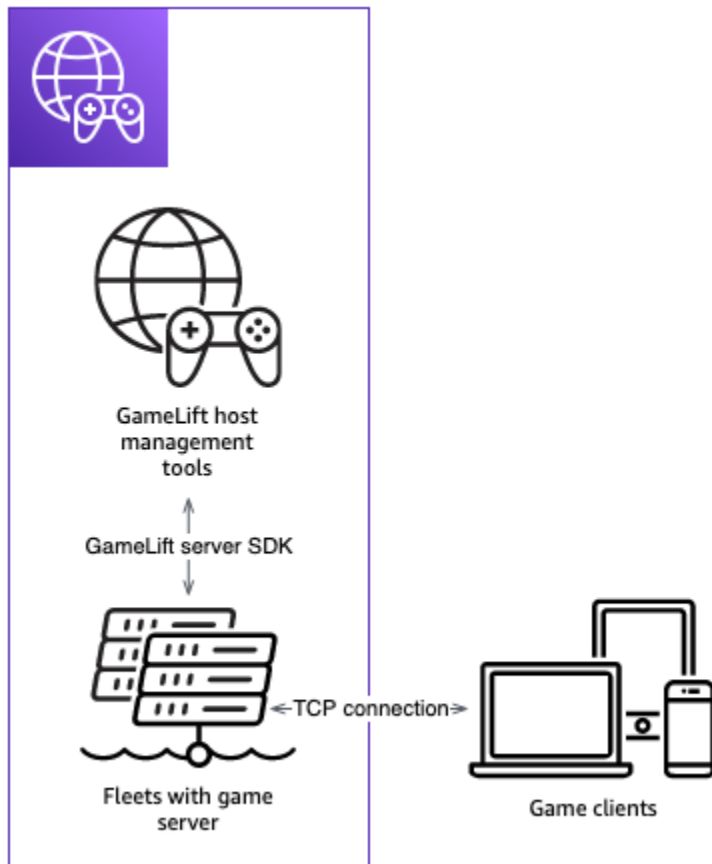
 Note

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 Servers プラグインについて説明します。

以下のタスクを完了して、Unity ゲームを Amazon GameLift Servers と統合してください。

- [Amazon GameLift Servers と Unity ゲームサーバープロジェクトを統合する](#)
- [Amazon GameLift Servers と Unity ゲームクライアントプロジェクトを統合する](#)

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、ゲームサーバーのあるフリートが Amazon GameLift Servers にデプロイされています。ゲームクライアントはゲームサーバーと通信し、ゲームサーバーは Amazon GameLift Servers と通信します。



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 Servers プラグインについて説明します。

Unity 用 Amazon GameLift Servers プラグインには、ゲームと Amazon GameLift Servers を統合するための基本を探索するために使用できるサンプルゲームが含まれています。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 Servers Local を使用してローカルでテストします。

前提条件

- [AWS ユーザーアカウントを設定する](#)
- [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 Servers 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 Servers] 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 Servers] を選択し、[Windows サンプルサーバービルド設定を適用] または [MacOS サンプルサーバービルド設定を適用] 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 Servers 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 Servers] を選択し、[Windows サンプルクライアントビルド設定を適用] または [MacOS サンプルクライアントビルド設定を適用] 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、[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 Servers 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。次の手順では、ゲームを実行し、ゲームが Amazon GameLift Servers とどのように連携するかを確認します。

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 Servers プラグインについて説明します。

Amazon GameLift Servers 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. [ローカルテスト] ウィンドウで、[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。
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 Servers ローカルログウィンドウを開きます。Windows には、ゲームサーバーと Amazon GameLift Servers Local との間で送信されるメッセージを含む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 Servers サンプルゲーム] で、電子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 Servers サンプルゲーム] で、[スタート] 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: playerId
0 joined
2021-09-15T19:58:0335 PID:20728 Log SERVER: Frame: 8696 Connection accepted: playerId
1 joined
2021-09-15T19:58:0338 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from playerId 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 playerId 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 Servers ローカルログのサンプル

```
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 Servers 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、[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、[ローカルテスト] ウィンドウで [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。

Amazon GameLift Servers でゲームサーバーの統合

ホスティングリソースでデプロイおよび実行されている場合、カスタムゲームサーバーは Amazon GameLift Servers とやり取りできる必要があります。このセクションのトピックでは、ゲームサーバープロジェクトをサーバー SDK for Amazon GameLift Servers と統合して、必須およびオプションの機能を追加する方法について説明します。このセクションには、任意の言語向けサーバー SDK で使用できる一般的な手順が含まれています。また、Unreal Engine および Unity のゲームプロジェクトを統合するための詳細なガイダンスも含まれています。

トピック

- [サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)
- [ゲームエンジン用のサーバー SDK を統合する](#)

サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。

このトピックでは、ゲームサーバーコードに追加する必要があるサーバー SDK 機能に関する一般的なガイダンスを提供します。この機能により、ゲームサーバープロセスは Amazon GameLift Servers サービスと通信できます。Amazon GameLift Servers マネージド EC2 フリート、マネージドコンテナフリート、または Anywhere フリートにデプロイする予定のゲームサーバーには、以下の手順を使用します。

ゲームサーバープロセスは Amazon GameLift Servers サービスと通信して、サービスから指示を受け取り、サーバープロセスのヘルスとゲームセッションのステータスを報告します。ゲームホスティングソリューションコンポーネント (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 Servers) 間のインタラクションの詳細については、[Amazon GameLift Servers とゲームクライアント / サーバーのやり取り](#)を参照してください。

サーバー SDK の入手

ゲームをホストする準備を整えるには、サーバー SDK for Amazon GameLift Servers をゲームサーバープロジェクトに追加します。サーバー SDK には複数の言語が用意されています。サーバー SDK を含むゲームサーバーのツールサポートの詳細については、「[Amazon GameLift Servers 開発用ツールを取得する](#)」を参照してください。

開発言語用サーバー SDK を github.com/amazon-gamelift からダウンロードします。サーバー SDK バージョン、プラグイン、その他の開発ツールのリポジトリがあります。

Note

Unreal Engine または Unity を使用して開発している場合は、これらのゲームエンジンの Amazon GameLift Servers プラグインをダウンロードします。サーバー SDK は組み込まれており、すぐに使用できます。詳細については、以下のリンクを参照してください。

- Unreal Engine ([プラグインのダウンロード](#)) ([統合ガイド](#))
- Unity ([プラグインのダウンロード](#)) ([統合ガイド](#))

サーバー SDK API リファレンス

- [C++ サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)
- [C# サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)
- [C++ \(Unreal\) サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)
- [Go サーバー SDK for Amazon GameLift Servers -- アクション](#)

サーバープロセスを初期化する

Amazon GameLift Servers サービスとの通信を確立し、ゲーム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 Servers 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 Servers API クライアントを初期化する。Amazon GameLift Servers マネージド EC2 フリートで実行するゲームサーバーを準備する場合は、パラメータなしでデフォルト `InitSDK()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) を使用します。API クライアントは、Amazon GameLift Servers サービスへの接続を自動的に処理します。

- Note** Amazon GameLift Servers Anywhere フリートで使用するゲームサーバーを準備する場合:

次の `ServerParameters` で `InitSdk()` を呼び出して Amazon GameLift Servers API クライアントを初期化します。

- ゲームサーバーへの接続に使用される WebSocket 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 Servers Anywhere コンピューティングを含むフリートの ID。
- Amazon GameLift Servers オペレーション [GetComputeAuthToken](#) によって生成された認可トークン。

2. サーバードロセスがゲームセッションをホストできる状態になったことを、サービスに通知します。ProcessReady() ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) を次の ProcessParameters を使用して呼び出します。各ゲームサーバードロセスはProcessReady() を 1 回だけ呼び出す必要があります。

- サーバードロセスが使用するポート番号。サーバードロセスはゲームセッションを開始すると、Amazon GameLift Servers サービスへのポートを提供し、ゲームセッション情報を更新します。ゲームはこの情報を取得してゲームクライアントに提供し、これを使用してサーバードロセスに接続してゲームセッションに参加できます。
- Amazon GameLift Servers に保存させたいファイルの場所。これには、ゲームセッションログや、ゲームセッション中にサーバードロセスが生成するその他のファイルが含まれる場合があります。Amazon GameLift Serversは、これらのファイルをサーバードロセスが実行されているコンピューティングに一時的に保存しますが、これらはインスタンスがシャットダウンするまでのみ使用できます。これらのファイルにアクセスするには、[\[Amazon GameLift Servers コンソール\]](#) 経由または Amazon GameLift Servers API オペレーション [GetGameSessionLogUrl\(\)](#) を呼び出します。

i Amazon GameLift Servers マネージドコンテナフリートで使用するゲームサーバーを準備する場合:

コンテナフリートのログパラメータを指定する必要はありません。代わりに、ゲームセッションやその他のログデータを標準出力に送信します。コンテナフリートは、すべてのコンテナ標準出力をログストリームとして自動的にキャプチャします。

- Amazon GameLift Serversがゲームサーバードロセスにメッセージまたはプロンプトを送信できるようにする次のコールバック関数。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、ProcessParameters([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) を参照してください。
- (オプション) onHealthCheck – Amazon GameLift Servers は、この関数を定期的呼び出して、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 Servers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 Servers 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 Servers 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他の AWS リソースに安全にアクセスできるようにすることもできます。詳細については、「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」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

ゲームサーバーにコードを追加してコールバック関数 `onHealthCheck()` を実装します。Amazon GameLift Servers は定期的にこのコールバックメソッドを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 Servers 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 Servers はサーバープロセスを正常とみなします。

Amazon GameLift Servers 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、サービス 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。サービス 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。ファイルのロケーションを取得するには、`GetComputeCertificate()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) 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 Servers はこのコールバックを呼び出して、サーバープロセスでゲームセッションを開始します。

`onStartGameSession` 関数は [GameSession](#) 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がプレイヤーを受け入れる準備が整ったら、Amazon GameLift Servers サービスに通知します。サーバー API オペレーション `ActivateGameSession()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) を呼び出します。呼び出しの成功に応じて、サービス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している場合は、新しいプレイヤーがゲームサーバーに接続するときにそのプレイヤーを検証するコードを追加します。Amazon GameLift Servers 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、ゲームセッションに参加しようとするゲームクライアントにはプレイヤーセッション ID が含まれている必要があります。Amazon GameLift Servers は、ゲームが [StartGameSessionPlacement\(\)](#) または [StartMatchmaking\(\)](#) を呼び出して新しいゲームセッションを開始すると、この ID を生成します。これらのリクエストでは、ゲームセッションのオープンスロットがプレイヤーセッション用に予約されます。

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、プレイヤーセッション ID が付いた状態で `AcceptPlayerSession()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) が呼び出されます。それに応じて、Amazon GameLift Servers 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 Servers がプレイヤーセッ

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 Servers 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 Servers 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の切断を検出するたびに実行する必要があります。Amazon GameLift Servers はこの通知を使用して、ゲームセッションの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 が付いた状態で、コードに、サーバー API オペレーション `RemovePlayerSession()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) を追加します。

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 Servers 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、ゲームセッションの完了後に各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](#)) ([Go](#)) 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 Servers に通知します。Amazon GameLift Servers はゲームセッションのステータスと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。

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 Servers サービス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。この通知は、サーバープロセスが異常を一貫して報告する場合、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送信されます。Amazon GameLift Servers は、容量のスケールダウンイベントの一部として、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できます。スポットインスタンスの中断は 2 分間の通知を提供します。これにより、サーバープロセスがプレイヤーを適切に切断し、ゲーム状態データを保持し、その他のクリーンアップタスクを実行する時間が提供されます。

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()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) 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。

- ゲームサーバー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](#)) ([Go](#)) を呼び出します。Amazon GameLift Servers 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](#)) ([Go](#)) 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 Servers サービスに通知します。その後、サービス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。

ゲームエンジン用のサーバー SDK を統合する

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 (O3DE、Unreal Engine、Unity など) で、マネージド Amazon GameLift Servers 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 Servers 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 GameLift Servers 開発用ツールを取得する](#)」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 Servers 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。

- [サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#) – Amazon GameLift Servers をゲームサーバーに統合する詳しい手順です。
- [Amazon GameLift Servers ゲームクライアント機能の統合する](#) – 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。

Unreal Engine

ゲームサーバー

Amazon GameLift Servers でホストするゲームサーバーを準備するために、[サーバー SDK for Amazon GameLift Servers for Unreal Engine](#) 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 Servers コードの追加については、「[Amazon GameLift Servers 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

使用可能なゲームセッションの検索、新しいゲームセッションの作成、プレイヤーのゲームへの追加など、ゲームクライアントやゲームサービスが Amazon GameLift Servers サービスとやり取りできるようにします。コアクライアント機能については、「[AWS SDK for C++](#)」で説明されています。Amazon GameLift Servers を Unreal Engine ゲームプロジェクトに統合するには、「[Amazon GameLift Servers ゲームクライアント機能の統合する](#)」を参照してください。

Unity

ゲームサーバー

Amazon GameLift Servers でホストするゲームサーバーを準備するために、[サーバー SDK for Amazon GameLift Servers 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 の設定と Amazon GameLift Servers コードの追加については、「[Amazon GameLift Servers を Unity 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

使用可能なゲームセッションの検索、新しいゲームセッションの作成、プレイヤーのゲームへの追加など、ゲームクライアントやゲームサービスが Amazon GameLift Servers サービスとやり取りできるようにします。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 Servers を Unity ゲームプロジェクトに統合するには、「[Amazon GameLift Servers ゲームクライアント機能の統合する](#)」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 Servers SDK リストについては、「[the section called “開発用ツールを取得する”](#)」を参照してください。

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

Amazon GameLift Servers SDK for Unreal Engine をゲームプロジェクトに統合して、サーバーの SDK 機能セット全体にアクセスする方法について説明します。

ヒント

デプロイを高速化するには、Unreal Engine 用の Amazon GameLift Servers スタンドアロンプラグインを試してください。最小限のセットアップでゲームサーバーをすばやくデプロイするためのガイド付き UI ワークフローを提供するため、ゲームコンポーネントを実際に試

すことができます。「[Unreal Engine 用 Amazon GameLift Servers プラグイン](#)」を参照してください。

その他のリソース:

- [C++ \(Unreal\) サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)
- [Amazon GameLift Servers 開発用ツールを取得する](#)

サーバー SDK for Unreal をインストールする

[GitHub](#) からオープンソース Amazon GameLift Servers SDK for Unreal Engine を取得します。リポジトリの readme ファイルには、前提条件とインストール手順が含まれています。

ビルドターゲットとモジュールルールを設定する

ゲームプロジェクトファイルを変更して、Amazon GameLift Servers で使用するビルドコンポーネントを適切に生成します。

クライアントとサーバーのビルドターゲットを追加するには:

1. ゲームプロジェクトのコードファイルを開き、ファイル `.../Games/[your application name]Source/[your application name]Target.cs` ファイルを見つけます。例: `.../Source/GameLiftUnrealAppTarget.cs`。(Visual Studio を使用する場合は、プロジェクトの `.sln` ファイルを開きます。)
2. このファイルをコピーして、Source/ ディレクトリに 2 つの新しいターゲットファイルを作成します。
 - クライアントターゲット - 新しいファイルの名前を `[your application name]Client.Target.cs` に変更します。次のサンプルコードに示すように、コンテンツを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。

```
using UnrealBuildTool;
using System.Collections.Generic;

public class GameLiftUnrealAppClientTarget : TargetRules
{
    public GameLiftUnrealAppClientTarget ( TargetInfo Target ) : base
    ( Target )
    {
```

```

        Type = TargetType.Client;
        DefaultBuildSettings = BuildSettingsVersion.V2;
        IncludeOrderVersion = EngineIncludeOrderVersion.Unreal5_1;
        ExtraModuleNames.Add( "GameLiftUnrealApp");
    }
}

```

- サーバータゲット - 新しいファイルの名前を *[your application name]*Server.Target.cs に変更します。次のサンプルコードに示すように、コンテンツを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。

```

using UnrealBuildTool;
using System.Collections.Generic;

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

```

3. プロジェクトファイルを再生成します。Visual Studio を使用している場合は、ゲームプロジェクトの .uproject ファイルを右クリックし、[Visual Studio プロジェクトファイルの生成] を選択します。

ゲームプロジェクトモジュールルールを更新するには:

ゲームプロジェクトのモジュールルールを更新して、プラグインに依存します。

1. ゲームプロジェクトのコードファイルを開き、ファイル .../Games/*[your application name]*Source/*[your application name]*.Build.cs ファイルを見つけます。例えば、.../Source/GameLiftUnrealApp.Build.cs などです。(Visual Studio を使用する場合は、プロジェクトの .sln ファイルを開きます。)
2. 次のサンプルコードに示すように、ModuleRules クラスを見つけて更新します。

```

using UnrealBuildTool;

```

```

public class GameLiftUnrealApp : ModuleRules
{
    public GameLiftUnrealApp ( ReadOnlyTargetRules Target ) : base ( Target )
    {
        PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;
        PublicDependencyModuleNames.AddRange( new string[] { "Core",
"CoreUObject", "Engine", "InputCore", "HeadMountedDisplay",
"EnhancedInput" });
        // Add the following section
        if (Target.Type == TargetType.Server)
        {
            PublicDependencyModuleNames.Add("GameLiftServerSDK");
        }
        else
        {
            PublicDefinitions.Add("WITH_GAMELIFT=0");
        }
        bEnableExceptions = true;
    }
}

```

3. 新しいターゲットファイルを作成し、モジュールルールを変更したら、ゲームプロジェクトを再構築します。

サーバーコードにゲームホスティング機能を追加する

サーバー SDK のインストールとセットアップが完了したら、次のステップとして、ゲームホスティング機能をサーバーコードに統合します。サーバー SDK を使用すると、ゲームサーバーは Amazon GameLift Servers サービスと通信し、ゲームセッションの指示を受け取り、ステータスと状態を報告し、その他のアクションを実行できます。

このトピックでは、Amazon GameLift Servers でゲームをホストするために必要な最小限の機能を追加するサンプルコードを提供します。

ステップ 1: **GameMode** ヘッダーファイルを更新する

1. ゲームプロジェクトのコードファイルを開き、ファイル *Your-application-name*GameMode.h ファイルを見つけます。例: GameLiftUnrealAppGameMode.h。Visual Studio を使用する場合は、ゲームプロジェクトの .sln ファイルを開きます。

2. ヘッダーファイルを、次のサンプルコードを含むように変更します。GameLiftUnrealApp を独自のアプリケーション名に必ず置き換えてください。

GameMode.h のサンプルコード

```
// 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 "GameLiftUnrealAppGameMode.generated.h"

struct FProcessParameters;

DECLARE_LOG_CATEGORY_EXTERN(GameServerLog, Log, All);

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

public:
    AGameLiftUnrealAppGameMode();

protected:
    virtual void BeginPlay() override;

private:
    void InitGameLift();

private:
    TSharedPtr<FProcessParameters> ProcessParameters;
};
```

ステップ 2: ゲームサーバーコードに必要なサーバー SDK 呼び出しを追加

このセクションのサンプルコードを使用して、Amazon GameLift Servers で使用するゲームサーバーコードを統合します。コードの動作の詳細については、「[サーバープロセスを初期化する](#)」および「[C++ \(Unreal\) サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)」を参照してください。

 Note

WITH_GAMELIFT プリプロセッサフラグには 2 つの目的があります。

- Amazon GameLift Servers バックエンド API コールを Unreal サーバービルドのみに制限します
- さまざまな Unreal ビルドターゲット間の互換性を確保する

1. 関連するソースファイルの *Your-application-name*GameMode.cpp を開きます。例: GameLiftUnrealAppGameMode.cpp。
2. コードを次のサンプルコードに合わせて変更します。「GameLiftUnrealApp」のインスタンスは必ず独自のアプリケーション名に置き換えてください。

提供されたコードサンプルは、Amazon GameLift Servers との統合に必要な要素を追加する方法を示しています。具体的には次のとおりです。

- Amazon GameLift Servers API クライアントを初期化します。
- OnStartGameSession、OnProcessTerminate、onHealthCheck など、Amazon GameLift Servers 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。
- ゲームセッションをホストする準備ができた時点でProcessReady() を呼び出し、Amazon GameLift Servers サービスに通知します。

GameMode.cpp のサンプルコード

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

#include "GameLiftUnrealAppGameMode.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);

AGameLiftUnrealAppGameMode::AGameLiftUnrealAppGameMode() :
    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 AGameLiftUnrealAppGameMode..."));
}

void AGameLiftUnrealAppGameMode::BeginPlay()
{
    Super::BeginPlay();

#ifdef WITH_GAMELIFT
    InitGameLift();
#endif
}

void AGameLiftUnrealAppGameMode::InitGameLift()
{
#ifdef 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;
    }
#endif
}
```

```
}

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.
        FString TimeString = FString::FromInt(std::time(nullptr));
        FString ProcessId = "ProcessId_" + TimeString;
        ServerParametersForAnywhere.m_processId = TCHAR_TO_UTF8(*ProcessId);
    }

    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);
    }

    FString glAnywhereAwsRegion = "";
    if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereAwsRegion="),
glAnywhereAwsRegion))
    {
        ServerParametersForAnywhere.m_awsRegion =
TCHAR_TO_UTF8(*glAnywhereAwsRegion);
    }

    FString glAnywhereAccessKey = "";
    if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereAccessKey="),
glAnywhereAccessKey))
    {
        ServerParametersForAnywhere.m_accessKey =
TCHAR_TO_UTF8(*glAnywhereAccessKey);
    }

    FString glAnywhereSecretKey = "";
    if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereSecretKey="),
glAnywhereSecretKey))
    {
        ServerParametersForAnywhere.m_secretKey =
TCHAR_TO_UTF8(*glAnywhereSecretKey);
    }

    FString glAnywhereSessionToken = "";
    if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereSessionToken="),
glAnywhereSessionToken))
    {
        ServerParametersForAnywhere.m_sessionToken =
TCHAR_TO_UTF8(*glAnywhereSessionToken);
    }

    UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_YELLOW);
```

```

        UE_LOG(GameServerLog, Log, TEXT(">>>> WebSocket 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, Log, TEXT(">>>> Aws Region: %s"),
*ServerParametersForAnywhere.m_awsRegion);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Access Key: %s"),
*ServerParametersForAnywhere.m_accessKey);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Secret Key: %s"),
*ServerParametersForAnywhere.m_secretKey);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Session Token: %s"),
*ServerParametersForAnywhere.m_sessionToken);
        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, Amazon GameLift Servers 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. Amazon GameLift Servers 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 Amazon GameLift Servers we are indeed going to
    shutdown.
    ProcessParameters->OnTerminate.BindLambda([=]()
    {
        UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
        // First call ProcessEnding()
        FGameLiftGenericOutcome processEndingOutcome = GameLiftSdkModule-
>ProcessEnding();
        // Then call Destroy() to free the SDK from memory
        FGameLiftGenericOutcome destroyOutcome = GameLiftSdkModule->Destroy();
        // Exit the process with success or failure
        if (processEndingOutcome.IsSuccess() && destroyOutcome.IsSuccess()) {
            UE_LOG(GameServerLog, Log, TEXT("Server process ending successfully"));
        }
        else {
            if (!processEndingOutcome.IsSuccess()) {
                const FGameLiftError& error = processEndingOutcome.GetError();
                UE_LOG(GameServerLog, Error, TEXT("ProcessEnding() failed. Error:
%s"),
                    error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
*error.m_errorMessage);
            }
            if (!destroyOutcome.IsSuccess()) {
                const FGameLiftError& error = destroyOutcome.GetError();

```

```
        UE_LOG(GameServerLog, Error, TEXT("Destroy() failed. Error: %s"),
            error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
            *error.m_errorMessage);
    }
}

});

//This is the HealthCheck callback.
//Amazon GameLift Servers 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. Amazon GameLift
Servers 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 Amazon GameLift Servers where to find game session
log files.
```

```
//At the end of a game session, Amazon GameLift Servers uploads everything in the
specified
//location and stores it in the cloud for access later.
TArray<FString> Logfiles;
Logfiles.Add(TEXT("GameLiftUnrealApp/Saved/Logs/server.log"));
ProcessParameters->logParameters = Logfiles;

//The game server calls ProcessReady() to tell Amazon GameLift Servers 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
}
```

ステップ 3: ゲームプロジェクトを再構築する

- 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、アプリの /Games/ フォルダの下にプロジェクトを構築します。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。

ホスティング用にゲームサーバーをパッケージ化します。

ゲームサーバーコードが最小限のサーバー SDK 機能と統合されたため、Unreal Editor を使用してゲームサーバービルドをパッケージ化する準備が整いました。

ゲームサーバービルドをパッケージ化するには

1. Unreal Editor でゲームプロジェクトを開きます。
2. Unreal Editor の手順に従って、ゲームサーバーをパッケージ化します。
 - ターゲットプラットフォーム (Windows または Linux) を選択します。
 - サーバービルドターゲット (*[your application name]*Server) を選択します。

パッケージングプロセスでは、ゲームサーバーの実行可能ファイル *[your application name]*Server.exe が生成されます。

3. ホスティングリソースへのデプロイ用にゲームサーバービルドを準備します。ビルドには以下のファイルが含まれている必要があります。
 - ゲームサーバー実行可能ファイル
 - Unreal Engine バージョン 5.5 以前を使用している場合、Windows ビルドには次のファイルを含めます。これらのファイルは、Unreal Engine のソースビルド版に含まれています
 - VC_redist.x64.exe (UnrealEngine\Engine\Source\Programs\PrereqInstaller\Resources\VCRedist\)
 - UEPrereqSetup_x64.exe or UE5PrereqSetup_x64.exe (UnrealEngine\Engine\Extras\Redist\en-us\)
 - ゲームサーバーに必要なその他すべての依存関係。
 - 必要に応じて OpenSSL ライブラリ。ゲームサーバーが Amazon GameLift Servers サーバー SDK バージョン 5.3 以降と統合されている場合は、このステップをスキップできます。[最新のサーバー SDK バージョンは、こちらで入手できます。](#)

サーバー SDK for Unreal、バージョン 5.2 以前で作成されたゲームサーバーに必要な OpenSSL ライブラリ

Unreal でゲームサーバーをパッケージ化するときに使用したのと同じバージョンの OpenSSL ライブラリを含める必要があります。これらのライブラリはゲームエンジンソースにあります。場所は開発環境によって異なります。

Windows の場合:

- [ENGINE_ROOT_DIR]\Engine\Extras\ThirdPartyNotUE\libimobiledevice\x64\libssl-1_1-x64.dll
- [ENGINE_ROOT_DIR]\Engine\Extras\ThirdPartyNotUE\libimobiledevice\x64\libcrypto-1_1-x64.dll

Linux の場合:

- Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libssl.so.1.1
- Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libcrypto.so.1.1

OpenSSL ライブラリをゲームサーバー実行可能ファイルと同じディレクトリのゲームビルドパッケージにコピーします。

次の手順

Amazon GameLift Servers でホスティングするために必要な最小限の機能を備えたゲームサーバービルドを準備したので、次のステップを検討してください。

- テストと開発用の統合ゲームサーバーをデプロイします。Anywhere フリートを使用すると、ローカルマシンをホスティングリソースとして設定し、それを使用してゲームサーバーとゲームクライアントの接続をテストできます。クラウドベースのホスティングの場合は、ゲームサーバーをマネージド EC2 またはマネージドコンテナフリートにデプロイします。ガイダンスについては、以下のトピックを参照してください。
 - [Amazon GameLift Servers Anywhere を使用した反復型開発の準備](#)
 - [Amazon GameLift Servers Anywhere フリート](#)
 - [Amazon GameLift Servers マネージド EC2 フリート](#)
 - [Amazon GameLift Servers マネージドコンテナフリート](#)
- オプション機能を追加して、ゲームサーバー統合をカスタマイズします。たとえば、一意のプレイヤー IDs でプレイヤーセッションを追加したり、マッチメイキングバックフィルを設定したり、他の AWS リソース (データベースやコンテンツストレージサービスなど) へのゲームサーバーアクセスを管理したりできます。ガイダンスについては、以下のトピックを参照してください。
 - [サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)
 - [C++ \(Unreal\) サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)

- ゲームクライアントコンポーネントをカスタマイズして、ゲームセッションをリクエストし、接続情報を受信し、ゲームサーバーに直接接続してゲームをプレイします。ガイダンスについては、以下のトピックを参照してください。
- [Amazon GameLift Servers ゲームクライアント機能の統合する](#)

Amazon GameLift Servers を Unity プロジェクトに統合する

Amazon GameLift Servers SDK for Unity をゲームプロジェクトに統合してサーバー SDK 機能セット全体にアクセスする方法について説明します。

ヒント

デプロイを高速化するには、Unity 用の Amazon GameLift Servers スタンドアロンプラグインを試してください。最小限のセットアップでゲームサーバーをすばやくデプロイするためのガイド付き UI ワークフローを提供するため、ゲームコンポーネントを実際に試すことができます。「[Unity 用 Amazon GameLift Servers プラグイン \(サーバー SDK 5.x\)](#)」を参照してください。

その他のリソース:

- [C# サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)
- [the section called “開発用ツールを取得する”](#)

サーバー SDK for Unity をインストールする

[GitHub](#) から Unity 用のオープンソース Amazon GameLift Servers を取得します。リポジトリの readme ファイルには、前提条件とインストール手順が含まれています。

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

開発ワークステーションを Amazon GameLift Servers Anywhere ホスティングフリートとしてセットアップして、Amazon GameLift Servers 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イキングリクエストを Amazon GameLift Servers 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 Servers 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 Servers Anywhereでローカルテストを設定する](#)」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 Servers 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 Servers 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。

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

ゲームサーバーは Amazon GameLift Servers 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 Servers 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 を使用した Amazon GameLift Servers 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 Servers 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 Servers API クライアントを初期化します。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 [テスト用に Amazon GameLift Servers Anywhere フリートをセットアップする](#) 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 Servers 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 Servers サービスに通知します。

提供されたサンプルコードは、Amazon GameLift Servers サービスとの通信を確立します。また、Amazon GameLift Servers 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は、「[C# サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)」を参照してください。

サンプルコードは、「[Amazon GameLift Servers をゲームサーバーに追加する](#)」で説明されているように、必要な機能を追加する方法を示しています。サーバー SDK アクションの詳細については、「[C# サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)」を参照してください。

統合サンプルコード

```
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 Servers 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 an Amazon GameLift Servers Anywhere fleet.
        //They are not required for an Amazon GameLift Servers managed EC2 fleet.
```

```

ServerParameters serverParameters = new ServerParameters(
    websocketUrl,
    processId,
    hostId,
    fleetId,
    authToken);

//InitSDK establishes a local connection with an Amazon GameLift Servers agent
//to enable further communication.
var initSDKOutcome = GameLiftServerAPI.InitSDK(serverParameters);
if (initSDKOutcome.Success)
{
    //Implement callback functions
    ProcessParameters processParameters = new ProcessParameters(
        //Implement OnStartGameSession callback
        (gameSession) => {
            //Amazon GameLift Servers 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) => {
            //Amazon GameLift Servers 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
            //Amazon GameLift Servers 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 Amazon GameLift Servers
it is shutting down.
        GameLiftServerAPI.ProcessEnding();
    },
    () => {
        //Implement callback function OnHealthCheck
        //Amazon GameLift Servers 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 Amazon
GameLift Servers 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 Amazon GameLift Servers where to find
game session log files.
        //At the end of a game session, Amazon GameLift Servers 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 Amazon GameLift Servers 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 Servers 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);
        }
    }
}
```

次の手順

Amazon GameLift Servers でホスティングするために必要な最小限の機能を備えたゲームサーバービルドを準備したので、次のステップを検討してください。

- テストと開発用の統合ゲームサーバーをデプロイします。Anywhere フリートをを使用すると、ローカルマシンをホスティングリソースとして設定し、それを使用してゲームサーバーとゲームクライアントの接続をテストできます。クラウドベースのホスティングの場合は、ゲームサーバーをマネージド EC2 またはマネージドコンテナフリートにデプロイします。ガイダンスについては、以下のトピックを参照してください。
 - [Amazon GameLift Servers Anywhere を使用した反復型開発の準備](#)
 - [Amazon GameLift Servers Anywhere フリート](#)
 - [Amazon GameLift Servers マネージド EC2 フリート](#)
 - [Amazon GameLift Servers マネージドコンテナフリート](#)

- オプション機能を追加して、ゲームサーバー統合をカスタマイズします。たとえば、一意のプレイヤー IDs でプレイヤーセッションを追加したり、マッチメイキングバックフィルを設定したり、他の AWS リソース (データベースやコンテンツストレージサービスなど) へのゲームサーバーアクセスを管理したりできます。ガイダンスについては、以下のトピックを参照してください。
- [サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)
- [C++ \(Unreal\) サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)
- ゲームクライアントコンポーネントをカスタマイズして、ゲームセッションをリクエストし、接続情報を受信し、ゲームサーバーに直接接続してゲームをプレイします。ガイダンスについては、以下のトピックを参照してください。
- [Amazon GameLift Servers ゲームクライアント機能の統合する](#)

デプロイ用にゲームサーバービルドをパッケージ化する

ゲームサーバーソフトウェアをホスティングリソースにデプロイする準備をします。デプロイされると、ソフトウェアが各ホスティングリソースにインストールされ、1 つ以上のゲームサーバープロセスが起動され、プレイヤーのゲームセッションをホストする準備が整います。

ゲームサーバービルドの準備は、使用している Amazon GameLift Servers ホスティングオプションのタイプによって異なります。[Amazon GameLift Servers でゲームサーバーの統合](#)で説明されているように、すべてのゲームサーバービルドは Amazon GameLift Servers のサーバー SDK と統合する必要があります。

このセクションのトピックでは、以下のシナリオでソフトウェアのデプロイを準備する方法を説明します。

- マネージド EC2 ホスティングの場合は、サーバーソフトウェアをパッケージ化し、デプロイのために Amazon GameLift Servers にアップロードします。
- マネージドコンテナホスティングの場合は、サーバーソフトウェアを使用してコンテナイメージを構築し、Amazon Elastic Container Registry に保存してデプロイします。
- Amazon GameLift Servers Anywhere でホスティングする場合は、必要に応じてサーバーソフトウェアをパッケージ化し、独自のホスティングリソースにインストールします。

Note

Amazon GameLift Servers Realtime 設定済みスクリプトをデプロイする場合は、「[Amazon GameLift Servers Realtime 用スクリプトのアップロード](#)」を参照してください。

トピック

- [Amazon GameLift Servers のゲームサーバービルドを作成する](#)
- [Amazon GameLift Servers 用のコンテナイメージを構築する](#)

Amazon GameLift Servers のゲームサーバービルドを作成する

ゲームサーバーを Amazon GameLift Servers と統合した後 ([Amazon GameLift Servers でホストするゲームを準備します](#) を参照)、ホスティング用のコンピューティングリソースにゲームサーバーソフトウェアをインストールします。このプロセスは、使用している Amazon GameLift Servers ホスティングのタイプによって異なります。

マネージドホスティング用にデプロイする

Amazon GameLift Servers のマネージド EC2 ホスティングを使用している場合、ゲームサーバーソフトウェアをパッケージ化し、Amazon GameLift Servers にアップロードする必要があります。マネージドフリートを作成すると、Amazon GameLift Servers はそのフリートを各フリートインスタンスに自動的にデプロイします。

このセクションのトピックでは、アップロード用にビルドファイルをパッケージ化し、オプションのビルドインストールスクリプトを作成してから、[AWS Command Line Interface \(AWS CLI\)](#) または AWS SDK を使用してファイルをアップロードする方法について説明します。

Anywhere ホスティング用にデプロイする

フルマネージドホスティングで Amazon GameLift Servers Anywhere フリートを使用する場合、フリート内の各コンピューティングにゲームサーバーソフトウェアをインストールし、最新の状態に保つのはユーザーの責任です。

合されたゲームサーバープロセスの実行が開始されると、Amazon GameLift Servers サービスとの通信が自動的に初期化され、確立されます。サーバープロセスは、Amazon GameLift Servers からのプロンプトでゲームセッションを開始し、アクティビティをサービスに報告します。

トピック

- [ゲームビルドファイルをパッケージ化する](#)
- [ビルドインストールスクリプトを追加する](#)
- [マネージドホスティング用のAmazon GameLift Serversビルドリソースを作成する](#)

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 Servers 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。このプロセスは、EC2 マネージドフリートでホスティングする場合の要件であり、Anywhere フリートでホスティングする場合のベストプラクティスです。ビルドディレクトリには、ゲームサーバーの実行とゲームセッションのホストに必要なすべてのコンポーネントが含まれている必要があります。これには次のものが含まれる場合があります。

- 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 GameLift Servers開発用ツールを取得する](#)」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++ 用サーバー SDK for Amazon GameLift Servers 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 Servers 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です。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 Servers はフリート作成の一環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を設定して、他の AWS サービスでリソースに安全にアクセスできます。これを行う方法については、「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」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、必要な依存関係がすべて含まれていることと、インストールスクリプトが正確であることを検証します。

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 Servers 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 フリーッ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 フリーットの 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。
- Linux では、Amazon GameLift Servers は bash などの一般的なシェルインタープリタ言語をサポートします。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 Servers 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 Servers 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 Servers はサービス VPC で実行されているため、ユーザーに代わって AWS Identity and Access Management (IAM) ロールを引き受けられるように、Amazon GameLift Servers にアクセス許可を付与する必要があります。Amazon GameLift Servers がロールを引き受けることを許可するには、AWS 管理ポリシーを含むロールを作成し CloudWatchAgentAdminPolicy、フリートの作成時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

```

マネージドホスティング用のAmazon GameLift Serversビルドリソースを作成する

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。

どちらの方法でも、Amazon GameLift Servers は一意のビルド ID とその他のメタデータを持つ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 Servers 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 Servers フリートにデプロイできます。詳細については、[Amazon GameLift Servers マネージド EC2 フリートを作成する](#) を参照してください。Amazon GameLift Servers 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を含む任意の場所に保存された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 Servers 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 server SDK for Amazon GameLift Servers 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 Servers サーバー SDK のバージョン。Amazon GameLift Servers の値を指定しない場合、4.0.2 はデフォルト値を使用します。正しくないサーバー SDK バージョンを指定すると、Amazon GameLift Servers サービスへの接続を確立するために InitSdk を呼び出した際に、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_2022 \  
  --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 Servers 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 Servers 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

ビルドファイルを Amazon S3 に保存し、そこから Amazon GameLift Servers にアップロードできます。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 Servers 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、 の S3 バケットにアップロードします AWS アカウント。バケットラベルとファイル名を書き留めておきます。Amazon GameLift Serversビルドの作成時に必要になります。
2. Amazon GameLift Servers にビルドファイルへのアクセスを許可します。「[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 Servers 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 Servers 向け 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 Servers 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 Servers 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. [作成] を選択します。

Amazon GameLift Servers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 server SDK for Amazon GameLift Servers 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 Servers と統合するために使用したサーバー SDK for Amazon GameLift Servers のバージョンです。Amazon GameLift Servers の値を指定しない場合、4.0.2 はデフォルト値を使用します。
- 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_2022 \  
  --storage-location  
  "Bucket"="my_game_build_files","Key"="mygame_build_101.zip","RoleArn"="arn:aws:iam::111111111111: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。

Amazon GameLift Servers 用のコンテナイメージを構築する

このトピックでは、Amazon GameLift Servers で使用する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する方法について説明します。ゲームサーバーコンテナイメージには、ゲームサーバー実行可能ファイルと実行する必要がある依存関係が含まれます。ゲームサーバーコンテナイメージは、Amazon GameLift Servers マネージドコンテナホスティングソリューションで使用されます。完全なソリューションの構築の詳細については、以下を参照してください。

- [Amazon GameLift Servers マネージドコンテナでホスティングするための開発ロードマップ](#)
- [Amazon GameLift Servers におけるコンテナの仕組み](#)

ゲームサーバーコンテナイメージを Amazon GameLift Servers コンテナフリートにデプロイする準備を整えるには、次のタスクを実行します。これらのタスクを開始する前に、ゲームサーバーコードを Amazon GameLift Servers サーバー SDK と統合します。

トピック

- [ゲームサーバーコンテナイメージを作成する](#)

- [コンテナイメージを Amazon ECR にプッシュする](#)

ゲームサーバーコンテナイメージを作成する

Linux ベースのプラットフォームで作業するとき、または Docker がインストールされている Windows Subsystem for Linux (WSL) を使用するとき、次の手順に従います。

ゲームサーバーコンテナイメージを作成するには

1. ゲームサーバーソフトウェアを使用して作業ディレクトリを準備します。ローカルマシンで、作業ディレクトリを作成して、ゲームサーバーコンテナのファイルを整理します。コンテナイメージは、ホスティング用の Amazon GameLift Servers リソースにコンテナをデプロイするときに、このファイル構造を使用します。例えば、次のようになります。

```
[~/]$ mkdir -p work/glc/gamebuild && cd work && find .  
.  
./glc  
./glc/gamebuild
```

Note

この機能を試みているが、ゲームサーバーをまだ構築していない場合は、GitHub で利用可能なサンプルゲームサーバー [SimpleServer](#) をお試しください。

2. 提供されたテンプレートを使用して新しい Dockerfile を作成します。
3. Dockerfile テンプレートの指示に従って、独自の使用のために更新します。
 - 必要に応じてベースイメージを更新します。
 - ゲームサーバービルドの環境変数を設定します。
4. コンテナイメージを構築します。独自のリポジトリ名を指定して、`docker build` を実行します。例えば、次のようになります。

```
[~/work/glc]$ docker build -t <local repository name>:<optional tag> .
```

この例に示すように、`docker images` コマンドを使用してリポジトリとイメージ ID を表示できます。

ゲームサーバーコンテナイメージの Dockerfile テンプレート

このテンプレートには、ゲームサーバーコンテナが Amazon GameLift Servers 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。ゲームサーバーに必要なコンテンツを変更します。

```
# Base image
# -----
# Add the base image that you want to use,
# Make sure to use an image with the same architecture as the
# Instance type you are planning to use on your fleets.
FROM public.ecr.aws/amazonlinux/amazonlinux
#
# Game build directory
# -----
# Add your gamebuild directory to the env variable below.
# The game build provided here needs to be integrated with server sdk for Amazon
GameLift Servers.
ENV GAME_BUILD_DIRECTORY="<ADD_GAME_BUILD_DIRECTORY>" \
#
# Game executable and launch parameters
# -----
# Add the relative path to your executable in the 'GAME_EXECUTABLE' env variable
below.
# The game build provided over here needs to be integrated with server sdk for Amazon
GameLift Servers.
# This template assumes that the executable path is relative to the game build
directory.
# Add any launch parameters to pass into your executable in the 'LAUNCH_PARAMS' env
variable below.
# Add 'HOME_DIR' to identify where the game executable and logs exist.
GAME_EXECUTABLE="<ADD NAME OF EXECUTABLE WITHIN THE GAME DIRECTORY>" \
LAUNCH_PARAMS=<ADD LAUNCH PARAMETERS> \
HOME_DIR="/local/game" \

# Install dependencies as necessary
RUN yum install -y shadow-utils

RUN mkdir -p $HOME_DIR
COPY ./$GAME_BUILD_DIRECTORY/ $HOME_DIR

# Change directory to home
```



```
WORKDIR $HOME_DIR

# Set up for 'gamelift' user
RUN useradd -m gamelift && \
    echo "gamelift ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers && \
    chown -R gamelift:gamelift $HOME_DIR

# Add permissions to game build
RUN chmod +x ./$GAME_EXECUTABLE

USER gamelift

# Check directory before starting the container
RUN ls -lhrt .

# Check path before starting the container
RUN echo $PATH

# Start the game build
ENTRYPOINT ["/bin/sh", "-c", "./$GAME_EXECUTABLE", "$LAUNCH_PARAMS"]
```

コンテナイメージを Amazon ECR にプッシュする

Amazon GameLift Servers にデプロイするコンテナイメージを作成したら、そのイメージを Amazon ECR のパブリックリポジトリまたはプライベートルポジトリに保存します。このリポジトリには URI 値が割り当てられ、Amazon GameLift Servers を使用してコンテナフリートにデプロイするためのイメージのスナップショットを作成します。

Note

Amazon ECR プライベートルポジトリがまだない場合は、[作成します](#)。

コンテナイメージを Amazon ECR にプッシュする

1. Amazon ECR 認証情報を取得します。コンテナイメージを Amazon ECR にプッシュする前に、まず一時的な形式で AWS 認証情報を取得し、Docker に提供します。Docker では、ログインにこれらの認証情報が必要です。

```
[~/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

2. 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。
3. コンテナイメージに Amazon ECR タグを付与します。

Example

```
[~/work/glc]$ docker tag <IMAGE ID from above> <Amazon ECR private repository URI>:<optional tag>
```

4. 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 Servers とゲームクライアント / サーバーのやり取り

Amazon GameLift Servers ホスティングソリューションのコンポーネントは、プレイヤーの需要に応じてゲームセッションを実行するために、特定の 방법으로相互に連携します。このトピックでは、ゲームサーバーが Amazon GameLift Servers マネージド EC2 フリート、セルフマネージド Amazon GameLift Servers Anywhere フリート、またはハイブリッドソリューションでホストされているときに、コンポーネントが相互に通信する方法について説明します。

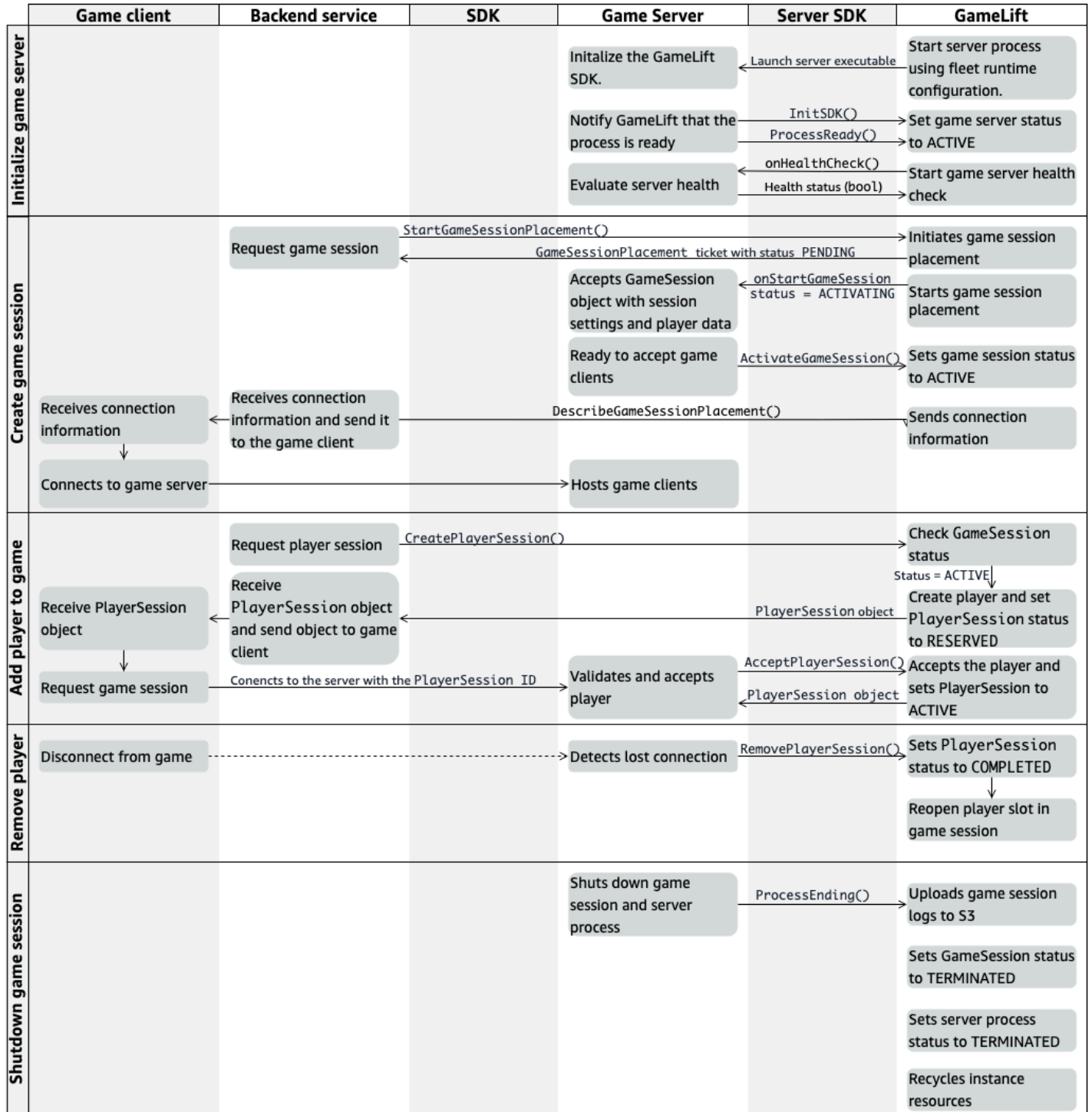
ホスティングソリューションコンポーネントには、ゲームサーバー、Amazon GameLift Servers サービス、クライアント側のバックエンドサービス、ゲームクライアントが含まれます。ゲームサーバーは、[Amazon GameLift Servers サーバー SDK](#) を使用して Amazon GameLift Servers サービスと連携します。バックエンドサービスは、[Amazon GameLift Servers サービス API](#) (AWS SDK の一部) を使用して、ゲームクライアントに代わってサービスとやり取りします。ゲームセッションに参加すると、ゲームクライアントはゲームセッションの一意の IP アドレスとポート番号を使用してゲームセッションに直接接続します。

トピック

- [連携図](#)
- [連携の動作](#)

連携図

次の図は、Amazon GameLift Servers サービスがプレイヤーの需要に応じてゲームサーバーの可用性のステータスを追跡し、ゲームセッションを開始できるように、ゲームホスティングコンポーネントがどのように相互に連携するかを示しています。



連携の動作

以下のセクションでは、主な各種連携におけるイベントの順番について説明します。

ゲームサーバープロセスの初期化

スタートアップ時、ゲームサーバープロセスは Amazon GameLift Servers サービスとの通信を確立し、ゲームセッションをホストする準備ができている状態として報告します。

1. ゲームサーバー実行可能ファイルの新しいプロセスが、ホスティングリソースで実行を開始します。
2. ゲームサーバープロセスは、次のサーバー SDK オペレーションを順番に呼び出します。
 - a. `InitSDK()` は、サーバー SDK を初期化し、サーバープロセスを認証して Amazon GameLift Servers サービスとの通信を確立します。
 - b. `ProcessReady()` は、ゲームセッションをホストする準備状況を伝えます。この呼び出しは、ゲームクライアントがゲームセッションの接続に使用するプロセスの接続情報やその他の情報も報告します。

次に、サーバープロセスは、Amazon GameLift Servers サービスからのプロンプトを待ちます。

3. Amazon GameLift Servers は、サーバープロセスのステータスを `ACTIVE` に更新して、新しいゲームセッションをホストできるようにします。
4. Amazon GameLift Servers は、サーバープロセスからヘルスステータスをリクエストするために `onHealthCheck` コールバックを定期的に呼び出すようになります。これらの呼び出しは、サーバープロセスがアクティブステータスのままでも続行されます。サーバープロセスは、1 分以内に正常か正常でないかを報告する必要があります。サーバープロセスが正常でないと報告した場合、または応答しない場合、ある時点で、Amazon GameLift Servers サービスは、サーバープロセスのアクティブステータスを変更し、ゲームセッションを開始するリクエストの送信を停止します。

ゲームセッションの作成

Amazon GameLift Servers サービスは、プレイヤーからのゲームをプレイするリクエストに応じて新しいゲームセッションを開始します。

1. ゲームクライアントを使用しているプレイヤーがゲームセッションへの参加をリクエストします。ゲームがプレイヤー参加プロセスを処理する方法に応じて、ゲームクライアントはバックエンドサービスにリクエストを送信します。

2. プレイヤー参加プロセスで新しいゲームセッションを開始する必要がある場合、バックエンドサービスは新しいゲームセッションのリクエストを Amazon GameLift Servers サービスに送信します。このリクエストは、サービス API オペレーション `StartGameSessionPlacement()` を呼び出します。(代わりに、バックエンドサービスは `StartMatchmaking()` または `CreateGameSession()` を呼び出す場合があります)。
3. Amazon GameLift Servers サービスは、ステータスが PENDING の新しい `GameSessionPlacement` チケットを作成することで応答します。チケット情報がバックエンドサービスに返されるため、バックエンドサービスは配置チケットステータスを追跡し、プレイヤーがゲームセッションを開始できる状態になったタイミングを判断できるようになります。詳細については、「[ゲームセッション配置のイベント通知を設定](#)」を参照してください。
4. Amazon GameLift Servers サービスは、ゲームセッション配置プロセスを開始します。ゲームセッションをホスティングしていないアクティブなサーバープロセスについて、どのフリートを調べ、それらのフリートを検索するかを特定します。使用可能なサーバープロセスが見つかったら、Amazon GameLift Servers サービスは以下を実行します。
 - a. ゲームセッション設定と、配置リクエストからのプレイヤーデータがある `GameSession` オブジェクトを作成し、ステータスを ACTIVATING に設定します。
 - b. ゲームセッションを開始するようにサーバープロセスにプロンプトします。サービスは、サーバープロセスの `onStartGameSession` コールバックを呼び出し、`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。サーバープロセスがプレイヤーの接続を受け入れる準備ができたなら、サーバー SDK オペレーション `ActivateGameSession()` を呼び出し、プレイヤーが接続されるまで待機します。
6. Amazon GameLift Servers サービスは、`GameSession` オブジェクトをサーバープロセスの接続情報 (`ProcessReady()` への呼び出しで報告されたもの) で更新し、ゲームセッションステータスを ACTIVE に設定します。また、`GameSessionPlacement` チケットステータスを FULFILLED に更新します。
7. バックエンドサービスは `DescribeGameSessionPlacement()` を呼び出し、チケットのステータスを確認し、ゲームセッション情報を取得します。ゲームセッションがアクティブな場合、バックエンドサービスはゲームクライアントに通知し、ゲームセッションの接続情報を渡します。
8. ゲームクライアントは、接続情報を使用して、ゲームサーバープロセスに直接接続し、ゲームセッションに参加します。

プレイヤーをゲームに追加する

ゲームはオプションとして、プレイヤーセッション使用し、ゲームセッションへのプレイヤー接続を追跡できます。プレイヤーセッションは個別に作成するか、ゲームセッション配置リクエストの一環として作成できます。

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 オペレーション `CreatePlayerSession()` を呼び出します。
2. Amazon GameLift Servers サービスは、ゲームセッションのステータスを確認し (ACTIVEである必要があります)、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、サービス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. プレイヤーセッション情報を使用してバックエンドサービスリクエストに応答します。
3. バックエンドサービスは、プレイヤーセッション情報をゲームセッション接続情報と共にゲームクライアントに渡します。
4. ゲームクライアントは接続情報とプレイヤーセッション ID を使用して、ゲームサーバープロセスに直接接続し、ゲームセッションの参加を要求します。
5. ゲームクライアントからの参加試行に応じて、ゲームサーバープロセスは、サービス 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。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。
6. Amazon GameLift Servers サービスは、次のいずれかを行います。
 - a. 接続が許可されると、Amazon GameLift Servers は `PlayerSession` ステータスを `ACTIVE` に設定し、`PlayerSession` をゲームサーバープロセスに渡します。
 - b. ゲームサーバープロセスが、元の `CreatePlayerSession()` リクエスト後、一定期間内にプレイヤーセッション ID に対して `AcceptPlayerSession()` を呼び出さない場合、Amazon GameLift Servers サービスは、`PlayerSession` ステータスを `TIMEDOUT` に変更し、ゲームセッションのプレイヤースロットを再び開放します。

プレイヤーの削除

プレイヤーセッションを使用するゲームの場合、ゲームサーバープロセスはプレイヤーが切断されたときに Amazon GameLift Servers サービスに通知します。サービスはこの情報を使用してゲームセッションのプレイヤースロットのステータスを追跡し、新しいプレイヤーが開いているスロットを使用できるようにします。

1. プレイヤーがゲームから切断されます。
2. ゲームサーバープロセスは、失われた接続を検出し、サーバー SDK オペレーション `RemovePlayerSession()` を呼び出します。
3. Amazon GameLift Servers サービスは、プレイヤーセッションステータスを `COMPLETED` に変更し、ゲームセッションのプレイヤースロットを再び開放します。

ゲームセッションをシャットダウンする

ゲームセッションの終了時、またはゲームセッションのシャットダウン時に、サーバープロセスは、ゲームセッションステータスを Amazon GameLift Servers サービスに通知します。

1. ゲームサーバープロセスは、ゲームセッションを終了し、サーバー SDK オペレーション `ProcessEnding()` を呼び出してプロセスのシャットダウンを開始します。
2. ゲームサーバープロセスは `Destroy()` を呼び出しサーバー SDK をメモリから解放します。このステップは、[テレメトリメトリクス](#)がクラッシュ時に通常のプロセスの終了を報告しないようにするために必要です。
3. Amazon GameLift Servers サービスは、以下を実行します。
 - 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. ゲームセッションのステータスを `TERMINATED` に変更します。
 - c. サーバープロセスのステータスを `TERMINATED` に変更します。
 - d. ホスティングソリューションの設計方法に応じて、新しく利用可能なホスティングリソースが割り当てられ、新しいゲームサーバープロセスを実行します。

Amazon GameLift Serversのバックエンドサービスを構築する

プレイヤーを認証し、Amazon GameLift Servers 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 Servers が新しいゲームセッションのプレイヤーをグループ化する方法、または既存のゲームセッションに追加する方法を制御します。

- クライアントをそのまま信頼するのではなく、自前のリソースから情報を収集して、マッチメイキング用のスキルレーティングなどのゲームセッションプレイヤー属性を提供します。

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 Servers 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。

プレイヤーの認証

Amazon Cognito および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、カスタム認可ロジックに 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 デベロッパーガイド)
- [ユーザープールの開始方法](#) (Amazon Cognito デベロッパーガイド)
- [Amazon Cognito でプレイヤー認証を設定する方法](#) (ゲームブログAWS 用)

Amazon GameLift Servers ゲームクライアント機能の統合する

ゲームセッション情報の取得や新しいゲームセッションの作成など、Amazon GameLift Servers サービスと通信する必要があるソリューションコンポーネントに Amazon GameLift Servers ゲームホスティング機能を統合します。ほとんどの場合、この機能はバックエンドサービスコンポーネントに組み込まれています。次のようなコアタスクの機能を追加します。

- アクティブなゲームセッションの情報とステータスをリクエストします。
- 既存のゲームセッションで新しいプレイヤーのスロットを予約します。
- プレイヤーグループの新しいゲームセッションを作成します。
- 1人以上のプレイヤーのマッチメイキングをリクエストします。
- 既存のゲームセッションの更新データを提供します。

Amazon GameLift Servers SDK を通じてゲームホスティングコンポーネントがどのようにやり取りするかの詳細については、「[Amazon GameLift Servers とゲームクライアント / サーバーのやり取り](#)」を参照してください。

Amazon GameLift Servers APIをセットアップします。

AWS SDK for C++ と をプロジェクトに追加Amazon GameLift Serversします。

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

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 GameLift Servers に連絡するとき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。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 Servers クライアントを初期化します。[Aws::GameLift::GameLiftClient\(\)](#) (C++) または [AmazonGameLiftClient\(\)](#) (C#) を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使用します。
3. プレイヤーセッションを使用している場合は、各プレイヤーに一意の識別子を生成する仕組みを追加します。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 – 特定のフリートでゲームセッションを手動で作成する場合は、ターゲットフリートを指すフリート ID またはエイリアス ID を指定します。フリートがマルチロケーションの場合は、フリートロケーションを指定します。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 – 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、配置リクエストを送信するキュー名を指定します。
 - AWS 認証情報 – 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Amazon GameLift Servers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のユーザーアクセス許可を管理する場合は、Amazon GameLift Servers 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。

アクティブなゲームセッションの取得

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、または検索条件セットを満たす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。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、ポート番号が含まれます。

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。例えば、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"`。

ゲームセッションデータの管理

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](#) – ゲームセッションのメタデータと設定を必要に応じて変更します。
- [GetGameSessionLogUrl](#) – 保存済み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、ゲームホスティングソリューションがゲームセッションの配置を管理する方法に応じて、2つのオプションがあります。

マルチフリートキューで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 GameLift Servers が新しいゲームセッション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、「[ゲームセッションの配置を設定する](#)」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た場合、Amazon GameLift Servers はこの情報を使用して、できる限り最適なゲームプレイエクスペリエンスをプレイヤーに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。

正確なレイテンシー測定値を取得するには、Amazon GameLift Servers の UDP ping ビーコンを使用します。これらのエンドポイントを使用すると、プレイヤーデバイスと潜在的なホスティングロケーション間の実際の UDP ネットワークレイテンシーを測定できるため、ICMP ping を使用するよりも正確な配置決定を行うことができます。UDP ping ビーコンを使用してレイテンシーを測定する方法の詳細については、「[UDP ping ビーコン](#)」を参照してください。

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。また、[StopGameSessionPlacement](#) を使用してリクエストをいつでもキャンセルすることができます。プレイメントリクエストのステータスを確認するには、[DescribeGameSessionPlacement](#) を呼び出します。

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 GameLift Servers 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、そのゲームセッションへプレイヤーの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 Servers リソース保護機能を使って、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。このアクションは、ゲームが一意的なプレイヤー IDs を持つプレイヤーセッションを使用している場合に使用できます。プレイヤーセッションの詳細については、「[Amazon GameLift Servers とプレイヤーエクスペリエンス](#)」を参照してください。

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プレイヤーセッションを作成します。これには、以下の 2 つの方法があります。

- [StartGameSessionPlacement](#) または [StartMatchmaking](#) を使用してゲームセッションを作成する場合、リクエストに 1 人以上のプレイヤーを含めて、新しいゲームセッションでプレイヤーセッションを作成できます。
- 既存のゲームセッションにプレイヤーを追加するには、[CreatePlayerSession](#) または [CreatePlayerSessions](#) をゲームセッション ID で呼び出します。

プレイヤーセッションリクエストには、一意のプレイヤー ID を含める必要があります。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。リクエストを受信する

と、Amazon GameLift Servers はゲームセッションが新規プレイヤーを受け入れており、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 GameLift Servers はそのプレイヤー用のスロットを確保し、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。

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ます。Amazon GameLift Servers は、プレイヤーがゲームセッションに直接接続するときに、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場合は、各プレイヤーデータ文字列をプレイヤー ID に対応付けます。

2. ゲームセッションに接続する

バックエンドサービスにコードを追加して PlayerSession オブジェクトを取得し ([DescribePlayerSessions\(\)](#) を呼び出すなど)、ゲームクライアントに渡します。このオブジェクトには、DNS 名、IP アドレス、ポートが含まれます。ゲームクライアントは、この情報を使用してサーバーへの直接接続を確立できます。

- 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。

接続すると、ゲームクライアントとサーバープロセスは Amazon GameLift Servers を関与させずに直接相互に通信します。サーバーは、Amazon GameLift Servers との通信を維持して、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の接続を検証する場合、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かを確認し、接続を許可または拒否します。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。

カスタムゲームデータをゲームセッションに渡す

ゲームクライアントは、ゲームセッションにデータを渡すことができます。ゲームセッションを作成するときは、一連のゲームプロパティ (キーと値のペア) またはゲームセッションデータ (文字列値) をリクエストに含めることができます。新規または更新されたゲームデータを使用して、既存のゲー

ムセッションを更新することもできます。このデータは、ゲームサーバーコードで使用するために、ゲームセッションをホストしているゲームサーバープロセスに渡されます。ゲームプロパティは削除できません。

例として、ゲームで難易度レベルを Novice、Easy、Intermediate、Expert と定義しているとします。プレイヤーは Easy のゲームへの参加を選択します。ゲームクライアントは、バックエンドサービスを通じて、ゲームプロパティ {"Key": "Difficulty", "Value": "Easy"} を使用して新しいゲームセッションをリクエストします。レスポンスとして、Amazon GameLift Servers 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するように促し、GameSession オブジェクトを渡します。ゲームサーバープロセスは、提供されたゲームプロパティを使用してゲームセッションの難易度レベルを設定します。

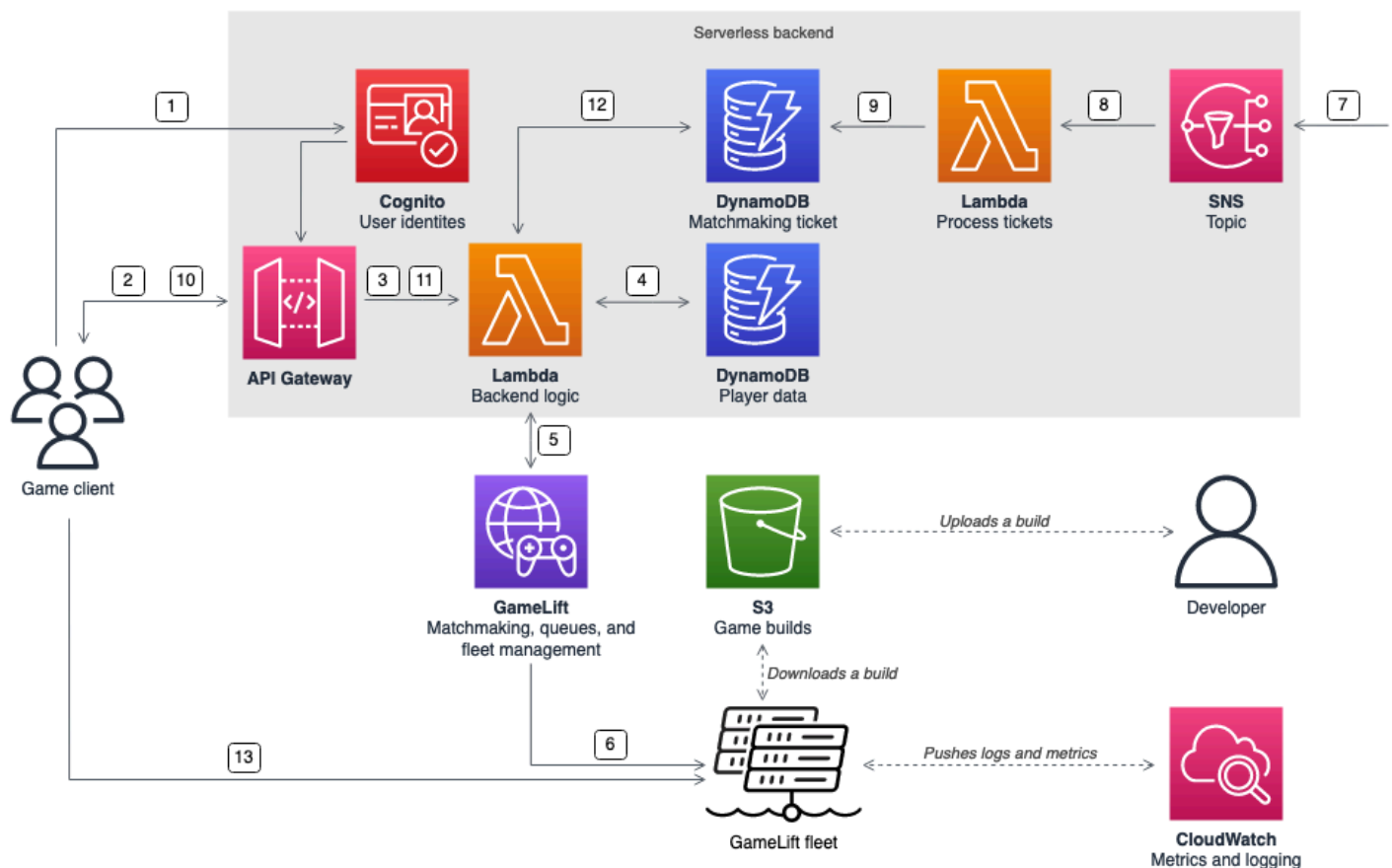
詳細情報

- [GameProperty データ型](#)
- [SearchGameSessions \(\) の例](#)
- [UpdateGameSession \(\) GameProperties パラメータ](#)

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 Servers 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イ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、プレイヤーを Amazon GameLift Servers フリートで実行されているゲームにマッチング AWS のサービス [Amazon GameLift](#) 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 NoSQL 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 Servers FlexMatch マッチメイキングを通じてマッチをリクエストします。
6. FlexMatch は、適切なレイテンシーを持つプレイヤーのグループを照合し、Amazon GameLift Serversキューを介してゲームセッションの配置をリクエストします。キューには、1 つ以上の AWS リージョン 場所があるフリートがあります。
7. Amazon GameLift Servers 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 Servers は Amazon Simple Notification Service (Amazon SNS) トピックにイベント通知を送信します。

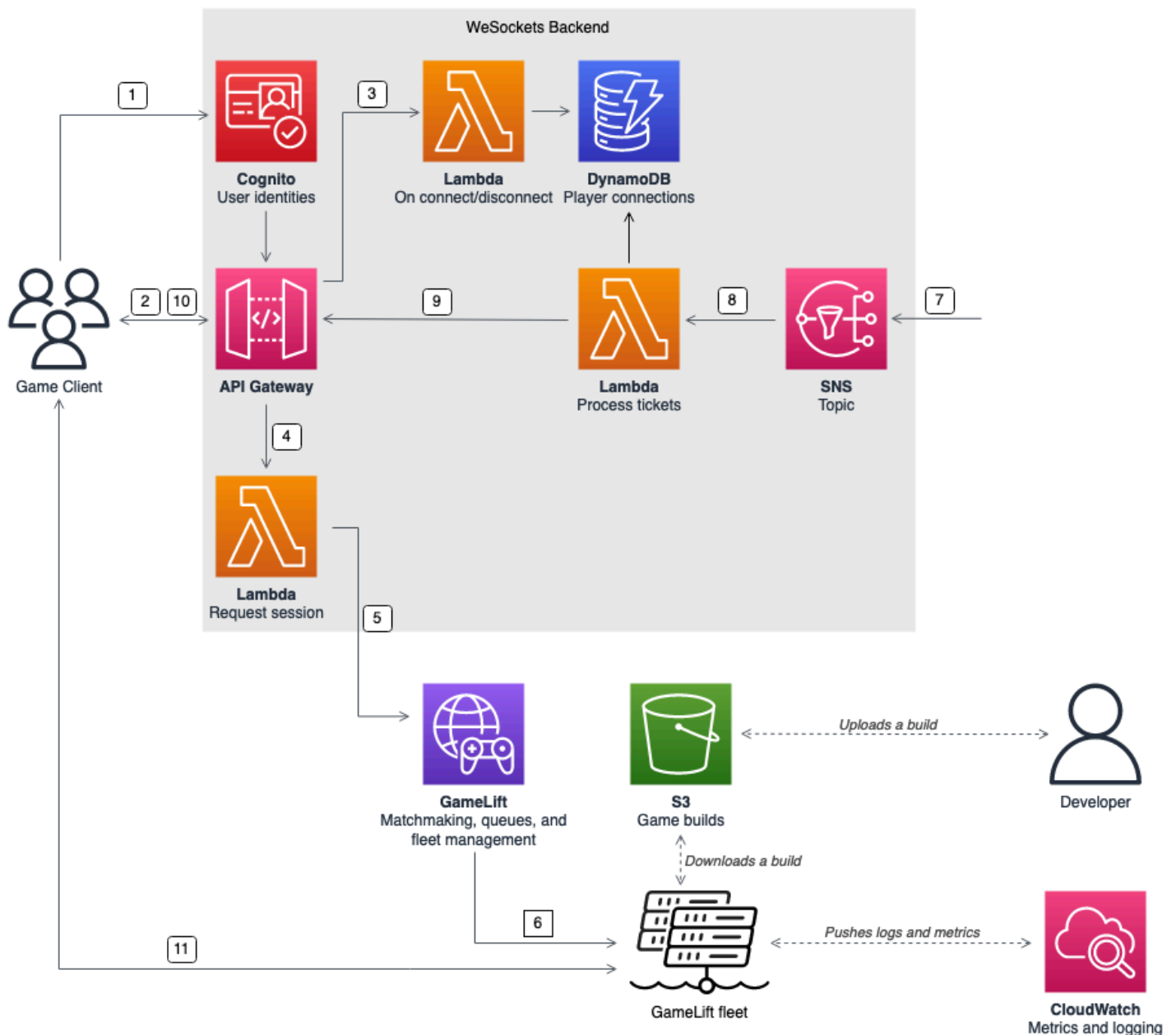
8. Lambda 関数は Amazon SNS イベントを受け取り、それを処理します。
9. マッチメイキングチケットが MatchmakingSucceeded イベントの場合、Lambda 関数はゲームサーバーのポートと IP アドレスと一緒に結果を DynamoDB に書き込みます。
10. ゲームクライアントは API Gateway に対して署名付きリクエストを行い、特定の間隔でマッチメイキングチケットのステータスを表示します。
11. API Gateway は、マッチメイキングチケットのステータスをチェックする Lambda 関数を使用します。
12. Lambda 関数は DynamoDB 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ーヤーセッション ID をゲームサーバーに送信し、ゲームサーバー側でサーバー SDK for Amazon GameLift Servers を使用してその ID を検証します。

WebSocket 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 でマッチメイ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、マッチメイ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 などを使用してプレーヤー AWS のサービスを Amazon GameLift Servers フリットで実行されているゲームにマッチングする 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、接続で AWS Lambda 関数を呼び出します。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、Amazon GameLift Servers FlexMatch マッチメイキングを通じて一致をリクエストします。
6. がプレイヤーのグループ FlexMatch を照合すると、は Amazon GameLift Servers キューを介してゲームセッションの配置を FlexMatch リクエストします。
7. Amazon GameLift Servers 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 Servers 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. マッチメイキングチケットが 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イ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、ゲームサーバー側でのサーバー SDK for Amazon GameLift Servers を使用してその ID を検証します。

Amazon GameLift Servers のホスティングフリートをデプロイする

ホスティングリソースをデプロイするには、ゲームサーバーを実行するコンピューティングインフラストラクチャを作成および設定する必要があります。Amazon GameLift Servers には、フルマネージド AWS クラウド リソースからクラウドインフラストラクチャとオンプレミスインフラストラクチャを組み合わせたハイブリッドソリューションまで、さまざまなホスティングニーズに合わせて複数のフリートタイプが用意されています。

コスト、制御、スケーラビリティ、地理的分散の要件に最適なフリートタイプを選択します。また、複数のフリートタイプを 1 つのホスティングソリューションに結合して、さまざまなシナリオやプレイヤー集団に合わせて最適化することもできます。

フリートの特性

Amazon GameLift Servers フリートは、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。フリートは、使用するコンピューティン

グリソースのタイプとフリートの管理方法によって異なります。フリートのサイズ - サポートできるゲームセッションとプレイヤーの数のことです。フリートに割り当てるコンピューティングリソースの数によって決まります。すべての Amazon GameLift Servers フリートには、次の特性があります。

- すべてのフリートで実行されるゲームサーバープロセスは、サーバー SDK for Amazon GameLift Servers と統合され、同じ方法で Amazon GameLift Servers サービスと通信します。ゲームサーバーは、ゲームセッションやプレイヤーをホストできる状態かどうかを報告し、ゲームセッションの開始や停止を促すプロンプトに応答するなど、さまざまなやり取りを行います。
- Amazon GameLift Servers は、すべてのフリートのゲームセッション配置を同じ方法で処理します。Amazon GameLift Servers はフリートのゲームサーバーのステータスを追跡し、利用可能なゲームサーバーから選択して新しいゲームセッションをホストします。このプロセスは、ゲームがゲームセッションを 1 つのフリートに配置する場合でも、[ゲームセッションキュー](#)を使用して複数のフリート間でホスティングを分散させる場合でも使用されます。キューを使用すると、リソースコストやレイテンシなどの要因を考慮して配置を決定するようカスタマイズすることもできます。
- すべてのフリートは、ゲームセッション配置キューと連携した FlexMatch マッチメーカーの使用をサポートしています。Amazon GameLift Servers サービスは、プレイヤーマッチリクエストを受け取り、マッチングを行い、これをゲームセッションキューに渡して利用可能なゲームサーバーを見つけます。
- Amazon GameLift Servers は、フリートに関するさまざまなメトリクスを収集します。これには、コンピューティングとサーバープロセスのステータスメトリクス、ゲームセッションとプレイヤーアクティビティの使用状況メトリクスが含まれます。使用可能なメトリクスの全一覧については、「[Amazon CloudWatch で Amazon GameLift Servers を監視する](#)」を参照してください。

トピック

- [決定ガイド: ホスティングオプションを選択する](#)
- [Amazon GameLift Servers マネージド EC2 フリート](#)
- [Amazon GameLift Servers マネージドコンテナフリート](#)
- [Amazon GameLift Servers Anywhere フリート](#)
- [ハイブリッドホスティングソリューションの構築](#)

決定ガイド: ホスティングオプションを選択する

Amazon GameLift Servers には複数のホスティングオプションがあり、それぞれが異なるホスティングシナリオと要件に合わせて設計されています。このガイドは、ゲームの技術的ニーズ、運用上の好み、ビジネス上の制約に基づいてフリートのタイプを選択するのに役立ちます。大まかな説明については、[Amazon GameLift Servers ゲームホスティングオプション](#) を参照してください。

フリート比較

次の表は、各ホスティングオプションの主な特性を比較したものです。

ホスティングオプション	最適な用途	管理レベル	主な利点
マネージド EC2	フル AWS 管理による本番ホスティング	フルマネージド型	自動スケーリング、グローバルリーチ、最小限の運用オーバーヘッド
マネージドコンテナ	オーケストレーションを必要とするコンテナ化されたアプリケーション	フルマネージド型	コンテナオーケストレーション、リソース効率、最新のデプロイ
Anywhere	オンプレミス、ハイブリッド、または開発環境	セルフマネージド型	ハードウェア制御、コンプライアンス要件、コスト最適化

主な決定要因

ホスティングオプションを選択するときは、次の要素を考慮してください。

運用の複雑さ

どのくらいのインフラストラクチャ管理を処理したいですか？ マネージドフリートは運用上のオーバーヘッドを最小限に抑え、Anywhere フリートは完全な制御を提供しますが、より多くの管理を必要とします。

スケーラビリティ要件

プレイヤーの需要に基づいて自動スケーリングが必要ですか？ マネージドフリートは組み込みの自動スケーリングを提供し、Anywhere フリートには手動容量管理が必要です。

地理的分布

プレイヤーの所在地はどこですか？ マネージドフリートは複数の AWS リージョンにグローバルにデプロイできますが、Anywhere フリートは独自のインフラストラクチャロケーションに依存します。

コンプライアンスとセキュリティについて

特定のコンプライアンス要件があるか、データをオンプレミスに保持する必要がありますか？ Anywhere フリートを使用すると、インフラストラクチャとデータの場所を完全に管理できます。

コストに関する考慮事項

予算とコストの最適化戦略は何ですか？ オプションを比較するときは、インフラストラクチャコストと運用オーバーヘッドの両方を考慮してください。

ハイブリッドソリューション

ゲームホスティングソリューションには、フリートのタイプを 1 つだけ選択する必要はありません。多くのゲームは、複数のフリートタイプを組み合わせることで多様なシナリオに最適化するハイブリッドアプローチを使用しています。

- 開発とテスト: Anywhere フリートを開発とテストに使用し、本番稼働用のマネージドフリートにデプロイします。
- リージョンの最適化: 既存のインフラストラクチャがあるリージョンでは、プレイヤー密度の高い AWS リージョンでマネージドフリートと Anywhere フリートを使用します。
- コスト最適化: ベースライン容量にはマネージドフリートを使用し、オーバーフローまたは特定のワークロードには Anywhere フリートを使用します。

Amazon GameLift Servers マネージド EC2 フリート

Amazon GameLift Servers マネージド EC2 フリートは、本番ホスティング用のクラウドベースのリソースを提供します。マネージドフリートを使用すると、マルチプレイヤーゲームホスティング

に最適化された AWS クラウド リソースの柔軟性、セキュリティ、信頼性が得られます。Amazon GameLift Serversは堅牢なホスト管理ツールを提供します。

マネージド EC2 フリートは、設定に基づいて Amazon GameLift Servers が管理・運用する Amazon Elastic Compute Cloud (Amazon EC2) インスタンスの集合です。これらのインスタンスは、サポートされている AWS リージョン またはローカルゾーンに物理的に配置されます。フリートを作成するときは、コンピューティング能力、メモリ、ストレージ、ネットワーク機能に関するゲームサーバーの要件を満たす EC2 インスタンスタイプを選択します。

フリート内の各インスタンスを起動すると、Amazon GameLift Servers は必要なランタイム環境でゲームサーバービルドをデプロイします。ランタイム環境は、フリートの作成時に利用可能な最新の Amazon マシンイメージ (AMI) バージョンを使用します。フリート内のすべてのインスタンスは、同じ AMI バージョンを使用します。

 Note

ベストプラクティスとして、ホストされているゲームサーバーでの安全で最新のランタイム環境を維持するために、30 日ごとにフリートを置き換えることをお勧めします。そのためには、新しいフリートを作成し、プレイヤートラフィックをそのフリートに移行する必要があります。ガイダンスについては、「[Amazon GameLift Servers のセキュリティに関するベストプラクティス](#)」を参照してください。

ランタイム環境とゲームサーバービルドをインスタンスにインストールすると、Amazon GameLift Servers はゲームサーバープロセスの起動を開始します。各ゲームサーバープロセスは、Amazon GameLift Servers サービスへの接続を確立し、ゲームセッションをホストする準備状況を報告して、ヘルスステータスの通信を開始します。Amazon GameLift Servers は、サーバープロセスにゲームセッションを開始するよう促すことができます。

フリートのデプロイに加えて、Amazon GameLift Servers は、ユーザーに代わって次のホスト管理タスクを処理します。

- フリート内のすべてのコンピューティングのステータスを追跡し、古いコンピューティングや異常なコンピューティングを置き換える。
- サーバープロセスと Amazon GameLift Servers サービス間の通信の認証を処理する。
- ランタイム構成に基づいて、各コンピュートでゲームサーバープロセスを自動的に起動および停止します。

- プレイヤーの需要に対応できるよう、フリート容量を動的に調整する自動スケーリングツールを提供します。
- フリートの EC2 インスタンスのパフォーマンスメトリクスを報告する。

マネージド EC2 フリートをセットアップおよび維持する方法については、以下のトピックを参照してください。

- [Amazon GameLift Servers マネージド EC2 でホスティングするための開発ロードマップ](#)
- [Amazon GameLift Servers マネージド EC2 フリートを作成する](#)
- [Amazon GameLift Servers を使用したゲームホスティング容量のスケーリング](#)
- [ホスティングリソースのカスタマイズ](#)
- [Amazon GameLift Servers フリート設定を更新する](#)

マネージド EC2 フリート作成のワークフロー

マネージドフリートの場合、Amazon GameLift Servers がフリートリソースをセットアップし、ゲームサーバーソフトウェアをインストールして実行したうえで一連のコンピューティングリソースのセットをデプロイします。作成ワークフローが正常に完了すると、フリートにはフリートのホームリージョンにアクティブな EC2 インスタンスが 1 つ、フリートのリモートロケーションにそれぞれ 1 つあります。すべてのインスタンスで、ゲームセッションをホストする準備が整います。

1. Amazon GameLift Servers は、フリートのホームリージョンにフリートリソースを作成し、各ロケーションの希望する容量を 1 インスタンスに設定します。フリートとロケーションのステータスは [新規] に設定されます。
2. Amazon GameLift Servers 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。
3. Amazon GameLift Servers はフリートステータスを [ダウンロード中] に設定し、デプロイ用のゲームサーバーソフトウェアの準備を開始します。
 - a. アップロードされたゲームサーバービルドを取得し、圧縮ファイルを抽出します。
 - b. 指定されている場合は、インストールスクリプトを実行します。
 - c. フリートステータスを [検証中] に設定し、ビルドファイルのダウンロードとインストール時にエラーが発生していないかどうかの検証を開始します。
4. Amazon GameLift Servers は、フリートステータスを [構築中] に切り替え、フリートハードウェアを設定し、フリートインスタンスごとに 1 つの EC2 インスタンスを割り当てます。

5. Amazon GameLift Servers 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。フリートのランタイム手順に従って各インスタンスでゲームサーバープロセスを起動し、ビルドと Amazon GameLift Servers サービス間の接続をテストします。
6. 各インスタンスでゲームサーバープロセスが接続を確立し、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と、Amazon GameLift Servers はフリート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。この時点で、フリートはゲームセッションをホストする準備ができていと見なされます。

Amazon GameLift Servers マネージド EC2 フリートを作成する

このトピックでは、Amazon GameLift Servers マネージド EC2 フリートを作成する方法を説明します。マネージドフリートは、マルチプレイヤーゲームホスティング用に最適化された Amazon Elastic Compute Cloud (Amazon EC2) コンピューティングインスタンスを使用します。コンピューティングを および でサポートされている AWS リージョン ローカルゾーンにグローバルにデプロイするマネージドフリートを作成できます Amazon GameLift Servers。

新しいマネージド EC2 フリートリソースを作成すると、フリート作成の最初のフェーズがすぐに開始されます。マネージドフリートの作成は、Amazon GameLift Servers が EC2 インスタンスをデプロイし、ランタイム環境とゲームサーバービルドをインスタンスにインストールし、ゲームサーバーの起動を開始するとき、いくつかのフェーズを通過します。ゲームサーバーが構築するランタイム環境に応じて、 はフリートの作成時に最新バージョンの Amazon マシンイメージ (AMI) を Amazon GameLift Servers デプロイします (フリート内のすべての将来のインスタンスは同じバージョンを使用します)。フリートのステータスは、コンソールまたは AWS Command Line Interface () を使用してモニタリングできます AWS CLI。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。マネージドフリートの作成についての詳細は、以下のトピックを参照してください。

Note

ベストプラクティスとして、ホストされているゲームサーバーでの安全で最新のランタイム環境を維持するために、30 日ごとにフリートを置き換えることをお勧めします。そのためには、新しいフリートを作成し、プレイヤートラフィックをそのフリートに移行する必要があります。ガイダンスについては、「[Amazon GameLift Servers のセキュリティに関するベストプラクティス](#)」を参照してください。

マネージド EC2 フリートを作成するには

Amazon GameLift Servers コンソールまたは AWS Command Line Interface (AWS CLI) を使用して、マネージド EC2 フリートを作成します。

Console

[\[Amazon GameLift Servers コンソール\]](#) のナビゲーションペインで、[マネージド EC2、フリート] ページを開きます。[マネージド EC2 フリートの作成] を選択して、フリート作成ワークフローを開始します。

ステップ 1 マネージド EC2 フリートの詳細を定義する

i 最小限のフリート設定の場合:

- フリート名を指定します。
- バイナリタイプ: ビルドを選択します。以前に Amazon GameLift Servers にアップロードしたサーバービルドのリストからビルドを選択します。
- デフォルト設定を使用するには、[その他の詳細] セクションと「タグ」セクションをスキップします。

1. [フリートの詳細] セクションに以下の内容を入力します。

- a. フリートの [名前] を入力します。フリート名は説明のみであり、フリートのリストをソートまたはフィルタリングするのに役立ちます。一意である必要はありません。
- b. フリートについてキャプチャするカスタム情報を含めるには、短いフリートの説明を指定します。
- c. [バイナリタイプ、ビルド] を選択し、ドロップダウンリストを使用して、以前に Amazon GameLift Servers にアップロードしたゲームサーバー build を選択します。

2. 必要に応じて [その他の詳細] を設定します。

- a. ゲームサーバー実行可能ファイルがアカウント内の他の AWS リソースにアクセスする必要がある場合は、必要なアクセス許可を持つ IAM インスタンスロールを指定します。他のサーバー側アプリケーション (CloudWatch エージェントなど) を認可する方法などの詳細については、「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」を参照してください。この設定は、フリートの作成後に変更することはできません。

ロールを使用するフリートを作成する前に、ロールを作成する必要があります。さらに、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、AWS ユーザーに IAM PassRole アクセス許可が必要です (「[Amazon GameLift Servers 向け IAM アクセス許可の例](#)」を参照)。

- b. ゲームの認証と暗号化を設定するには、[TLS 証明書の生成] オプションをオンにします。接続時にこの証明書を使用してゲームサーバーを認証し、クライアント/サーバー間のすべての通信を暗号化します。TLS 対応のフリートのインスタンスごとに、Amazon GameLift Servers 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。この設定は、フリートの作成後に変更することはできません。
 - c. Amazon GameLift Servers は個々のフリートのメトリクスデータを送信します。複数のフリートのメトリクスデータを組み合わせる場合は、[メトリクスグループ] 名を指定します。メトリクスを組み合わせるすべてのフリートに同じメトリクスグループ名を使用します。CloudWatch を使用して、集計されたメトリクスグループデータを表示します。
3. (オプション) フリートリソースに [タグ] を追加します。各タグはキーとオプションの値で構成され、どちらもお客様側が定義します。目的、所有者、環境など、分類する AWS リソースにタグを割り当てます。追加するタグごとに [新しいタグを追加] を選択します。
 4. [次へ] を選択してワークフローを続行します。

ステップ 2 インスタンスの詳細を定義する

このステップでは、ゲームサーバーに使用するハードウェアリソースのタイプと、リソースをデプロイするロケーションを指定します。複数のロケーションを選択すると、ゲームサーバーをデプロイする地理的範囲を広げられるため、サーバーをプレイヤーに近づけ、レイテンシーを最小限に抑えることができます。フリート内のすべての EC2 インスタンスは同じタイプである必要があります。ロケーションによっては使用できないインスタンスタイプがあります。

最小限のフリート設定の場合:

- まず、ホームロケーションにのみデプロイします。初期開発とテストにリモートロケーションを追加する必要はありません。
- フリートタイプを「オンデマンド」に設定します。スポットフリートには追加のセットアップ作業が必要です。

- インスタンスタイプを「c5.large」に設定します。この一般的に使用されるインスタンスタイプは、すべてので使用できます AWS リージョン。ハードウェアの使用量は後で最適化できます。

1. [インスタンスデプロイ] で、フリートのロケーションとタイプを指定します。
 - a. フリートインスタンスをデプロイする追加の [ロケーション] を 1 つまたは複数選択します。これらのリモートロケーションは、フリートのホームロケーション (事前選択済み) に追加されます。これは、このフリートを作成する AWS リージョン です。が Amazon GameLift Servers サポートするすべての AWS リージョン およびローカルゾーンからリモートロケーションを選択できます。

デフォルトで有効 AWS リージョン になっていない の使用方法など、サポートされている場所の詳細については、マネージドホスティング [Amazon GameLift Servers サービスロケーション](#) の「」を参照してください。また、フリートごとのロケーションに対する Amazon GameLift Servers [クォータ](#) も確認してください。
 - b. このフリートで [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。
2. ニーズを満たし、選択したすべてのロケーションで利用可能な Amazon EC2 [インスタンスの設定] を選択します。このリストは、現在のロケーションとフリートタイプの選択に基づいてフィルタリングされます。インスタンスタイプファミリーやアーキテクチャなどの他の要因によってさらにフィルタリングできます。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。

一部のロケーションでは、インスタンスタイプのオプションが制限されています。希望するインスタンスタイプがすべてのロケーションで利用できない場合は、ロケーションの可用性値を選択して詳細を表示します。すべてのロケーションに対応するために、異なるインスタンスタイプで個別のフリートを作成する必要がある場合があります。

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。Amazon EC2 Arm アーキテクチャの詳細については、「[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 Servers がサポートするインスタンスタイプの完全な一覧については、[EC2InstanceType](#) (CreateFleet()) の API リファレンスを参照してください。

Note

Graviton Arm インスタンスには、Linux AMI 用のサーバービルドが必要です。C++ および 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) および Amazon Linux 2 (AL2)での Mono インストールは標準ではサポートされません。

3. [次へ] を選択してワークフローを続行します。

ステップ 4 ランタイムを設定する

このステップでは、フリート内の各インスタンスでゲームサーバーソフトウェアを実行する方法を説明します。インスタンスで実行する実行可能ファイルごとに個別のサーバープロセス行項目を定義し、同時に実行する各サーバープロセスの数を決定します。各インスタンスでポートを開き、プレイヤーがゲームサーバーに直接接続できるようにします。これらのフリート設定はいつでも更新できます。

最小限のフリート設定の場合:

- ゲームサーバー実行可能ファイルの単一のサーバープロセス行項目を定義します。ゲームサーバーで他のプロセスを実行する必要がある場合は、これらの各プロセスの定義も作成します。
- 各行項目には、デフォルトの同時プロセス数 (1) を使用します。
- ゲームセッションのアクティベーション設定は省略します。
- 1 つのポート番号を指定します。
- ゲームセッションのリソース設定は省略します。

1. [ランタイム設定] を作成して、フリート内の各インスタンスでサーバープロセスを実行する方法について Amazon GameLift Servers に指示します。フリートのランタイム設定は、デプロイ後いつでも変更できます。
 - a. ビルドの実行可能ファイルへの [起動パス] 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 にビルドされます。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` または `/local/game/MyGame/server.exe`。

- b. オプションで、ゲーム実行可能ファイルに渡す [起動パラメータ] を入力します。例えば、`+sv_port 33435 +start_lobby` などです。
- c. 各インスタンスで実行する [同時プロセス] の数を指定します。ゲームサーバー実行可能ファイルの場合、各プロセスで 1 つのゲームセッションをホストできるため、同時プロセスにより、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が決まります。

インスタンスごとのサーバープロセスで Amazon GameLift Servers [クォータ](#)を確認します。このクォータは、すべての設定の同時プロセスの合計に適用されます。フリートの設定がクォータを超えると、フリートはアクティブ化されません。

2. [ゲームセッションアクティベーション] のデフォルトを使用するか、ゲームに合わせてカスタマイズします。ランタイム設定で、インスタンスごとに複数のゲームサーバープロセスを同時に実行する必要がある場合、これらの設定により、新しいゲームセッションの起動速度が決まります。
 - a. [最大の同時ゲームセッションアクティベーション] を設定して、新しいゲームセッションを準備している、インスタンス上のゲームサーバーの数を制限します。この設定は、複数の新しいゲームセッションを起動する際にリソースが大量に消費され、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を与える可能性がある場合に便利です。
 - b. 新しいゲームセッションがアクティベーションを完了し、プレイヤーをホスト可能な状態になったことを報告するまでに要する最大時間を反映するように、[新しいアクティベーションタイムアウト] を設定します。Amazon GameLift Servers がこの値を超えた場合、ゲームセッションのアクティベーションは終了されます。
3. [EC2 ポート設定] を開き、インバウンドトラフィックがフリート上のサーバープロセスにアクセスできるようにします。この設定はフリートの作成には必要ありませんが、プレイヤーがフリートのゲームセッションに接続する前に設定する必要があります。

ポート設定ごとに、ゲームクライアントとゲームサーバー間の通信に使用するデータ転送プロトコルの [タイプ] を選択します。[ポート範囲] (nnnnn[-nnnnn] 形式) と、CIDR 表記を使用した [IP アドレス範囲] (誰にでもアクセスを許可する `0.0.0.0/0` など) を指定します。

連続しない範囲を複数設定する必要がある場合は、複数のポート設定を作成します。

4. オプションの [ゲームセッションリソース設定] を指定します。この設定は、デプロイ後にいつでも更新できます。
 - a. フリート内のすべてのインスタンスで [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。スケールダウンイベントの間、アクティブなゲームセッションをホストしている保護されたフリートインスタンスは、Amazon GameLift Servers によって終了されません。
 - b. 指定された期間に 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する場合は、[リソース作成制限] を設定します。
5. [次へ] を選択してワークフローを続行します。

ステップ 5 確認して作成する

フリートを作成する前に、設定を確認します。一部の設定は後で更新できますが (「[Amazon GameLift Servers フリート設定を更新する](#)」を参照)、次の設定はフリートの作成後に変更することはできません。

- コンピューティングタイプ: マネージド EC2 フリートを Anywhere フリートに変換することはできません。
- ゲームサーバーバイナリ: ゲームサーバービルドに更新をデプロイするには、新しいフリートを作成する必要があります。
- インスタンスロールや TLS 証明書の生成などの追加オプション。
- フリートタイプ (スポットまたはオンデマンド) や EC2 インスタンスタイプなど、インスタンスの詳細。

新しいフリートをデプロイする準備ができたなら、[作成] を選択します。Amazon GameLift Servers はフリートのアクティベーションプロセスを直ちに開始し、一意の ID を割り当ててフリートを NEW ステータスに移行します。フリートの進行状況は [フリート] ページからを追跡します。フリートの詳細ページを表示し、[イベント] タブに移動します。

フリートが [アクティブ] ステータスになると、フリートのホスティング容量を調整できます。Amazon GameLift Servers は、各フリートロケーションに 1 つのインスタンスを持つフリートを初期デプロイします。その後、各ロケーションにインスタンスを追加して容量を調整します。詳細については、「[Amazon GameLift Servers を使用したゲームホスティング容量のスケールリング](#)」を参照してください。

AWS CLI

[create-fleet](#) コマンドを使用して、コンピューティングタイプ EC2 のフリートを作成します。Amazon GameLift Servers は、現在のデフォルト AWS リージョン にフリートリソースを作成します (または、`--region` タグを追加して別の AWS リージョンを指定することもできます)。

最小限の設定でマネージドフリートを作成する

次のリクエスト例では、ゲームクライアントが接続できる実行中のゲームサーバーでフリートをデプロイするために必要な最小限の設定で新しいフリートを作成します。新しいフリートには次の特徴があります。

- ゲームサーバービルドを指定します。このビルドは Amazon GameLift Servers にアップロードされ、READY ステータスになっています。
- 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。
- フリートのホーム AWS リージョン を に設定 `us-west-2` し、そのリージョンにのみインスタンスをデプロイします。
- ランタイム設定に基づいて、フリート内の各コンピューティングは 1 つのゲームサーバープロセスを実行します。つまり、各コンピューティングは一度に 1 つのゲームセッションのみをホストできます。ゲームセッションのアクティベーションタイムアウトはデフォルト値の 300 秒に設定され、同時アクティベーションの数に制限はありません。
- プレイヤーは、33435 の単一のポート設定を使用してゲームサーバーに接続できます。
- その他の機能はすべてオフにするか、デフォルト設定を使用します。

```
aws gamelift create-fleet \  
  --name MinimalFleet123 \  
  --description "A basic test fleet" \  
  --region us-west-2 \  
  --ec2-instance-type c5.large \  
  --fleet-type ON_DEMAND \  
  --build-id build-1111aaaa-22bb-33cc-44dd-5555eeee66ff \  
  --runtime-configuration "ServerProcesses=[{LaunchPath=C:\game\Bin64.dedicated\  
\MultiplayerSampleProjectLauncher_Server.exe, ConcurrentExecutions=10}]" \  
  --ec2-inbound-permissions  
  "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"
```

完全に設定されたマネージドフリートを作成する

次のリクエスト例では、すべてのオプション機能の設定を使用して本番稼働用フリートを作成します。新しいフリートには次の特徴があります。

- ゲームサーバービルドを指定します。このビルドは Amazon GameLift Servers にアップロードされ、READY ステータスになっています。
- 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。
- フリートのホーム AWS リージョン をに設定us-west-2し、インスタンスをホームリージョンと1つのリモートロケーション にデプロイしますsa-east-1。
- ランタイム設定により:
 - フリート内の各コンピューティングは、同じ起動パラメータで 10 のゲームサーバープロセスを実行します。つまり、各コンピューティングは最大 10 のゲームセッションを同時にホストできます。
 - 各コンピューティングで、同時にアクティブ化できるゲームセッションは 2 つのみです。アクティブ化中のゲームセッションは、300 秒 (5 分) 以内にプレイヤーをホストできる状態になるか、終了されます。
- プレイヤーは、33435 to 33535 の範囲のポートを使用してゲームサーバーに接続できます。
- ゲームクライアントとサーバー間の暗号化された通信用の TLS 証明書を生成します。
- このフリートの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。
- 各プレイヤーが 15 分以内に作成できる新しいゲームセッションは 3 つまでです。
- このフリートのメトリクスは、メトリクスグループ AMERfleets に含まれます。このメトリクスグループは、北米、中米、南米のフリートグループのメトリクスを集計します (この例の場合)。

```
aws gamelift create-fleet \  
  --name ProdFleet123 \  
  --description "A fully configured prod fleet" \  
  --ec2-instance-type c5.large \  
  --region us-west-2 \  
  --locations "Location=sa-east-1" \  
  --fleet-type ON_DEMAND \  
  --build-id build-1111aaaa-22bb-33cc-44dd-5555eeee66ff \  
  --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=33535,IpRange=0.0.0.0/0,Protocol=UDP" \
--metric-groups "AMERfleets"
```

フリート作成のリクエストが成功すると、リクエストした構成設定と新しいフリート ID が含まれるフリート属性のセットが Amazon GameLift Servers から返されます。Amazon GameLift Servers は直ちにフリートアクティベーションプロセスを開始し、フリートのステータスとロケーションステータスを [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 Servers マネージドコンテナフリート

Amazon GameLift Servers マネージドコンテナフリートは、コンテナ化されたゲームサーバーソフトウェアをホストするためのクラウドベースのリソースを提供します。マネージドコンテナフリートを使用すると、マルチプレイヤーゲームホスティング用に最適化された AWS クラウド リソースの柔軟性、セキュリティ、信頼性が得られます。Amazon GameLift Serversは堅牢なホスト管理ツールを提供します。

i マネージドコンテナ用のこれらのツールを使用して、オンボーディングを高速化します。

- [コンテナスターターキット](#)は、統合とフリートのセットアップを合理化します。ゲームサーバーに重要なゲームセッション管理機能を追加し、事前設定されたテンプレートを使用して、ゲームサーバー用のコンテナフリートと自動デプロイパイプラインを構築します。デプロイ後、Amazon GameLift Servers コンソールと API ツールを使用して、フリートのパフォーマンスのモニタリング、ゲームセッションの管理、メトリクスの分析を行います。
- Unreal Engine または Unity 開発者の場合は、ゲームエンジン[Amazon GameLift Servers プラグインとサーバー SDK](#)を使用してゲームサーバーを統合し、ゲームエンジンの開発環境内からコンテナフリートを構築します。プラグインのガイド付きワークフローは、マネージドコンテナを使用したクラウドベースのホスティングによる高速でシンプルなソリューションの作成に役立ちます。この基盤をもとに、ゲーム用のカスタムホスティングソリューションを作成できます。

マネージドコンテナフリートは、Linux を実行する Amazon Elastic Compute Cloud (Amazon EC2) インスタンスのセットであり、Amazon GameLift Servers が設定に基づいて管理および運用します。これらのインスタンスは、サポートされている AWS リージョン またはローカルゾーンに物理的に配置されます。コンテナフリートを作成するときは、コンピューティング能力、メモリ、ストレージ、ネットワーク機能に関するゲームサーバーの要件を満たす EC2 インスタンスタイプを選択します。

マネージドコンテナフリートの場合、Linux ベースのコンテナイメージを Amazon Elastic Container Registry (Amazon ECR) リポジトリに保存し、コンテナグループ定義を作成してコンテナアーキテクチャを記述します。フリートを作成すると、Amazon GameLift Servers は Linux Amazon マシンイメージ (AMI) の最新バージョンを使用してフリートインスタンスをプロビジョニングし、コンテナグループ定義を使用してコンテナイメージをデプロイします。コンテナグループ定義を更新したり、コ

コンテナイメージを変更したりしても、コンテナフリート内のすべてのインスタンスは同じ AMI バージョンを使用します。

 Note

ベストプラクティスとして、ホストされているゲームサーバーでの安全で最新のランタイム環境を維持するために、30 日ごとにフリートを置き換えることをお勧めします。そのためには、新しいフリートを作成し、プレイヤートラフィックをそのフリートに移行する必要があります。ガイダンスについては、「[Amazon GameLift Servers のセキュリティに関するベストプラクティス](#)」を参照してください。

コンテナ化されたインスタンスをデプロイすると、コンテナはゲームサーバープロセスの起動を開始します。各ゲームサーバープロセスは、Amazon GameLift Servers サービスへの接続を確立し、ゲームセッションをホストする準備状況を報告して、ヘルスステータスの通信を開始します。Amazon GameLift Servers は、サーバープロセスにゲームセッションを開始するよう促すことができます。

フリートのデプロイに加えて、Amazon GameLift Servers は、ユーザーに代わって次のホスト管理タスクを処理します。

- フリート内のすべてのコンテナのステータスを追跡し、古くなったコンテナや異常なコンテナを入れ替えます。
- サーバープロセスと Amazon GameLift Servers サービス間の通信の認証を処理する。
- プレイヤーの需要に合わせてフリート容量を動的に調整する自動スケーリングツールを提供する。
- フリート内の EC2 インスタンス、コンテナ、サーバープロセスのパフォーマンスメトリクスを報告します。

マネージドコンテナフリートのセットアップと管理方法については、次のトピックを参照してください。

- [Amazon GameLift Servers マネージドコンテナでホスティングするための開発ロードマップ](#)
- [Amazon GameLift Servers マネージドコンテナフリートを作成する](#)
- [Amazon GameLift Servers コンテナフリートをカスタマイズする](#)
- [Amazon GameLift Servers を使用したゲームホスティング容量のスケーリング](#)
- [Amazon GameLift Servers マネージドコンテナフリートを更新する](#)

Amazon GameLift Servers におけるコンテナの仕組み

Amazon GameLift Servers 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柔軟にデプロイおよびスケーリングできるように設計されています。Amazon Elastic Container Service (Amazon ECS) を使用して、タスクのデプロイメントと実行をユーザーの Amazon GameLift Servers フリート向けに管理します。このトピックでは、Amazon GameLift Servers 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示し、いくつかの主要な概念の概要を示します。

 マネージドコンテナ用のこれらのツールを使用して、オンボーディングを高速化します。

- [コンテナスターターキット](#)は、統合とフリートのセットアップを合理化します。ゲームサーバーに重要なゲームセッション管理機能を追加し、事前設定されたテンプレートを使用して、ゲームサーバー用のコンテナフリートと自動デプロイパイプラインを構築します。デプロイ後、Amazon GameLift Servers コンソールと API ツールを使用して、フリートのパフォーマンスのモニタリング、ゲームセッションの管理、メトリクスの分析を行います。
- Unreal Engine または Unity の開発者の場合、[Amazon GameLift Servers プラグイン](#)を使用してゲームサーバーを統合し、ゲームエンジンの開発環境内からコンテナフリートを構築します。プラグインのガイド付きワークフローは、マネージドコンテナを使用したクラウドベースのホスティングによる高速でシンプルなソリューションの作成に役立ちます。次に、この基盤をもとに、ゲーム用のカスタムホスティングソリューションを作成します。

コンテナフリートコンポーネント

Fleet

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ホストするための Amazon EC2 インスタンスのコレクションです。これらのインスタンスは、Amazon GameLift Servers によってユーザーに代わって管理されます。フリートを作成するときは、ゲームサーバーソフトウェアを含むコンテナアーキテクチャを各フリートインスタンスにどのようにデプロイするかを設定します。1 つまたは複数の地理的な場所にインスタンスを持つコンテナフリートを作成できます。Amazon GameLift Servers 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自動的にスケールできます。

インスタンス

Amazon EC2 インスタンスは、ゲームホスティング用のコンピューティング容量を提供する仮想サーバーです。Amazon GameLift Servers では、さまざまなインスタンスタイプから選択できます。各インスタンスタイプごとに、CPU、メモリ、ストレージ、ネットワーキングの異なる組み合わせが提供されます。

コンテナフリートを作成すると、Amazon GameLift Servers が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コンテナをデプロイします。デプロイされた各フリートインスタンスは同一であり、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容量が決まります。

コンテナグループ

Amazon GameLift Servers 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、コンテナの「タスク」や「ポッド」に近い概念です。各コンテナグループ内では、コンテナの動作、依存関係、使用可能な CPU とメモリリソースの共有方法を定義できます。

各フリートインスタンスには、次のタイプのコンテナグループを含めることができます。

- ゲームサーバーコンテナグループは、ゲームサーバーアプリケーションおよび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。ゲームセッションやプレイヤーをホストするには、コンテナフリートにこのタイプのコンテナグループが少なくとも 1 つ必要です。ゲームサーバーコンテナグループは、フリートインスタンス間で複製できます。フリートインスタンスごとのゲームサーバーグループの複製数は、ソフトウェアのコンピューティング要件と、インスタンスで利用できるコンピューティングリソースによって変わります。
- オプションであるインスタンス単位のコンテナグループを使用すると、各フリートインスタンスで追加のソフトウェアを実行できます。これらは、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、インスタンス単位のコンテナグループのプロセスに直接依存しません。インスタンス単位のコンテナグループは、各フリートインスタンスに 1 つだけデプロイされます。

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 つ含まれます。必須コンテナは、コンテナグループのライフサイクルを制御します。必須コンテナに障害が発生すると、コンテナグループ全体が再起動します。

コンテナ

コンテナは、コンテナベースのアーキテクチャを構成する最も基本的な要素です。これには、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が含まれます。ソフトウェアの動作方法やAmazon GameLift Serversとの連携方法を設定するため、コンテナを定義します。

Amazon GameLift Servers は 2 種類のコンテナを定義します。

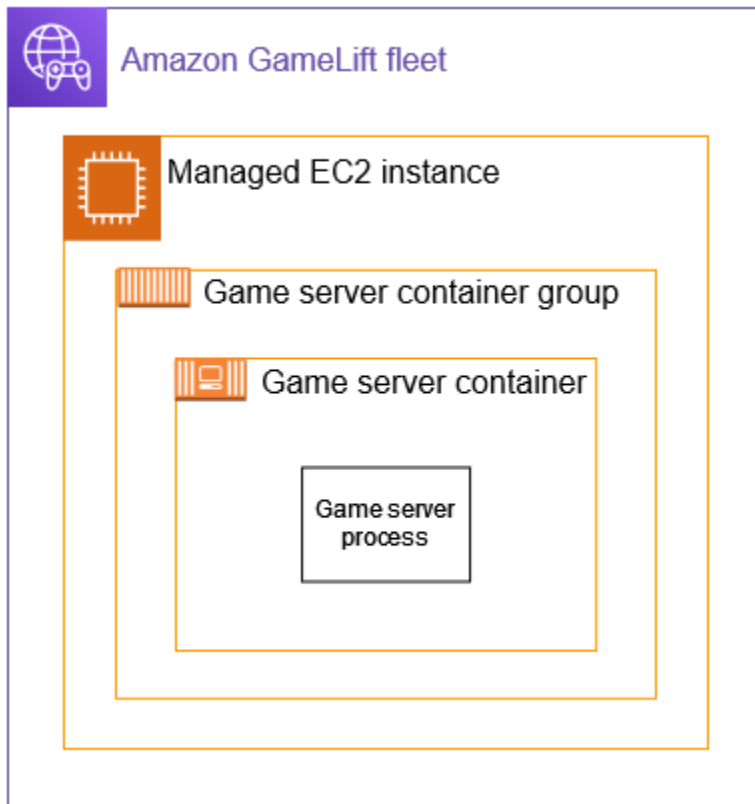
- ゲームサーバーコンテナには、ゲームサーバープロセスを実行し、プレイヤーのゲームセッションをホストするために必要なものがすべて含まれています。これには、ゲームサーバーのビルドと依存ソフトウェアが含まれます。フリートのゲームサーバーコンテナ用に、1 つのゲームサーバーコンテナを定義します。ゲームサーバーコンテナは、コンテナグループの必須として扱われます。
- サポート用コンテナは、ゲームサーバーをサポートする追加のソフトウェアを実行します。これは「sidecar」コンテナの概念に似ています。これにより、サポートソフトウェアをゲームサーバーと並行して実行・スケールさせつつ、別のコンテナとして管理できます。ゲームサーバーコンテナグループでは、サポートコンテナを 0 個以上定義できます。インスタンス単位のコンテナグループでは、すべてのコンテナがサポート用コンテナです。どのサポート用コンテナでも、必須として指定できます。

コンピューティング

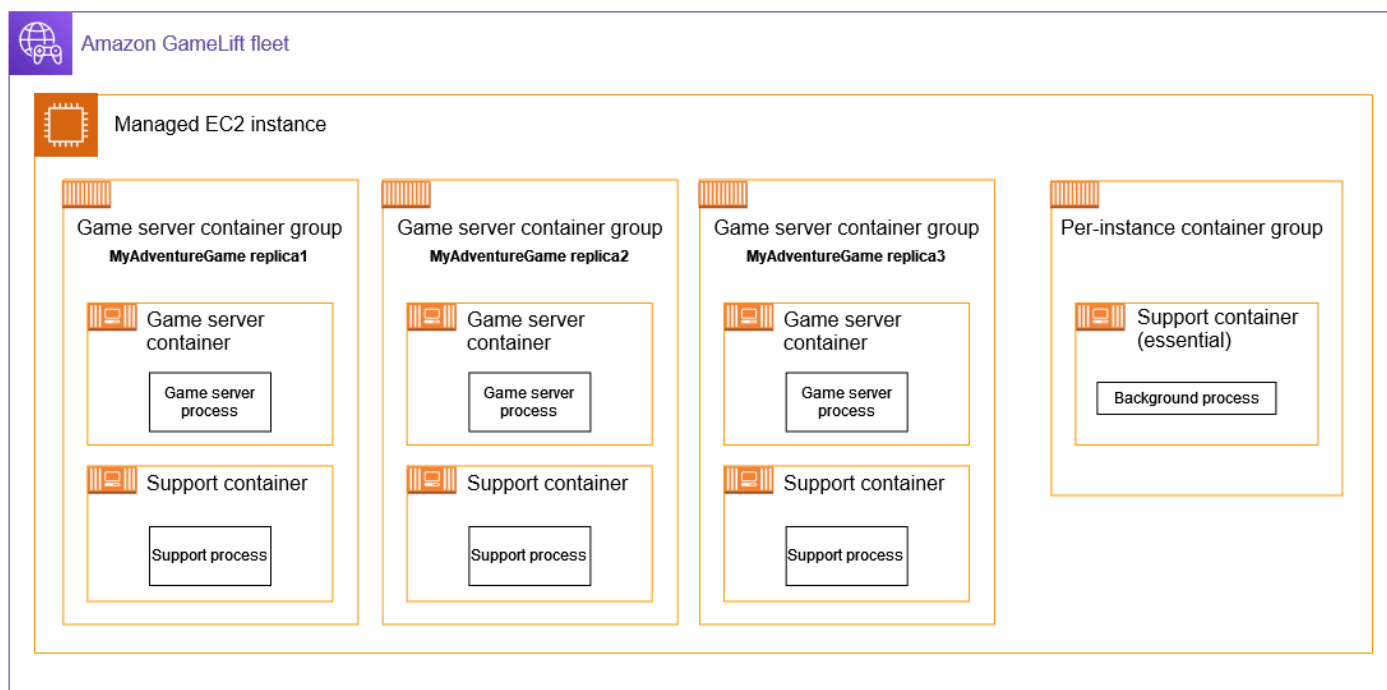
コンピューティングは、フリートインスタンス上のゲームサーバーコンテナグループのコピーを表します。

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がゲームサーバーコンテナグループの 1 つのコピーを保持します。このコンテナグループには、1 つのゲームサーバープロセスを実行する 1 つのゲームサーバーコンテナが含まれます。この例では、コンテナフリートは、インスタンスごとにゲームサーバーコンテナグループのコピーを 1 つ配置するように構成されています。このアーキテクチャでは、各インスタンスが 1 つのゲームサーバープロセスを実行します。



2 番目の図は、より複雑なコンテナフリートアーキテクチャを示しています。この構造では、フリートにゲームサーバーコンテナグループとインスタンス単位のコンテナグループの両方が含まれます。ゲームサーバーコンテナグループには、ゲームサーバープロセス用とサポートプロセス用の個別のコンテナがあります。フリートは、各フリートインスタンスにゲームサーバーコンテナグループのコピーを 3 つ配置するよう構成されています。インスタンス単位のコンテナグループは、レプリケートされません。この例では、コンテナフリートは、インスタンスごとにゲームサーバーコンテナグループのコピーを 3 つ配置するよう構成されています。このアーキテクチャでは、各インスタンスが 3 つのゲームサーバープロセスを実行します。



主要機能

このセクションでは、Amazon GameLift Servers がいくつかの基本的なコンテナの概念をどのように実装するかをまとめています。コンテナフリートの操作方法については、このガイド内の関連トピックを参照してください。

アクティブなフリートの更新

マネージドコンテナは、ホストされたソフトウェアとコンテナアーキテクチャのライフサイクル管理を支援する高度なサポートを提供します。コンテナイメージを含むコンテナ定義を更新し、既存のフリートに変更をデプロイできます。この機能を利用すると、開発中のコンテナへの変更をより迅速かつ容易に反映できます。また、ソフトウェアバージョンのビルド、デプロイ、および追跡を支援する機能も提供します。これには次の機能が含まれます。

- コンテナグループ定義の更新とバージョニングを管理します。コンテナイメージや設定を含む、コンテナグループ定義のほぼすべてのプロパティを更新できます。コンテナを更新すると、Amazon GameLift Servers が更新にバージョン番号を自動的に割り当て、デフォルトですべてのバージョンを保持します。任意のバージョンにアクセスでき、必要に応じてバージョンを削除できます。コンテナフリートを作成するときに、デプロイするコンテナグループ定義とバージョンを指定できます。
- 既存のコンテナフリートを、新しいコンテナグループ定義と設定で更新します。フリートインスタンスにデプロイ済みのフリートに、コンテナ更新をデプロイできます。AWS マネジメントコン

ソール または AWS SDK と CLI を使用して、各フリートロケーションの更新デプロイのステータスを追跡できます。

- フリートの更新をアクティブなフリートにどのようにデプロイするかを設定します。
 - ゲームセッション保護。アクティブなゲームセッションが終了するまで、そのセッションを実行しているフリートインスタンスを保護すること (安全なデプロイ) を選択できます。または、ゲームセッションの状況に関係なくフリートインスタンスを置き換えること (安全でないデプロイ) を選択できます。開発フェーズやテストフェーズでは、安全でないデプロイを使用してデプロイ時間を短縮できます。
 - 最小正常率。デプロイ中に維持したい正常なタスクの割合を指定します。この機能を使用すると、デプロイによって影響を受けるフリートインスタンスの数を決定できます。値を低くするとデプロイの速度が優先され、値を高くするとデプロイ全体を通じてゲームサーバーの可用性が高く維持されます。
 - デプロイの失敗戦略。デプロイが失敗した場合に実行するアクションを決定します。デプロイの失敗は、更新されたコンテナの一部のステータスチェックが失敗し、異常状態と見なされることを意味します。すべてのフリートインスタンスを以前のデプロイ状態に自動的にロールバックするよう、デプロイを設定できます。あるいは、デバッグ用途のために、障害状態のフリートインスタンスの一部を維持することもできます。

アクティブなフリートを更新できる機能は、ゲームサーバーソフトウェアに更新をデプロイしたい場合に非常に有用です。ゲームサーバー用に新しいコンテナイメージをビルドした後、デプロイは 2 つのステップで行います。まず、新しいイメージでコンテナグループ定義を更新し、次にコンテナフリートを更新します。Amazon GameLift Servers は必要に応じて残りのタスクを処理します。

コンテナのパッキング

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、共通の目標は、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、各フリートインスタンスに可能な限り多くの ゲームサーバーコンテナグループ をパッキングします。

Amazon GameLift Servers は、次の情報に基づいてインスタンスあたりの最大ゲームサーバーコンテナグループ数を計算し、この作業を支援します。

- フリートのインスタンスタイプと、その vCPU およびメモリリソース。
- ゲームサーバーコンテナグループ内のすべてのコンテナに必要な vCPU とメモリ要件。

インスタンス単位のコンテナグループが存在する場合、そのグループ内のすべてのコンテナに必要な vCPU とメモリ要件。

コンテナフリートを作成するときは、計算された最大値を使用するか、必要な数を指定できます。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テストし、最適なゲームサーバーパフォーマンスに必要なリソース要件を判断する計画を立ててください。

容量スケーリング

フリート容量は、フリートが同時にホストできるゲームセッション数を表します。また、フリートがサポートできる同時プレイヤー数に基づいて容量を算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。

コンテナフリートは、各フリートインスタンスで特定数の同時ゲームサーバープロセスを実行するように構成されています。(これは、(1) インスタンスあたりの ゲームサーバーコンテナグループ 数と、(2) 各コンテナグループで実行されるゲームサーバープロセスの数に基づいて計算できます。) インスタンスあたりの同時ゲームサーバー数は、それぞれのフリートインスタンスを追加または削除した場合の影響を示します。例えば、コンテナフリートが各ゲームサーバーコンテナグループで 1 つのゲームサーバープロセスを実行し、各フリートインスタンスが 100 のゲームサーバーコンテナグループを保持している場合、フリートの同時ゲームセッションのホスティング容量は 100 単位で増減します。各ゲームセッションが 10 個のプレイヤースロットを持つ場合、プレイヤーをホストするフリート容量は 1000 単位で増減します。

コンテナフリートでは、Amazon GameLift Servers が提供する任意の容量スケーリング方法を使用できます。具体的には次のとおりです。

- 必要な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をターゲットとして、自動スケーリングを設定します (ターゲット追跡)。この方法は、参加プレイヤーがすぐにゲームに入れるよう、一定量のアイドルホスティングリソースを自動的に維持します。プレイヤーの需要の増減に応じて、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。この方法では、選択したフリートメトリクスに基づいてスケーリングできます。

ゲームクライアント/サーバー接続

Amazon GameLift Servers が管理するフリートでは、ゲームクライアントがクラウドホスト型ゲームサーバーに直接接続します。ゲームクライアントがゲームへの参加をリクエストすると、Amazon GameLift Servers はゲームセッションを検索し、接続情報 (IP とポート) をゲームクライアントに提供します。フリートの特定のポート範囲 (インバウンドアクセス許可) を開くことで、フリートインスタンスへの外部アクセスを制御できます。インバウンドアクセス許可は、受信トラフィックに対し

て開いているポートを決定します。すべてのポートをすばやくシャットダウンしたり、開くポートを数個に制限したり、すべてのポートを開くことができます。

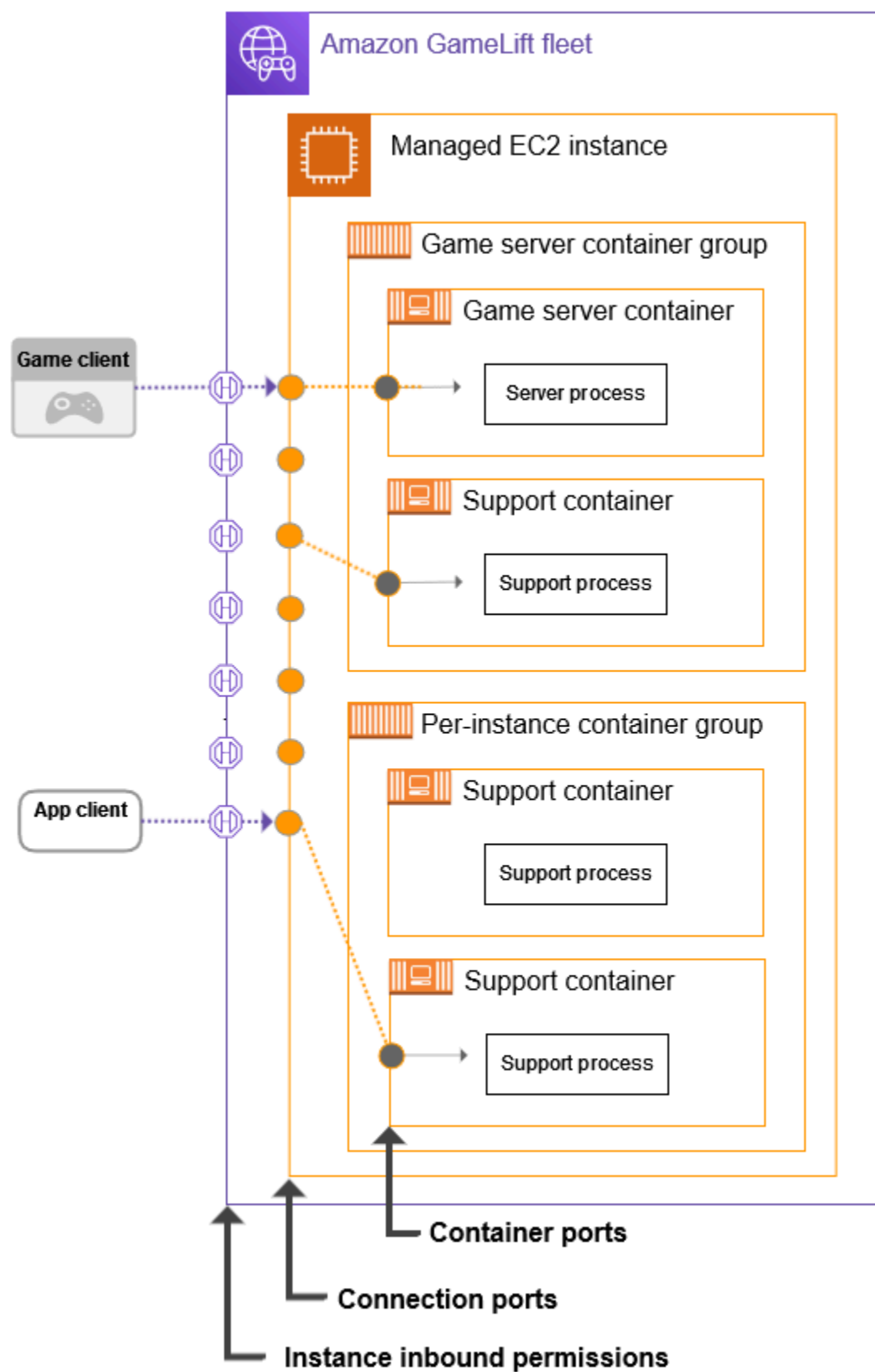
マネージドコンテナフリートには、コンテナで実行されているプロセスへのアクセスを許可する追加の設定が必要です。コンテナ定義を作成する際は、接続を受け付ける各プロセスごとに、使用するポートのセットを指定します。これには、以下が含まれます。

- ゲームサーバーコンテナで同時に実行されるすべてのゲームサーバープロセス。ゲームセッションに参加するため、ゲームサーバープロセスはゲームクライアントからの接続を受け入れる必要があります。
- 外部ソースが接続する必要があるサポートコンテナ内のプロセス。例えば、テストアプリケーションにリモート接続できます。

内部向けコンテナポートを設定すると、Amazon GameLift Servers はその設定を使用して、ゲームクライアントやその他のアプリケーションが接続できる外部向けインバウンドアクセス許可を算出します。また Amazon GameLift Servers は、インバウンドアクセス許可と、プレイヤーがコンテナ内のゲームセッションへアクセスする際に使用する個々のコンテナポートとのマッピングも管理します。この内部マッピングは、ゲームサーバーがコンテナポートへの直接アクセスから保護されることで、追加のセキュリティレイヤーを提供します。必要に応じて、フリートの外部向けポート設定をカスタマイズできます。コンテナフリートポートを手動で設定する方法の詳細については、「[ネットワーク接続を構成する](#)」を参照してください。

コンテナフリートのポート設定はいつでも変更できます。この変更には、フリート更新のデプロイが必要です。

次の図は、コンテナフリート間のポート接続の役割を示しています。図に示すように、個々のコンテナにポートを設定すると、Amazon GameLift Servers がその情報を使用して、各コンテナポートに対応させるために必要な数のポートをフリートインスタンス上に構成します。外部向けインスタンスのインバウンドアクセス許可と、内部向け接続ポートの両方は、手動で設定しない限り、Amazon GameLift Servers によってフリート用に算出されます。



コンテナのログ記録

マネージドコンテナフリートでは、標準出力 (および標準エラー) ストリームがすべてのコンテナでキャプチャされます。これには、ゲームサーバーのゲームセッションログが含まれます。コンテナフリートは、複数のオプションのいずれかを使用して出力ストリームを処理するように設定できます。

- コンテナ出力を Amazon CloudWatch ログストリームとして保存します。各ログストリームはフリート ID とコンテナを参照します。フリートに対してこのログ記録オプションを選択した場合、フリートからのすべてのログストリームを整理する CloudWatch ロググループを指定します。その後、CloudWatch 機能を使用して、必要に応じてログデータを検索および分析できます。
- コンテナ出力を Amazon Simple Storage Service (Amazon S3) バケットに保存します。必要に応じて、コンテンツを表示、共有、またはダウンロードできます。
- ログ記録をオフにします。このシナリオでは、コンテナ出力は保存されません。

Amazon GameLift Servers は、マネージドコンテナフリートから AWS アカウントの CloudWatch または Amazon S3 サービスにログデータを送信します。データを表示するには、AWS または他の AWS マネジメントコンソール ツールを使用してアカウントにサインインし、個々の サービスを操作します。コンテナフリートのサービスロールを作成することで、これらのアクションを実行するための制限付きアクセスを Amazon GameLift Servers に付与します。

コンテナフリートのログ記録設定はいつでも変更できます。この変更には、フリート更新のデプロイが必要です。

コンテナフリートと Amazon GameLift Servers エージェント

一般的に使用されるコンテナアーキテクチャは、コンテナごとに 1 つのプロセスを実行します。Amazon GameLift Servers コンテナフリートでは、ゲームサーバーグループに 1 つのゲームサーバーコンテナがあり、そのコンテナで 1 つのゲームサーバープロセスを実行します。このアーキテクチャでは、Amazon GameLift Servers が フリートインスタンス 上の各 ゲームサーバーコンテナグループ 内にある単一の 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。

各 ゲームサーバーコンテナグループで複数の ゲームサーバー プロセスを実行する コンテナアーキテクチャを構築する場合、すべての プロセスのライフサイクルを管理する方法が必要です。これは、必要に応じてプロセスを起動・停止・置き換え、同時に実行するプロセス数を管理し、障害発生時の対処を行うといった作業を含みます。

これらのタスクに Amazon GameLift Servers エージェントを使用することを選択できます。コンテナフリートでは、エージェントが「実行指示 (ランタイム指示)」を処理し、どの実行ファイルを何個

起動するかを指定し、起動時のパラメータを渡し、ゲームサーバーの起動ルールを設定します。例えば、ランタイム指示により、本番環境用として 10 個のゲームサーバープロセスを維持し、テスト用として特別な起動パラメータを設定したゲームサーバープロセスを 1 個維持するよう、エージェントに指示できます。

コンテナフリートでエージェントを使用するには、コンテナイメージにエージェントを組み込み、必要なランタイム指示一式を含めます。エージェントの詳細については、「[Amazon GameLift Servers エージェントの操作](#)」を参照してください。

Amazon GameLift Servers マネージドコンテナフリートを作成する

Amazon GameLift Servers マネージドコンテナフリートを作成して、コンテナ化されたゲームサーバーを AWS クラウドにデプロイしてホストします。コンテナフリートを作成するときは、1 つ以上のコンテナイメージ (ゲームサーバービルドを含む 1 つ以上のイメージ) と設定を指定するコンテナグループ定義を指定します。

新しいマネージドコンテナフリートリソースを作成すると、フリート作成の最初のフェーズがすぐに開始されます。マネージドフリートの作成は、Amazon GameLift Servers が EC2 インスタンスをプロビジョニングし、ランタイム環境をインストールし、コンテナグループをインスタンスにデプロイして、ゲームサーバープロセスの起動を開始するときに、いくつかのフェーズを通過します。ゲームサーバーが構築するランタイム環境に応じて、Amazon GameLift Servers はフリートの作成時に最新バージョンの Amazon マシンイメージ (AMI) をデプロイします (フリート内のすべての将来のインスタンスは同じバージョンを使用します)。フリートのステータスは、コンソールまたは AWS Command Line Interface (AWS CLI) を使用してモニタリングできます。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 Servers フリートの問題をデバッグする](#)」を参照してください。

空のコンテナフリートを作成し、後でフリートのコンテナグループ定義を追加または更新できます。コンテナグループ定義なしでフリートを作成すると、フリートはアクティブステータスに到達しません。

Note

ベストプラクティスとして、ホストされているゲームサーバーでの安全で最新のランタイム環境を維持するために、30 日ごとにフリートを置き換えることをお勧めします。そのためには、新しいフリートを作成し、プレイヤートラフィックをそのフリートに移行する必要があります。ガイダンスについては、「[Amazon GameLift Servers のセキュリティに関するベストプラクティス](#)」を参照してください。

コンテナフリートを作成するには、[Amazon GameLift Servers コンソール](#)または AWS Command Line Interface AWS CLI を使用します。

Console

[Amazon GameLift Servers 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、[マネージドコンテナ: フリート] を選択します。[フリート] ページで、[コンテナフリートの作成] を選択します。

ステップ 1: マネージドコンテナフリートの詳細を定義する

1. [コンテナフリートの詳細] セクションに、フリートの説明を入力します。
2. フリート用の IAM ロールを指定します。このロールには、Amazon GameLift Servers がコンテナフリートを管理するために必要なアクセス許可があります。必要なサービスロールの作成については、「[Amazon GameLift Servers 用に IAM サービスロールをセットアップする](#)」を参照してください。
3. ログ設定オプションを選択します。デフォルトでは、CloudWatch オプションが選択されています。選択したオプションに基づいて必要な情報を入力します。
4. フリートにコンテナグループを追加します。これは任意の手順です。後で追加する計画を持つコンテナグループなしでフリートを作成できます。コンテナグループのないフリートは、フリートインスタンスをデプロイせず、ゲームをまだホストできませんが、フリートリソースが作成されます。
 - ゲームサーバー用コンテナグループ定義を選択します。任意で、デプロイする定義のバージョンを指定します。バージョン番号を指定しない場合は、Amazon GameLift Servers が自動的に最新バージョンを使用します。
 - 任意で、インスタンスごとのコンテナグループ定義とバージョンを追加します。バージョン番号を指定しない場合は、Amazon GameLift Servers が自動的に最新バージョンを使用します。
5. [その他の詳細]で、いくつかのオプションカスタマイズを設定できます。コンテナフリートの作成にこれらの設定は必要ありません。

ステップ 2: インスタンスの詳細を定義する

1. インスタンスのデプロイでは、インスタンスをデプロイするリモートロケーションを 1 つ以上選択します。ホームリージョンが自動的に選択されます (これはフリートを作成するリージョンです)。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、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この要件の影響を受けません。
- 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。詳細については、「AWS Local Zones ユーザーガイド」の「[Local Zones の開始方法](#)」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、各コンテナグループに設定した合計制限に基づいて自動的に計算します。コンソールは、リソース要件と入力したロケーションに基づいて、利用可能なインスタンスタイプの一覧をフィルタリングします。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。選択に応じて、フリートインスタンスごとの希望するゲームサーバーコンテナグループ設定を確認してください。選択に応じて、インスタンスごとに必要なゲームサーバーコンテナグループの設定を確認することを検討してください。

ステップ 4: 確認して作成する

- フリートの設定内容を確認します。

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 Servers フリート設定を更新する](#)」を参照してください。フリート容量を更新できるのは、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 Servers を使用したゲームホスティング容量のスケーリング](#)」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。

完了したら、[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。初期状態では、ステータスは NEW です。Amazon GameLift Servers はフリート作成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。フリートがゲームセッションをホストできる状態になると、フリートのステータスが ACTIVE になります。

AWS CLI

AWS CLI でコンテナ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-container-fleet` コマンドを使用します。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-container-fleet](#)」を参照してください。

次の `create-container-fleet` リクエスト例では、以下のような特徴を持つ新しいコンテナフリートを作成します。

- ContainerGroupsConfiguration は、ゲームサーバーコンテナグループ定義のみを指定します: MyAdventureGameContainerGroup。各フリートインスタンスにデプロイされるゲームサーバーコンテナグループの数は、Amazon GameLift Servers によって計算されます。
- フリートはデフォルトで `c5.large` オンデマンドインスタンスを使用します。
- デフォルトでは、フリートは Amazon GameLift Servers によって計算された接続ポートとインバウンドアクセス許可ポートのセットを開きます。コンテナグループを次の場所にデプロイします。

```
aws gamelift create-container-fleet \  
  --fleet-role-arn arn:aws:iam::MyAccount:role/MyContainersRole \  
  --game-server-container-group-definition-name "rn:aws:gamelift:us-  
west-2:111122223333:containergroupdefinition/MyAdventureGameContainerGroup:2" \  
  --fleet-name MyAdventureGameContainerGroup
```


create-fleet のリクエストが成功すると、リクエストした構成設定と新しいコンテナフリート ID が含まれたフリート属性のセットが Amazon GameLift Servers から返されます。Amazon GameLift Servers は即座にフリートアクティベーションプロセスを開始し、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-container-fleet](#)
- [describe-fleet-capacity](#)
- [describe-fleet-port-settings](#)
- [describe-fleet-utilization](#)
- [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-container-fleet](#)
- [update-fleet-capacity](#)
- [update-fleet-port-settings](#)
- [create-fleet-locations](#)
- [delete-fleet-locations](#)

コンテナフリートAmazon GameLift Servers用のコンテナグループ定義を作成する

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記述します。これは、どのコンテナイメージをフリートにデプロイし、どのように実行するかを Amazon GameLift Servers に示すブループリントです。コンテナフリートを作成するときに、どのコンテナグループ定義をデプロイするかを指定します。コンテナイメージの詳細については、[コンテナフリートコンポーネント](#)を参照してください。

開始する前に

コンテナグループ定義の作成を開始する前にすべきことに関するヒント:

- コンテナイメージを確定し、コンテナグループを作成する予定 AWS リージョン と同じ の Amazon Elastic Container Registry (Amazon ECR) リポジトリにプッシュします。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をAmazon GameLift Serversキャプチャし、コンテナフリートにデプロイするときにスナップショットを使用します。「[Amazon GameLift Servers 用のコンテナイメージを構築する](#)」を参照してください。

- コンテナ定義を JSON ファイルとして作成します。コンテナグループ定義には、1 つまたは複数のコンテナ定義が含まれます。AWS CLI for for を使用してコンテナグループ定義を作成する場合は、JSON ファイルを使用できますAmazon GameLift Servers。
- AWS ユーザーに Amazon ECR リポジトリにアクセスするための IAM アクセス許可があることを確認します。「[Amazon GameLift Servers 向け IAM アクセス許可の例](#)」を参照してください。

ゲームサーバーコンテナグループ定義を作成する

ゲームサーバーコンテナグループは、ゲームサーバーソフトウェアを実行します。ゲームサーバーコンテナグループには、ゲームサーバー実行ファイルを実行するゲームサーバーコンテナが 1 つあります。また、1 つ以上のサポートコンテナを持ち、ゲームサーバーを補助する追加ソフトウェアを実行することもできます。(これらは「サイドカー」コンテナと呼ばれることもあります。)

このトピックでは、Amazon GameLift Serversコンソールまたは AWS CLI ツールを使用してシンプルなゲームサーバー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。オプション機能の詳細については、[Amazon GameLift Servers コンテナフリートをカスタマイズする](#)を参照してください。

Note

ほとんどのコンテナグループ定義とコンテナ定義の設定は、作成後に変更できます。コンテナ定義を変更すると、Amazon GameLift Serversは更新されたコンテナイメージの新しいスナップショットをキャプチャします。

シンプルなゲームサーバーコンテナグループ定義を作成するには:

次の手順では、最小限必要なパラメータとAmazon GameLift Serversデフォルト値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。

Console

[Amazon GameLift Servers 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、マネージドコンテナ: グループ定義を選択します。コンテナグループ定義ページで、グループ定義の作成を選択します。

ステップ 1: コンテナグループ定義の詳細を定義する

1. コンテナグループ定義名を入力します。エイリアス名は AWS アカウント およびリージョンで一意であることが必要です。
2. ゲームサーバーコンテナグループタイプを選択します。
3. Total memory limitには、コンテナグループ内のすべてのコンテナで利用できる最大メモリリソースを入力します。この値の計算については、[リソース制限の設定](#)を参照してください。
4. Total vCPU limitには、コンテナグループ内のすべてのコンテナが利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、[リソース制限の設定](#)を参照してください。

ステップ 2: コンテナ定義を追加する

少なくとも、ゲームサーバーコンテナグループにはゲームサーバーコンテナが 1 つあります。コンソールで最初に作成されるコンテナ定義は、ゲームサーバーコンテナです。このステップでは、ゲームサーバーコンテナ定義に必要な最小設定を定義する方法について説明します。

1. コンテナ定義名を入力します。グループに定義された各コンテナには、一意の名前値が必要です。
2. ゲームサーバービルドに対応するコンテナイメージをリンクします。パブリックリポジトリまたはプライベートリポジトリのコンテナイメージに対する Amazon ECR image URI を入力します。以下の形式を使用できます。
 - イメージ URI のみ: [AWS #####].dkr.ecr.[AWS #####].amazonaws.com/[repository ID]
 - イメージ URI + digest: [AWS #####].dkr.ecr.[AWS #####].amazonaws.com/[repository ID]@[digest]
 - Image URI + tag: [AWS #####].dkr.ecr.[AWS #####].amazonaws.com/[repository ID]:[tag]
3. ゲームサーバービルドで使用する Amazon GameLift Servers Server SDK バージョンを指定します。コンテナフリートの場合、この値は 5.2.0 以上である必要があります。
4. 内部コンテナポート範囲 で、プロトコルを設定し、ポート範囲を定義します。範囲サイズは、このコンテナ内で実行される同時ゲームサーバープロセス数よりも大きくある必要があ

ります。ゲームサーバーコンテナがコンテナごとに 1 つのサーバープロセスのみを実行する場合、このポート数に必要なポートはわずかです。詳細については、[ネットワーク接続を構成する](#)を参照してください。

5. 必要に応じてコンテナを追加して、追加のサポートソフトウェアを実行します。追加のコンテナは自動的にサポートコンテナとして指定されます。ゲームサーバーコンテナグループは、1 つのゲームサーバーコンテナと最大 8 つのサポートコンテナを持つことができます。次の最小必須設定を指定します。

- コンテナ定義の名前
- ECR イメージの URI。
- 内部コンテナポート (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これを含めます。)

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、オプションで各コンテナ間の依存関係を設定できます。詳細については、[コンテナの依存関係の設定](#)を参照してください。

ステップ 3: 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。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. 完了したら、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。最初、ステータスは COPYING です。Amazon GameLift Servers は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を開始します。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 READY に変わります。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 ファイルに維持します。CLI コマンドでファイルを参照できます。スキーマの例については、[コンテナ定義 JSON ファイルを作成する](#)を参照してください。

コンテナグループ定義の作成

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、「AWS CLI コマンドリファレンス」の「[create-code-signing-config](#)」を参照してください。

この例では、ゲームサーバーコンテナグループ定義のリクエストを示しています。このグループのコンテナ定義を含む JSON ファイルを作成済みであ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 \  
  --container-group-type GAME_SERVER \  
  --total-memory-limit-mebibytes 4096 \  
  --total-vcpu-limit 1 \  
  --game-server-container-definition file://MyAdventureGameContainers.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で 1 つの JSON ファイルを作成することをお勧めします。ファイルを維持することで、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。例えば、次のようになります。

```
[~/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 Servers API リファレンスの [ContainerDefinitionInput](#) 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: ゲームサーバーコンテナ定義

Example

この例では、ゲームサーバーコンテナグループに必要な主要コンテナについて説明します。重要な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 Servers 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ます。

```
{
  "ContainerName": "MyAdventureGameServer",
  "ImageUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/gl-
containers:myadventuregame-server",
  "PortConfiguration": {
    "ContainerPortRanges": [
      {
        "FromPort": 2000,
        "Protocol": "TCP",
        "ToPort": 2010
      }
    ]
  },
  "ServerSdkVersion": "5.2.0"
}
```

Amazon GameLift Servers Anywhere フリート

自前のホスティングリソースで Amazon GameLift Servers 機能を利用する場合は、Anywhere フリートを使用します。Anywhere フリートは一般的に、反復型開発のテスト環境として、またはハイブリッドホスティングソリューションでマネージドフリートと共に使用されます。

Anywhere フリートは、ユーザーが提供および管理する一連のコンピューティングリソース (仮想または物理) で構成されます。コンピューティングは、ローカルのノートパソコンから遠隔のアウトポストまで、接続があればどのような地理的場所にでも配置できます。Anywhere フリートを立ち上げるときは、Amazon GameLift Servers を使用してフリートを登録することで、フリートにコンピューティングを追加します。各コンピューティングはそれぞれの IP アドレス (または DNS 名) で登録されるため、Amazon GameLift Servers はコンピューティングとの接続を確立できます。

ゲームサーバーソフトウェアを Anywhere フリートにデプロイするには、各コンピューティングにインストールし、ゲームサーバープロセスを起動します。起動された各ゲームサーバープロセス

は、Amazon GameLift Servers サービスへの接続を確立し、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します。既存の設定管理とデプロイツールを使用して、最初のデプロイとホスト管理タスクを処理できます。Amazon GameLift Servers で使用するには、次のような追加のタスクがいくつか必要です。

- コンピューティングを登録および登録解除して、フリートに追加または削除する。
- すべてのコンピューティングで最新の認証トークンを維持する。コンピューティング上のサーバープロセスは、Amazon GameLift Servers サービスに接続するときにこのトークンを使用します。

Note

Amazon GameLift Servers エージェントを使用して Anywhere フリートをデプロイし、こうした重要な管理タスクを自動化することもできます。「[Amazon GameLift Servers エージェントの操作](#)」を参照してください。

Anywhere フリートをセットアップおよび維持する方法については、以下のトピックを参照してください。

- [Amazon GameLift Servers Anywhere でホスティングするための開発ロードマップ](#)
- [Amazon GameLift Serversを利用するハイブリッドホスティングの開発ロードマップ](#)
- [Amazon GameLift Servers Anywhere を使用した反復型開発の準備](#)
- [Amazon GameLift Servers Anywhere フリートを作成する](#)
- [Amazon GameLift Servers フリート設定を更新する](#)

Anywhere フリート作成のワークフロー

Anywhere フリートの場合、Amazon GameLift Servers はフリートリソースのみをセットアップします。ユーザーがコンピューティングをセットアップしてフリートに登録し、ゲームサーバーソフトウェアをインストールして、ゲームセッションをホストするためのゲームサーバープロセスを開始します。

1. Amazon GameLift Servers は、フリートのホームリージョンにフリートリソースを作成します。フリートステータスとカスタムロケーションステータスは、[新規] に設定されます。
2. Amazon GameLift Servers 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。

3. フリートリソースが作成されると、Amazon GameLift Servers はフリートステータスをアクティブに設定します。この時点で、新しいコンピューティングをフリートに登録できます。

Amazon GameLift Servers Anywhere フリートを作成する

このトピックでは、Amazon GameLift Servers Anywhere フリートを作成する方法を説明します。Anywhere フリートを使用すると、独自のコンピューティングリソースで Amazon GameLift Servers ゲームセッションをホスティングしながら、コアとなるゲームセッション管理機能を利用できます。オンプレミスハードウェアまたはその他のクラウドベースリソース用の Anywhere フリートを作成します。

Anywhere フリートは一般的に、ハイブリッドホスティングソリューションで Amazon GameLift Servers マネージドフリートと共に使用されます。また、Amazon GameLift Servers でホスティングするためのゲームを開発する際にテスト環境としても使用できます。以下のトピックでは、Amazon GameLift Servers Anywhere フリートをゲームホスティングソリューションに組み込むべき場合やその方法の詳細について説明します。

- [ハイブリッドホスティング](#)
- [Amazon GameLift Servers のホスティングフリートをデプロイする](#)
- [Amazon GameLift Servers Anywhere を使用した反復型開発の準備](#)

Anywhere フリートはセルフマネージド型であるため、フリートのセットアップには追加の作業が必要です。Anywhere フリートをゲームセッションとプレイヤーをホストできる状態にするには、次のタスクを完了する必要があります。

トピック

- [開始する前に](#)
- [カスタムロケーションを作成する](#)
- [Anywhere フリートを作成する](#)
- [フリートにコンピューティングを追加する](#)
- [ゲームサーバーを起動する](#)

開始する前に

Anywhere フリートを作成する前に、次のタスクを実行します。詳細については、「[Amazon GameLift Servers Anywhere でホスティングするための開発ロードマップ](#)」または「[Amazon GameLift Serversを利用するハイブリッドホスティングの開発ロードマップ](#)」を参照してください。

- ゲームサーバーコードを Amazon GameLift Servers サーバー SDK バージョン 5.x (またはそれ以降) と統合します。すべてのゲーム統合タスクを完了する必要はありません。ゲームサーバービルドに必要なタスクのみでかまいません。一般的な手法は、ローカルマシンを Anywhere フリートとして設定し、コマンドラインインターフェイスを使用してゲームサーバーの統合をテストする方法です (「[Amazon GameLift Servers Anywhereでローカルテストを設定する](#)」を参照)。追加のコンポーネント (Amazon GameLift Servers 対応ゲームクライアントなど) は、開発とともに組み込むことができます。
- Anywhere フリートコンピューティングにインストールするためのゲームサーバーソフトウェアをパッケージ化します。パッケージには、統合されたゲームサーバービルドと、ゲームサーバーの実行に必要なすべてのサポートソフトウェアが含まれている必要があります。
- Anywhere フリートで Amazon GameLift Servers エージェントを使用するかどうかを決定します。エージェントは、Amazon GameLift Servers で使用するサーバープロセスとコンピューティングの管理に関連する主要なタスクの一部を自動化する、コンピューティング上のプロセス管理ツールです。詳細については、「[Amazon GameLift Servers エージェントの操作](#)」を参照してください。

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。Anywhere フリートを作成するときは、少なくとも 1 つのカスタムロケーションが既に定義されている必要があります。追加のカスタムロケーションを作成し、いつでも既存のフリートに追加できます。

カスタムロケーションを作成するには

カスタムロケーションは、Amazon GameLift Servers コンソールまたは AWS Command Line Interface (AWS CLI) を使用して作成できます。

Console

[\[Amazon GameLift Servers コンソール\]](#) のナビゲーションペインを使用して、[ロケーション] ページを開きます。[ロケーションを作成] をクリックして、[作成] ダイアログボックスを開きます。

1. ダイアログボックスに、[ロケーション名] を入力します。ベストプラクティスとして、コンピューティングリソースの場所をわかりやすく説明する名前を使用します。地理的な場所、データセンター名、またはその他の場所識別子である場合があります。Amazon GameLift Servers はカスタムの場所の名前に custom- を追加します。
2. (オプション) カスタムロケーションにタグを追加します。タグはそれぞれ、1 つのキーとオプションの 1 つの値で設定されており、どちらもお客様側が定義します。タグを割り当てることで、AWS リソースを目的、所有者、環境など有用な方法で分類することができます。追加するタグごとに [新しいタグを追加] を選択します。
3. [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。custom- で始まる location-name 値を指定します。ベストプラクティスとして、コンピューティングリソースの場所をわかりやすく説明する名前を使用します。地理的な場所やデータセンター名など、その場所の識別子を使用するとよいでしょう。

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

Output

```
{  
  "Location": {  
    "LocationName": "custom-location-1",  
    "LocationArn": "arn:aws:gamelift:us-east-1:111122223333:location/custom-  
location-1"  
  }  
}
```

Anywhere フリートを作成する

ご自身で所有するコンピューティングリソース用に Anywhere フリートを作成します。新しい Anywhere フリートは最初は空で、フリートにコンピューティングを登録することで追加します。

新しく Anywhere フリートを作成すると、フリートステータスは NEW から ACTIVE へすばやく移行します。フリートが ACTIVE に達したら、コンピューティングを追加できます。

Anywhere フリートを作成するには

Anywhere フリートを作成するには、Amazon GameLift Servers コンソールまたは AWS Command Line Interface (AWS CLI) を使用します。

Console

[\[Amazon GameLift Servers コンソール\]](#) のナビゲーションペインを使用して、[フリート] ページを開きます。[フリートの作成] を選択して、フリート作成ワークフローを開始します。

ステップ 1 コンピューティングタイプを選択する

[Anywhere] オプションを選択し、[次へ]を選択します。

ステップ 2 フリートの詳細を定義する

このステップでは、フリート全体の主要な設定をいくつか指定します。

1. [フリートの詳細] セクションに以下の内容を入力します。
 - a. フリートの [名前] を入力します。フリートの一覧を表示したときにフリートタイプを識別しやすいフリート命名パターンを使用することをお勧めします。
 - b. [説明] には、フリートの簡単な説明を入力します。
2. 必要に応じて、オプションの [その他の詳細] を設定します。これらのフリート設定は後で更新できます。
 - a. 本番稼働用または本番稼働前テスト用のフリートを作成するときは、この設定を使用してフリートのコンピューティングの時間あたりのコスト値を指定します。Amazon GameLift Servers は、ゲームセッション配置プロセス中にこの情報を使用して、コストに基づいてホスティングリソースを選択できます。
 - b. このフリートのメトリクスデータとその他のメトリクスデータを組み合わせる場合は、[メトリクスグループ] 名を指定します。組み合わせるすべてのフリートに同じメトリクスグループ名を使用します。メトリクスグループのメトリクスを表示して、集計データを確認します。
3. オプションで、カスタムロケーションにタグを追加します。タグはそれぞれ、1 つのキーとオプションの 1 つの値で設定されており、どちらもお客様側が定義します。タグを割り当てることで、AWS リソースを目的、所有者、環境など有用な方法で分類することができます。追加するタグごとに [新しいタグを追加] を選択します。
4. [次へ] を選択してワークフローを続行します。

ステップ 3 カスタムロケーションを選択する

このステップでは、このフリートに追加する予定のコンピューティングの物理的な場所を特定します。ここで 1 つまたは複数のロケーションを指定し、必要に応じて後でロケーションを追加または削除できます。

1. [カスタムロケーション] で、フリートのコンピューティングに 1 つまたは複数のロケーションを選択します。リストには、現在選択されている AWS リージョン で定義されているすべてのカスタムロケーションが含まれます。フリートに追加する新しいカスタムロケーションを定義するには、[ロケーションを作成] を選択します。
2. [次へ] を選択してワークフローを続行します。

ステップ 4 確認して作成する

フリートを作成する前に、設定を確認します。

新しいフリートをデプロイする準備ができたなら、[作成] を選択します。Amazon GameLift Servers はフリートのアクティベーションプロセスを直ちに開始し、一意の ID を割り当ててフリートを NEW ステータスに移行します。フリートの進捗状況は [フリート] ページで追跡できます。

AWS CLI

[create-fleet](#) コマンドを使用して、コンピューティングタイプ ANYWHERE のフリートを作成します。名前と少なくとも 1 つのカスタムロケーションを指定します。Amazon GameLift Servers は、現在のデフォルト AWS リージョン に Anywhere フリートリソースを作成します (または、`--region` タグを追加して別の AWS リージョン を指定することもできます)。

次のリクエスト例では、最小限の必要な設定で新しいフリートを作成します。*FleetName* と *custom-location* を自分の情報で置き換えます。

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

レスポンスの例

```
{
```

```
"FleetAttributes": {
  "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
  "FleetArn": "arn:aws:gamelift:us-west-2:111122223333:fleet/fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
  "Name": "HardwareAnywhere",
  "CreationTime": "2023-02-23T17:57:42.293000+00:00",
  "Status": "ACTIVE",
  "MetricGroups": [
    "default"
  ],
  "CertificateConfiguration": {
    "CertificateType": "DISABLED"
  },
  "ComputeType": "ANYWHERE"
}
```

作成時、新しい Anywhere フリートは、すばやくフリートステータス ACTIVE に移行します。フリートが ACTIVE に達したら、コンピューティングを追加できます。

レスポンスにはフリートのロケーションが含まれていないことに注目してください。フリートの全詳細については、[describe-fleet-attributes](#) および [describe-fleet-location-attributes](#) を呼び出すことで取得できます。

フリートにコンピューティングを追加する

コンピューティングリソースをフリートに追加し、ゲームセッションをホストできる状態にするには、次のタスクを実行します。

- コンピューティングをフリートに登録します。登録すると、どの物理ホスティングリソースがフリートに含まれるかを Amazon GameLift Servers に通知します。
- コンピューティングの認証トークンをリクエストします。コンピューティングで実行される各ゲームサーバーは、Amazon GameLift Servers サービスに接続するためにこのトークンが必要です。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。

Note

Amazon GameLift Servers エージェントを使用してゲームサーバーソフトウェアをデプロイする場合は、このステップを省略できます。エージェントは各コンピューティングを自動的

に登録し、コンピューティング用の有効な認証トークンを維持します。「[Amazon GameLift Servers エージェントの操作](#)」を参照してください。

AWS CLI を使用するか、AWS SDK for Amazon GameLift Servers にプログラムから呼び出しを行うことで、コンピューティングに登録し、認証トークンをリクエストできます。これらのアクションは、Amazon GameLift Servers コンソールで行うことはできません。

ベストプラクティスとして、各コンピューティングにスタートアップスクリプトを追加して、このタスクの両方を自動化することをお勧めします。スタートアップスクリプトは、`register-compute` と `get-compute-auth-token` コマンドの両方を自動的に呼び出します。また、タスクを自動化して、コンピューティングの存続期間を通じて認証トークンを定期的に更新し、シャットダウン時にコンピューティングの登録を解除することもできます。

各スタートアップアクションは、コンピューティング上に保存する必要があるコンピューティング固有の値を返します。ゲームサーバープロセスがコンピューティングで起動すると、Amazon GameLift Servers サービスへの接続を初期化する際に、これらの値をサーバーパラメータとして渡す必要があります (サーバー SDK リファレンスの [ServerParameters](#) を参照)。このコンピューティング固有の値 (またはその保存場所) を環境変数として設定することをお勧めします。Amazon GameLift Servers エージェントを使用している場合、このタスクはユーザーに代わって処理されます。コンピューティング固有の値は次のとおりです。

- `register-compute` は `GameLiftServiceSdkEndpoint` の値を返します。この値を `webSocketUrl` サーバーパラメータに設定します。
- `compute-auth-token` は認証トークンを返します。この値を `authToken` サーバーパラメータに設定します。

AWS CLI

以下の手順では、AWS CLI を使用して各リクエストを手動で送信する方法を説明します。

コンピューティングに登録するには

[register-compute](#) を呼び出してコンピューティングに登録します。コンピューティングを追加するフリートの ID を特定します。コンピューティング情報 (わかりやすい名前、IP アドレス、ロケーション) を指定します。コンピューティングのロケーションは、フリートに既に関連付けられているカスタムロケーションである必要があります。別のカスタムロケーションを使用する場合は、Amazon GameLift Servers コンソールを使用してフリートを更新するか、AWS CLI コ

マンド [create-fleet-locations](#) を呼び出してフリートにカスタムロケーションを追加します。

次の例では、コンピューティングとフリートのプレースホルダー値を置き換えます。Anywhere フリートを作成すると、fleet-id 値が返されます。フリートの全詳細については、[describe-fleet-attributes](#) および [describe-fleet-location-attributes](#) を呼び出すことで取得できます。

```
aws gamelift register-compute \  
  --compute-name HardwareAnywhere \  
  --fleet-id arn:aws:gamelift:us-east-1:111122223333:fleet/  
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \  
  --ip-address 10.1.2.3 \  
  --location custom-location-1
```

出力例

```
{  
  "Compute": {  
    "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
    "FleetArn": "arn:aws:gamelift:us-west-2:111122223333:fleet/  
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
    "ComputeName": "HardwareAnywhere",  
    "ComputeArn": "arn:aws:gamelift:us-west-2:111122223333: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-west-2.api.amazongamelift.com"  
  }  
}
```

認証トークンをリクエストするには

[get-compute-auth-token](#) を呼び出して有効な認証トークンをリクエストします。コンピューティングを登録します。フリート ID とコンピューティング名を特定します。

次の例では、コンピューティングとフリートのプレースホルダー値を置き換えます。Anywhere フリートを作成すると、fleet-id 値が返されます。フリートの全詳細については、[describe-fleet-attributes](#) を呼び出すことで取得できます。コンピューティング情報を検索するに

は、フリート ID を使用して [list-compute](#) を呼び出し、フリートに登録されているすべてのコンピューティングを確認します。

```
aws gamelift get-compute-auth-token \  
  --fleet-id arn:aws:gamelift:us-east-1:111122223333:fleet/  
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \  
  --compute-name HardwareAnywhere
```

出力例

```
{  
  "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
  "FleetArn": "arn:aws:gamelift:us-east-1:111122223333:fleet/  
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
  "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"  
}
```

ゲームサーバーを起動する

Anywhere フリートを作成し、フリートに 1 つまたは複数のコンピューティングを追加したら、ゲームサーバーの実行を開始する準備が整います。

ステップ 1 ゲームサーバーソフトウェアをインストールする

ゲームサーバービルドとすべての依存ソフトウェアを Anywhere フリートの各コンピューティングにインストールします。ゲームサーバービルドは、Amazon GameLift Servers サービスと通信するために必要な最小限の機能を備えた Amazon GameLift Servers サーバー SDK バージョン 5.x (またはそれ以降) と統合する必要があります。

ステップ 2 ゲームサーバーを実行するためのコンピューティングを準備する

各コンピューティングが登録されており、有効な認証トークンがあることを確認します。スクリプトを使用してこれらのタスクを管理する場合は、ゲームサーバープロセスを開始する前に、スクリプトが各コンピューティングで実行されていることを確認します。

ゲームサーバーソフトウェアで Amazon GameLift Servers エージェントをデプロイした場合は、エージェントの実行可能ファイルが起動することを確認します。

ステップ 3 ゲームサーバープロセスを起動する

ゲームサーバーの実行ファイルのインスタンスをコンピューティングで実行します。ゲームサーバービルドが適切に統合されている場合、ゲームサーバープロセスは有効な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()` を呼び出します。サーバープロセスがゲームセッションをホストできる状態になると、`ProcessReady()` を呼び出します。

Note

Amazon GameLift Servers エージェントでゲームサーバーソフトウェアをデプロイした場合は、このステップを省略できます。エージェントは、指定したランタイム手順に基づいてゲームサーバープロセスを自動的に起動します。

サーバープロセスのアクティブ化とアクティブなサーバープロセスのメトリクスを確認することで、進行状況をモニタリングできます。「[フリートの Amazon GameLift Servers メトリクス](#)」を参照してください。ゲームサーバープロセスが初期化に失敗した場合は、プロセスが実行中のコンピューティングに適したサーバーパラメータ値を取得していることを確認します。

ハイブリッドホスティングソリューションの構築

ハイブリッドホスティングソリューションは、複数のゲームホスティングリソースのソースを組み合わせ、ゲームをホストします。これには、で実行される Amazon GameLift Servers マネージドフリート AWS クラウド や、自分で提供および管理しているリソースが含まれます。このトピックでは、ハイブリッドソリューションを構築するための一般的なパターンについて説明し、セルフマネージドゲームホスティングと Amazon GameLift Servers が管理するクラウドベースのゲームホスティングを正常に組み合わせる方法に関するヒントを提供します。

一般的なハイブリッドパターン

- **コスト最適化:** ベースライン容量には Anywhere フリート、ピーク需要にはマネージドフリートなど、シナリオごとに最も費用対効果の高いフリートタイプを使用します。
- **地理的柔軟性:** 既存のインフラストラクチャや特定のコンプライアンス要件があるトラフィックの多いフリート AWS リージョン や Anywhere フリートにマネージドフリートをデプロイします。
- **リスク軽減:** 複数のフリートタイプとプロバイダーに負荷を分散することで、単一のホスティングアプローチへの依存を軽減します。

- 段階的な移行: サービスの継続性を維持しながら、オンプレミスまたは他のホスティングソリューションから AWS 段階的に移行します。

開発と本番の分割

開発環境と本番環境で異なるフリートタイプを使用します。

- 開発: 費用対効果の高い開発とテストのための Anywhere フリート
- 本番稼働: スケーラブルで信頼性の高い本番ホスティング向けの、マネージド Amazon EC2 またはコンテナフリートです。

リージョンの最適化

AWS リージョン 特性に基づいてフリートタイプを最適化します。

- トラフィックが多い AWS リージョン: 需要の変化に応じて自動スケーリングを行うマネージドフリート
- 特化 AWS リージョン: コンプライアンス、データ主権、または既存のインフラストラクチャのための Anywhere フリート

キャパシティ階層化

容量階層ごとに異なるフリートタイプを使用します。

- ベースライン容量: Anywhere フリートまたはリザーブドインスタンスによる予測可能な負荷
- バーストキャパシティ: ピーク需要に合わせて自動スケーリングを行うマネージドフリート
- オーバーフロー容量: スポットインスタンス、または極端なピーク AWS リージョン 時の追加

実装の考慮事項

ハイブリッドソリューションを構築するときは、次の主要な要因を考慮してください。

ゲームセッションキューの設定

すべてのフリートタイプを含めるようにゲームセッションキューを設定し、ハイブリッドインフラストラクチャ全体で最適な配置を実現するために、適切な優先順位とレイテンシーの設定を行います。

モニタリングとオブザーバビリティ

ハイブリッドソリューションのパフォーマンス、容量、コストを可視化するため、すべてのフリートタイプに対して包括的なモニタリングを実装します。

運用の複雑さ

異なるデプロイプロセス、モニタリングツール、トラブルシューティング手順など、複数のフリートタイプを管理することで増大する運用の複雑さを考慮します。

ネットワーク接続

特にオンプレミスまたは異なるクラウドプロバイダーにある Anywhere フリートの場合、さまざまなホスティング環境間の信頼性の高いネットワーク接続を確保します。

ハイブリッドホスティングの開始方法

ハイブリッドホスティングソリューションを実装するには:

1. シンプルに開始: 1 つのフリートタイプから始めて、要件が明確になると徐々に他のフリートを追加します。
2. アーキテクチャを計画する: コスト、パフォーマンス、コンプライアンス、運用の複雑さに関する特定の要件に基づいてハイブリッドアーキテクチャを設計します。
3. キューを設定する: 適切な優先順位と配置戦略で、さまざまなフリートタイプにまたがるゲームセッションキューを設定します。
4. 徹底的にテストする: 本番環境に移行する前に、ハイブリッドインフラストラクチャ全体でゲームセッションの配置とフェイルオーバーのシナリオをテストします。
5. モニタリングと最適化: すべてのフリートタイプでパフォーマンスとコストを継続的にモニタリングし、必要に応じて設定を調整します。

ヒント

- マネージドホスティングリソースとセルフマネージドホスティングリソースの両方で、同じゲームクライアントとサーバーコンポーネントを使用できます。すべてのホスティングリソースで統一されたプレイヤーエクスペリエンスを提供できます。
- すべてのホスティングリソースで同じ FlexMatch マッチメーカーを使用してマッチを配置できます。
- ハイブリッドホスティングリソースを世界中にデプロイしながら、そのリソースを一元管理できます。

- プレイヤーの需要が変動すると、マネージドリソースとセルフマネージドリソース間でゲームセッションの負荷をシームレスに管理できます。
- Amazon GameLift Servers エージェントを使用すると、同じツールを使用して、あらゆる種類のホスティングリソース上にあるゲームサーバーのライフサイクルを管理できます。
- すべてのホスティングリソースでゲームとプレイヤーのメトリクスとログを収集できます。Amazon GameLift Servers 機能やその他の AWS サービスを活用してデータを組み合わせ、まとまりのあるオブザーバビリティソリューションを開発します。

ゲームセッションの配置を設定する

ゲームセッション配置は、新しいゲームセッションをホストするために利用可能なゲームサーバーを見つけるプロセスです。Amazon GameLift Servers はゲームセッションキューを使用して、プレイヤーのレイテンシー、フリート容量、コスト最適化などの要因を考慮して、フリート全体にゲームセッションをインテリジェントに配置します。

適切に設定されたゲームセッションキューにより、プレイヤーは利用可能な最良のゲームサーバーに接続され、ホスティングリソースを効率的に活用しながら最適なパフォーマンスを提供します。キューは複数のフリートと AWS リージョンにまたがってグローバルカバレッジと冗長性を提供できます。

ゲームセッションキューは、Amazon GameLift Servers が利用可能なゲームサーバーを検索し、新しいゲームセッションのホスト用にそれらを選択するために使用する主要なメカニズムです。キューは、多数のゲームセッションリクエストを処理し、複数のホスティングリソースのフリート間でそれらの配置を見つけるために、はるかに効率的な方法を提供します。ホスティングソリューションが複数のフリートを使用していて、大量のリクエストを処理している場合は、キューが必要になる可能性があります。

ゲームがプレイヤーの新しいゲームセッションを開始する場合、Amazon GameLift Servers サービスにプレイスメントリクエストを送信し、キューにファネルします。キューの設定によって、リクエストがいつどのように処理されるかが決まります。プレイスメントリクエストを処理すると、Amazon GameLift Servers はゲームセッションをホストするゲームサーバーのフリートのセットを検索します。プレイスメントは、Amazon GameLift Servers が利用可能なゲームサーバーを見つけ、ゲームセッションを開始するように促すと成功します。

トピック

- [キューの特性](#)
- [Amazon GameLift Servers のゲームセッションキューに関するベストプラクティス](#)

- [ゲームセッションキューを作成する](#)
- [ゲームセッション配置のイベント通知を設定](#)

キューの特性

Amazon GameLift Servers ゲームセッションキューは AWS クラウドリソースです。キューは、Amazon GameLift Servers がサポートする任意の AWS リージョン で作成できます (「[Amazon GameLift Servers サービスロケーション](#)」を参照)。ゲームセッション配置リクエストはその場所に送信され、そこで処理されます。

キューを使用したゲームセッション配置の自動化は、ゲーム開発者やプレイヤーにとって大きなメリットとなります。具体的には次のとおりです。

- キューは「可能な限り最適な」配置を指定します。ゲームセッション配置リクエストを処理する際、キューは、Amazon GameLift Servers FleetIQ アルゴリズムを使用して、コスト、ロケーション、プレイヤーのレイテンシーなどの定義済みの優先設定一式に基づき、キューの配置を優先付けます。
- キューはスポットフリートをサポートしているため、ゲームホスティングコストを削減できます。AWS スポットフリートを使用してキューを設定できます。スポットフリートを使用すると、多くの場合、ホスティングコストとオンデマンドフリートを大幅に削減できます。低コストは配置の主要な基準の 1 つであるため、キューは常にコストの違いを活用できます。
- キューは需要の多いときに新規のゲームをより速く配置できます。複数のフリートでキューを設定することで、ゲームセッション配置のより柔軟なオプションを提供できます。ただし、追加のフリートは、需要の増加に応じて必要に応じてバックアップ容量も提供します。配置リクエストでは、Amazon GameLift Servers がゲームセッションを最適な場所に配置できない場合、自動的に他の場所の評価に移行します。
- キューはゲームサーバーの稼働率をより回復性のあるものにできます。停電は起こり得ます。マルチフリート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。異なる AWS リージョン およびアベイラビリティゾーンに容量があるフリートでキューを設定することで、プレイヤーが参加するゲームセッションを常に見つけられるようにできます。
- 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を取得します。Amazon GameLift Servers はキューのメトリクスを提供します。これには、配置の成功と失敗、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 Servers 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。

基本的なスターターキューを作成して開始するには、「[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。

トピック

- [Amazon GameLift Servers のゲームセッションキューに関するベストプラクティス](#)
- [ゲームセッションキューを作成する](#)
- [ゲームセッション配置のイベント通知を設定](#)

Amazon GameLift Servers のゲームセッションキューに関するベストプラクティス

ゲームセッションキューには、Amazon GameLift Servers が新しいゲームセッションを配置できるフリートの一覧が含まれます。各フリートは、複数の地理的なロケーションにデプロイされるホスティングリソースを持つことができます。配置を選択すると、キューはフリートに設定した一連の優先順位に基づいてフリートとフリートの場所を選択します。

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容量を増やすことができます。
- ゲームセッションの配置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イ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。

ゲームセッションキューを作成する

キューは、複数のフリートとロケーションに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。ゲームは、キューにプレイスメントリクエストを送信することで新しいゲームセッションを開始します。キューには、リクエストを処理する方法の手順が設定されています。ゲームセッションの配置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](#) を参照してください。

ゲームセッションキューを作成する方法

これらの手順は、最小限の構成設定とデフォルト設定でシンプルな作業キューを作成する方法を示しています。キュー設定をカスタマイズするには、いくつかのオプションがあります。これらのオプションは、ゲームのニーズに基づいて可能な限り最適な配置を行うのに役立ちます。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をカスタマイズする](#)」を参照してください。ほとんどのキュー設定はいつでも更新できます。

Amazon GameLift Servers コンソールまたは AWS CLI を使用してゲームセッションキューを作成できます。

Console

[Amazon GameLift Servers コンソール](#)で、作業する AWS リージョンを選択します。コンソールの左側のナビゲーションバーを開き、[キュー] を選択します。

1. [キュー] ページで、[キューの作成] を選択し、ワークフローを開始します。
2. [キュー設定] で、次の設定を入力します。
 - a. キューの名前を入力してください。この名前は、キューを作成する AWS リージョンに一意である必要があります。
 - b. デフォルトのタイムアウト設定は 600 秒 (または 10 分) のままにします。この値は、Amazon GameLift Servers が停止する前に新しいゲームセッションを配置しようとする時間を制御します。Amazon GameLift Servers は、リクエストがタイムアウトする

まで利用可能なリソースを検索します。キューのタイムアウト設定はいつでも更新できます。

- c. 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。キューは、プレイヤーレイテンシーデータを含むプレースメントリクエストを受け取った場合にのみレイテンシーポリシーを使用します。レイテンシーポリシーはいつでもキューに追加できます。レイテンシーポリシーの作成の詳細については、「[プレイヤーレイテンシーポリシーを作成する](#)」を参照してください。
3. [ゲームセッションの配置場所] セクションをスキップして、[すべての場所] のデフォルト設定を使用します。この設定では、キューが配置できる場所 (フィルター設定とも呼ばれます) の許可リストを作成できます。ロケーションとフィルター設定による優先順位付けの詳細については、「[場所による配置の優先順位付け](#)」を参照してください。
4. [送信先の順序] で、キューに 1 つ以上のフリートを追加します。フリートを識別するには、フリート ID または ARN、またはフリートエイリアスを使用します。複数のフリートを追加する場合、それらはすべて同様のゲームビルドを実行し、このキューを使用するゲームクライアントと互換性がある必要があることに注意してください。さらに、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。
 - a. フリートまたはエイリアスが作成されたリージョンを選択します。マルチロケーションフリートの場合、これは「ホーム」リージョンです。
 - b. 送信先タイプで、フリートまたはエイリアスを選択します。
 - c. リージョンとタイプの選択により、既存のフリートまたはエイリアスのドロップダウンリストが入力されます。キューの送信先として指定するものを選択します。
 - d. キューに別のフリートまたはエイリアスを指定するには、[送信先の追加] を選択し、前のステップを繰り返します。
 - e. 送信先のリストを追加したら、ドラッグアンドドロップ機能を使用して送信先の順序を変更します。Amazon GameLift Servers は、送信先ごとに配置を優先するときこの順序を使用します。
5. [ゲームセッション配置優先順位] セクションをスキップして、デフォルトの優先順位を維持します。この設定により、Amazon GameLift Servers が新しいゲームセッション配置で利用可能なホスティングリソースを検索する場所を選択する方法をカスタマイズできます。配置の優先順位付けの詳細については、「[ゲームセッション配置に優先順位を付ける](#)」を参照してください。キューの配置の優先順位はいつでも更新できます。
6. [ロケーションの順序] で、デフォルト値のままにします。この設定は、フリートのロケーション別に優先順位を付ける場合に使用されます。使用するロケーションの順序を提供しま

す。デフォルトの優先度設定を使用する場合、優先する送信先が複数のロケーションを持つフリートである場合、ロケーションはタイブレーカーとして使用されます。

7. オプションの [イベント通知設定] セクションをスキップします。大量のプレイメントリクエストを処理するキューには、イベント通知が必要です。開発やテストなどの目的で少量を処理するキューの場合、[DescribeGameSessionPlacement](#) でポーリングすることでプレイメントリクエストのステータスを追跡できます。詳細については、[ゲームセッション配置のイベント通知を設定](#)を参照してください。キューのイベント通知設定はいつでも更新できます。
8. [作成] を選択して、最小限のカスタマイズで新しいキューを生成します。

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。
- フィルタリングして、次の場所への配置のみを許可します: us-east-1、us-east-2、us-west-2、ca-central-1。
- 指定された順序でのコストそして次にロケーションに基づく優先順位。

```
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="COST","LOCATION",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 Servers 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 Servers 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ト通知を設定

イベント通知を使用して、個々のプレイ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 GameLift Servers から Amazon Simple Notification Service (Amazon SNS) トピックにイベント通知を発行します。
- 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式を使用します。

Amazon GameLift Servers から生成されるゲームセッションの配置イベントのリストについては、「[ゲームセッション配置イベント](#)」を参照してください。

Important

ハイボリュームプレイメントシステムでは、FIFO トピックではなく、標準 (非FIFO) Amazon SNS トピックを使用することをお勧めします。FIFO トピックは標準トピックよりも発行制限が低く、高負荷時にスロットリング例外が発生する可能性があります。FIFO トピックでスロットリングが発生すると、キュー配置通知が失われる可能性があります。

SNS トピックの設定

Amazon GameLift Servers 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 Servers イベント通知用の SNS トピックをセットアップするには

1. にサインイン AWS マネジメントコンソール し、<https://console.aws.amazon.com/sns/v3/home> で Amazon SNS コンソールを開きます。
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。

JSON

```
{
  "Version": "2012-10-17",
  "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:us-east-1:111122223333: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:us-east-1:111122223333:your_topic_name",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:gamelift:us-east-1:111122223333: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. [トピックを作成] 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 Notification Service デベロッパーガイド」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 デベロッパーガイド」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 Servers は、すべてのゲームセッションの配置イベントを 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。たとえば、ゲームセッションへの接続の前にタスクを処理する PlacementFulfilled AWS Lambda 関数にイベントを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 Servers キューで使用する EventBridge ルールの例を以下に示します。

すべての Amazon GameLift Servers キューのイベントに一致します

```

{
  "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"
  ]
}

```

```
}
```

を使用したリファレンス実装 AWS CDK

Amazon SNS、および Amazon DynamoDB を使用したイベントベースのゲームセッション配置の完全なリファレンス実装については AWS Lambda、Amazon GameLift Toolkit の「[イベントベースのゲームセッション配置ガイド](#)」を参照してください。このガイドには、ゲームセッション配置イベントを処理する AWS Lambda 関数、ゲームセッション配置状態を追跡するための Amazon DynamoDB テーブル、キュー Amazon GameLift Servers 設定のベストプラクティスを使用して、完全なイベント駆動型ゲームセッション配置システムをデプロイするための AWS CDK テンプレートが含まれています。

ゲームホスティングソリューションをカスタマイズする

基本的なゲームホスティングソリューションを導入したら、次のトピックを使用してカスタマイズおよび強化し、プレイヤーエクスペリエンスの向上、コストの最適化、高度な機能の追加を行います。このセクションでは、主に影響を受けるコンポーネント別に整理されたさまざまなカスタマイズオプションについて説明します。ゲームの要件とプレイヤーベースに最適なカスタマイズを選択します。

トピック

- [ゲームサーバー構築のカスタマイズ](#)
 - [Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)
 - [ゲームサーバーが Amazon GameLift Servers フリートデータにアクセスできるようにする](#)
 - [Amazon GameLift Servers に対する VPC ピアリングのセットアップ](#)
- [プレイヤーセッションとマッチメイキングのカスタマイズ](#)
 - [プレイヤー ID を生成する](#)
 - [FlexMatch マッチメイキングを Amazon GameLift Servers に追加する](#)
- [ゲームセッション配置のカスタマイズ](#)
 - [ゲームセッションキューをカスタマイズする](#)
 - [ゲームセッション配置に優先順位を付ける](#)
 - [スポットインスタンスのキューの設計](#)
- [ホスティングリソースのカスタマイズ](#)
 - [マネージドフリートのコンピューティングリソースを選択する](#)
 - [Amazon GameLift Servers コンテナフリートをカスタマイズする](#)

- [スポットフリートによるゲームホスティングコストの削減](#)
- [マネージド Amazon GameLift Servers でのゲームサーバーランタイム設定の最適化](#)
- [Amazon GameLift Servers エージェントの操作](#)
- [エイリアスを使用して Amazon GameLift Servers フリートの指定を抽象化する](#)

ゲームサーバー構築のカスタマイズ

ゲームサーバービルドのカスタマイズは、他の AWS サービスを活用するなど、ゲームサーバーの機能を強化する機会を提供します。これらのカスタマイズにより、ゲームサーバー機能が基本的なホスティングを超えて拡張され、高度な機能と統合がサポートされます。

Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する

Amazon GameLift Servers 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 Servers はゲームホスティングフリートを管理するため、Amazon GameLift Servers 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 フリートおよび Anywhere フリートからメトリックス、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 Servers 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認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、ロールのアクセス許可を使用する Amazon GameLift Servers 権限を付与する信頼ポリシーを持つ IAM ロールを作成します。

IAM ロールのセットアップ方法については「[Amazon GameLift Servers 用に IAM サービスロールをセットアップする](#)」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

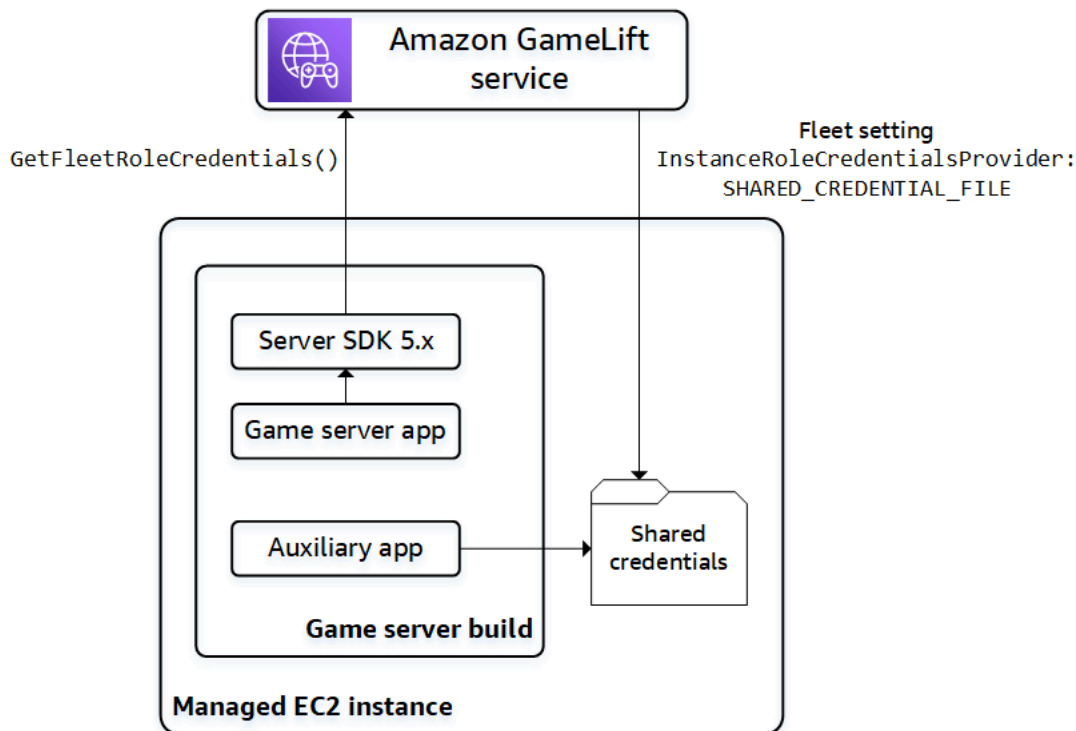
このステップでは、IAM ロールのセキュリティ認証情報を取得し、AWS リソース を操作するときにそれらを使用するようにアプリケーションを設定します。以下の表を参照して、(1) アプリケーションのタイプ、(2) ゲームが Amazon GameLift Servers との通信に使用するサーバー SDK バージョンに基づいてアプリケーションを変更する方法を決定してください。

	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 <code>GetFleetRoleCredentials()</code> 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 AssumeRole を使用して AWS Security Token Service (AWS STS) を呼び出します。	ロール ARN AssumeRole を使用して AWS Security Token Service (AWS STS) を呼び出します。

Note

コンテナフリートの場合、`FleetRoleArn` 認証情報は各コンテナに挿入されます。アプリケーションは、デフォルトの AWS 認証情報プロバイダーを使用してこれらの認証情報にアクセスできます。を呼び出しても `GetFleetRoleCredentials()`、同じ認証情報が返されます。これらのフリートロールの認証情報は、コンテナ内でのみアクセスできます。

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 Servers 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) (Go) を呼び出し、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 Servers は各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 Servers 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:

```
[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 Servers 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 Servers 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

アプリケーションにコードを追加して IAM ロールを引き受け、AWS リソースを操作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 Servers 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 ロールを引き受けすることができます。

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、アプリケーションは 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、アプリケーションが AWS リソースにアクセスすることを許可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 リソースで一時的な認証情報の使用」](#)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ときは、次のパラメータを設定します。

コンテナフリートの場合:

- [FleetRoleArn](#) – このパラメータを IAM ロールの ARN に設定します。

他のフリートタイプの場合:

- [InstanceRoleArn](#) – このパラメータを IAM ロールの ARN に設定します。
- [InstanceRoleCredentialsProvider](#) – Amazon GameLift Servers 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) ピアリングを使用して、Amazon GameLift Servers 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 リソース間で通信できます。VPC は、を通じて管理される一連のリソースを含む、ユーザーが定義する仮想プライベートネットワークです。AWS アカウント。Amazon GameLift Servers 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、フリートと他の AWS リソースの VPC 間の直接ネットワーク接続を確立できます。

Amazon GameLift Servers 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 Servers に対する VPC ピアリングのセットアップ](#)」を参照してください。

 Note

VPC ピアリングはコンテナフリートではサポートされていません。

ゲームサーバーが Amazon GameLift Servers フリートデータにアクセスできるようにする

カスタムゲームビルドまたは Amazon GameLift Servers Realtime スクリプトが、Amazon GameLift Servers 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。)
- マッチメイキング、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、フリー[Amazon GameLift Servers](#) [トリソースの属性](#)が含まれます。

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",
  "name": "ReallyFunGameTestFleet08",
  "fleetType": "ON_DEMAND",
  "instanceRoleArn": "arn:aws:iam::123456789012:role/S3AccessForGameLift",
  "instanceType": "c5.large",
  "serverLaunchPath": "/local/game/reallyfungame.exe"
}
```

Amazon GameLift Servers に対する VPC ピアリングのセットアップ

このトピックでは、Amazon GameLift Servers ホストゲームサーバーと他の Amazon GameLift Servers 以外のリソースとの VPC ピアリング接続を設定する方法に関するガイダンスを提供します。Amazon Virtual Private Cloud (VPC) ピアリング接続を使用して、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 リソースと直接およびプライベートに通信できるようにします。で実行 AWS され、アクセスできる AWS アカウントによって管理されている任意のリソースと VPC ピア接続を確立できます。

Note

VPC ピアリング接続は高度な機能です。ゲームサーバーが他の AWS リソースと直接およびプライベートに通信できるようにする推奨オプションについては、「」を参照してください。[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する。](#)

Amazon VPC と VPC ピアリングにすでに精通している場合は、Amazon GameLift Servers 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 Servers 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、最初に Amazon GameLift Servers サービスへのピア接続リクエストを受け入れるために、Amazon GameLift Servers 以外のリソースで VPC を事前承認します。次に、Amazon GameLift Servers をトリガーして、先ほど認可した VPC ピアリング接続をリクエストします。Amazon GameLift Servers 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 ピアリング接続を設定するには

1. AWS アカウント ID (複数可) と認証情報を取得します。

次の AWS アカウントの ID とサインイン認証情報が必要です。AWS にサインイン[AWS マネジメントコンソール](#)してアカウント設定を表示することで、アカウント IDs を見つけることができます。認証情報を取得するには、IAM コンソールに移動します。

- AWS Amazon GameLift Servers ゲームサーバーの管理に使用する アカウント。
- AWS 以外の Amazon GameLift Servers リソースの管理に使用する アカウント。

同じアカウントを Amazon GameLift Servers リソースと Amazon GameLift Servers 以外のリソースに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 Servers ゲームサーバー用の VPC - ユーザーの Amazon GameLift Servers 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 Servers にデプロイされます。フリートは、Amazon GameLift Servers サービスによって管

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 Servers AWS リソース以外の VPC – で実行 AWS され、アクセスできる AWS アカウントによって管理されている任意のリソースと VPC ピア接続を確立できます。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 マネジメントコンソール](#) 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 Servers 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 ピアリング接続を認可します。

このステップでは、Amazon GameLift Servers 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、Amazon GameLift Servers 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 ピア接続を承認するには、サービス API [CreateVpcPeeringAuthorization\(\)](#) を呼び出すか AWS、CLI コマンド を使用します `create-vpc-peering-authorization`。Amazon GameLift Servers 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 Servers 以外のリソース用の VPC の ID。
- Amazon GameLift Servers AWS アカウント ID – Amazon GameLift Servers フリーットの管理に使用するアカウントです。

VPC ピアリング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 ピアリング接続承認を管理できます。

- [DescribeVpcPeeringAuthorizations\(\)](#) (AWS CLI `describe-vpc-peering-authorizations`)。
- [DeleteVpcPeeringAuthorization\(\)](#) (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 Servers にピア接続の確立をリクエストできます。

VPC ピア接続をリクエストするには、サービス API [CreateVpcPeeringConnection\(\)](#) AWS を呼び出するか、CLI コマンド `create-vpc-peering-connection` を使用します。Amazon GameLift Servers 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 Servers 以外のリソースの VPC と、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 - Amazon GameLift Servers ゲームサーバー用の 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 ピアリング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 ピアリング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 ピアリング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。

- [DescribeVpcPeeringConnections\(\)](#) (AWS CLI `describe-vpc-peering-connections`)。
- [DeleteVpcPeeringConnection\(\)](#) (AWS CLI `delete-vpc-peering-connection`)。

新しいフリートとの VPC ピアリング接続を設定するには

新しい Amazon GameLift Servers 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 (複数可) と認証情報を取得します。

次の 2 つの AWS アカウントの ID とサインイン認証情報が必要です。AWS にサインイン [AWS マネジメントコンソール](#) し、アカウント設定を表示することで、アカウント IDs を見つけることができます。認証情報を取得するには、IAM コンソールに移動します。

- AWS Amazon GameLift Servers ゲームサーバーの管理に使用する アカウント。
- AWS 以外の Amazon GameLift Servers リソースの管理に使用する アカウント。

同じアカウントを Amazon GameLift Servers リソースと Amazon GameLift Servers 以外のリソースに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 Servers AWS リソース以外の 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。以外の Amazon GameLift Servers リソースが 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/ユーザーグループで管理されている場合は Amazon GameLift Servers、次のステップで認可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 Servers 以外のリソースとの VPC ピアリング接続を承認します。

Amazon GameLift Servers 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 Servers 以外のリソース用の 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。

以外の Amazon GameLift Servers リソースを管理するアカウント認証情報を使用して、サービス API [CreateVpcPeeringAuthorization\(\)](#) AWS を呼び出すか、CLI コマンド を使用します `create-vpc-peering-authorization`。以下の情報を識別します。

- ピア VPC ID - Amazon GameLift Servers 以外のリソース用の VPC の ID。
- Amazon GameLift Servers AWS アカウント ID – Amazon GameLift Servers フリートの管理に使用するアカウントの ID。

VPC ピアリング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 ピアリング接続承認を管理できます。

- [DescribeVpcPeeringAuthorizations\(\)](#) (AWS CLI `describe-vpc-peering-authorizations`)。
 - [DeleteVpcPeeringAuthorization\(\)](#) (AWS CLI `delete-vpc-peering-authorization`)。
4. [AWS 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 Servers 以外のリソース用の VPC を管理するために使用するアカウントの ID。
 - `peer-vpc-id` - Amazon GameLift Servers 以外のアカウント用の VPC の ID。

VPC ピアリング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 ピアリング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 ピアリング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 ピアリング接続は確立されません。同様に、VPC ピアリング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 ピアリング接続は、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。事前に確立された VPC は、非 Amazon GameLift Servers 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 ピアリング接続に関する問題のトラブルシューティング

Amazon GameLift Servers 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 Servers 以外の VPC の VPC 認可のステータスを確認します。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 Servers 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 Servers 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 Servers 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 Servers 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 ピアリング接続のリクエスト時)。新しいフリートが [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。

プレイヤーセッションとマッチメイキングのカスタマイズ

プレイヤーセッションとマッチメイキングのカスタマイズにより、バランスの取れた魅力的なマルチプレイヤーエクスペリエンスを提供するのに役立つ微妙なマッチメイキングシステムなど、高度なプレイヤー管理ワークフローを開発できます。

プレイヤー ID を生成する

Amazon GameLift Servers はプレイヤーセッションを使用して、ゲームセッションに接続されたプレイヤーを表します。Amazon GameLift Servers は、プレイヤーが Amazon GameLift Servers と統合されたゲームクライアントを使用してゲームセッションに接続するたびにプレイヤーセッションを作成します。プレイヤーがゲームを終了すると、プレイヤーセッションは終了します。Amazon GameLift Servers はプレイヤーセッションを再利用しません。

Important

FlexMatch マッチメイキングを使用する場合、既存のアクティブなマッチメイキングリクエストにすでに含まれているプレイヤー ID を含む新しいマッチメイキングリクエストを作成した場合、既存のリクエストは自動的にキャンセルされます。ただし、キャンセルされたり

クエストには MatchmakingCancelled イベントは送信されません。既存のマッチメイキングリクエストのステータスをモニタリングするには、[DescribeMatchmaking](#) を使用して、頻度の低い間隔 (30 ~ 60 秒) でリクエストステータスをポーリングします。キャンセルされたリクエストには、CANCELLED のステータスと理由「Cancelled due to duplicate player」が表示されます。

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、「[Amazon GameLift Servers コンソールでのゲームセッションとプレイヤーセッション](#)」を参照してください。

FlexMatch マッチメイキングを Amazon GameLift Servers に追加する

Amazon GameLift Servers FlexMatch を使用して、Amazon GameLift Servers でホストされているゲームにプレイヤーマッチメイキング機能を追加します。FlexMatch は、カスタムゲームサーバーまたは Amazon GameLift Servers Realtime のいずれでも使用できます。

FlexMatch では、マッチメイ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。ゲームに適したプレイヤー属性とゲームモードに基づいてプレイヤーと一緒にマッチングする方法を設計します。FlexMatch は、ゲームを探しているプレイヤーを評価し、1 つ以上のチームとマッチングを行い、マッチングをホストするゲームセッションを開始するための作業を管理します。

フル FlexMatch サービスを使用するには、ホスティングリソースをキューで設定する必要があります。Amazon GameLift Servers 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見つけます。特に、Amazon GameLift Servers キューはゲームクライアントから提供されるレイテンシーデータを利用して、プレイヤーがプレイ時に可能な限り低いレイテンシーで済むように、ゲームセッションの配置場所を決定します。

FlexMatch に関する詳細、特にマッチメイキングをゲームに統合するための詳細なヘルプについては、「[Amazon GameLift Servers FlexMatch デベロッパーガイド](#)」の各トピックを参照してください。

- [Amazon GameLift Servers FlexMatch の仕組み](#)
- [FlexMatch 統合ステップ](#)

ゲームセッション配置のカスタマイズ

ゲームセッション配置のカスタマイズにより、配置システムをファインチューニングして、プレイヤーベースに可能な限り最高のゲームプレイエクスペリエンスを提供することができます。プレイヤーの互換性と設定、コスト効率、地理的分散、サービスの中断に対する耐障害性などの運用上の考慮事項に合わせて配置を最適化できます。

ゲームセッションキューをカスタマイズする

このトピックでは、ゲームセッションキューをカスタマイズして、ゲームセッションの配置に関する最善の決定を行う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、「[ゲームセッションの配置を設定する](#)」を参照してください。

以下の Amazon GameLift Servers 機能にはキューが必要です:

- [FlexMatch でのマッチメイキング](#)
- [スポットインスタンスのキューの設計](#)

トピック

- [キューのスコープを定義する](#)
- [マルチロケーションキューを構築する](#)
- [キューメトリクスの評価](#)

キューのスコープを定義する

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。

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 Servers 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。ゲームセッションの配置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 Servers メトリクス](#)」を参照してください。

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、プレー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、「[スポットフリートによるゲームホスティングコストの削減](#)」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、配置場所 AWS リージョン として us-east-1、us-east-2ca-central-1、および us-west-2 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 Servers 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 Servers 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				

Game session placement priority

The values that GameLift uses to prioritize game session placement. The default order is latency, cost, destination, and location.



Latency

Prioritize locations with the lowest average player latency.



Cost

Prioritize destinations with the lowest current hosting cost.



Destination

Prioritize based on the defined destination order.



Location

Prioritize based on the defined location order.

▼ Location order

An ordered list of locations that the queue can use for game session placement.

Location

ca-central-1



Remove

us-east-1



Remove

us-east-2



Remove

us-west-2



Remove

Add locations

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。キューに関するメトリクスは、[Amazon GameLift Servers 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 Servers 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、プレイメントが失敗するタイミングと理由を特定するのに役立ちます。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容量をいつ調整する必要があるかを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配置リクエストの場合、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の配置を見つける頻度を示します。プレイメントでは、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 Servers が理想的なプレイ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 Servers を監視する](#)」を参照してください。
- [ロケーション固有のプレイメント] – 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容量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。

ゲームセッション配置に優先順位を付ける

Amazon GameLift Servers はアルゴリズムを使用して、キューの送信先に優先順位を付ける方法と、新しいゲームセッションを配置する場所を決定します。アルゴリズムは、順序付けられた一連の基準

に基づいています。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。キューの優先順位はいつでも編集できます。

デフォルトの優先順序

1. レイテンシー – ゲームセッション配置リクエストにプレイヤーのロケーション固有のレイテンシーデータが含まれている場合、Amazon GameLift Servers は各ロケーションの平均プレイヤーレイテンシーを計算し、平均が最も低いフリートロケーションにゲームセッションを配置しようとします。
2. コスト – リクエストにレイテンシーデータが含まれていない場合、または複数のフリートのレイテンシーが等しい場合、Amazon GameLift Servers は各フリートのホスティングコストを評価します。フリートのホスティングコストは、フリートタイプ (スポットまたはオンデマンド)、インスタンスタイプ、ロケーションによって異なります。
3. 送信先 - 複数のフリートのレイテンシーとコストが等しい場合、Amazon GameLift Servers は、キュー設定にリストされている送信順序に基づいてフリートに優先順位を付けます。
4. 場所 – マルチロケーションフリートを持つキューの場合、他のすべての条件が等しい場合、Amazon GameLift Servers はアルファベット順に基づいてフリートの場所の優先順位を付けます。

キューがゲームセッション配置の優先順位を付ける方法をカスタマイズする

キューが配置条件の優先順位を付ける方法をカスタマイズできます。キューは、受信するすべてのゲームセッション配置リクエストにカスタム優先順位を適用します。

Note

カスタム優先度設定を作成し、4 つの条件をすべて含めない場合、Amazon GameLift Servers は欠落している条件をデフォルトの順序で自動的に追加します。

キューの優先度設定をカスタマイズする方法

[Amazon GameLift Servers コンソール](#)または AWS Command Line Interface (AWS CLI) を使用して、カスタム優先度設定を作成します。

Console

[Amazon GameLift Servers コンソール](#)では、新しいキューの作成時または既存のキューの更新時に、キューの優先順位をカスタマイズできます。作業する AWS リージョンを選択します。

コンソールの左側のナビゲーションバーを開き、[キュー] を選択します。[キュー] ページで、既存のキューを選択し、[編集] を選択します。

1. [ゲームセッション配置の優先度] セクションに移動します。各優先条件をドラッグアンドドロップして、必要な順序を作成します。
2. [ロケーションの順序] セクションに移動します。優先する場所を追加します。このリストは、キューに複数のロケーションを持つフリートがある場合に便利です。少なくとも 1 つの場所を指定する必要があります。ここで指定した場所が最初に優先され、次にキューの送信先の他のすべての場所が優先されます。
3. [Save changes] (変更の保存) をクリックします。

AWS CLI

コマンド [update-game-session-queue](#) と `--priority-configuration` オプションを使用して、キューの優先順位をカスタマイズします。は現在のデフォルト AWS リージョンのキュー Amazon GameLift Servers を更新するか、`--region` タグを追加して別の AWS リージョンを指定できます。

次のリクエスト例では、指定されたキューの優先度設定を追加または更新します。

```
aws gamelift update-game-session-queue \
  --name "example-queue-with-priority"
  --priority-configuration
  PriorityOrder="COST", "LOCATION", "DESTINATION", LocationOrder="us-east-1", "us-
  east-2", "ca-central-1", "us-west-2" \
```

プレイヤーのレイテンシー別に配置を優先する

プレイヤーに可能な限り最高のプレイヤーエクスペリエンスを提供し、レイテンシーを最小限に抑えるには、ゲームセッション配置システムを設定するときに次の手順を実行します。

- ゲームセッションを配置する場所を選択するときにレイテンシーを優先するようにキューを設定します。デフォルトでは、レイテンシーは優先度リストの最上位にあります。キューの優先度設定をカスタマイズし、レイテンシーを優先度順に配置する場所を選択することもできます。
- キューのプレイヤーレイテンシーポリシーを設定します。レイテンシーポリシーを使用すると、ゲームセッション配置で許容するレイテンシーの量にハードリミットを設定できます。Amazon GameLift Servers が制限を超えることなくゲームセッションを配置できない場合、プレイスメントリクエストはタイムアウトして失敗します。単一のレイテンシーポリシーを設定することも、時

間の経過とともにレイテンシー制限を徐々に緩和する一連のポリシーを作成することもできます。一連のポリシーを使用すると、初期レイテンシーの制限を非常に低く指定でき、短い遅延後にレイテンシーが高いプレイヤーに対応できます。レイテンシーポリシーの作成の詳細については、「[プレイヤーレイテンシーポリシーを作成する](#)」を参照してください。

- ゲームセッションの配置リクエストを行うときは ([「StartGameSessionPlacement」](#)を参照)、各プレイヤーのレイテンシーデータを含めます。プレイヤーレイテンシーデータには、ゲームセッションを配置できるすべての場所の値が含まれます。たとえば、AWS リージョン us-east-2 と ca-central-1 にゲームセッションを配置するキューの場合、レイテンシーデータは次のようになります。

```
"PlayerLatencies": [  
  { "LatencyInMilliseconds": 100, "PlayerId": "player1", "RegionIdentifier": "us-east-2" },  
  { "LatencyInMilliseconds": 100, "PlayerId": "player1", "RegionIdentifier": "ca-central-1" },  
  { "LatencyInMilliseconds": 150, "PlayerId": "player2", "RegionIdentifier": "us-east-2" },  
  { "LatencyInMilliseconds": 150, "PlayerId": "player2", "RegionIdentifier": "ca-central-1" }  
]
```

正確なレイテンシー測定値を取得するには、Amazon GameLift Servers の UDP ping ビーコンを使用します。これらのエンドポイントを使用すると、プレイヤーデバイスと潜在的な各ホスティングロケーション間の実際の UDP ネットワークレイテンシーを測定できるため、ICMP ping を使用するよりも正確な配置決定を行うことができます。UDP ping ビーコンを使用してレイテンシーを測定する方法の詳細については、「[UDP ping ビーコン](#)」を参照してください。

場所による配置の優先順位付け

優先順位付けされた地理的位置のリストに基づいてゲームセッションを配置するようにキューを設定できます。場所は、キューが新しいゲームセッションを配置する場所を選択する方法を決定する基準の 1 つです。デフォルトでは、場所はレイテンシー、コスト、送信先の後、4 番目に優先されます。

ゲームセッションの配置では、送信先と場所の意味が多少異なります。

- 送信先とは、特定のフリートを指し、デプロイされているすべてのフリートのホスティングリソースが含まれます。送信先で優先順位を付けると、Amazon GameLift Servers はフリート内の任意の場所に配置できます。マルチロケーションマネージドフリートと Anywhere フリートは、1 つ以上の場所にデプロイされるホスティングリソースを持つことができます。

- ロケーションとは、フリートのホスティングリソースがデプロイされる特定の地理的位置を指します。フリートには、Local Zones AWS リージョンやカスタムロケーション (Anywhere フリートの場合) など、複数のロケーションを含めることができます。単一ロケーションのマネージドフリートには 1 つのロケーションがあり、常に AWS リージョンです。マルチロケーションマネージドフリートは、ホームリージョンに加え、リモートロケーションを持つことができます。Anywhere フリートには 1 つ以上のカスタムロケーションがあります。

ロケーションごとにプレイメントを優先順位付けする場合、Amazon GameLift Servers は優先ロケーションを含むキューの送信先を検索し、使用可能なホスティングリソースを検索します。優先ロケーションを持つ送信先が複数ある場合、Amazon GameLift Servers は次の優先度基準 (コスト、レイテンシー、送信先) に進みます。

キューのロケーションの優先順位付け方法に影響を与えるには、いくつかの方法があります。

- キューがすべてのゲームセッションの配置リクエストを処理する方法を設定します。
 - キューに優先度設定を追加します。キューの優先度設定には、順序付けられたロケーションのリストが含まれます。優先順位を付けるロケーションを 1 つ以上指定できます。このリストはロケーションを除外するものではなく、利用可能なホスティングリソースを最初に探す場所を Amazon GameLift Servers に指定するだけです。順序付けられたロケーションリストの一般的な用途は、ほとんどのトラフィックを 1 つ以上の特定の地理的場所にファネルし、バックアップ容量として追加の場所を使用する場合です。[UpdateGameSessionQueue](#) を呼び出して優先度設定を追加します。
 - フィルター設定をキューに追加します。フィルター設定は、キューの許可リストです。使用可能なホスティングリソースを検索するときに、リストにない場所を無視するように Amazon GameLift Servers に指示します。フィルター設定には 2 つの一般的な用途があります。まず、複数のロケーションを持つフリートの場合、フィルターを使用してフリートのロケーションの一部を除外できます。次に、特定のロケーションでの配置を一時的に禁止する場合があります。たとえば、ロケーションで一時的な問題が発生している可能性があります。キューのフィルター設定はいつでも更新できるため、必要に応じてロケーションを簡単に追加および削除できます。[UpdateGameSessionQueue](#) を呼び出してフィルター設定を追加します。
- 個々のプレイメントリクエストには特別な手順を使用します。
 - ゲームセッションの配置リクエストに優先度のオーバーライドリストを含めます。[StartGameSessionPlacement](#) リクエストを使用して、ロケーションの代替優先度リストを指定できます。このリストは、その 1 つのリクエストのみのロケーションに対してキューが設

定した優先順位を効果的に置き換えます。他のリクエストには影響しません。このオーバーライド機能にはいくつかの要件があります。

- オーバーライドリストは、優先度設定が LOCATION であるキューを最優先としてのみ使用します。
- 同じプレイメントリクエストにプレイヤーレイテンシーデータを含めないでください。Amazon GameLift Servers が解決できないロケーションを優先する場合、レイテンシーデータを含めると競合が発生します。
- 優先度のオーバーライドリストで使用可能なリソースが見つからない場合の Amazon GameLift Servers の処理方法を決定します。キューの他のロケーションにフォールバックするか、オーバーライドリストへの配置を制限するかを選択します。デフォルトでは、Amazon GameLift Servers はキューの他のロケーションへの配置を試みるためにフォールバックします。
- オーバーライドリストにロケーションを追加するなど、必要に応じてキューのフィルター設定を更新します。オーバーライドリストはフィルターリストを無効にしません。

プレイヤーレイテンシーポリシーを作成する

プレイ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、Amazon GameLift Servers は、すべてのプレイヤーに対して、平均レイテンシーが最小の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 Servers 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 Servers 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 Servers 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 Servers 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 Servers 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。

たとえば、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。

Create queue

Queue settings

Name

The name must be unique and have 1-128 characters. Valid characters: A-Z, a-z, 0-9, and - (hyphen).

Timeout

Specify how long GameLift tries to place a game session before stopping.

 seconds

Must be 10-600 seconds.

 We recommend setting player latency policies, unless you're using GameLift FlexMatch.



Player latency policies - *optional*

Add policies to help place players into games with lower latency. Use multiple policies to reduce latency requirements per policy so that each player eventually finds a match.

0 seconds left to allocate

100%

Period start

Seconds

Period end

Seconds

Max player latency

Milliseconds

Remove

Seconds

Seconds

Milliseconds

Remove

Seconds

Seconds

Milliseconds

Remove

Add policy

スポットインスタンスのキューの設計

スポットフリートを使用すると、ホスティングコストを大幅に削減できます。スポットフリートとその使用方法の詳細については、「[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。

ゲームホスティングソリューションにスポットフリートが含まれている場合は、ゲームセッションプレイメントキューを使用する必要があります。Amazon GameLift Servers はキューを使用して複数のゲームホスティングリソースを検索し、新しいゲームセッションをホストするために利用可能な最適なリソースを選択します。スポットフリートでは、キューはホスティングコストを最小限に抑え、スポットの中断を回避するために特に重要です。このトピックは、中断、スローダウン、停止が発生した場合でもプレイヤーのゲームをホストし続けることができる回復力のあるキューを設定するのに役立ちます。キューが利用可能なホスティングリソースに優先順位を付ける方法は、ホスティングコストなどのいくつかの要因に基づいてカスタマイズできます。

マッチメイキングに FlexMatch 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。

スポットフリートの実装タスク

スポットフリートを使用するようにゲームホスティングソリューションを作成または更新する場合は、次のタスクを完了します。スポットの可用性と回復性を最適化するキューを構築する方法の詳細については、「[スポットフリートによるゲームホスティングコストの削減](#)」を参照してください。

1. ゲームセッションキューのフリート送信先のセットを選択して作成します。

まず、キューでゲームセッションを配置する場所を決定します。キューは複数のフリートを検索して、最適な配置を見つけることができます。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。ロケーションタイプとインスタンスタイプの両方で異なるフリートを持つキューは、プレイメントを成功させる可能性が高くなります。効果的で回復力のあるスポット最適化キューを設計するためのベストプラクティスを参照してください。

2. スポット最適化ゲームセッションキューを作成します。

キューを作成し、スポットフリート用に設定します。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。Amazon GameLift Servers コンソールまたは AWS CLI を使用して、キューを作成または編集できます。

- ステップ 1 からフリートの送信先を追加します。
- 必要に応じて送信先の順序に優先順位を付けます。デフォルトでは、Amazon GameLift Servers は送信先のコストで優先順位を付けるため、送信先間のコストが最も低い場合にのみ送信先の順序が使用されます。
- プレイヤーのレイテンシーの前にゲームホスティングコストを優先する場合は、カスタムプレイメントの優先順位を指定します。「[ゲームセッション配置に優先順位を付ける](#)」を参照してください。

3. ソリューション内の他のコンポーネントを更新して、新しいキューを使用します。

ソリューションがスポット最適化キューを使用して新しいゲームセッションを開始すると、キューは中断の可能性の高いフリートでゲームセッションを配置することを自動的に回避します。代わりに、プレイヤーのレイテンシー、ホスティングコスト、送信先の順序など、定義された優先順位に一致するリソースについて、実行可能なすべてのフリートを検索します。

- FlexMatch を使用していない場合 – バックエンドサービスを更新して、ゲームセッションリクエストで新しいスポット最適化キューを指定します。バックエンドサービスは、ゲームクライアントに代わって (StartGameSessionPlacement()) を使用して) Amazon GameLift Servers に API リクエストを行い、各リクエストはキュー名を指定する必要があります。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。
- FlexMatch を使用している場合 – マッチメイキング設定を更新して、新しいスポット最適化キューにゲームセッションリクエストを送信します。マッチメイキングシステムがプレイヤーマッチを形成すると、指定されたキューにゲームセッション配置リクエストを送信して、マッチの新しいゲームセッションを開始します。FlexMatch モードが「マネージド」に設定されているマッチメイキング設定のみがプレイスメントキューを指定できます。CLI または AWS コンソールを使用してマッチメイキング設定を更新できます Amazon GameLift Servers (「[マッチメイキング設定の編集](#)」を参照)。

4. 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 Servers コンソールの Amazon GameLift Servers メトリクス、または Amazon CloudWatch を表示してパフォーマンスを確認できます。Amazon GameLift Servers メトリクスの詳細については、「[Amazon CloudWatch で Amazon GameLift Servers を監視する](#)」を参照してください。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。再要求されたインスタンスのゲームセッションのステータスは TERMINATED、ステータス理由は INTERRUPTED です。
- 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。

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

スポットインスタンスのフリートとキューを作成するときは、次のベストプラクティスを使用します。

- キューの地理的カバレッジを拡張します。プレイヤーが 1 つの にクラスター化されていても AWS リージョン、スポットフリートに隣接する場所を追加します。このアプローチにより、リージョンの速度低下、停止、スポットの中断時にキャパシティを維持するキューの機能が向上します。マルチロケーションフリートは、スポットインスタンスとオンデマンドインスタンスの両方で動作します。
- キューのインスタンスタイプのカバレッジを多様化します。 はインスタンスタイプに基づいてスポットの実行可能性Amazon GameLift Serversを評価するため、さまざまなインスタンスタイプのスポットフリートがあると、複数のスポットフリートが同時に実行できなくなる可能性が低くなります。各ロケーションに異なるインスタンスタイプを持つ少なくとも 2 つのスポットフリートを含めます。

Note

料金は、フリートの数ではなく、使用するインスタンスに基づいています。それぞれ 10 個のインスタンスで 5 つのフリートを実行することは、同様のコストで 50 個のインスタンスで 1 つのフリートを実行することと同じです。料金はインスタンスタイプ、サイズ、場所によって異なります。

スポットインスタンスタイプをグループ化するためのヒント:

- m6g.medium、m6g.large、m6g.xlarge など、同じファミリーでインスタンスタイプを使用します。インスタンスタイプが大きいほどコストは高くなりますが、一度により多くのゲームセッションをホストすることもできます。
- 広く利用可能なインスタンスタイプを選択します。通常、旧世代のファミリー (C5、M5、R5 など) と一般的なサイズ (.large、.xlarge、.2xlarge など) は可用性が高くなります。
- Amazon GameLift Servers コンソールで 30 ~ 90 日間の料金履歴を確認します。一貫した可用性パターンを持つインスタンスタイプを探します。
- Amazon GameLift Servers コンソール、フリート作成ツールを使用して、インスタンスタイプのロケーションカバレッジを調べます。
- バックアップ容量のオンデマンドフリートを追加します。ゲームホスティングは、スポットフリートが利用できないときはいつでもオンデマンドフリートに切り替えることができます。各ロケー

セッションに少なくとも 1 つのオンデマンドフリートを配置して、プレイヤーのレイテンシーを低く保ちます。バックアップオンデマンドフリートに自動スケーリングを追加すると、必要なまでスケールダウンしたままにできます。

- すべてのフリート送信先にエイリアスを割り当てます。キューの送信先ごとにエイリアスを作成します。エイリアスを使用すると、フリートを置き換える必要がある場合に簡単かつ効率的になります。
- キューの優先順位付け戦略を適用します。キューがゲームセッションの配置先を優先する方法をカスタマイズできます (詳細については、「[ゲームセッション配置に優先順位を付ける](#)」を参照)。スポット最適化キューの場合、コストで優先順位を付けると、可能な限り低コストのスポットフリートが使用されます。

送信先順序を指定して、特定のフリートに優先順位を付けることもできます。たとえば、一部のユーザーは、定期的に使用するプライマリフリートのセットと、バックアップとしてセカンダリフリートのセットを指定します。このシナリオでは、キューの送信先順序を設定して、プライマリフリートを最初に一覧表示します。次に、キューの優先順位を宛先とそれに続くコストで設定します。

ホスティングリソースのカスタマイズ

このセクションでは、特定のパフォーマンス、コスト、運用要件を満たすように Amazon GameLift Servers インフラストラクチャを設定および管理するための高度なオプションを提供します。特に、このセクションのトピックでは、ゲームとプレイヤーに最適な Amazon GameLift Servers マネージドホスティングリソースをカスタマイズする方法について説明します。

考慮すべき決定事項には、次のようなものがあります。

- プレイヤーのホスティングリソースをデプロイする場所とは ゲームプレイのレイテンシーはフリートの地理的場所を選択する上で重要な要素ですが、リソースタイプの可用性やコストなど、場所によって異なるその他の要因があります。
- ゲームに最適な EC2 インスタンスタイプはどれか。利用可能なインスタンスタイプの中から、コンピューティングアーキテクチャ、メモリ、ストレージ、ネットワーク容量の最適な組み合わせを使用するものを選択します。
- 必要なインスタンスタイプのサイズはどのくらいか。ゲームサーバーソフトウェアのリソース要件 (メモリと CPU) などの要因に基づいて、インスタンスタイプのサイズを選択します。

- フリートは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使用する必要があるか。低い Spot 料金を活用できるかどうか、また Amazon GameLift Servers がゲームセッションに対する Spot の中断リスクを十分に軽減できるかどうかを検討してください。
- 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でどのように実行するか。ランタイム設定は、実行するサーバーソフトウェアとその方法を Amazon GameLift Servers に指示します。
- コンテナフリートの場合、デフォルト設定がゲームで機能するかどうか。Amazon GameLift Servers はコンテナフリート設定を最適化するための作業の多くを代行しますが、ほとんどの設定はカスタマイズできます。

マネージドフリートのコンピューティングリソースを選択する

Amazon GameLift Servers マネージド EC2 やマネージドコンテナを含むマネージドホスティングの場合、サービスはゲームサーバーを AWS クラウドのコンピューティングリソースのフリートにデプロイします。マネージドフリートを作成する場合は、ゲームに最適なホスティングリソースを設定する必要があります。このトピックでは、ゲームホスティングフリートを選択および設定する際の重要な決定ポイントについて説明します。

Note

Anywhere フリートと Amazon GameLift Servers マネージドフリートの両方を使用してハイブリッドソリューションを構築する場合は、これらのトピックを参照して、セルフマネージドリソースを補完するための マネージドフリートを設計してください。「[Amazon GameLift Servers のホスティングフリートをデプロイする](#)」を参照してください。

トピック

- [地理的場所](#)
- [オペレーティングシステム](#)
- [インスタンスのタイプ](#)
- [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](#)
- [サービスクォータ](#)

地理的場所

ゲームサーバーをデプロイする予定の場所を検討してください。一般的に、可能な限り最高のプレイヤーエクスペリエンスを提供するために、ゲームサーバーをプレイヤーにできるだけ近づけたい

と考えています。Amazon GameLift Servers マネージドホスティングでは、サポートされている任意の AWS リージョン およびローカルゾーンにゲームサーバーを配置することを選択できます。ハイブリッドソリューションを構築する場合は、マネージドフリートのデプロイがセルフマネージド Amazon GameLift Servers Anywhere フリートの場所を補完する方法を検討してください。

ほとんどの開発およびテストシナリオでは、単一の場所にデプロイするのが理にかなっています。起動以降に備えると、複数の地理的な場所にデプロイする理由は多数あります。これには、幅広いプレイヤーグループをサポートし、ゲームホスティングの全体的な耐障害性と信頼性を向上させることが含まれます。複数のロケーションでは、ゲームセッションの配置を高速化し、レイテンシーとコストに合わせて配置を最適化するときにより多くの選択を可能にすることで、プレイヤーのエクスペリエンスを向上させることもできます。

Amazon GameLift Servers でサポートされているロケーションのリストと、すべてのフリートタイプのロケーションの詳細については、「[Amazon GameLift Servers サービスロケーション](#)」を参照してください。

マルチロケーションフリート

1 つのマネージドフリートで、複数の場所にリソースをデプロイできます。マルチロケーションフリート内の個々のロケーションに容量を手動で設定できます。

マルチロケーションフリートを使用する利点:

- フリートのデプロイと管理の簡素化 - ゲームサーバーソフトウェアとフリート設定を提供し、Amazon GameLift Servers が複数のロケーション (一度構築、任意の場所にデプロイ) のフリートインスタンスにデプロイします。本番フリートでは、各ロケーションが異なるリージョンにある複数のフリートを個別に管理する必要はなく、フリート内のすべてのロケーションを一覧表示して管理できます。
- ローカルゾーンの可用性 - ローカルゾーンを使用する場合は、AWS リージョン ホームロケーションとローカルゾーンをリモートロケーションとするマルチロケーションフリートを作成する必要があります。Local Zones は、それを必要とするエリアや顧客にレイテンシーをさらに短縮 AWS リージョン できる の拡張機能です。ローカルゾーンは任意のマルチロケーションフリートに追加できます。ローカルゾーンの親 AWS リージョンを含める必要はありません。
- ゲームセッションキューとの互換性 - 1 つ以上のマルチロケーションフリートを使用してゲームセッションプレイメントキューを構築できます。このアプローチにより、新しいゲームセッションをホストする場所を優先して選択する際のキューの柔軟性が得られます。
- 効率的なリソース使用率 - 自動スケーリングを有効にすると、Amazon GameLift Servers はフリート内のすべてのロケーションで容量スケーリングをより適切に最適化できます。

マルチロケーションフリートを使用するためのヒント:

- AWS リージョン または フリートあたりのロケーション数のクォータを確認します。「[Amazon GameLift Servers サービスのクォータ](#)」を参照してください。
- ロケーションによっては使用できないインスタンスタイプがあります。選択した場所によっては、インスタンスタイプのオプションが限られている場合があります。Amazon GameLift Servers コンソールには、場所とインスタンスタイプの適切なバランスを見つけるのに役立つ便利なツールが用意されています。
- [UDP ping ビーコン](#) を使用してすべてのフリートロケーションのプレイヤーレイテンシーデータを収集することを検討してください。Amazon GameLift Servers はこのデータを使用して、低レイテンシーのゲームセッションを最適化し、許容できないほど高いレイテンシーでプレイヤーがセッションに参加できないようにします。これらの特別なエンドポイントは、従来の ICMP ping の代わりに UDP メッセージを受け入れるため、正確なレイテンシー測定が可能になり、最適なフリートロケーションの選択に役立ちます。

オペレーティングシステム

マネージドフリート内のすべてのインスタンスは、ゲームサーバーソフトウェアの完全なランタイム環境を提供する Amazon マシンイメージ (AMI) でデプロイされます。マネージド EC2 フリートの場合、ビルドを Amazon GameLift Servers にアップロードするときにゲームサーバービルドのオペレーティングシステムを指定します。マネージドコンテナフリートの場合は、コンテナグループ定義でオペレーティングシステムを指定します。AMI のバージョンの詳細については、[Amazon GameLift Servers AMI バージョン](#) を参照してください。

AMI バージョンは定期的に更新されます。新しいフリートを作成すると、Amazon GameLift Servers はゲームビルド用に選択した AMI の最新バージョンを割り当てます。そのフリートにデプロイされているすべてのインスタンスは、同じバージョンを使用します。AMI バージョンを最新のセキュリティおよびソフトウェア更新で最新の状態に保つには、フリートを定期的に置き換える必要があります。ベストプラクティスとして、マネージドフリートのランタイム環境を最新に保つため、30日ごとにフリートを更新することをお勧めします。ガイダンスについては、「[Amazon GameLift Servers のセキュリティに関するベストプラクティス](#)」を参照してください。

インスタンスのタイプ

マネージドフリートのインスタンスタイプは、すべてのフリートインスタンスにデプロイされるハードウェアの種類を決定し、インスタンスタイプは一般的にさまざまなサイズで利用できます。すべての Amazon GameLift Servers マネージドフリートは Amazon EC2 インスタンスを使用し、コンピューティング能力、メモリ、ストレージ、ネットワーク機能のさまざまな組み合わせを提供する幅

広いインスタンスタイプをサポートします。インスタンスタイプの利用可否は、選択するロケーションによって異なります。

Amazon GameLift Servers コンソールには、ゲームビルドおよびデプロイ場所に適したインスタンスタイプを選択するのに役立つ便利なツールが用意されています。マネージドコンテナフリートの場合、コンソールはゲームの CPU 電源とメモリの要件に関するガイダンスも提供します。

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 AMI 用のサーバービルドが必要です。C++ および 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) および Amazon Linux 2 (AL2)での Mono インストールは標準ではサポートされません。

- 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。
- インスタンスタイプのサイズ。ゲームサーバーソフトウェア実行可能ファイルの要件を満たすだけでなく、より大きなインスタンスタイプでは、各インスタンスで複数のゲームサーバープロセスやコンテナを実行できます。検討すべき要素にはコストが含まれます (大きなインスタンスを少数実行する方が安いのか、小さなインスタンスを多数実行する方が安いのか)。フリートスケーリングイベントの間や異常のあるインスタンスをシャットダウンする際に、インスタンスを追加または削除すると、全体のゲームホスティング容量に大きく影響する可能性があります。各インスタンスが多数のゲームサーバープロセスを同時に実行する場合、インスタンスを追加または削除すると、同じホスティング容量に大きな影響を与える可能性があります。

インスタンスタイプの詳細については、「[Amazon EC2 インスタンスタイプ](#)」を参照してください。

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は従量課金制で、使用した時間分だけ料金が発生します。長期契約の必要はありません。

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方法を提供できます。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 2 分間の通知でスポットインスタンスを再利用 AWS で、再利用されたインスタンスでアクティブに実行されているゲームセッションは中断されます。

Amazon GameLift Servers には、ゲームセッションのスポット中断の可能性を軽減するのに役立つツールがいくつか用意されています。スポットバイアビリティアルゴリズムは、インスタンスタイプの履歴データを追跡して、中断のリスクが重要なポイントに達するタイミングを予測し、そのタイプのスポットインスタンスに新しいゲームセッションを配置しないようにします。中断が発生した場合、ゲームサーバーは通知を使用してプレイヤーのゲームセッションを適切に終了できます。

スポットフリートによるゲームホスティングでは、ゲームセッションの配置にキューを使用する必要があります。キューは、スポットフリートの実行可能性、コスト、その他の要因に基づいてゲームセッション配置に優先順位を付けることができます。ゲームサーバーホスティングでスポットを利用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。

- [スポットフリートによるゲームホスティングコストの削減](#)
- [スポットインスタンスのキューの設計](#)

サービスクォータ

次のツールを使用して、Amazon GameLift Servers のデフォルトのサービスクォータと、AWS アカウントの現在のクォータステータスを確認できます。

- Amazon GameLift Servers に関する一般的なサービスクォータ情報については、AWS 全般のリファレンスの「[Amazon GameLift Servers エンドポイントおよびクォータ](#)」を参照してください。
- アカウントのロケーションごとに利用可能なインスタンスタイプのリストについては、Amazon GameLift Servers コンソールの [\[Service Quotas\]](#) ページを開いてください。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。

ゲームをローンチする準備をするときに、[\[Amazon GameLift Servers コンソール\]](#) でローンチアンケートに記入してください。Amazon GameLift Servers チームはローンチ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 Servers コンテナフリートをカスタマイズする

このセクションのトピックでは、Amazon GameLift Servers マネージドコンテナのオプション機能の一部を説明します。これらの機能は、必要に応じて、いずれか単独でも、複数組み合わせても使用できます。

トピック

- [リソース制限の設定](#)
- [NVMe ドライブアクセスの設定](#)
- [必須コンテナの指定](#)
- [ネットワーク接続を構成する](#)
- [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の設定](#)
- [コンテナフリートを設定する](#)

リソース制限の設定

各コンテナグループについて、そのコンテナグループがソフトウェアを実行するために必要なメモリおよびコンピューティング能力を設定できます。Amazon GameLift Servers はこの情報を使用してグループ全体のリソースを管理します。さらに、この情報を使って、フリートイメージが保持できるゲームサーバーコンテナグループの数を算出します。個々のコンテナに対して制限を設定することもできます。

コンテナグループのメモリおよびコンピューティング能力の上限を設定できます。デフォルトでは、これらのリソースはグループ内のすべてのコンテナで共有されます。個々のコンテナに制限を設定することで、リソース管理をさらに細かく調整できます。

個々のコンテナにオプションの制限を設定

コンテナごとに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するかをより細かく管理できます。コンテナ固有の制限を設定しない場合、グループ内のコンテナはすべてグループのリソースを共有します。共有することで、必要なリソースを柔軟に使用しやすくなります。一方で、プロセス同士がリソースを奪い合い、結果的にコンテナが障害を起こす可能性も高まります。

任意のコンテナに対して、以下のいずれかの `ContainerDefinition` プロパティを設定できます。

- `MemoryHardLimitMebibytes` – 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。
- `Vcpu` 制限 – コンテナが専有して使用するための最小 vCPU リソースを予約します。コンテナには、この予約済みリソースが常に利用可能です。追加のリソースが利用可能な場合、この最小値を上回ることも可能です。(1024 CPU ユニットは 1 vCPU に相当します)。

コンテナグループの総合リソース制限を設定します。

個々のコンテナに制限を設定する場合、コンテナグループに必要なメモリや vCPU リソース量を調整する必要がある場合があります。目的は、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 Servers は、この制限情報を使用して、1 つのフリートインスタンスにどれだけのゲームコンテナサーバーグループを配置できるかを計算します。また、この情報はコンテナフリートのインスタンスタイプを選択する際にも使用します。

コンテナグループに必要な合計メモリと vCPU を計算します。以下の点を考慮してください。

- コンテナグループ内のすべてのコンテナで実行されるすべてのプロセスは何ですか。これらのプロセスに必要なリソースを加算します。コンテナ固有の制限に注意してください。
- 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数はいくつですか。これはゲームサーバーコンテナイメージで決定します。

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。

- `TotalMemoryLimitMebibytes` – 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 制限を設定する場合、合計メモリはそのコンテナ固有の最大メモリ 制限以上である必要があります。
- `TotalVcpuLimit` – コンテナグループの最大 vCPU 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。個々のコンテナ制限を設定する

場合、合計 CPU 制限は、すべてのコンテナ固有の CPU 制限の合計以上である必要があります。ベストプラクティスとして、この値をコンテナ 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 つのコンテナを使用して、ゲームサーバーコンテナグループを定義するとします。

- コンテナ A は ゲームサーバーコンテナ です。1 つのゲームサーバーのリソース要件は、512 MiB と 1024 CPU で見積もられます。コンテナで 1 つ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、桃利制限と vCPU リザーブ制限は設定しません。
- コンテナ B は、リソース要件が 1024 MiB および 1536 CPU と推定されるサポートコンテナです。メモリ上限は 2048 MiB、CPU リザーブの上限は 1024 CPU ユニットに設定されています。
- コンテナ C は別の サポート コンテナ です。メモリのハードリミットは 512 MiB、CPU リザーブの上限は 512 CPU ユニットに設定されています。

この情報を使用して、コンテナグループに対して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、メモリの上限 (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。

NVMe ドライブアクセスの設定

d タイプのインスタンスでは、ホストの起動時に NVMe ドライブが /data ディレクトリに自動的にマウントされます。コンテナが SSD ストレージにアクセスできるようにするには、次の `ContainerGroupDefinition` プロパティ を設定します `MountPoints`。

- `InstancePath` – ホストインスタンスの自動マウント NVMe ドライブを参照 /data するには、 に設定します。
- `AccessLevel` – コンテナのニーズに適したアクセスレベルを選択します (例: `READ_ONLY` または `READ_WRITE`)。
- `ContainerPath` – (オプション) インスタンスパスをコンテナ内にマウントするパスを指定します。指定しない場合、デフォルトでインスタンスパスになります。

マウントポイントの詳細については、「Amazon GameLift Servers API リファレンス」の [ContainerMountPoint](#) を参照してください。

必須コンテナの指定

各インスタンスのコンテナグループごとに、各コンテナをEssential (必須) または Non-essential (非必須) として指定します。インスタンス単位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、コンテナグループの重要な処理を担当します。必須コンテナは、常に稼働していることが期待されます。失敗すると、コンテナグループ全体が再起動します。

ContainerDefinitionプロパティEssentialを各コンテナの true または false に設定します。

ネットワーク接続を構成する

外部トラフィックがコンテナフリート内の任意のコンテナに接続できるように、ネットワークアクセスをカスタマイズできます。例えば、ゲームクライアントがゲームに参加できるようにするには、ゲームサーバープロセスを実行するコンテナに対してネットワーク接続を構成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game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には行われません。内部的には、コンテナ内のプロセスはコンテナポートで待ち受けます。外部からのトラフィックは、接続ポートを使用してフリート内のインスタンスに接続されます。Amazon GameLift Servers は、内部コンテナポートと外部公開ポートのマッピングを管理し、トラフィックがインスタンス上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 Servers は、ネットワーク接続に対する追加の制御レイヤーを提供します。各コンテナフリートには受信許可設定があり、外部公開ポートごとのアクセス制御が可能です。例えば、すべての接続ポートの許可を無効にして、フリート内のコンテナへのアクセスを完全に遮断することも可能です。

フリートの受信許可、接続ポート、およびコンテナポートを更新できます。

Warning

カスタムの InstanceConnectionPortRange または InstanceInboundPermissions を指定した場合、Amazon GameLift Servers はフリートでこれらの値を管理しなくなります。意図しない動作を避けるために、両方のフィールドを設定する必要があります。

コンテナポート範囲の設定

各コンテナ定義の一部として、コンテナポート範囲を設定します。これは、コンテナグループ定義に必要なパラメータです。外部アクセスが必要なすべての同時実行プロセスに対応できるよ

う、十分な数のポートを設定する必要があります。一部のコンテナにはポートが不要な場合もあります。

ゲームサーバーコンテナには、同時に実行される各ゲームサーバープロセスに対して、それぞれポートが必要です。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、その情報を Amazon GameLift Servers に報告します。

接続ポート範囲の設定

接続ポートのセットを使用して コンテナフリート を構成します。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 GameLift Servers は接続ポートを割り当て、必要に応じてコンテナポートにマッピングします。

デフォルトでは、Amazon GameLift Servers がすべてのコンテナグループに必要なポート数を計算し、それに応じたポート範囲を設定します。コンテナ グループ定義に更新をデプロイすると更新される、Amazon GameLift Servers によって計算された値を使用することを強く推奨します。接続ポート範囲をカスタマイズする必要がある場合は、次のガイダンスを使用します。

コンテナフリートを作成する際は、接続ポート範囲を定義します

([ContainerFleet:InstanceConnectionPortRange](#) を参照)。フリート内のすべてのコンテナグループに含まれるコンテナポートにマッピングできるよう、ポート範囲に十分な数のポートが含まれてい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 game server container group] * [Number of game server container groups per instance] + [Total number of container ports defined for containers in the per-instance 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

接続ポート数は、インスタンスあたりのゲームサーバーコンテナグループの数を制限する可能性があります。フリートに、インスタンスごとに1つのゲームサーバーコンテナグループしか対応できないだけの接続ポートしかない場合、たとえインスタンスに複数のゲームサーバーコンテナグループを実行できる十分な計算リソースがあっても、Amazon GameLift Servers は 1 つのゲームサーバーコンテナグループしかデプロイしません。

受信許可を設定する

受信許可は、受信トラフィックに対して開放する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できます。

デフォルトでは、Amazon GameLift Servers がすべてのコンテナグループに必要なポート数を計算し、それに応じたポート範囲を設定します。コンテナグループ定義に更新をデプロイすると更新される、Amazon GameLift Servers によって計算された値を使用することを強く推奨します。接続ポート範囲をカスタマイズする必要がある場合は、次のガイダンスを使用します。

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権限を定義します ([ContainerFleet::InstanceInboundPermissions](#) を参照)。受信許可ポートは、フリートの接続ポート範囲と一致している必要があります。

Note

コンテナポートは InstanceConnectionPortRange からランダムに選ばれるため、セッション接続を確実に行うには、InstanceConnectionPortRange に含まれるすべてのポートが InstanceInboundPermissions によって許可されている必要があります。

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どのように設定するかを示します。

- フリートのゲームサーバーコンテナグループには、1 つのコンテナがあり、1 つのゲームサーバープロセスを実行しています。

ゲームサーバー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インスタンス単位のコンテナグループもあります。このグループには、ネットワークアクセスを必要とするプロセスが1つ含まれています。インスタンス単位のコンテナ定義では、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"} ] }
```

- フリートでは、1 インスタンスあたり20個のゲームサーバーコンテナグループが構成されています。この情報に基づいて、必要な接続ポート数を次の計算式で求めることができます。
- 最小: 21 ポート $(1 \text{ ゲームサーバーコンテナ port} \times \text{インスタンスあたり } 20 \text{ ゲームサーバーコンテナ groups} + 1 \text{ per-instance コンテナ ポート})$
- ベストプラクティス: 42 ポート $[\text{最小ポート数} \times 2]$

コンテナフリートを作成するときは、InstanceConnectionPortRange パラメータを次のように設定します。

```
"InstanceConnectionPortRange": { "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は、InstanceInboundPermissions パラメータを次のように設定します。

```
"InstanceInboundPermissions": [  
  { "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

コンテナは、ターミナル障害が発生して実行が停止すると、自動的に再起動します。コンテナが必須 (Essential) と見なされている場合、そのコンテナが停止するとコンテナグループ全体が再起動されます。

すべてのゲームサーバーコンテナは、自動的に必須コンテナとして扱われます。サポートコンテナも必須に設定できますが、その場合は正常性を報告する仕組みが必要です。必須でないサポートコンテナにも、ヘルスチェックを設定できます。

コンテナの正常性を評価するための、独自のカスタム基準を定義し、その基準を検証するためにヘルスチェックを使用できます。コンテナのヘルスチェックは、Docker コンテナイメージまたはコンテナ定義内に記述できます。コンテナ定義にヘルスチェックを設定すると、コンテナイメージ内の設定は上書きされます。

コンテナヘルスチェックには、次の SupportContainerDefinition プロパティを設定します。

- **Command** — コンテナの状態の一部を確認するコマンドを指定します。正常性の基準は、任意に定義できます。コマンドの終了コードは、次のいずれかである必要があります: 1 (異常) または 0 (正常)。
- **StartPeriod** — ヘルスチェックの失敗カウントを開始するまでの遅延時間を指定します。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ための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指定します。コンテナ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までの成功・失敗の待機時間を設定します。ヘルスチェックコマンドの完了に必要な時間はどれくらいですか？
- **Retries** — 失敗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る必要がありますか？

コンテナの依存関係の設定

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。依存関係を設定することで、他のコンテナの状態に応じて、依存先コンテナの起動やシャットダウンのタイミングを制御できます。

依存関係の主なユースケースの 1 つは、コンテナグループのスタートアップおよびシャットダウンの順序を構成することです。

たとえば、コンテナ A を最初に起動し、B や C が起動する前に正常終了させたい場合があります。これを実現するには、まずコンテナ B に対してコンテナ A への依存関係を設定します。条件として、コンテナ A が正常に完了することを指定します。続けて、同じ条件でコンテナ C にもコンテナ A への依存関係を設定します。スタートアップシーケンスは、シャットダウン時に逆順で実行されます。

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際は、以下の設計上の判断項目を考慮してください。これらの判断項目の多くは、コンテナのアーキテクチャと設定内容に依存します。

フリートをどこにデプロイするかを決定します。

一般的には、レイテンシーを最小限に抑えるため、プレイヤーに近い地域にデプロイするのが理想です。コンテナフリートは、[Amazon GameLift Servers サポート AWS リージョン](#) する任意の [リージョン](#) にデプロイできます。同じゲームサーバーを追加の地理的場所にデプロイする場合は、[AWS リージョン](#) および [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を持つフリートでは、各ロケーションで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のロケーションの詳細については、「[Amazon GameLift Servers サービスロケーション](#)」を参照してください。

ネットワークのレイテンシーデータを収集するには、[UDP ping ビーコン](#) を使用して、各地域のレイテンシーを測定し、プレイヤーデバイスとフリート候補ロケーション間の遅延を予測することを検討してください。これらの特別なエンドポイントは、従来の ICMP ping の代わりに UDP メッセージを受け入れるため、正確なレイテンシー測定が可能になり、最適なフリートロケーションの選択に役立ちます。

フリートで使用するインスタンスタイプとサイズを選択する

Amazon GameLift Servers は、幅広い Amazon EC2 インスタンスタイプをサポートしており、これらはすべてコンテナフリートで利用できます。利用可能なインスタンスタイプと料金は、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 Servers コンソールの「リソース、インスタンス、およびサービスのクォータ」内で、ロケーション別にフィルタリングして確認できます。

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では、CPU、メモリ、ストレージ、ネットワーキング機能をさまざまな組み合わせで提供しています。[EC2 インスタンスファミリー](#)の詳細情報を取得します。各ファミリー内では、さまざまなインスタンスサイズを選択できます。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を教えてください。この情報を活用して、小さすぎるインスタンスタイプを除外しましょう。
- コンテナアーキテクチャに適したインスタンスタイプサイズは何ですか？理想的には、ゲームサーバーコンテナグループの複数のコピーを無駄なスペースを最小限に抑えて収容できるサイズを選択します。
- ゲームに適したスケーリングの粒度 (細かさ) はどれくらいですか？フリートの容量スケーリングにはインスタンスの追加や削除が伴い、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持ちます。インスタンスごとに、どれだけの容量を追加・削除したいかを検討しましょう。プレイヤーの需要が分単位で数千規模に変動する場合は、数百から数千のゲームセッションをホストできる大規模インスタンスの利用が有効です。一方で、小さなインスタンスタイプを用いることで、より細かくスケーリング制御ができる場合もあります。
- インスタンスサイズによって、コスト削減の可能性はありますか？一部のインスタンスタイプでは、利用可能性によりロケーションごとにコストが異なることがあります。

その他のフリート設定を構成する

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。「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」を参照してください。
- スケールダウン時に、アクティブなプレイヤーが参加しているゲームセッションが途中終了しないように保護します。
- 一定期間内に 1 人のユーザーがフリート上で作成できるゲームセッション数を制限します。

スポットフリートによるゲームホスティングコストの削減

Amazon GameLift Servers マネージドホスティングを使用してマルチプレイヤーゲームサーバーをホストする場合、スポットインスタンスはオンデマンドインスタンスの代わりにコスト効率の高い方法を提供できます。スポット料金モデルは、オンデマンドと同じハードウェアとパフォーマンスを提供しますが、コストを大幅に削減できる可能性があります (最大 70~90%)。ただし、容量を戻す AWS 必要がある場合、2 分間の中断通知でこれらのインスタンスを再利用できます。

Amazon GameLift Servers は、ゲームサーバーホスティングの中断リスクを軽減します。Amazon GameLift Servers は、スポットインスタンスタイプの中断の可能性を予測し、すべてのインスタンスにゲームセッションを危険にさらすことを回避します。まれな中断が発生した場合、通知によりプレイヤーのゲームセッションを正常に終了できます。

Amazon GameLift Servers がスポットフリートと連携する方法

ゲームホスティング用にスポットフリートを設定すると、Amazon GameLift Servers はゲームホスティングの実行可能性についてスポットフリートのインスタンスタイプと場所を継続的に評価します。

- スポット実行可能性アルゴリズムは、スポットインスタンスタイプの最近の可用性パターンと過去の中断率をロケーション別に分析します。
- この分析に基づいて、Amazon GameLift Servers は、ゲームセッションの中断の可能性が許容できないスポットインスタンスタイプと場所を特定します。以下のアクションを実行します。
 - インスタンスタイプと場所の組み合わせを一時的に無効としてマークします。
 - 新しいゲームセッションを配置するとき、実行不可能なスポットフリートの場所が考慮から除外されます。その結果、ゲームセッションは、ゲームサーバーのホスティングが中断されない可能性が高いスポットフリートのロケーションにのみ配置されます。
- AWS が既存のインスタンスのスポットフリートの場所を再利用しない場合でもドレインするため、ゲームホスティングに使用できないインスタンスには料金が発生しません。ゲームセッション

ン保護が有効になっている場合、インスタンスはアクティブなゲームセッションが完了した後のみシャットダウンされます。

- Amazon GameLift Servers は、ゲームホスティングの実行可能性についてスポットフリートのインスタンスタイプと場所を継続的に再評価します。更新された履歴データに基づいて、以前は実行できなかったインスタンスタイプが再び有効になると、スポットフリートを再度スケールアップでき、Amazon GameLift Servers はそれを使用してゲームセッションの配置を再開します。

設計上の考慮事項

スポットフリートを使用するソリューションを設計するときは、次の問題を考慮してください。

- ゲームセッションの長さを評価する - ゲームセッションの平均長は、スポットがゲームに対してどの程度うまく機能するかに影響を与える可能性があります。ゲームセッションが短いほど、切り替えが速くなり、最新の履歴データに基づいて実行可能なインスタンスタイプでゲームセッションが実行されます。長いゲームセッションは、最新の実行可能性データを評価しないまま同じインスタンスタイプで実行され続けるため、時間の経過とともに中断するリスクが高まります。
- インスタンスタイプの可用性を評価する - すべてのフリートロケーションがすべてのインスタンスタイプをスポットとして提供するわけではありません。スポットフリートのインスタンスタイプを選択するときは、Amazon GameLift Servers コンソールフリート作成ツールを使用して、必要な場所にスポットインスタンスタイプを見つけることができます。このツールを使用すると、フリートロケーションを選択し、それらのロケーションでのインスタンスタイプの可用性を表示できます。
- マルチロケーションスポットフリートの作成 - 複数のロケーションでスポットフリートを作成できます。単一のマルチロケーションスポットフリートは、同じインスタンスタイプのインスタンスを複数の AWS リージョン またはローカルゾーンにデプロイします。スポット実行可能性アルゴリズムは、インスタンスタイプと場所の両方に基づいて実行可能性を評価します。スポットフリートの場所が実行不可能と評価された場合、フリート内の他の場所には影響しません。この場所はゲームセッションのホストに使用できます。
- スポットフリートの多様性を持つキューを作成する - ゲームホスティングにスポットフリートを使用している場合は、ゲームセッションプレイスメントキューを設定する必要があります。新しいゲームセッションリクエストごとに、キューは利用可能なゲームホスティングリソースを検索し、最適なオプションを選択します。スポットフリートでは、ロケーションやインスタンスタイプが異なる複数のフリートを検索できるキューが必要です。また、バックアップ容量として少なくとも 1 つのオンデマンドフリートを含める必要があります。さまざまな配置オプションを提供する適切に設計されたマルチフリートキューは、中断、速度低下、停止に対して高い耐障害性を備えます。スポットのキューの設計に関するその他のガイダンスについては、[スポットインスタンスのキューの設計](#) を参照してください。

- 中断を適切に処理する - スポットの中断時にプレイヤーへの影響を最小限に抑えるようにゲームサーバーを設定します。ガスロットインスタンス AWS を再利用すると、はサーバー SDK コールバック関数を使用して、影響を受けるすべてのサーバープロセスに終了通知をAmazon GameLift Servers渡しますonProcessTerminate()。ゲームセッションを適切に終了するためには、ゲームにこのコールバックを実装する必要があります。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。

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努めますが、警告が到着する前に ガスロットインスタンスを AWS 再利用する可能性があります。ゲームサーバーが予期しない中断を適切に処理できるように準備しておきます。

- バックアップフリートの自動スケーリングを設定して、スポットの中断中にサービスを維持します。ターゲット追跡の自動スケーリングは容量バッファを維持し、需要に応じて自動的にスケーリングします。自動スケーリングでは、バックアップフリート (スポットまたはオンデマンド) は、より多くのゲームセッションリクエストの受信を開始するたびに容量の増加を開始します。

スポットフリートが実行不能になったときに失われた容量をすばやく置き換えるために、カスタムスケーリングメカニズムは使用可能なキューとフリートのメトリクスを使用してバックアップフリートの迅速なスケールアウトを開始できます。スポットフリートが FirstChoiceOutOfCapacity、FirstChoiceNotViable、PercentAvailableGameSessions などのメトリクスで実行不能になったときを検出します。最近の PlacementsStarted メトリクスデータを分析して、代替容量のニーズを見積もります。即時の需要に対応するためにバックアップフリートをスケーリングすると、通常の自動スケーリングが引き継がれます。

- FlexMatch との統合 - ソリューションが FlexMatch マッチメーカーを使用している場合、スポットフリートに特別な要件はありません。スポットフリートでキューを使用するようにマッチメーカーを設定できます。Amazon GameLift Servers は、新しいゲームセッションを配置する場合や既存のゲームセッションで空のプレイヤースロットをバックフィルする場合など、スポットフリートとオンデマンドフリート間でマッチプレイメントを自動的に優先します。

マネージド Amazon GameLift Servers でのゲームサーバーランタイム設定の最適化

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 Servers 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 Servers 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 Servers 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 Servers 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 Servers 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 Servers 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 Servers 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上で実行され、サーバー SDK for Amazon GameLift Servers 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 以降と統合します。

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 Servers がプロセスを効率よくリサイクルできるよう、サーバープロセスのシャットダウンシナリオに適切に対応します。例えば、次のようになります。
 - 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 Servers からの終了リクエストを適切に処理するため、`OnProcessTerminate()` コールバック関数をゲームサーバーコードに実装する。
- Amazon GameLift Servers 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し、サーバーのヘルスステータスを Amazon GameLift Servers に報告します。Amazon GameLift Servers は、3 回連続で異常と報告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 Servers 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 Servers 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 Servers 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 GameLift Servers エージェントの操作

Amazon GameLift Servers エージェントは、Amazon GameLift Servers フリートでのゲームサーバープロセスの実行を監督します。エージェントはフリート内の各コンピューティングにデプロイされ、コンピューティングの自動プロセス管理、ホスティング管理、ログ記録を提供します。エージェントを使用するには、ゲームサーバービルドをバージョン 5.x 以降用のサーバー SDK for Amazon GameLift Servers と統合する必要があります。

Amazon GameLift Servers エージェントは、マネージド EC2 フリートではない Amazon GameLift Servers フリートで外部使用できます。(マネージド EC2 フリートは、エージェントのタスクを自動的に処理します。) エージェントの有無にかかわらず、Anywhere フリートを含む Amazon GameLift Servers フリートを実行できます。エージェントがない場合、必要なタスクを完了するための代替ソリューションを提供する必要があります。

コンピューティングにデプロイする場合は、ゲームサーバープロセスを開始する前に Amazon GameLift Servers エージェントを起動する必要があります。起動時に、エージェントは次のタスクを完了します。

- [RegisterCompute](#) API を使用して、コンピューティングを Amazon GameLift Servers Anywhere フリートに登録します。
- [GetComputeAuthToken](#) API を呼び出して認証トークンを取得し、コンピューティングで実行されているサーバープロセスでできるように保存します。
- コンピューティングの WebSocket URL 環境変数を設定し、Amazon GameLift Servers サービスへの WebSocket 接続を確立します。
- Amazon GameLift Servers サービスからフリートのランタイム設定の最新バージョンを要求します。
- ランタイム設定手順に従って、サーバープロセスを開始および停止します。

Amazon GameLift Servers エージェントのソースコードおよびビルドの手順は、[Amazon GameLift Servers エージェント](#) GitHub で入手できます。

エージェントについて

Amazon GameLift Servers エージェントは、フリートに対して次のタスクを処理するように設計されています。

プロセス管理

- ランタイム手順で定義されているように、新しいサーバープロセスを開始します。エージェントは、エージェントにデプロイされたカスタムランタイム設定を使用する場合があります。また、RuntimeConfiguration をフリート定義の一環として指定することもできます。このアプローチの利点として、いつでもフリートのランタイム設定を変更できることが挙げられます。エージェントは、定期的に Amazon GameLift Servers サービスから更新されたランタイム設定を要求します。
- サーバープロセスのアクティベーションを監視し、時間内にアクティブ化されない場合はプロセスを終了します。
- ハートビートを Amazon GameLift Servers に送信します。エージェントがハートビートを送信できない場合、コンピューティングは「古い」としてマークされる場合があります。
- サーバープロセスが終了したら、Amazon GameLift Servers に報告します。Amazon GameLift Servers はこの情報を使用して、ゲームセッション配置のゲームサーバーの可用性をモニタリングします。
- 以下を含むサーバープロセスのフリートイベントを出力します。
 - SERVER_PROCESS_INVALID_PATH: ゲームサーバープロセスの起動パラメータが正しく設定されていません。
 - SERVER_PROCESS_TERMINATED_UNHEALTHY: ゲームサーバープロセスが、アクティブ化されてから 3 分以内に有効なヘルスチェックを報告しなかったため、終了しました。
 - SERVER_PROCESS_FORCE_TERMINATED: OnProcessTerminate() が 30 秒以内に送信された後、ゲームサーバープロセスが正常に終了しませんでした。
 - SERVER_PROCESS_CRASHED: 何らかの理由でゲームサーバープロセスがクラッシュしました。

コンピューティング管理

- Amazon GameLift Servers サービスからメッセージを受信して、コンピューティングをシャットダウンします。
- Amazon GameLift Servers によって終了されることをコンピューティングにプロンプトします。

ログ記録

- AWS アカウントの Amazon S3 バケットにログをアップロードします。

エイリアスを使用して Amazon GameLift Servers フリートの指定を抽象化する

Amazon GameLift Servers エイリアスは、ホスティング先を抽象化するために使用されます。ホスティング先は、Amazon GameLift Servers に、プレイヤー用に新しいゲームセッションをホスティングするために利用できるリソースを検索する場所を伝えます。エイリアスは、次のシナリオで有用です。

- ゲームセッション配置にマルチフリートキューを使用しない場合、Amazon GameLift Servers フリート ID を指定して新しいゲームセッションをリクエストします。ゲームの開始から停止まで、サーバービルドの更新やホスティングハードウェアとオペレーティングシステムの更新、パフォーマンスの問題の解決のために何度もフリートを入れ替えることになります。エイリアスを使用してフリート ID を抽象化することで、プレイヤートラフィックを既存のフリートから新しいフリートにシームレスに切り替えることができます。
- ゲームクライアントが新しいゲームセッションをリクエストした際に、セッションを作成する以外のことを行う場合。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ウェブサイトへ誘導する場合などです。

エイリアスはルーティング戦略を指定する必要があります。ルーティング戦略には 2 種類あります。シンプルルーティング戦略は、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。このフリート ID を更新することでトラフィックをリダイレクトできます。ターミナルルーティング戦略は、新しいゲームセッションを作成する代わりに、メッセージをクライアントに渡します。エイリアスのルーティング戦略はいつでも変更できます。

ゲームセッション配置にキューを使用する場合、フリートを入れ替える際にトラフィックをリダイレクトするためのエイリアスは必要ありません。キューを使用すれば、新しいフリートを追加し、古いフリートを削除するだけで済みます。新しいゲームセッションリクエストは新しいフリートを使用して自動的に実行されるため、このアクションはプレイヤーからは見えません。既存のゲームセッションにも影響しません。キューの宛先は、フリート ID またはエイリアスを使用して識別できます。

トピック

- [Amazon GameLift Servers エイリアスの作成](#)

Amazon GameLift Servers エイリアスの作成

このトピックでは、ゲームセッションの配置で使用する Amazon GameLift Servers エイリアスの作成方法を説明します。

エイリアスを作成するには

エイリアスは、Amazon GameLift Servers コンソールまたは AWS Command Line Interface (AWS CLI) を使用して作成できます。

Console

[\[Amazon GameLift Servers コンソール\]](#) のナビゲーションペインを使用して、[エイリアス] ページを開きます。

1. [エイリアスを作成] を選択します。
2. エイリアスの [名前] を入力します。エイリアスの一覧を確認する際にわかりやすいように、エイリアス名には意味のある特徴を持たせることをお勧めします。
3. 必要に応じてエイリアスの [説明] を入力します。
4. エイリアスの [ルーティング戦略] を選択します。
 - a. [シンプル] ルーティング戦略を選択した場合は、このエイリアスに関連付けるフリート ID をリストから選択します。リストには、現在選択されている AWS リージョンのすべてのフリートが含まれます。エイリアスはフリートと同じリージョンに作成する必要があります。
 - b. [ターミナル] ルーティング戦略を選択した場合は、ゲームセッションリクエストに応じて Amazon GameLift Servers がゲームクライアントに返す文字列値を入力します。ターミナルエイリアスを持つリクエストは、メッセージが埋め込まれた例外をスローします。
5. (オプション) エイリアスリソースに [タグ] を追加します。タグはそれぞれ、1 つのキーとオプションの 1 つの値で設定されており、どちらもお客様側が定義します。タグを割り当てることで、AWS リソースを目的、所有者、環境など有用な方法で分類することができます。追加するタグごとに [新しいタグを追加] を選択します。
6. 新しいフリートをデプロイする準備ができたなら、[作成] を選択します。

AWS CLI

[create-alias](#) コマンドを使用してエイリアスを作成します。Amazon GameLift Servers は現在のデフォルト AWS リージョン にエイリアスリソースを作成します (または、`--region` タグを追加して別の AWS リージョン を指定することもできます)。

少なくとも、エイリアス名とルーティング戦略を含めます。シンプルルーティング戦略では、エイリアスと同じリージョンにあるフリートの ID を指定します。ターミナルルーティング戦略には、メッセージ文字列を指定します。

Amazon GameLift Servers ホスティングソリューションの管理(起動以降)

このセクションでは、Amazon GameLift Servers リソースを長期的に管理する方法に関するガイダンスを提供します。マルチプレイヤーゲームを正常に起動することは始まりにすぎません。本番ホスティングソリューションを管理するには、パフォーマンス、コストの最適化、セキュリティの更新、プレイヤーエクスペリエンスに継続的に注意する必要があります。このセクションでは、1 日目の起動アクティビティから長期のメンテナンスと最適化まで、本番環境での Amazon GameLift Servers の実行の運用面に関するガイダンスを提供します。

これらのトピックは、起動の準備、ライブゲームの管理、確立されたホスティングソリューションの最適化など、プレイヤーベースに合わせてスケールする信頼性が高く費用対効果の高いゲームホスティングを維持するのに役立ちます。

Amazon GameLift Servers を使用したホスティングリソースの管理

ビルド、フリート、ゲームセッションキュー、マッチメーカーなどの Amazon GameLift Servers リソースを管理します。Amazon GameLift Servers コンソールを使用して、リソースリストをスキャンしたり、メトリクスを含むリソースの詳細を表示したりできます。の AWS SDK または CLI AWS を使用してリソース情報を取得することもできます。Amazon GameLift Servers

Amazon GameLift Servers コンソールの機能

Amazon GameLift Servers コンソールを使用して、ゲームホスティングリソースと進行中のホスティングアクティビティをほぼリアルタイムで表示および管理します。コンソールには、Amazon GameLift Servers のサービス API のほとんどの機能用のグラフィカルインターフェイスが用意されています。コンソールでは次の操作が可能です。

- ダッシュボードを使用して高レベルのスナップショットを作成します。すべての Amazon GameLift Servers ホスティングリソースの番号と現在のステータスを確認し、リンクに従って個々のリソースの詳細を確認できます。
- 個々のホスティングリソースを管理します。すべての Amazon GameLift Servers リソースを作成、表示、削除し、変更可能なプロパティを更新できます。イベントやパフォーマンスメトリクスなど、特定のタイプのホスティングアクティビティを表示することもできます。
- ゲームとプレイヤーのセッションアクティビティを操作します。フリートごとにゲームセッションとプレイヤーセッションのアクティビティを追跡し、この情報を使用してゲームセッションの問題

をトラブルシューティングできます。ゲームセッションの詳細の表示、各ゲームセッションのプレイヤーセッションの表示、複数のゲームセッションでのプレイヤーアクティビティの検索を行います。必要に応じて、個々のゲームセッションをシャットダウンすることもできます。

トピック

- [Amazon GameLift Servers コンソールでのダッシュボードのホスティング](#)
- [ゲームサーバービルド](#)
- [Amazon GameLift Servers コンソールのフリート](#)
- [Amazon GameLift Servers コンソールのエイリアス](#)
- [Amazon GameLift Servers コンソールのゲームセッションキュー](#)

Amazon GameLift Servers コンソールでのダッシュボードのホスティング

Amazon GameLift Servers コンソールダッシュボードを使用して、AWS アカウント内のAmazon GameLift Serversホスティングリソースの現在のステータスに関する概要スナップショットを取得します。[Amazon GameLift Serversダッシュボード] には、以下の内容が表示されます。

- [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。
- 新しい機能とサービスのお知らせ。

Amazon GameLift Servers ダッシュボードを開くには

- [Amazon GameLift Servers 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。

GameLift > Dashboard

Dashboard

Build status overview

View builds

Viewing data for all builds in N. Virginia region.

Ready

1

Initialized

0

Failed

0

Fleet status overview

View fleets

Viewing data for all fleets in N. Virginia region.

Active

0

New

0

Deleting

0

Error

0

In progress

0

Terminated

0

Resources (1)

View service quotas

Resource type	Count
Builds	1
Scripts	0
Fleets	0
Aliases	0
Queues	0
Matchmaking rule sets	0
Matchmaking configurations	0

Prepare for your game launch [Learn more](#)

Fill out a launch questionnaire

Fill out our game launch questionnaire and email it to the GameLift launch team to ensure a smooth launch. The GameLift launch team will verify your GameLift setup and service limits, preparing you for launch.

[Prepare to launch](#)

Features spotlight

Updates on features available in N. Virginia region

March 22, 2022

Updates to Amazon GameLift FlexMatch for greater flexibility

October 28, 2021

New Asia Pacific (Osaka) region and Graviton2 support for Amazon GameLift

ゲームサーバービルド

ビルドリソースは、ゲームサーバーソフトウェアを表します。マネージド EC2 フリートにデプロイするために、ビルドパッケージを Amazon GameLift Servers にアップロードします。

ゲームサーバービルドに関する情報は、Amazon GameLift Servers コンソールで、または または AWS SDK for を使用して表示します Amazon GameLift Servers。

Console

[Amazon GameLift Servers コンソール](#)の [ビルド] ページでは、マネージド EC2 フリートにデプロイするために Amazon GameLift Servers にアップロードしたすべてのゲームサーバービルドを表示および管理できます。ナビゲーションペインで、[ホスティング]、[マネージド EC2]、[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。必要に応じて、[環境設定] ツールを使用してテーブルの内容を調整できます (テーブルの右上隅にある



アイコンを参照)。カスタム設定は AWS アカウントユーザーに保存され、このページを表示するたびに自動的に適用されます。

Note

ビルドページには、現在の AWS リージョンでのみビルドが表示されます。

- [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 Servers がバイナリを受信する前に構築がタイムアウトしました。
- [作成日] – ビルドを Amazon GameLift Servers 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 Servers 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 Servers 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行います。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、このビルドを実行しているフリートのリストが表示されます。また、[\[フリート\] ページ](#)と同じ概要情報が含まれます。

AWS SDK

次の AWS CLI コマンドを使用して、このリソースに関する情報を取得します。

- [ListBuild](#)
- [DescribeBuild](#)

ゲーム サーバグループを更新する

Amazon GameLift Servers マネージド EC2 ホスティング用にゲームサーバービルドをデプロイするときは、ゲームサーバーソフトウェアをアップロードし、Amazon GameLift Servers ビルドリソースを作成します。Amazon GameLift Servers ビルドを作成したら、ビルドのメタデータを更新できますが、ビルドファイル自体を更新することはできません。ゲームサーバービルドに更新をデプロイするには、新しいファイルのセットをアップロードし、AWS CLI コマンド[upload-build](#)を使用して新しいAmazon GameLift Serversビルドを作成する必要があります。または、[create-build](#) コマンドを使用して、自らが管理している Amazon S3 バケットから新しいビルドをアップロードすることもできます。次に、新しいフリートを作成して、新しいビルドをデプロイします。

名前や説明など、ビルドのメタデータを更新できます。これらのタスクでは、Amazon GameLift Serversコンソールまたは [update-build](#) AWS CLI コマンドを使用します。

ゲームビルドの更新を自動化する

Amazon GameLift Servers マネージドフリートのゲームサーバービルドの更新プロセスを自動化および効率化するためのヒントを紹介します。

- 必要に応じて、キューを使用し、フリートを交換する。Amazon GameLift Servers にゲームセッションリクエストを送信するときは、特定のフリートではなくゲームセッションキューを指定します。キューを使用する場合は、新しいビルドのフリートを追加し、古いフリートを削除します。詳細については、「[ゲームセッションの配置を設定する](#)」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。ゲームセッションリクエストを Amazon GameLift Servers に送信するときは、フリート ID ではなくフリートエイリアスを指定する。詳細については、「[Amazon GameLift Servers エイリアスの作成](#)」を参照してください。
- 反復型開発のための環境の設定。ゲーム開発中、高速反復型開発をサポートするホスト型テスト環境を立ち上げるためのオプションを検討します。「[Amazon GameLift Servers Anywhere を使用した反復型開発の準備](#)」を参照してください。

GitHub の [Amazon GameLift Servers Toolkit](#) には、以下のリソースが用意されています。

Fast Build Update Tool (開発専用)

このツールを使用すると、マネージド EC2 フリートのコンピューティングに既にデプロイされているゲームサーバービルドを変更できます。これにより、迅速なデプロイイテレーションにかかる時間を節約できます。このツールにはいくつかのオプションがあり、ゲームビルド全体を置き換えることも、特定のファイルを変更することもできるほか、更新後にゲームサーバープロセスを再起動する方法を管理できます。このツールを使用して、フリート内のすべてのコンピューティングを更新することも、個々のコンピューティングのみを更新することもできます。

[Fast Build Update Tool](#) を入手し、その使用方法の詳細を確認するには、GitHub にある Amazon GameLift Servers Toolkit リポジトリを参照してください。

本番デプロイ用のサンプルスクリプト

このスクリプトは、本番環境のマネージド EC2 フリートにデプロイされたゲームサーバービルドを更新するプロセスを自動化する方法を示しています。このスクリプトを使用するには、Amazon GameLift Servers ホスティングソリューションでエイリアスを使用してフリート ID を抽象化する必要があります。サンプルスクリプトは、更新されたビルドのアップロード、新しいビルドの作成と新しいフリートへのデプロイ、既存のフリートから新しいフリートへのプレイ

ヤートラフィックのリダイレクト、古いフリートの削除という基本的なステップを処理します。特定のデプロイ要件を満たすように、サンプルスクリプトをカスタマイズできます。

[本番デプロイ用のサンプルスクリプト](#)を入手し、その使用方法の詳細を確認するには、GitHub にある Amazon GameLift Servers Toolkit リポジトリを参照してください。

Amazon GameLift Servers コンソールのフリート

フリートリソースは、デプロイされ、ゲームサーバーを実行する一連のコンピューティングマシンを表します。ホスティングソリューションによっては、マネージド EC2 フリート、マネージドコンテナフリート、Anywhere フリート、またはそれらの組み合わせがある場合があります。

フリートに関する情報をAmazon GameLift Serversコンソールで、または または AWS SDK for を使用して表示しますAmazon GameLift Servers。

Console

AWS アカウントAmazon GameLift Serversで で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。コンソールの左側のナビゲーションで、必要なホスティングオプション (Anywhere、マネージド EC2、マネージドコンテナ) を見つけ、フリートを選択します。

各ホスティングオプションについて、フリートページには、現在選択されている AWS リージョンにあるフリートが一覧表示されます。[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] ページには、以下の概要情報がデフォルトで表示されます。必要に応じて、[環境設定] ツールを使用してテーブルの内容を調整できます (テーブルの右上隅にある



アイコンを参照)。カスタム設定は AWS アカウントユーザーに保存され、このページを表示するたびに自動的に適用されます。

- ID- フリートに割り当てられた識別子です。この ID は、フリート AWS リージョン が作成される 内で一意です。
- 名 - フリートに付けられたわかりやすい名前です。

- ステータス - フリートのステータスとして、[新規]、[ダウンロード中]、[構築中] および [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。

 Note

90 日より前に作成されたフリートには、警告アイコンが表示されます。ベストプラクティスとして、ホストされているゲームサーバーでの安全で最新のランタイム環境を維持するために、30 日ごとにフリートを置き換えることをお勧めします。ガイダンスについては、「[Amazon GameLift Servers のセキュリティに関するベストプラクティス](#)」を参照してください。

- フリートタイプ - ゲームのホストに使用されるインスタンスの可用性。ホスティングコストに影響する可能性があります。マネージドフリートは、オンデマンド (常に使用可能) またはスポット (可用性はさまざま) インスタンスを使用できます。
- 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。
- プレイヤーセッション - フリート内のアクティブなプレイヤーセッションの数。データには 5 分の遅延が適用されています。

AWS SDK

次の AWS CLI コマンドを使用して、このリソースに関する情報を取得します。

- マネージド EC2 フリート
 - [ListFleets](#)
 - [ListCompute](#)
 - [DescribeFleetAttributes](#)
 - [DescribeCompute](#)
 - [DescribeFleetCapacity](#)
 - [DescribeFleetUtilization](#)
- マネージドコンテナフリート
 - [ListContainerFleets](#)

- [DescribeContainerFleet](#)
- [ListContainerGroupDefinitions](#)
- [DescribeContainerGroupDefinition](#)
- [ListFleets](#)
- [ListCompute](#)
- [DescribeFleetAttributes](#)
- [DescribeCompute](#)

Amazon GameLift Servers フリート設定を更新する

Amazon GameLift Servers コンソールまたは AWS CLI を使用して、フリート設定の更新、リモートロケーションの変更、フリートの削除を行います。マネージドフリートの場合、フリートのゲームサーバービルドまたはインスタンスタイプを変更することはできません。代わりに、フリートを新しいものに置き換える必要があります。

Fast Build Update Tool (開発専用)

マネージド EC2 フリートの場合、ゲームサーバービルドの更新をデプロイするには、新しいビルドを 1 つずつ Amazon GameLift Servers にアップロードし、ビルドに対して新しいフリートを作成する必要があります。

Fast Build Update Tool を使用すると、開発中にこうした手順を回避できるため、時間を節約し、開発のイテレーションを高速化できます。このツールを使用すると、既存のフリート内のすべてのコンピューティングでゲームビルドファイルをすばやく更新できます。このツールにはいくつかのオプションがあり、ゲームビルド全体を置き換えたり、6 つの特定のファイルを変更したり、更新後にゲームサーバープロセスを再起動する方法を管理できます。このツールを使用して、フリート内の個々のコンピューティングを更新することもできます。Fast Build Update Tool を入手し、その使用方法の詳細を確認するには、GitHub にある [Fast Build Update Tool](#) の Amazon GameLift Servers リポジトリを参照してください。

Amazon GameLift Servers コンソールまたは AWS CLI を使用して、変更可能なフリート属性、ポート設定、ランタイム設定を更新できます。スケーリング制限を変更するには、「[Amazon GameLift Servers でのフリート容量の自動スケーリング](#)」を参照してください。

Console

1. [\[Amazon GameLift Servers コンソール\]](#) 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 Servers 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

次の 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 Servers 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。

Console

1. [\[Amazon GameLift Servers コンソール\]](#) 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、[追加]を選択します。Amazon GameLift Servers は新しいロケーションをリストに追加し、ステータスは NEW に設定されます。Amazon GameLift Servers は、追加された各ロケーションでインスタンスのプロビジョニングを開始し、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。Amazon GameLift Servers は削除された場所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 Servers 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

フリートの場所を更新するには、次の AWS CLI コマンドを使用します。

- [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、自動スケーリングを無効にして、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 Servers は、フリート削除時に 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 Servers コンソールまたは AWS CLI ツールを使用できます。

Console

1. [\[Amazon GameLift Servers コンソール\]](#) 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. [削除] を選択します。
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. [削除] を選択します。

AWS CLI

フリートを削除するには、次の AWS CLI コマンドを使用します。

- [delete-fleet](#)

Amazon GameLift Servers コンソールのフリートの詳細

[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。
- ターゲット追跡の自動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。

Details

フリート設定

- 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。この ID は、フリート AWS リージョン が作成される 内で一意です。
- 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。
- ARN - このフリートに割り当てられた一意の識別子です。フリートの ARN は、それをAmazon GameLift Servers 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。
- ステータス - フリートの現在のステータスとして、[新規]、[ダウンロード中]、[構築中] および [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。

Note

90 日より前に作成されたフリートには、警告アイコンが表示されます。ベストプラクティスとして、ホストされているゲームサーバーでの安全で最新のランタイム環境を維持するために、30 日ごとにフリートを置き換えることをお勧めします。ガイダンスについては、「[Amazon GameLift Servers のセキュリティに関するベストプラクティス](#)」を参照してください。

- 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、マネージド EC2、マネージドコンテナ、または Anywhere フリートです。
- オペレーティングシステム - フリート内のすべてのインスタンスにデプロイされる Amazon マシンイメージ (AMI)。AMI バージョンは、フリートの作成時に利用可能な最新バージョンです。

- 証明書タイプ - フリートがゲームサーバーの認証およびクライアント/サーバー間通信の暗号化に TLS 証明書を使用できる状態を示します。
- フリートタイプ - ゲームのホストに使用されるインスタンスの可用性。ホスティングコストに影響する可能性があります。フリートは、オンデマンド (常に使用可能) またはスポット (可用性は異なります) インスタンスを使用できます。
- EC2 インスタンスタイプ - フリート作成時に選択された Amazon EC2 の [インスタンスタイプ](#) です。
- インスタンスロール ARN - フリートの作成中に他の AWS リソースが提供された場合に、そのリソースへのアクセスを管理する AWS IAM ロール。
- インスタンスロール認証情報プロバイダー - フリートの作成中に他の AWS リソースが提供された場合に、そのリソースへのアクセスを管理する AWS IAM ロール。
- メトリクスグループ名 - 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が有効になっているかどうかを示します。これにより、スケールダウンイベント中にアクティブなゲームセッションが早期に終了するのを防ぎます。
- 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 - プレイヤーが作成するセッション数を制限するために使用される時間の長さ。

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 Servers でメトリクスを使用する方法の詳細については、「[Amazon CloudWatch で Amazon GameLift Servers を監視する](#)」を参照してください。

Events

[イベント] 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 Servers API リファレンスの「[イベント](#)」を参照してください。

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。容量制限の更新と自動スケーリング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、フリートのロ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 Servers を使用したゲームホスティング容量のスケーリング](#)」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 - そ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。

自動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。AWS SDK または CLI を使用して定義する必要があるフリートのルールベースのポリシー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 Servers でのフリート容量の自動スケーリング](#)」を参照してください。

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。

ロケーション

[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。

- 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。
- 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

[ゲームセッション] 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、「[Amazon GameLift Servers コンソールでのゲームセッションとプレイヤーセッション](#)」を参照してください。

Amazon GameLift Servers コンソールでのゲームセッションとプレイヤーセッション

Amazon GameLift Servers コンソールを使用して、ゲームセッションとプレイヤーセッションを操作できます。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、「[Amazon GameLift Servers とプレイヤーエクスペリエンス](#)」を参照してください。Amazon GameLift Servers コンソールには、ゲームセッションの問題を調査するのに役立つ情報とツールが用意されています。

できること:

- 特定のフリートでホストされているゲームセッションとプレイヤーセッションのアクティビティを調べます。
- 複数のフリートにまたがって、特定のプレイヤーのゲームセッションアクティビティを検索します。
- 特定のゲームセッションをシャットダウンします。

ゲームセッションの詳細を表示します。

ゲームセッションとプレイヤーセッションのデータは、ゲームセッションをホストするフリートによって整理されます。

ゲームセッションとプレイヤーセッションの情報にアクセスするには

1. [Amazon GameLift Servers コンソール](#)の左のナビゲーションペインを開きます。ホスティングソリューションタイプを選択し、フリートページを開きます。例えば、次のようになります。
 - ホスティング、Anywhere、フリート
 - ホスティング、マネージド EC2、フリート
 - ホスティング、マネージドコンテナ、フリート
2. 各フリートページには、現在選択されているフリートのリストが表示されます AWS リージョン。ゲームセッションデータを表示するフリートを選択します。
3. 選択したフリートの詳細ページで、[ゲームセッション] タブを開きます。このタブには、フリートにホストされていたすべてのゲームセッションと概要情報が表示されます。必要に応じて、[環境設定] ツールを使用してテーブルの内容を調整できます (テーブルの右上隅にある  アイコンを参照)。カスタム設定は AWS アカウントユーザーに保存され、このページを表示するたびに自動的に適用されます。
4. リストからゲームセッションを選択して、追加情報を表示します。

5. ゲームセッションにプレイヤーセッションデータが含まれている場合は、プレイヤーセッションの表示を選択して、ゲームセッション ID が自動的に入力されたプレイヤーセッション検索ツールを開きます。

Game sessions の詳細には、以下の情報が含まれます。

- 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 Servers 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 Servers 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 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。
- 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 Servers 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

文字列形式のゲームプロパティデータ。開始時にゲームセッションに送信します。

ゲームのプロパティ

キーと値のペアとしてフォーマットされたゲームプロパティデータ。開始時にゲームセッションに送信します。

マッチメイキングデータ

ゲームセッションが FlexMatch で作成された場合、マッチメイキングデータにはマッチメイキング設定とルールセットに関する情報が記載されます。これには、各マッチのプレイヤー属性とチーム割り当てが含まれます。データは JSON 形式です。FlexMatch マッチメイ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。

Look up player session data

ゲームホスティングソリューションがプレイヤーセッションを使用し、一意のプレイヤー ID を提供している場合、複数のフリートにわたり、過去または現在のゲームセッションにおけるプレイヤー固有のアクティビティを調べることができます。次のいずれかの方法で Player セッションルックアップツールを開きます。

- Amazon GameLift Servers コンソールで、左側のナビゲーションペインを開き、Player session lookupを選択し、使用するフィルタータイプを選択します。
- フリートのゲームセッションの詳細を表示するときは、View player sessionsを選択します。ルックアップツールが開くと、ゲームセッション ID フィルターが事前に選択され、ゲームセッションの値が入力された状態になっています。

ルックアップツールを使用する場合は、次の情報を指定できます。

- 特定のプレイヤーセッションに関する情報を取得するためのプレイヤーセッション ID。
- リクエストされたゲームセッションのすべてのプレイヤーセッションに関する情報を取得するためのゲームセッション ID。結果は、スロットを予約したか、ゲームセッションに接続したすべてのプレイヤーを表します。プレイヤーセッションのステータスで結果をフィルタリングすることもできます。
- リクエストされたプレイヤーのすべてのプレイヤーセッションに関する情報を取得するためのプレイヤー ID。結果は、プレイヤーが参加したすべてのゲームセッションを表します。

Note

ルックアップツールは、現在選択されている 全体のすべてのプレイヤーセッションアクティビティを検索します AWS リージョン。リージョンに複数のフリートがある場合、結果にはすべてのフリートのプレイヤーセッションアクティビティが含まれます。マルチロケーションフリートの場合、結果にはフリートのリモートロケーションにおけるプレイヤーセッションのアクティビティも含まれます。

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。
- 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。
- ゲームセッション ID - ゲームセッションに割り当てられた識別子。
- フリート ID - ゲームセッションをホストしたフリートに割り当てられた識別子。
- 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。
 - 完了済み - 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 - プレイ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接続した時間です。
- 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。
- 接続データ - プレイヤーがゲームセッションへの接続に使用した IP アドレス、DNS 名、ポート。
- 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。

Shut down a game session

Amazon GameLift Servers コンソールを使用して、特定のゲームセッションをシャットダウンします。この機能を使用すると、ゲームセッションを特定し、終了シグナルを送信するためのシンプルで迅速な方法が得られます。別の終了方法では、ゲームセッションが実行されているフリートインスタンスを特定し、そのインスタンスにリモートアクセスして、ゲームセッションを手動でシャットダウンする必要があります。

ゲームセッションは、どのような理由でもシャットダウンできます。最も一般的な理由は、自然にシャットダウンできないゲームセッションを終了させるためです。その結果、ゲームセッションのホスティングリソースが解放されず、新しいゲームセッションをホストできないため、フリートのホスティング容量が低下します。

Note

この機能は、ホスティングソリューションの特定の構成設定に依存します。この機能には次の制約事項があります。

- ゲームセッションは、サーバー SDK for Amazon GameLift Servers v5 以降を使用してビルドされたゲームサーバーを実行しているフリート上でホストする必要があります。ゲームサーバーが古いバージョンでデプロイされている場合は、リモートアクセスを使用してゲームセッションを削除する必要があります。
- ゲームセッションが Anywhere フリートでホストされている場合、フリートは Amazon GameLift Servers エージェントを使用してゲームサーバープロセスを管理する必要があります。

ゲームセッションを終了するには

1. [Amazon GameLift Servers コンソール](#)の左のナビゲーションペインを開きます。ホスティングソリューションタイプを選択し、フリートページを開きます。例えば、次のようになります。
 - ホスティング、Anywhere、フリート
 - ホスティング、マネージド EC2、フリート
 - ホスティング、マネージドコンテナ、フリート
2. 各フリートページには、現在選択されている のフリートのリストが表示されます AWS リージョン。終了するゲームセッションをホストしているフリートを選択します。
3. 選択したフリートの詳細ページで、[ゲームセッション] タブを開きます。ゲームセッションのリストで、終了するセッションを選択し、Terminateボタンを選択します。
4. [ゲームセッションを終了しますか?] ウィンドウで、正しいゲームセッションをシャットダウンすることを確認し、終了方法を選択します。
 - 通常のゲームセッションのシャットダウン - このオプションは、シャットダウンするためにゲームセッションをホストしているサーバープロセスにシグナルを送信します。ゲームサーバービルドが Amazon GameLift Servers に適切に統合されている場合、サーバープロセスはゲームセッションのシャットダウンシーケンスを開始し、終了Amazon GameLift Serversを通知して停止します。ゲームの設計によっては、シャットダウンシーケンスに、データの保存やアクティブなプレイヤーへの通知など、ゲームセッションを正常に完了するステップが含まれる場合があります。この方法では、ゲームセッションのシャットダウンシーケンスの完了にわずかな遅延が必要になる場合があります。
 - ゲームセッションの即時シャットダウン — このオプションは、ゲームセッションをホストしているサーバープロセスをシャットダウンするシグナルをプロセスマネージャーに送信しま

す。このオプションは、通常のゲームセッションのシャットダウンをバイパスします。サーバープロセスが応答できない場合でも、ゲームセッションを終了できます。

5. ゲームセッションの終了を確認します。[ゲームセッション] コンソールページで、シャットダウンの進行状況を確認できます。ゲームセッションのステータスは「Terminating」に変わり、シャットダウンが完了すると「Terminated」に変わります。

関連トピック

- AWS SDK とを使用してゲームセッションをシャットダウンすることもできます AWS CLI。詳細と例については、Amazon GameLift Servers API リファレンストピック「[TerminateGameSession](#)」を参照してください。
- ゲームサーバーの統合と、サーバープロセスが Amazon GameLift Servers サービスからのシグナルに応答する方法の詳細については、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。

Amazon GameLift Servers マネージドコンテナフリートを更新する

コンテナグループ定義など、マネージドコンテナフリートのほとんどのプロパティを更新できます。更新する設定によっては、フリートの更新によって新しいフリートのデプロイが開始される場合があります。フリートデプロイでは、フリート内のすべてのインスタンスが削除され、新しい設定のインスタンスに置き換えられます。デプロイが必要な設定は次のとおりです。

- コンテナイメージの更新を含むコンテナグループの定義
- 接続ポート範囲とインバウンドアクセス許可
- ログ設定

フリートデプロイのステータスは、[Amazon GameLift Servers コンソール](#)または AWS Command Line Interface (AWS CLI) で追跡して、コンテナフリートを作成できます。

Note

フリートのランタイム環境を更新することはできません。フリートの作成時に、フリートの Amazon マシンイメージ (AMI) は Linux AMI の利用可能な最新バージョンに設定されます。このフリートにデプロイされるすべてのコンテナイメージは、このバージョンと互換性がある必要があります。フリートの AMI を変更したり、新しいバージョンにアップグレードしたりするには、新しいフリートを作成する必要があります。ベストプラクティスとして、ホス

トされているゲームサーバーでの安全で最新のランタイム環境を維持するために、30 日ごとにフリートを置き換えることをお勧めします。ガイダンスについては、「[Amazon GameLift Servers のセキュリティに関するベストプラクティス](#)」を参照してください。

Console

[Amazon GameLift Servers 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、[マネージドコンテナ: フリート] を選択します。マネージドコンテナフリートページで、リストからフリートを選択し、[編集]を選択します。

1. 必要に応じてコンテナフリートの設定を更新します。完了したら、[作成] を選択します。
2. 更新でフリートのデプロイが必要な場合は、次のようにデプロイオプションを指定するように求められます。
 - ゲームセッション保護。アクティブなゲームセッションがあるフリートインスタンスを保護することを選択できます (安全なデプロイ)。この設定では、ゲームセッションが終了するまでフリートインスタンスは置き換えられません。または、ゲームセッションアクティビティ (安全でないデプロイ) に関係なく、フリートインスタンスを置き換えることもできます。安全でないデプロイは、デプロイ時間を短縮するために、開発フェーズとテストフェーズで役立ちます。
 - 最小正常率。フリートのインスタンスの置き換え速度を管理できます。この設定を使用して、デプロイ中に の正常なタスクを最小限に抑えます。値を低くするとデプロイの速度が優先され、値を高くするとデプロイ全体を通じてゲームサーバーの可用性が高く維持されます。
 - デプロイの失敗戦略。デプロイが失敗した場合に実行するアクションを決定します。デプロイの失敗は、更新されたコンテナの一部のステータスチェックが失敗し、異常状態と見なされることを意味します。すべてのフリートインスタンスを以前のデプロイ状態に自動的にロールバックするよう、デプロイを設定できます。あるいは、デバッグ用途のために、障害状態のフリートインスタンスの一部を維持することもできます。

リクエストが成功すると、コンソールにマネージドコンテナフリートのデプロイタブが表示されます。このタブを使用して、各デプロイのステータスを追跡します。フリートの新しいデプロイ

を開始すると、このアクションはフリートに対して現在進行中のデプロイを自動的にキャンセルします。

AWS CLI

AWS CLI でコンテナ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、`update-container-fleet` コマンドを使用します。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「[update-container-fleet](#)」を参照してください。

次の例では、既存のコンテナフリートを次の特性で更新します。

- バージョン 2 を使用するようにゲームサーバーコンテナグループ定義を更新します。
- 安全なデプロイオプションを指定します。

```
{
  "DeploymentConfiguration": {
    "ImpairmentStrategy": "ROLLBACK",
    "MinimumHealthyPercentage": 75,
    "ProtectionStrategy": "WITH_PROTECTION"
  },
  "FleetId": "containerfleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
  "GameServerContainerGroupDefinitionName": "arn:aws:gamelift:us-west-2:111122223333:containergroupdefinition/MyAdventureGameContainerGroup:2"
}
```

コンテナフリートの Amazon GameLift Servers コンテナグループ定義を更新する

個々のコンテナ定義を含め、コンテナグループ定義のほとんどのプロパティを更新できます。コンテナグループ定義にはバージョン番号があります。コンテナグループ定義を更新すると、Amazon GameLift Servers は更新を保存し、定義のバージョン番号を増分します。コンテナフリートを構成するときに、どのバージョンのコンテナグループ定義をデプロイするかを指定できます。

コンテナグループ定義を更新した後、新しいバージョンを新規または既存のコンテナフリートにデプロイできます。

ゲームサーバーコンテナグループ定義を更新する

このトピックでは、Amazon GameLift Servers コンソールまたは AWS CLI ツールを使用してゲームサーバーコンテナグループ定義を更新する方法について説明します。オプション機能の詳細については、「[Amazon GameLift Servers コンテナフリートをカスタマイズする](#)」を参照してください。

コンテナグループ定義を更新するには:

Console

[Amazon GameLift Servers 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、[マネージドコンテナ: グループ定義] を選択します。コンテナグループ定義ページで、更新するコンテナグループ定義とバージョンを選択します。

更新を保存したら、新しいバージョンを使用して新しいコンテナフリートを作成するか、既存のコンテナフリートに更新をデプロイできます。

ステップ 1: コンテナグループ定義の詳細を定義する

- 合計メモリと vCPU の制限設定を更新できます。

ステップ 2: コンテナ定義を追加する

次のコンテナ定義を更新できます。

- 既存のコンテナ定義を更新します。
 - 新しいサポートコンテナ定義を追加します。
 - サポートコンテナ定義を削除します。
1. ECR イメージ URI を更新できます。新しいイメージと一致するように Server SDK バージョン設定を更新してください。
 2. 必要に応じて内部コンテナのポート範囲を更新できます。これらの設定を変更すると、これらの変更がフリートにデプロイされると、コンテナフリートの接続ポート設定に影響する可能性があります。詳細については、[ネットワーク接続を構成する](#)を参照してください。

ステップ 3: 依存関係を設定する

- 必要に応じて依存関係を変更できます。詳細については、「[コンテナの依存関係の設定](#)」を参照してください。

ステップ 3: 確認して作成する

- コンテナグループ定義の更新を確認します。編集を使用して、任意のセクションで追加の変更を加えます。完了したら、作成を選択してコンテナグループ定義の新しいバージョンを生成します。

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。最初、ステータスは `COPYING` です。Amazon GameLift Servers は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を開始します。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 `READY` に変わります。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または更新する場合は、コンテナ定義の設定を別の JSON ファイルに維持します。CLI コマンドでファイルを参照できます。スキーマの例については、「[コンテナ定義 JSON ファイルを作成する](#)」を参照してください。

定義を更新するときは、更新する値を指定するだけで済みます。Amazon GameLift Servers は、更新リクエストに含まれていない値を保持します。コンテナ定義を変更する場合。ただし、コンテナ定義を変更する場合は、完全なセットを指定します。

コンテナグループ定義を更新するには

新しいコンテナグループ定義を更新するには、`update-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、「AWS CLI コマンドリファレンス」の「[update-container-group-definition](#)」を参照してください。

Example: ゲームサーバーコンテナグループ

コンテナグループ定義を取得、更新、または削除するとき、またはコンテナフリートを作成または更新するときに、コンテナグループ定義バージョンを指定できます。各コンテナグループ定義にはバージョンプロパティがあります。さらに、定義の ARN 値はバージョン番号を指定します。

この例は、ゲームサーバーコンテナグループ定義の変更要求を示しています。このグループのコンテナ定義を含む JSON ファイルを作成済みであることを前提としています。この例では、定義名に ARN 値を使用し、更新をバージョン 1 に指定します。

```
aws gamelift update-container-group-definition \  
  --name arn:aws:gamelift:us-west-2:111122223333:containergroupdefinition/  
MyAdventureGameContainerGroup:1 \  
  --operating-system AMAZON_LINUX_2023 \  
  --container-group-type GAME_SERVER \  
  --total-memory-limit-mebibytes 4096 \  
  --total-vcpu-limit 1 \  
  --container-definitions file://SimpleServer.json
```

コンテナグループ定義のクローンを作成する

Amazon GameLift Servers 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 Servers コンソール](#)の左側のナビゲーションペインで、[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、事前入力された設定でコンテナグループ作成ウィザードが開きます。
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。

コンテナフリートの Amazon GameLift Servers コンテナグループ定義を削除する

コンテナグループ定義を削除するには、いくつかのオプションがあります。コンテナグループ定義を削除すると、このアクションはコンテナグループ内のすべてのコンテナ定義も削除します。

コンテナグループ定義には複数のバージョンを含めることができます。コンテナグループバージョンの名前は同じですが、バージョン番号は異なります。コンテナグループ定義 ARN、名前とバージョンの両方を指定します。

コンテナグループ定義を取得、更新、または削除するとき、またはコンテナフリートを作成または更新するときに、コンテナグループ定義バージョンを指定できます。各コンテナグループ定義にはバージョンプロパティがあります。さらに、定義の ARN 値はバージョン番号を指定します。

コンテナグループ定義を削除するには、いくつかの方法があります。

- 特定の定義のすべてのバージョンを削除できます。
- 特定の定義の特定のバージョンを 1 つ削除できます。
- いくつかの最新バージョンを保持し、特定の定義の古いバージョンを削除できます。例えば、バージョン 5 より前のすべてのバージョンを削除できます。

コンテナグループ定義バージョンは、コンテナフリートで使用されていない場合にのみ削除できます。

コンテナグループ定義を削除するには

Console

[Amazon GameLift Servers 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、[マネージドコンテナ: グループ定義] を選択します。コンテナグループ定義ページで、変更する定義を選択し、削除を選択します。

削除のタイプを選択し、削除タイプに応じて追加の設定を指定するように求められます。

AWS CLI

- コンテナグループ定義を削除するには、delete-container-group-definition CLI コマンドを使用して、実行する削除のタイプの値を指定します。この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「[delete-bucket](#)」を参照してください。

この例では、バージョン 5 より前のゲームサーバーコンテナグループ定義のすべてのバージョンを削除するリクエストを示しています。

Example

```
aws gamelift delete-container-group-definition \  
  --name MyAdventureGameContainerGroup \  
  --version-count-to-retain 5 \  
  --arn arn:aws:gamekit:us-east-1:123456789012:container-group-definition/MyAdventureGameContainerGroup/1
```

Amazon GameLift Servers コンソールのエイリアス

エイリアスリソースは、フリートまたはターミナルの送信先へのポインタを表します。プレイヤートラフィックを特定のホスティング先に転送するために使用されます。エイリアスを使用してホスティング先を抽象化することで、ホスティングフリートを更新するときにトラフィックをより容易にリダイレクトできるようになります。

Amazon GameLift Servers コンソールで、または または AWS SDK for を使用して、エイリアスに関する情報を表示します Amazon GameLift Servers。

Console

エイリアスページには、トラフィック固有のホスティング先を誘導する Amazon GameLift Servers エイリアスに関する情報が表示されます。エイリアスを表示するには、ナビゲーションペインで [ホスティング]、[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。[エイリアスを作成] 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。必要に応じて、[環境設定] ツールを使用してテーブルの内容を調整できます (テーブルの右上隅にある



アイコンを参照)。カスタム設定は AWS アカウントユーザーに保存され、このページを表示するたびに自動的に適用されます。

- 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。[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。[削除] を選択します。

エイリアスの詳細情報を次に示します。

- ID – エイリアスを識別するために使用される一意の数値。
- 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。

AWS SDK

次の AWS CLI コマンドを使用して、このリソースに関する情報を取得します。

- [ListAliases](#)
- [DescribeAlias](#)

エイリアスの編集

Amazon GameLift Servers コンソールを使用するか、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。

このトピックでは、ゲームセッションの配置で使用する Amazon GameLift Servers エイリアスを編集する方法を説明します。以下のプロパティを編集できます。

エイリアスを編集するには

エイリアスは、Amazon GameLift Servers コンソールまたは AWS Command Line Interface (AWS CLI) を使用して編集できます。

Console

[\[Amazon GameLift Servers コンソール\]](#) のナビゲーションペインを使用して、[エイリアス] ページを開きます。

1. 編集するエイリアスを選択したら、[編集] を選択します。編集するエイリアスが表示されない場合は、現在選択されている AWS リージョン を確認してください。
2. [エイリアスの編集] ページで、以下の編集ができます。
 - エイリアス名を変更する。
 - エイリアスの説明を変更する。
 - ルーティング戦略をシンプルからターミナルに、またはターミナルからシンプルに変更する。
 - シンプルルーティング戦略が設定されているエイリアスで、エイリアスが関連付けられているフリート ID を変更する。
 - ターミナルルーティング戦略が設定されているエイリアスで、メッセージテキストを変更する。
3. [変更を保存] をクリックします。シンプルルーティング戦略のエイリアスのフリート ID を更新する場合、変更が完了するまでに最大 2 分かかることがあります。この間、古いフリートで新しいゲームセッション配置が行われる可能性があります。

AWS CLI

[update-alias](#) コマンドを使用して、エイリアスリソースを変更します。現在のデフォルト AWS リージョンでエイリアスリソースを更新するか、`--region` タグを追加して別の AWS リージョンを指定できます。

次のプロパティを変更できます。

- エイリアス名。
- エイリアスの説明。
- トラフィックルーティング戦略のタイプ。新しいルーティング戦略のフリート ID またはメッセージ文字列を必ず指定してください。
- 既存のシンプルルーティング戦略のフリート ID。フリート ID はエイリアスと同じリージョンに存在する必要があります。
- 既存のターミナルルーティング戦略のメッセージ文字列。

Amazon GameLift Servers コンソールのゲームセッションキュー

キューリソースは、ゲームセッション配置コンポーネントの設定を表します。設定の選択に基づいて、Amazon GameLift Servers が新しいゲームセッションをホストするための最適なコンピューティングリソースを検索して選択する方法を決定します。

Amazon GameLift Servers コンソールで、または または AWS SDK for を使用して、ゲームセッションキューに関する情報を表示しますAmazon GameLift Servers。

Console

新しいゲームセッションのリクエストを処理するために使用されるすべてのキューに関する情報を表示できます。キューページには、現在選択されている のゲームセッションキューが表示されます AWS リージョン。[キュー] 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、「[ゲームセッションの配置を設定する](#)」を参照してください。

キューページには、各キューに関する次の概要情報が表示されます。必要に応じて、[環境設定] ツールを使用してテーブルの内容を調整できます (テーブルの右上隅にある



アイコンを参照)。カスタム設定は AWS アカウントユーザーに保存され、このページを表示するたびに自動的に適用されます。

- 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。
- キュー内の送信先 - キュー設定にリストされているフリートの数。Amazon GameLift Servers は、キュー内の任意のフリートに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。[編集] を選択します。
- キューを削除します。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。[削除] を選択します。

**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。

Details

[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 Servers 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、ゲームセッションの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。

イベントの通知

[イベント通知] セクションには、Amazon GameLift Servers 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

[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、成功したプレイメントのリージョン別内訳など、キュー全体のプレイ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 Servers メトリクス](#)」を参照してください。

送信先

[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 Servers は、新しいゲームセッションをホスト可能な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。リストにある最初の送信先に容量がある限り、Amazon GameLift Servers はそこに新しいゲームセッションを配置します。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 Servers に、平均プレーヤレイテンシーが最も低い利用可能な宛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をカスタマイズする](#)」を参照してください。

セッションプレイスメント

ゲームの配置プロセスをカスタマイズする設定。

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。

ロケーション

[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。

AWS SDK

次の AWS CLI コマンドを使用して、このリソースに関する情報を取得します。

- [DescribeGameSessionQueues](#)

Amazon GameLift Servers ホスティングによる起動の準備

次のチェックリストを使用して、ゲームの各デプロイフェーズを検証できます。[重要] とマークされた項目は、本番起動開始に不可欠です。

[Amazon GameLift Servers コンソール](#)から Amazon GameLift Servers ローンチアンケートをダウンロードして記入します。当社は、Amazon GameLift Servers を使用するすべてのゲームデベロッパーがローンチ日をスムーズに進められるようにしたいと考えています。アンケートでリクエストされている情報は、今後の負荷テスト、ソフトローンチ、公開ローンチの準備に役立ちます。最初の負荷テストを実施する少なくとも 3 か月前に、記入済みのアンケートを送信するように計画してください。

トピック

- [ゲームの準備](#)
- [テストの準備](#)
- [ローンチの準備](#)
- [ローンチ後の更新の計画](#)

ゲームの準備

- [重要] ホスティングソリューションの[開発ロードマップのステップ](#)がすべて完了し、統合ゲームサーバー、ゲームクライアント向けのバックエンドサービス、ホスティングフリート、ゲームセッションプレイスメント方法 (キューなど) など、必要なコンポーネントがすべて整っていることを確認します。
- [重要] [ゲームサーバーが実行中に他のリソースにアクセスできるようにする Create AWS Identity and Access Management \(IAM\) ロール](#)。AWS
- [重要] 必要に応じて、他のホスティングリソースへのフェイルオーバーを設計して実装します。
- 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](#)。
- CloudFormation とを使用して、Infrastructure as Code (IaC) を使用して[デプロイを自動化します](#) AWS Cloud Development Kit (AWS CDK)。
- 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](#)。

テストの準備

- [重要] [ライブ環境が本番環境のニーズにスケールアップできるように、Amazon GameLift Servers サービスクォータやその他のクォータの引き上げ](#)をリクエストします。AWS のサービス
- [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。
- 使用しているコンピューティングリソースが、各コンピューティングで同時に実行するサーバープロセスの数をサポートでき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 では、[レイテンシルール](#)を使用して、地理的に近いプレイヤー同士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q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。
- サポート 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、 が問題や停止時に対応 AWS できるようにします。

ローンチの準備

- [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。

ローンチ後の更新の計画

- プレイヤーの使用状況に基づいてアイドル容量を最小限に抑えるため、[スケーリングポリシーを調整します](#)。
- プレイヤーのレイテンシーデータと改訂された要件を基に [FlexMatchルールを変更する](#) か、[ホスティング場所を追加](#)します。
- 各コンピューティングリソースでできるだけ多くのゲームセッションを実行するためにランタイム設定を最適化します。この方法でパフォーマンス効率を最適化すると、同じコンピューティングリソースでより多くのサーバープロセスを実行できるため、フリートコストに直接影響する可能性があります。
- [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。

を使用してAmazon GameLift Serversホスティングリソースを管理する CloudFormation

CloudFormation を使用して Amazon GameLift Serversリソースを管理できます。では CloudFormation、各リソースをモデル化するテンプレートを作成し、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、テンプレートを変更し、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。

CloudFormation を使用してAmazon GameLift Serversホスティングリソースを維持すると、AWS リソースのセットをより効率的に管理できます。バージョニングを使用して、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。

詳細については CloudFormation、[AWS CloudFormation 「ユーザーガイド」](#) を参照してください。Amazon GameLift Servers リソースのテンプレート情報を表示するには、[Amazon GameLift Servers 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

の使用に関する詳細なガイダンスについては CloudFormation、AWS CloudFormation ユーザーガイドの[CloudFormation 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 Servers と特別な関連性があります。

- CloudFormationによりリソースを一貫して管理する。リソースの外部でリソースを変更する CloudFormation と、リソーステンプレートと同期しなくなります。
- CloudFormation スタックと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します。
 - 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します。テンプレートを更新してビルドを置き換える場合、はビルドに接続されているフリートを CloudFormation 置き換えます。CloudFormation その後、は既存のエイリアスを更新して新しいフリートを参照します。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 CloudFormation スタックをデプロイする場合は、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。キューやマッチメイ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。
- マッチメイキング設定は、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します。
- スタック内のフリートごとに個別のエイリアスを作成します。

CloudFormation スタックの使用

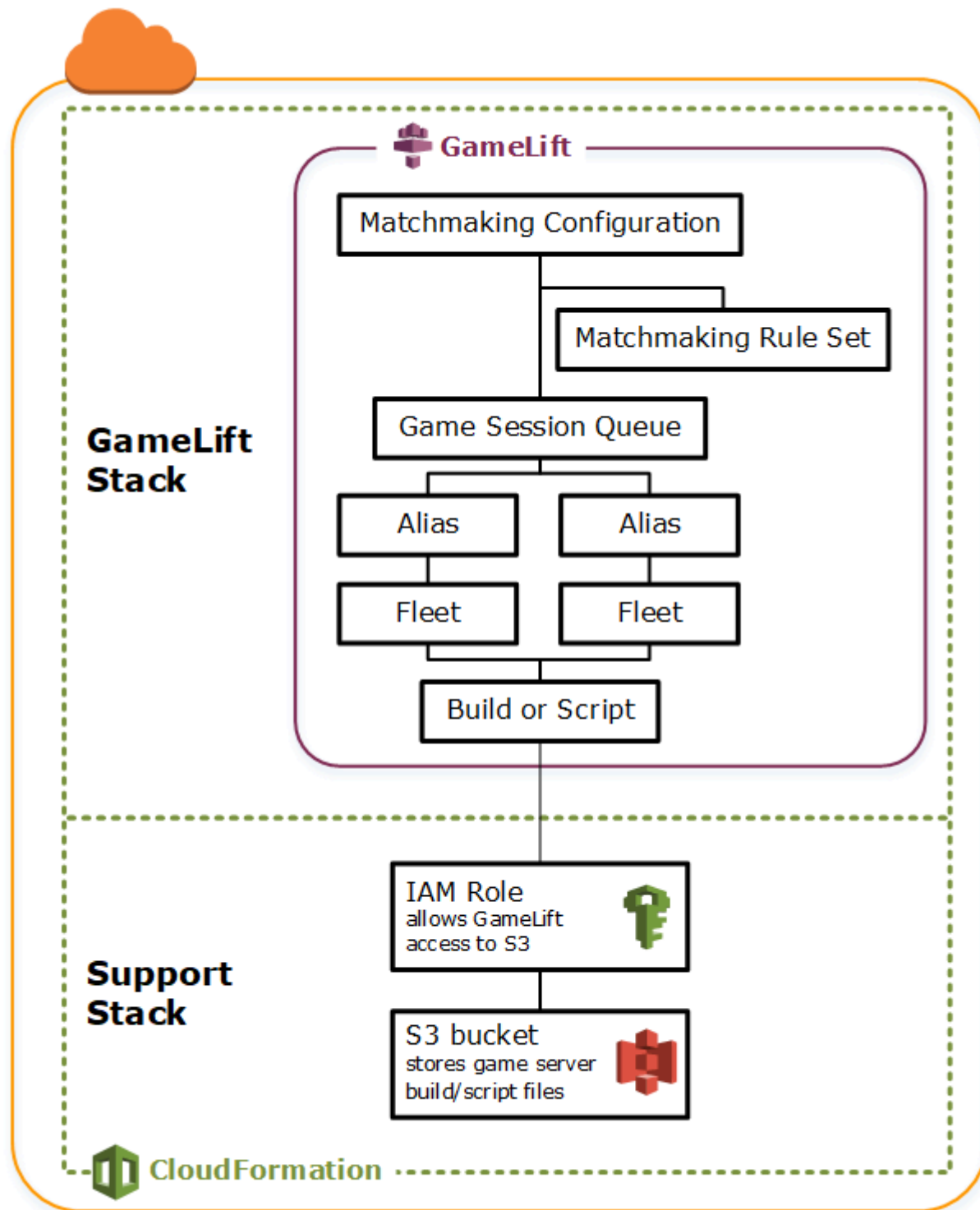
Amazon GameLift Servers リソースの CloudFormation スタックを設定するときは、次の構造を使用することをお勧めします。最適なスタック構造は、ゲームを 1 つのロケーションにデプロイするか、複数のロケーションにデプロイするかによって異なります。

1 つのロケーションのスタック

1 つのロケーションで Amazon GameLift Servers 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 Servers 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。スタックには、Amazon GameLift Serversビルドまたはスクリプトリソースの作成時に S3 バケットからファイルを取得するAmazon GameLift Serversアクセス許可を付与する IAM ロールも含める必要があります。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 リソースが含まれている場合もあります。
- Amazon GameLift Servers スタック - このスタックには、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を含むすべてのAmazon GameLift Serversリソースが含まれます。は、S3 バケットの場所に保存されたファイルを含むビルドまたはスクリプトリソース CloudFormation を作成し、ビルドまたはスクリプトを 1 つ以上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。マッチメイキングに FlexMatch を使用する場合、このスタックにはマッチメイキング設定とルールセットも含まれます。

次の図は、単一の AWS リージョンにリソースをデプロイするための 2 スタック構造を示しています。



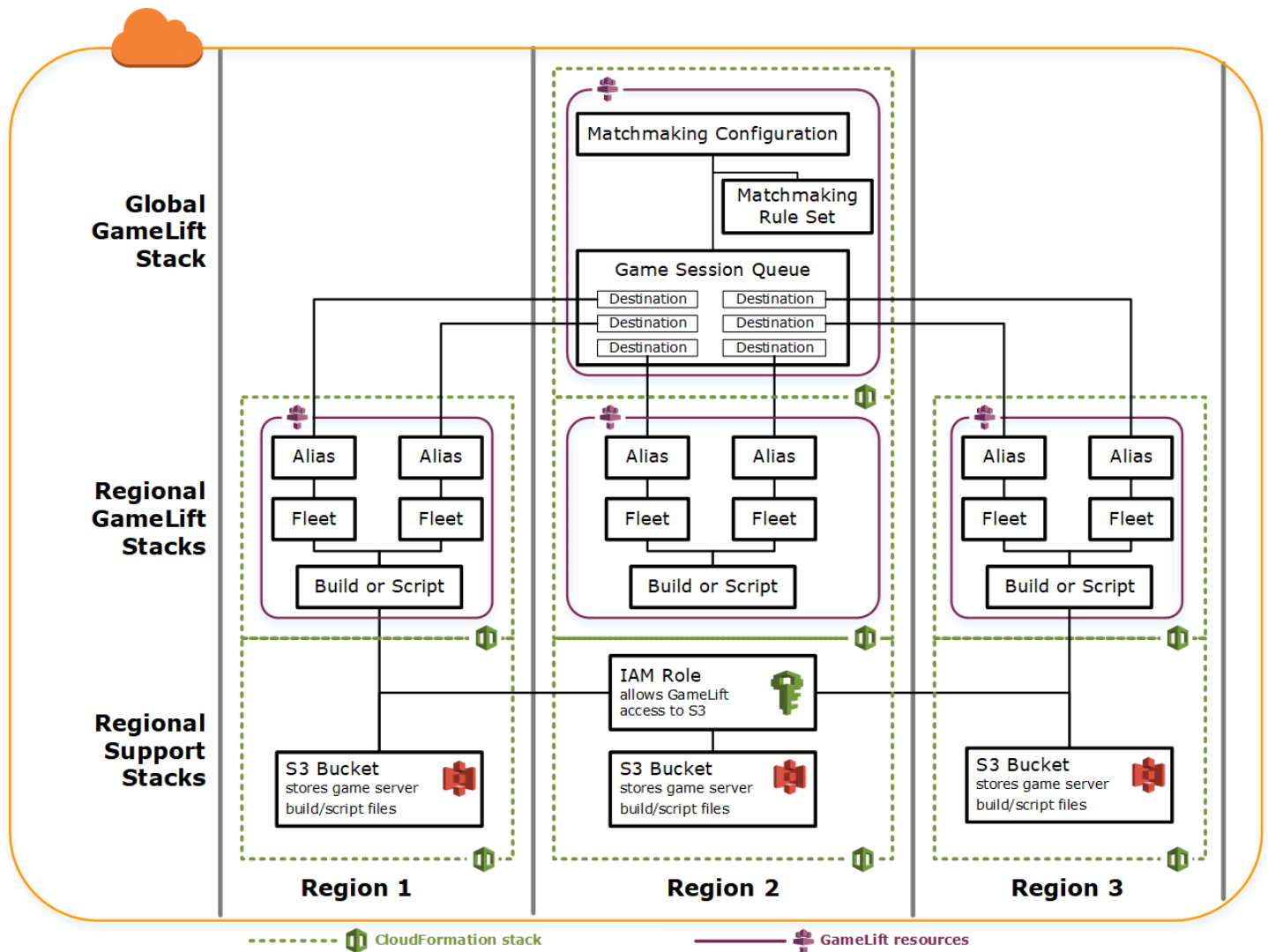
リージョンが複数の場合のスタック

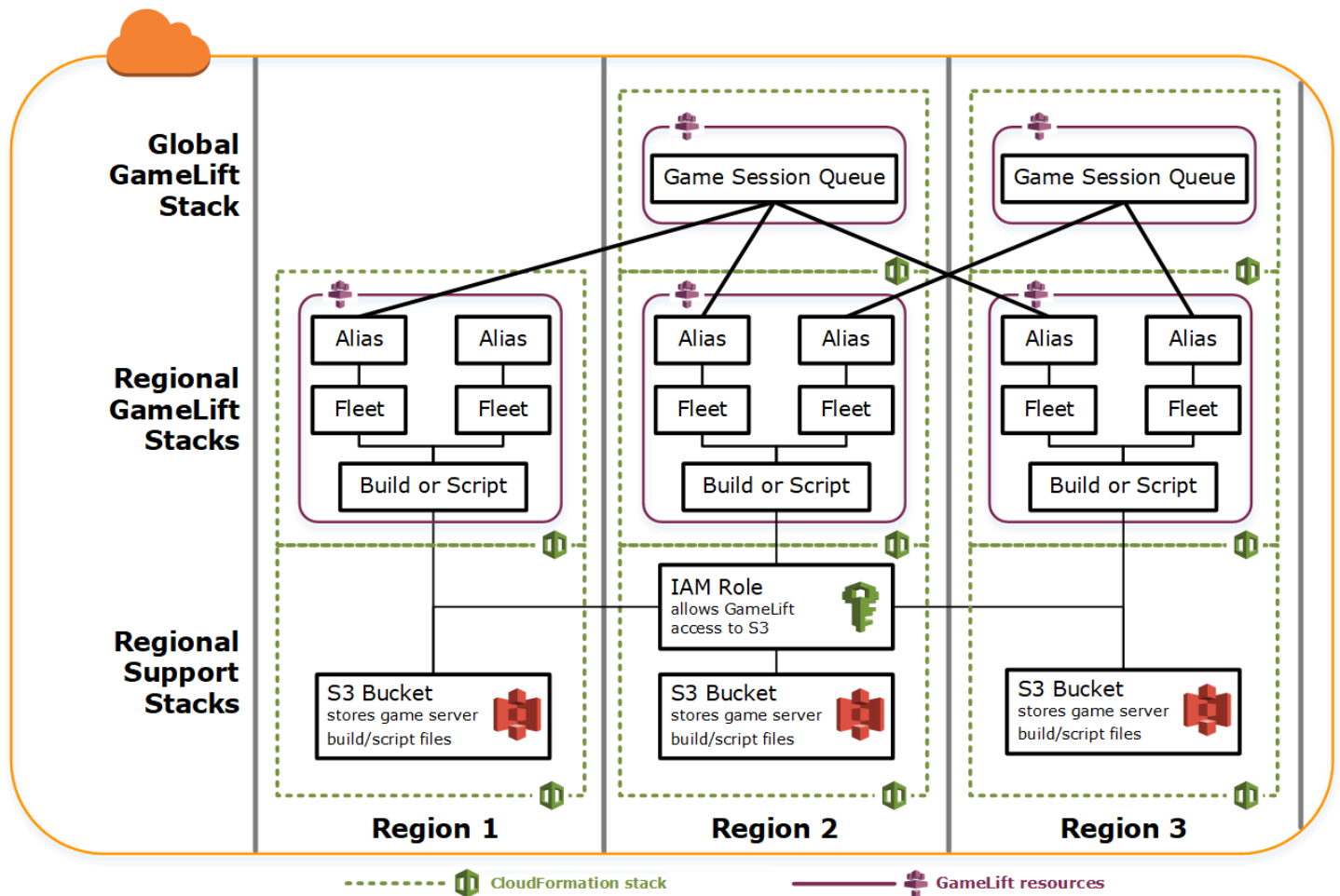
ゲームを複数のリージョンにデプロイする場合、リソースがリージョン間でどのようにやり取りするかに留意してください。Amazon GameLift Servers フリートなどの一部のリソースは同じリージョン内の他のリソースのみを参照できます。Amazon GameLift Servers キューなどの他のリソースは、

リージョンに依存しません。複数のリージョンで Amazon GameLift Servers 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 Servers 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。また、DynamoDB テーブル、Amazon Redshift クラスター、Lambda 関数など、ゲームの他の AWS リソースが含まれている場合もあります。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 Servers 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 Amazon GameLift Servers スタック - このスタックには、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、ゲームがデプロイされている各リージョンに存在する必要がある Amazon GameLift Servers リソースが含まれています。は、S3 バケットの場所にファイルを含むビルドまたはスクリプトリソース CloudFormation を作成し、ビルドまたはスクリプトを 1 つ以上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 Servers 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。

以下の図は、複数の AWS リージョンにリソースをデプロイするための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。





ビルドの更新

Amazon GameLift Servers ビルドは不変です。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場合、デフォルトの CloudFormation 動作では、これらのステップを順番に自動的に実行します。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 の場所を指すように 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 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 Servers 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。自動スケーリングは、プレイヤーのトラフィックの変動に応じ、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、後で自動スケーリングを有効にすることもできます。

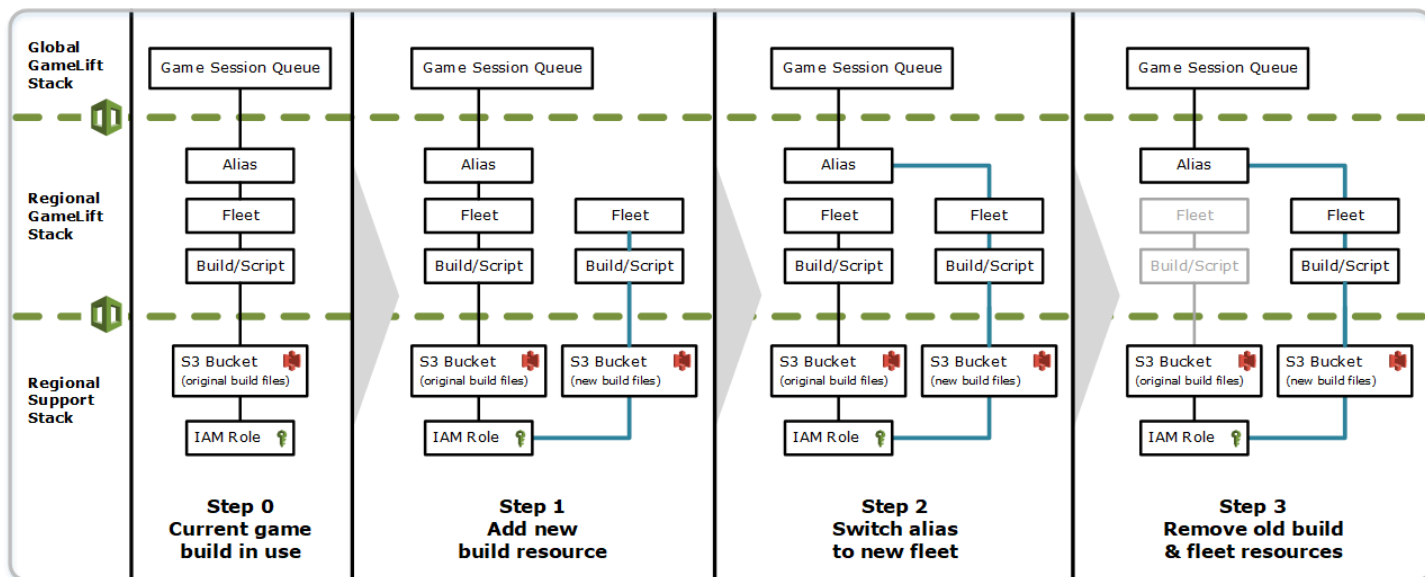
リソースを削除する代わりに `CloudFormation` 保持することもできます。詳細については、AWS CloudFormation API リファレンスの「[RetainResources](#)」を参照してください。

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 Servers 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、2 番目のビルドリソースと対応するフリート CloudFormation を作成します。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、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 Servers 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。

による DDoS 保護 Amazon GameLift Servers

Amazon GameLift Servers プレイヤーゲートウェイは、でホストされているゲームクライアントとゲームサーバー間で UDP トラフィックをルーティングするリレーベースのネットワークです Amazon GameLift Servers。プレイヤーゲートウェイは、ゲームサーバーに到達する前にトラフィックを検証し、プレイヤートラフィックをレート制限し、ゲームサーバーの IP アドレスをパブリックから非表示にして、リレーエンドポイントが異常になったときに更新されたエンドポイントを提供することで、プロアクティブ DDoS 保護を提供します。この機能は、わずかなレイテンシーでボリウム攻撃からゲームサーバーを保護するのに役立ちます。

プレイヤーゲートウェイでは、ゲームサーバーを Linux ベースの Amazon GameLift Servers Managed EC2 フリートまたはコンテナフリートで実行する必要があります。フリートの作成時にプレイヤーゲートウェイを有効にし、この機能を使用するにはゲームクライアントとゲームバックエンドを更新する必要があります。

主な利点

Player Gateway には、次のセキュリティと運用上の利点があります。

- ゲームサーバーの IP アドレスを非表示にする – ゲームクライアントは、ゲームサーバーに直接接続するのではなくリレーエンドポイントを介して接続し、ゲームサーバーアドレスをパブリックから非表示にします。
- トラフィックの検証 – プレイヤーゲートウェイ経由のすべてのトラフィックにはプレイヤーゲートウェイトークンが必要で、認証されたプレイヤーからのトラフィックのみがゲームサーバーに到達できるようにします。
- 動的エンドポイントの置き換え – リレーエンドポイントが異常になると、はエンドポイント Amazon GameLift Serversをすばやく置き換え、次の呼び出しで更新された正常なエンドポイントを提供してプレイヤー接続の詳細を更新します。
- プレイヤートラフィックを分散する – リレーエンドポイントはプレイヤーによって異なり、同じゲームセッション内の他のプレイヤーに対する単一の異常なリレーエンドポイントの影響を軽減します。

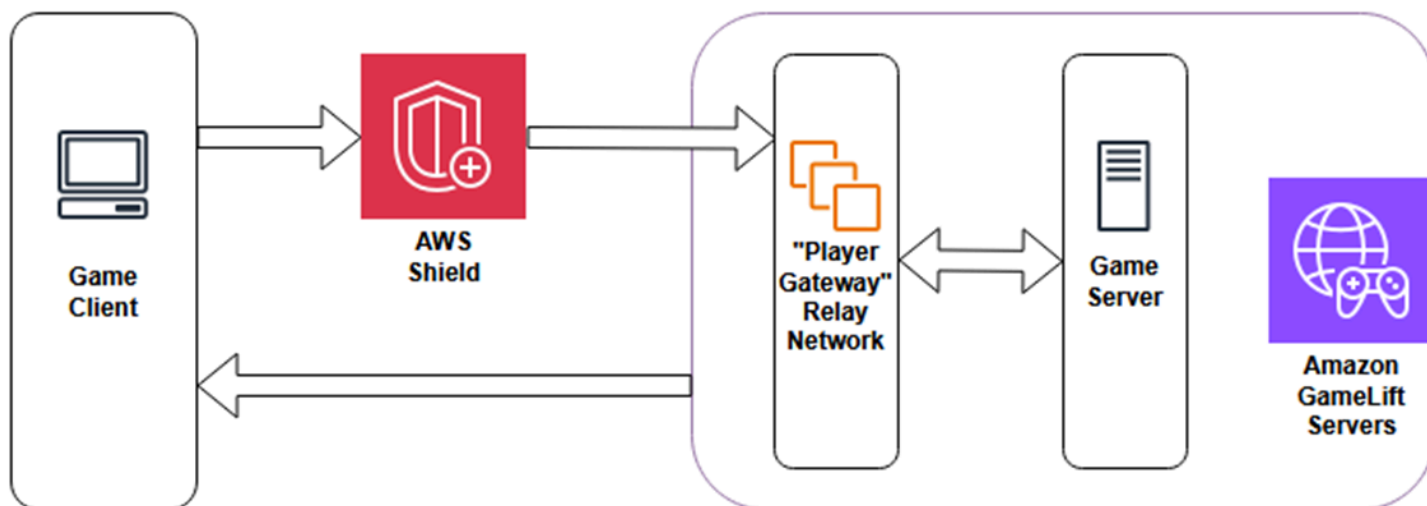
プレイヤーゲートウェイの仕組み

プレイヤーゲートウェイはリレーネットワークを使用して、ゲームクライアントとゲームサーバー間で UDP トラフィックをルーティングします。これにより、ゲームサーバーに到達する前にトラ

フィックを検証し、プレイヤートラフィックをレート制限し、ゲームサーバーの IP アドレスをパブリックから非表示にし、リレーエンドポイントが異常になったときに更新されたエンドポイントを提供することで、DDoS 攻撃から保護されます。

トラフィックフロー

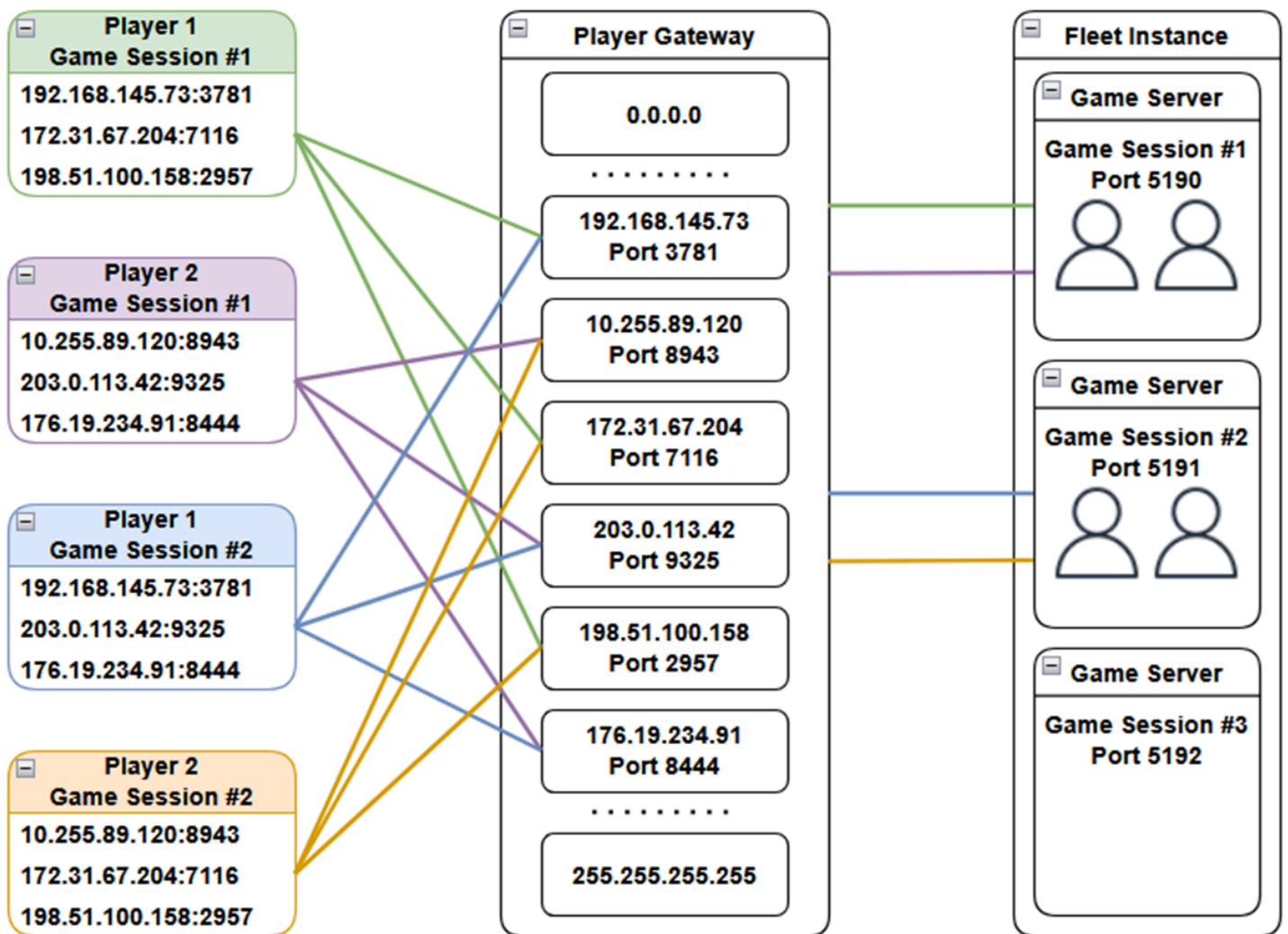
プレイヤーがゲームセッションに接続すると、ゲームバックエンドは `GetPlayerConnectionDetails` API からリレーエンドポイントとプレイヤーゲートウェイトークンを取得し、ゲームクライアントに送信します。ゲームクライアントは、UDP パケットにプレイヤーゲートウェイトークンを付加し、パケットをリレーエンドポイントに送信します。リレーネットワークはトークンを検証し、正当なトラフィックをゲームサーバーにルーティングします。配信前に、リレーネットワークはプレイヤーゲートウェイトークンを削除して、ゲームサーバーが raw ゲームクライアントペイロードを受信し、コードの変更を必要としないようにします。ゲームサーバーからゲームクライアントへの通信は、リレーネットワークを介して変更なしで返されます。



重要な概念

リレーエンドポイント

リレーエンドポイントは、ゲームクライアントがプレイヤーゲートウェイを介してトラフィックを送信するために使用する IP アドレスとポートの組み合わせです。各プレイヤーは、トラフィックを分散し、同じゲームセッション内の他のプレイヤーへの攻撃の影響を軽減するために、プレイヤー間で異なる複数のエンドポイントを受け取ります。



プレイヤーゲートウェイトークン

プレイヤーゲートウェイトークンは、プレイヤーがプレイヤーゲートウェイを介してゲームセッションにトラフィックを送信することを許可する暗号化されたバイト配列です。GetPlayerConnectionDetails API は、base64 でエンコードされた文字列としてトークンを返します。ゲームクライアントは、すべての UDP パケットにプレイヤーゲートウェイトークンを付加する必要があります。リレーネットワークはトークンを検証し、トークンが無効または欠落しているパケットを拒否します。

⚠ Important

プレイヤーゲートウェイトークンは暗号化してはならず、ゲームクライアントによって送信される各 UDP パケットの先頭に表示される必要があります。ゲームがペイロードを暗号化

する場合は、ゲームデータを暗号化した後に、変更されていないプレイヤーゲートウェイトークンを付加します。

プレイヤーゲートウェイトークンは少なくとも 3 分間有効です。リレーエンドポイントが異常になったときにプレイヤーが更新されたエンドポイントを受信できるように、トークンを 60 秒ごとに更新することをお勧めします。

GetPlayerConnectionDetails API

ゲームバックエンドは GetPlayerConnectionDetails API を呼び出して、ゲームセッションのプレイヤーの接続の詳細を取得します。API は、リレーエンドポイントとプレイヤーゲートウェイトークンを返すか、直接接続のためにゲームサーバーの IP アドレスとポートにフォールバックします。ゲームクライアントは、両方の接続タイプを処理するように設計されている必要があります。リレーエンドポイントが異常になったときに更新されたエンドポイントを受信するには、ゲームセッション全体でこの API を定期的に呼び出します (60 秒ごとに推奨)。

詳細については、「[GetPlayerConnectionDetails API](#)」を参照してください。

プレイヤーゲートウェイのモニタリング

Player Gateway は Amazon CloudWatch にメトリクスを発行して、ネットワークトラフィックパターンのモニタリング、潜在的な DDoS 攻撃の特定、リレーパフォーマンスの追跡を支援します。メトリクスには、パケットとバイトの入出力、スロットリングされたトラフィック、プレイヤーセッションが含まれます。プレイヤーゲートウェイメトリクスの完全なリストについては、「」を参照してください[DDoS 保護 \(プレイヤーゲートウェイ\) メトリクス](#)。

IPv4 と IPv6 の互換性

ゲームクライアントは IPv4 を使用して通信します。プレイヤーゲートウェイは IPv6 を使用してゲームサーバーと通信します。は、フリート設定に基づいて IPv4 と IPv6 間の変換 Amazon GameLift Servers を自動的に処理します。

フリートでプレイヤーゲートウェイを設定する方法の詳細については、「[CreateFleet](#)」または「[CreateContainerFleet](#)」を参照してください。IPv4 および IPv6 のサポートの詳細については、「」を参照してください[IPv4 と IPv6 の互換性](#)。

フリートでプレイヤーゲートウェイを有効にする

フリートの作成時にプレイヤーゲートウェイを有効にして、受信 UDP トラフィックを検証してゲームサーバーにレート制限するリレーエンドポイントを介してゲームトラフィックをルーティングします。

前提条件

プレイヤーゲートウェイを有効にする前に、次の要件を確認してください。

- サポートされているロケーション – プレイヤーゲートウェイをサポートするロケーションのリスト AWS については、「」を参照してください [Amazon GameLift Servers サービスロケーション](#)。
- フリートタイプ – プレイヤーゲートウェイは、Linux ベースの Managed EC2 フリートと Managed Container フリートをサポートします。
- Server SDK – プレイヤーゲートウェイには Amazon GameLift Servers Server SDK 5.0 以降が必要です。

プレイヤーゲートウェイモード

PlayerGatewayMode プロパティを使用して、フリートのプレイヤーゲートウェイの互換性を設定します。

DISABLED (デフォルト)

フリートはプレイヤーゲートウェイと互換性がありません。

有効

フリートは、サポートされている場所でプレイヤーゲートウェイと互換性があります。この機能をサポートしていないフリートの場所は引き続きフリートに追加できますが、プレイヤーゲートウェイを利用せず、ゲームサーバーの IP アドレスを直接利用します。

REQUIRED

フリートはプレイヤーゲートウェイと互換性があります。この機能をサポートしていないフリートロケーションをフリートに追加することはできません。

IPv4 と IPv6 の互換性

ゲームクライアントは IPv4 経由でプレイヤーゲートウェイと通信する必要があります。プレイヤーゲートウェイリレーネットワークは、IPv6 を使用してゲームサーバーと通信します。マネージドコンテナフリートは、IPv6 トラフィックを IPv4 に自動的に変換します。マネージド EC2 フリートの場合、の `GameServerIpProtocolSupported` プロパティを使用して、フリートが IPv6 トラフィックを処理する方法を設定します `PlayerGatewayConfiguration`。

IPv4 (デフォルト)

ゲームサーバーは受信 IPv4 トラフィックのみを受け入れます。軽量 IP 変換ソフトウェアがインスタンスにインストールされて実行され、受信 IPv6 トラフィックを受信して IPv4 に変換します。

デュアルスタック

ゲームサーバーは受信 IPv6 トラフィックと互換性があります。IP 翻訳ソフトウェアはインスタンスにインストールまたは実行されません。IPv6 をネイティブにサポートするゲームサーバーは、パフォーマンスがわずかに向上する可能性があります。

プレイヤーゲートウェイをゲームに統合する

プレイヤーゲートウェイ対応フリートを作成したら、ゲームクライアントとゲームバックエンドを更新してプレイヤーゲートウェイと統合する必要があります。ゲームバックエンドはリレーエンドポイントとプレイヤーゲートウェイトークンを取得し、ゲームクライアントに提供します。ゲームクライアントは、すべての UDP トラフィックをこれらのリレーエンドポイントに送信します。

統合は次の呼び出しパスに従います。

1. ゲームバックエンドは `GetPlayerConnectionDetails` API を呼び出して、ゲームセッションの各プレイヤーのリレーエンドポイントとプレイヤーゲートウェイトークンを取得します。
2. ゲームバックエンドは、リレーエンドポイントとプレイヤーゲートウェイトークンをゲームクライアントに送信します。
3. ゲームクライアントは、すべての送信 UDP パケットにプレイヤーゲートウェイトークンを附加し、そのパケットをリレーエンドポイントに送信します。
4. リレーネットワークはプレイヤーゲートウェイトークンを検証し、トラフィックをゲームサーバーにルーティングします。

5. ゲームサーバーは、同じリレーパスを介してトラフィックをゲームクライアントに送り返します。

バックエンド統合

ゲームバックエンドは `GetPlayerConnectionDetails` API を呼び出して、プレイヤーのリレーエンドポイントとプレイヤーゲートウェイトークンを取得する必要があります。その後、バックエンドはこの情報をゲームクライアントに提供します。

GetPlayerConnectionDetails API

`GetPlayerConnectionDetails` API は、ゲームセッションのプレイヤーの接続の詳細を返します。

リレーエンドポイントとトークン

プレイヤーゲートウェイが有効で、ロケーションでサポートされている場合、は以下を返します。

- リレーエンドポイント – プレイヤーによって異なる複数のリレーエンドポイント (IP アドレスとポート)
- プレイヤーゲートウェイトークン – クライアントがすべての UDP パケットの前に付加する必要があるトークン (少なくとも 3 分間有効)
- 有効期限 – プレイヤーゲートウェイトークンの有効期限タイムスタンプ

直接接続

プレイヤーゲートウェイがロケーションで有効になっていないか、サポートされていない場合、はゲームサーバーの IP アドレスとポートを返します。

ゲームクライアントは、両方の接続タイプを処理するように設計されている必要があります。

API コールの例:

```
// C++ example using AWS SDK
Aws::GameLift::GameLiftClient client;
Aws::GameLift::Model::GetPlayerConnectionDetailsRequest request;
request.SetGameSessionId(gameSessionId);
request.SetPlayerIds(playerIds); // Vector of player IDs
```

```
auto outcome = client.GetPlayerConnectionDetails(request);
if (outcome.IsSuccess()) {
    auto result = outcome.GetResult();
    auto connectionDetails = result.GetPlayerConnectionDetails();

    // Process each player's connection details
    for (const auto& detail : connectionDetails) {
        std::string playerId = detail.GetPlayerId();

        // Get relay endpoints (IP address and port)
        auto endpoints = detail.GetEndpoints();
        for (const auto& endpoint : endpoints) {
            std::string ipAddress = endpoint.GetIpAddress();
            int port = endpoint.GetPort();
        }

        // Get player gateway token
        auto token = detail.GetPlayerGatewayToken();

        // Get expiration time
        auto expiration = detail.GetExpiration();

        // Send endpoints and token to game client
    }
}
```

Amazon GameLift Servers API リファレンスの[GetPlayerConnectionDetails](#)」を参照してください。

リレーエンドポイントを更新する

リレーエンドポイントが異常になったときにGetPlayerConnectionDetails、定期的に を呼び出して更新されたエンドポイントをプレイヤーに提供します。ゲームセッションのすべてのプレイヤーに対して 60 秒ごとに API を呼び出すことをお勧めします。API の使用を最適化するには、1 回の API コールで複数のプレイヤー IDs をバッチ処理します。

Important

定期的な更新呼び出しは、更新されたエンドポイントをプレイヤーに配信するための主要なメカニズムです。プレイヤーゲートウェイトークンは少なくとも 3 分間有効ですが、60 秒ごとに更新すると、プレイヤーは定期的に更新されたエンドポイントを受信します。

クライアント統合

ゲームクライアントは、すべての送信 UDP パケットにプレイヤーゲートウェイトークンを付加し、提供されたリレーエンドポイントにパケットを送信する必要があります。Amazon GameLift Servers クライアント SDK for C++ および Unreal Engine は、この統合を簡素化するためのユーティリティを提供します。

クライアント要件

プレイヤーゲートウェイを介してトラフィックをルーティングするには、ゲームクライアントが以下を行う必要があります。

- プレイヤーゲートウェイトークンを準備 – すべての送信 UDP パケットにプレイヤーゲートウェイトークンを準備します。プレイヤーゲートウェイトークンは暗号化してはならず、各パケットの先頭に表示される必要があります。有効なプレイヤーゲートウェイトークンがないパケットは削除されます。
- リレーエンドポイントに送信する – すべての UDP パケットを指定されたリレーエンドポイントに送信します。
- 接続アクティビティを維持する – ゲームクライアントがゲームサーバーにパケットを送信するか、ゲームサーバーが少なくとも 30 秒に 1 回、プレイヤーにパケットを送信することを確認します。このアクティビティは、リレーネットワークを介した接続を維持します。
- エンドポイントの更新を処理する – 更新されたリレーエンドポイントとプレイヤーゲートウェイトークンをバックエンドから受け入れ (60 秒ごとに推奨)、接続を中断せずに新しいエンドポイントに移行します。

クライアント SDK

Amazon GameLift Servers クライアント SDK for C++ および Unreal Engine は、プレイヤーゲートウェイの統合を簡素化するユーティリティを提供します。

- トークン管理 – すべての送信 UDP パケットにプレイヤーゲートウェイトークンを付加します。
- エンドポイントの選択 — 設定可能なアルゴリズムを使用してトラフィックをリレーエンドポイントにルーティングします。
- エンドポイントの更新 – 定期的なコールバックをスケジュールして、更新されたリレーエンドポイントとプレイヤーゲートウェイトークンをバックエンドから取得します。

エンドポイント選択アルゴリズム

クライアント SDK には、使用するリレーエンドポイントを選択するための 2 つの組み込みアルゴリズムが用意されています。

フォールバックアルゴリズム

1 つのエンドポイントが異常になるまで使用し、別のエンドポイントに切り替えます。短時間の中断が許容されるメニュー、ロビー、ターンベースのゲームに最適です。エンドポイントのフェイルオーバー中に、設定されたタイムアウト期間 (デフォルト: 2 秒) パケットが失われる可能性があります。

予測ローテーションアルゴリズム

利用可能なすべてのエンドポイントを継続的にローテーションし、障害が発生する前に障害を予測します。一貫したパケット配信が重要なファーストパーソンシューティングゲームやレースゲームなどのリアルタイムのゲームプレイに最適です。ゲームサーバーは、一貫した頻度でメッセージを送信する必要があります。

SDK のベースアルゴリズムクラスを拡張することで、カスタムアルゴリズムを実装することもできます。

クライアント SDK リソース

完全な統合手順とサンプルコードについては、次のリソースを参照してください。

C++

クライアント SDK for C++ は [Amazon GameLift Servers](#)、[クライアント SDK for C++](#) で利用できます。

Unreal Engine

Unreal Engine 用プラグインは、[Amazon GameLift Serversクライアント SDK for Unreal Engine](#) で利用できます。アクセスには、GitHub の Epic Games 組織にメンバーシップが必要です。詳細については、[GitHub の「Unreal Engine」](#) を参照してください。

統合をテストする

フリートにデプロイする前に、プレイヤーゲートウェイテストツールを使用してプレイヤーゲートウェイの統合をローカルでテストします。このツールは、プレイヤーゲートウェイ UDP プロキシイ

ンフラストラクチャをシミュレートし、ゲームクライアントがプレイヤーゲートウェイトークンの先頭を正しく付加し、複数のエンドポイントを介してトラフィックをルーティングし、ネットワークの低下を処理することを確認するのに役立ちます。

セットアップと使用方法については、GitHub の [プレイヤーゲートウェイテストアプリ](#) を参照してください。

ベストプラクティス

プレイヤーゲートウェイを統合するときは、次のベストプラクティスに従ってください。

- 60 秒ごとにエンドポイントを更新する – 60 秒 `GetPlayerConnectionDetails` ごとに を呼び出して、プレイヤーが定期的に更新されたエンドポイントを受信できるようにします。
- バッチ API コール – 同じゲームセッションで複数のプレイヤー `GetPlayerConnectionDetails` に対して を呼び出す場合は、呼び出しを一括して API オーバーヘッドを減らします。
- 接続アクティビティを維持する – 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が少なくとも 30 秒に 1 回流れることを確認します。これは、client-to-server トラフィックまたは server-to-client トラフィックのいずれかです。
- エンドポイントの更新を適切に処理する – ゲームクライアントがバックエンドから更新されたエンドポイントを受け取ったら、プレイヤーの接続を切断せずに新しいエンドポイントに移行します。
- 適切なアルゴリズムを選択する – ゲームの要件に最も適したエンドポイント選択アルゴリズムを選択します。ターンベースのゲームにはフォールバックアルゴリズムを使用し、リアルタイムゲームには予測ロケーションアルゴリズムを使用します。

テストとトラブルシューティング Amazon GameLift Servers

このセクションでは、Amazon GameLift ホスティングソリューションの問題の診断と解決に関するガイダンスを提供します。開発、デプロイ、または本稼働オペレーション中に問題が発生した場合でも、これらのリソースは根本原因を特定し、効果的なソリューションを実装するのに役立ちます。

- 開発環境とテスト環境 - 本番環境を反映する反復的な開発ワークフローを設定し、開発サイクルの早い段階で統合の問題を検出して解決できるようにします。
- フリートのデプロイと管理 - 一般的なフリート作成の失敗、サーバープロセスのアクティベーションの問題、スケーリングの問題、およびゲームサーバーが正しく実行されない可能性があるデプロイエラーを診断します。
- ランタイム診断とモニタリング - リアルタイムのトラブルシューティング、パフォーマンス分析、設定検証のためにライブフリートインスタンスにアクセスして、プレイヤーエクスペリエンスに影響する問題を解決します。

これらのリソースを Amazon GameLift のモニタリングおよびログ記録機能と一緒に使用して、マルチプレイヤーゲームの最適なパフォーマンスと可用性を維持します。

Amazon GameLift Servers Anywhere を使用した反復型開発の準備

Amazon GameLift Servers には、ゲーム開発時に使用するテスト用のホスト環境の設定に役立つツールとソリューションが用意されています。これらのツールを使用すると、Amazon GameLift Servers によるマネージドホスティングの実際のプレイヤー体験を反映し、迅速かつ反復的な開発プロセスをサポートする環境を作成できます。

別のテスト環境で、テスト中に Amazon GameLift Servers マネージドフリートのオーバーヘッドを削除します。新しいゲームサーバービルドのイテレーションをアップロードしたり、新しいフリートを作成したり、アクティブ化するまで 15 分以上待つたりする必要がなくなります。代わりに、新しいビルドを作成し、新しいビルドでテストフリートをすばやく更新し、新しいビルドを起動してテストを開始できます。

Amazon GameLift Servers Anywhere フリートを使用すると、開発ワークステーションなどのローカルデバイスを使用してテスト環境を設定できます。クラウドベースのホスティングリソースを使用してテスト環境を設定することもできます。

Anywhere テスト環境をセットアップして、次のようなさまざまなシナリオを開発してテストします。

- ゲームサーバー統合と Amazon GameLift Servers サーバー SDK をテストします。AWS CLI 呼び出しを使用して新しいゲームセッションを開始し、ゲームセッションイベントを追跡することで、ゲームクライアントが動作しなくてもテストできます。
- ゲーム用コンポーネントを開発する際に、ゲームクライアント、バックエンドサービスそして Amazon GameLift Servers サービス間のインタラクションをテストします。ゲームに参加するためのプレイヤー体験をファインチューニングします。
- FlexMatch マッチメーカー設計を試します。ルールセットのバリエーションやその他のマッチメイキング機能の実装を試します。マッチメイキングバックフィルを設定してテストします。
- ゲームサーバーライフサイクル管理用に、(Amazon GameLift Serversエージェントでの) ランタイム設定など別の Amazon GameLift Servers ホスティング機能を試します。
- 実際のホスト環境で、マルチプレイヤーインタラクションを含むゲームでのプレイヤー体験のあらゆる側面をすばやく構築、テスト、および繰り返して検証します。

後で、ゲームの起動準備をする際に、Amazon GameLift Servers マネージドフリートを追加してホスティング設定をファインチューニングし、以下を含む追加のシナリオをテストします。

- マルチロケーションフリート、スポットフリート、オンデマンドフリート、複数のインスタンスタイプの使用など、ゲームセッションキュー設計を試してテストします。
- オプションのレイテンシーポリシーやフリートの優先順位付け設定の使用など、マネージドフリートでゲームセッション配置オプションを試します。
- 自動または手動スケーリングオプションを使用して、プレイヤーの需要を満たすようにキャパシティスケーリングを設定します。
- Amazon GameLift Servers マネージドフリート AWS CloudFormation を使用して をセットアップし、ホスティングリソースを長期的に管理します。

Fast Build Update Tool (開発専用)

マネージド EC2 フリートの場合、ゲームサーバービルドの更新をデプロイするには、新しいビルドを 1 つずつ Amazon GameLift Servers にアップロードし、ビルドに対して新しいフリートを作成する必要があります。

Fast Build Update Tool を使用すると、開発中にこうした手順を回避できるため、時間を節約し、開発のイテレーションを高速化できます。このツールを使用すると、既存のフリート内のすべてのコンピューティングでゲームビルドファイルをすばやく更新できます。このツールにはいくつかのオプションがあり、ゲームビルド全体を置き換えたり、6 つの特定のファ

イルを変更したり、更新後にゲームサーバープロセスを再起動する方法を管理できます。このツールを使用して、フリート内の個々のコンピューティングを更新することもできます。Fast Build Update Tool を入手し、その使用方法の詳細を確認するには、GitHub にある [Fast Build Update Tool](#) の Amazon GameLift Servers リポジトリを参照してください。

トピック

- [クラウドベースのテスト環境を構築する](#)
- [Amazon GameLift Servers Anywhereでローカルテストを設定する](#)
- [Amazon GameLift Servers Local を使用して統合をテストする](#)

クラウドベースのテスト環境を構築する

Note

このトピックでは、Amazon GameLift Servers バージョン 5.x 用のサーバー SDK と統合されたゲームの反復テストについて説明します。ゲームでサーバー SDK バージョン 4.x 以前を使用している場合は、「[Amazon GameLift Servers Local を使用して統合をテストする](#)」を参照してください。

Amazon GameLift Servers Anywhere フリートを使用して、クラウドベースのホスト環境でゲームコンポーネントを繰り返し構築およびテストします。ホスティングリソースと Amazon GameLift Servers サービスへの接続を使用して Anywhere フリートを作成し、それらの上でゲームサーバーを実行し、必要に応じてゲーム機能をテストします。

Amazon GameLift Servers エージェントを使用して Anywhere フリートをデプロイする

ゲームサーバービルドが Amazon GameLift Servers SDK 5.x 以降と統合されている場合は、Amazon GameLift Servers エージェントを使用してクラウドベースの Anywhere フリートにデプロイできます。エージェントは、フリート内の各コンピューティングのゲームサーバーのライフサイクルやその他のタスクを管理するバックグラウンドプロセスです。これらのタスクには、Anywhere フリートへのコンピューティングの登録、認証トークンの取得、一連の手順に基づくゲームサーバープロセスの起動/停止が含まれます。エージェントは、フリートのランタイム設定によって制御されます。この設定は、フリートの有効期間中いつでも更新できます。(エージェントはマネージド EC2 フリートに自動的にデプロイされます。) 詳細とエージェントのダウンロード方法については、「[Amazon GameLift Servers GitHub リポジトリ](#)」を参照してください。

Amazon EC2 で反復テストを設定する

この [Amazon GameLift Servers ツールキットソリューション](#) のガイド付きワークフローを使用して、Amazon GameLift Servers でのマネージドホスティングエクスペリエンスを反映するクラウドベースのホスティング環境を設定します。

GitHub リポジトリには、Amazon GameLift Servers Anywhere と Amazon GameLift Servers エージェントを使用してテスト環境をセットアップするためのほとんどのプロセスを自動化するスクリプト一式が用意されています。また、新しいゲームサーバービルドをテストするたびに環境を更新するためのガイダンスも提供します。サンプルゲームサーバービルドを使用してテスト環境をデプロイする単一のスクリプトを実行することも、各手順を実行して独自のゲームサーバービルドでセットアップすることもできます。

このワークフローでは、を使用してスクリプトを実行し AWS マネジメントコンソール、コマンドラインタスクを完了 AWS CloudShell することで、で完全に作業します。

Note

このチュートリアルタスクでは、、、Amazon S3 Amazon GameLift Servers AWS CloudShell、、、Amazon EC2 AWS Systems Manager、およびのサービスに対するアクセス許可を持つ AWS アカウントユーザーが必要です AWS Identity and Access Management。Amazon S3 AWS アカウントへの管理者レベルのアクセス権を持つユーザーには、必要なアクセス許可が既に付与されています。

ワークフローでは、次のタスクについて説明します。

- Amazon GameLift Servers のゲームサーバービルドをパッケージ化します。このワークフローには、サンプル C++ ゲームサーバーを構築するためのスクリプトが用意されています。このゲームサーバーは、サーバー SDK for Amazon GameLift Servers バージョン 5.x と既に統合されており、ホスティングの準備ができています。または、統合が完了したら、独自のゲームプロジェクトで作業することもできます。
- ゲームサーバーのビルドと依存関係を保存する Amazon Simple Storage Service バケットを設定します。ゲームビルドの新しいバージョンを作成すると、それらを S3 に保存し、スクリプトを使用してゲームテスト用の Anywhere フリートを更新できます。
- Amazon GameLift Servers エージェントを取得してビルドします。エージェントは、設定に基づいてホスティングリソースのゲームサーバープロセスを管理します。これは、同じロジックを使用し、Amazon GameLift Servers マネージド EC2 ホスティングと同じように動作します。

- ホスティングリソースの Anywhere フリートを設定します。Anywhere フリートの場合、Amazon GameLift Servers サービスを使用して、Amazon GameLift Servers が管理していないリソースをホスティングできます。この手順では、ランタイム設定も設定します。ランタイム設定では、ゲームサーバープロセスを開始するタイミングと方法を Amazon GameLift Servers エージェントに指示します。
- Amazon EC2 インスタンスをセットアップします。これは反復テスト用のテスト環境です。(本番稼働レベルの使用状況に合わせて最適化された) フルマネージド Amazon GameLift Servers インスタンスの代わりに、標準の EC2 インスタンスを使用する方がはるかに高速です。標準の EC2 インスタンスを使用すると、必要に応じてゲームサーバーをすばやく継続的に更新できます。
- ゲームサーバービルドと Amazon GameLift Servers エージェントを Amazon EC2 インスタンスにデプロイします。ワークフローには、ゲームビルドの最新バージョンとすべての依存関係を取得し、EC2 インスタンスにインストールするスクリプトが用意されています。このワークフローでは、依存関係には Amazon GameLift Servers エージェントと CloudWatch エージェントが含まれます。
- Amazon GameLift Servers エージェントを開始します。インストールされると、エージェントが自動的に開始され、手順を実行します。具体的には次のとおりです。
 - Amazon GameLift Servers Anywhere フリートで EC2 インスタンスをコンピューティングとして登録します。
 - WebSocket と Amazon GameLift Servers サービス間の接続を確立し、最新のランタイム設定を取得します。
 - ランタイム設定の手順に基づいて、ゲームサーバープロセスを起動します。このワークフローでは、エージェントはゲームサーバー実行可能ファイルの単一のプロセスを開始するように指示されます。
- ゲームシナリオをテストします。テスト環境をセットアップし、最新のゲームサーバービルドをインストールしたら、テストを開始できます。このワークフローでは、ゲームセッションの開始など、テストに関するいくつかの手順について説明します。CloudWatch ゲームサーバーログにアクセスして、ゲームセッションの起動とプレイヤーの受け入れ準備ができたときの進行状況を追跡します。

ゲームクライアントやクライアント側のバックエンドサービスなど、ゲームコンポーネントを開発する際に、これらをテストシナリオに含めることができます。ゲームクライアントを使用してゲームセッションをリクエストし、Amazon GameLift Servers サービスから接続情報を取得し、ゲームセッションに直接接続します。

- 新しいゲームサーバーのビルドと繰り返しテストをデプロイします。ゲームの開発時に、新しいゲームサーバービルドを生成すると、テストを実行する EC2 テスト環境にすばやくデプロイでき

ます。これらを Amazon S3 バケットにアップロードし、ワークフロースクリプトを使用してテスト環境を更新します。

ゲームを Amazon GameLift Servers マネージドフリートに移行する

開発テストを完了し、起動の準備が整ったら、Amazon GameLift Servers マネージドフリートに切り替える良いタイミングとなります。マネージドフリートを使用して、ゲームホスティングリソースをファインチューニングおよびテストします。ゲームセッションの配置ソリューション (キューとマッチメーカー) を実装し、最適なホスティングハードウェア (スポットフリートを含む) と場所を選択し、容量のスケーリング戦略を選択します。また、フリート、キュー、マッチメーカーなど、すべてのゲームホスティングリソースのライフサイクルをより効率的に管理 AWS CloudFormation するために、 の使用を開始することもできます。

これにより、クラウドベースの Anywhere テストフリートから Amazon GameLift Servers マネージドフリートへの移行に対する工数は最小限に抑えられます。ゲームコードを変更する必要はありません。同じキューとマッチメーカーを再利用できます。次のいずれかのタスクを実行します。

- Amazon GameLift Servers ビルドリソースを作成します。Anywhere テストフリートでは、ゲームサーバーのビルドと依存関係を各フリートコンピューティングに手動でデプロイする必要があります。マネージドフリートでは、ゲームビルドパッケージを Amazon GameLift Servers にアップロードします。これにより、すべてのフリートコンピューティングに自動的にデプロイされます。ゲームビルドファイルのパッケージ化と、Amazon S3 バケット内のファイルを使用したビルドリソース作成の詳細については、「[Amazon GameLift Servers のゲームサーバービルドを作成する](#)」を参照してください。
- マネージドフリートを作成します。コンソールまたは AWS CLI を使用し、EC2 マネージドフリートを指定してフリートを作成します。このタイプのフリートには、ビルドリソースとインスタンスタイプの指定など、追加の設定が必要です。同じランタイム設定を使用すると、各フリートコンピューティングのゲームサーバーライフサイクルを管理できます。マネージドフリートの作成の詳細については、「[Amazon GameLift Servers マネージド EC2 フリートを作成する](#)」を参照してください。
- フリートエイリアスをリダイレクトします (オプション)。Anywhere フリートで使用するエイリアスを設定すると、マネージドフリートに同じエイリアスを再利用できます。エイリアスの作成または更新の詳細については、「[Amazon GameLift Servers エイリアスの作成](#)」を参照してください。

Amazon GameLift Servers Anywhereでローカルテストを設定する

Note

このトピックでは、サーバー SDK for Amazon GameLift Servers バージョン 5.x 用サーバー SDK と統合されたゲームのローカルテストについて説明します。ゲームでサーバー SDK バージョン 4.x 以前を使用している場合は、「[Amazon GameLift Servers Local を使用して統合をテストする](#)」を参照してください。

Amazon GameLift Servers Anywhere フリートと独自のハードウェアを使用して、シミュレートされたホスト環境でゲームコンポーネントを繰り返し構築およびテストします。フリートを設定し、ローカルデバイスを登録して、Amazon GameLift Servers サービスへの接続を確立します。ゲームサーバービルドをデバイスにインストールし、ゲームサーバープロセスを開始し、必要に応じてゲーム機能をテストします。ゲームサーバービルドは、新しいビルドのイテレーションをテストするために、必要な回数だけ更新できます。

Anywhere フリートでは、CLI AWS またはテストスクリプトを使用してテストできます。ゲームクライアントを Amazon GameLift Servers と統合した場合は、同じローカルデバイスまたは別のデバイスでクライアントを実行できます。

フリートによるローカルテストは、Amazon GameLift Servers とのゲームサーバー統合をテストするために特に役立ちます。ローカルマシンのすべてのホスティングアクティビティと、イベントとログデータを完全に可視化できます。

Note

Unreal Engine または Unity に Amazon GameLift Servers プラグインを使用していますか？これらのツールには、Anywhere フリートでローカルテストを設定するためのガイド付きワークフローが含まれています。[Unity 用プラグイン: Amazon GameLift Servers Anywhere を使用してローカルテストを設定する](#) または [Unreal 用プラグイン: Amazon GameLift Servers Anywhere でゲームをローカルでホストする](#) のドキュメントに従います。

トピック

- [ローカルの Anywhere フリートを設定する](#)
- [ゲームサーバーを更新してインストールする](#)

- [ゲームセッションアクティビティをテストする](#)
- [ゲームサーバーでのイテレーション](#)
- [ゲームを Amazon GameLift Servers マネージドフリートに移行する](#)

ローカルの Anywhere フリートを設定する

ローカルワークステーションの Anywhere フリートを作成するには、次の手順を実行します。AWS CLI または AWS マネジメントコンソール の を使用する詳細な手順については Amazon GameLift Servers、「」を参照してください [Amazon GameLift Servers Anywhere フリートを作成する](#)。

Anywhere フリートを作成するには

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。(AWS CLI またはコンソール)。カスタムロケーションは、Anywhere フリートに含める予定のコンピューティングリソースのラベルにすぎません。カスタムロケーション名は custom- で始まる必要があります。例: custom-my_laptop。「[カスタムロケーションを作成する](#)」を参照してください。
2. Anywhere フリート (AWS CLI またはコンソール) を作成します。この手順では、ローカルワークステーションのカスタムロケーションを使用してフリートリソースを作成します。「[Anywhere フリートを作成する](#)」を参照してください。

新しいフリートの ID または ARN 値をメモします。この値は次のステップで必要になります。

3. ローカルワークステーションをフリートコンピューティングとして登録します (AWS CLI のみ)。ゲームサーバーをホストするには、Anywhere フリートに少なくとも 1 つのコンピューティングリソースが必要です。「[フリートにコンピューティングを追加する](#)」を参照してください。フリートにコンピューティングを追加するには、次の情報が必要です。
 - コンピューティング名。フリート内の各コンピューティングには、一意の名前が必要です。
 - Anywhere フリート識別子。FleetID または FleetArn のいずれかを使用できます。
 - コンピューティングの接続情報。IpAddress または DnsName のいずれかを指定します。これは、Amazon GameLift Servers とゲームクライアントがゲームサーバーに接続する方法です。
 - Anywhere フリート内のカスタムロケーション。

GameLiftServiceSdkEndpoint の戻り値をメモします。Anywhere フリートで実行するようにゲームサーバーを更新するときは、この値が必要です。

ゲームサーバーを更新してインストールする

このタスクは、ゲームサーバービルドを Amazon GameLift Servers サーバー SDK 5.x と既に統合していることを前提としています。統合プロセスでは、Amazon GameLift Servers サービスとやり取りしてゲームセッションを開始および管理できるように、ゲームサーバーにコードを追加します。

Anywhere フリートの場合、特定のゲームサーバー設定を手動で設定する必要があります。Amazon GameLift Servers マネージドフリートでは、これらの設定は自動的に設定されます。

Anywhere フリート用にゲームサーバーを準備する

1. 認証トークンを取得します。ゲームサーバーには、Amazon GameLift Servers サービスとのすべての通信に認証トークンを含める必要があります。Amazon GameLift Servers 認証トークンは有効期間が短く、定期的に更新する必要があります。

ベストプラクティスとして、スクリプトを作成して次のタスクを完了します。

- CLI AWS アクション を呼び出します `get-compute-auth-token`。
- ローカルコンピューティングの環境変数など、ゲームサーバープロセスが取得できるトークンの戻り値を保存します。

ゲームサーバーと一緒にスクリプトをコンピューティングにインストールします。最初のゲームサーバープロセスを開始する前に、実行するスクリプトを設定します。ゲームサーバープロセスがアクティブな間は、スクリプトを定期的に実行して有効な認証トークンを維持します。コンピューティング上のすべてのゲームサーバープロセスは、同じ認証トークンを使用できます。

2. Amazon GameLift Servers ゲームサーバーコードを更新します。ゲームサーバーコードをサーバー SDK for Amazon GameLift Servers と統合すると、アクション `InitSdk()` への呼び出しが追加されます。ゲームサーバーを Anywhere フリートで実行する場合、この呼び出しには、追加のサーバーパラメータが必要です。詳細については、開発言語に該当する「[サーバープロセスを初期化する](#)」および「[サーバー SDK 5.x for Amazon GameLift Servers](#)」を参照してください。サーバーパラメータは次のとおりです。

- `websocketUrl` – このパラメータを、フリートにコンピューティングを登録したときに返される `GameLiftServiceSdkEndpoint` 値に設定します。
- `hostId` – このパラメータを、Anywhere フリートにコンピューティング登録した時に指定されるコンピューティング名に設定します。
- `fleetId` – このパラメータを Anywhere フリートの ID に設定します。

- `authToken` – このパラメータを、コンピューティングの認証トークンを取得するリクエストに 응답して返されるトークンに設定します。
- `processId` – このパラメータを設定して、ローカル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を特定します。同時ゲームサーバープロセスごとに一意のプロセス ID が必要です。

各ゲームサーバープロセスが使用するサーバーパラメータ値は、プロセスが実行されている Anywhere フリートコンピューティングに固有である必要があります。コンピューティングに適切な値を取得する方法の詳細については、「[フリートにコンピューティングを追加する](#)」を参照してください。ベストプラクティスとして、`websocketUrl`、`hostId`、`fleetId`、および `authToken` をローカルコンピューティングの環境変数として設定します。コンピューティングで実行されるすべてのサーバープロセスは、これらの値を使用します。

3. ローカルコンピューティングにゲームサーバービルドをインストールします。ゲームサーバーの実行に必要なすべての依存関係を含めます。
4. ローカルコンピューティングで実行されている 1 つ以上のゲームサーバープロセスを開始します。ゲームサーバープロセスがサーバー SDK アクション `ProcessReady()` を呼び出すと、プロセスはゲームセッションをホストする準備が整います。

ゲームセッションアクティビティをテストする

ゲームセッションを使用して、ゲームサーバーの統合をテストします。Amazon GameLift Servers 機能と統合されたゲームクライアントがない場合は、AWS CLI を使用するとゲームセッションを開始できます。次のシナリオを試行します。

- ゲームセッションの作成。[create-game-session](#) コマンド (または [CreateGameSession](#) API オペレーション) を呼び出します。Anywhereフリートの ID とカスタムロケーションを指定します。この呼び出しは、新しいゲームセッションの一意の識別子を返します。
- ゲームセッションステータスを確認します。[describe-game-sessions](#) コマンド (または [DescribeGameSessions](#) API アクション) を呼び出します。ゲームセッション ID を指定します。この呼び出しは、ゲームセッションステータスを含む詳細なゲームセッション情報を返します。アクティブ状態のゲームセッションは、プレイヤーによる接続ができることを示します。フリートのすべてのゲームセッションのリストを取得するには、[list-game-sessions](#) コマンド (または [ListGameSessions](#) API アクション) を呼び出します。
- ゲームセッションに接続します。ゲームクライアントにゲームセッションに参加する機能がある場合は、ゲームセッション情報に含まれる接続情報を使用します。

ゲームサーバーでのイテレーション

同じ Anywhere フリートとコンピューティングを使用すると、ゲームサーバーぶり度の別のバージョンをテストできます。

1. 既存の **GameSession** をクリーンアップします。ゲームサーバープロセスがクラッシュした場合または、`ProcessEnding()` を呼び出さない場合は、ゲームサーバーがヘルスチェックの送信を停止した後に、Amazon GameLift Servers が **GameSession** をクリーンアップします。
2. 新しいゲームサーバービルドを生成します。ゲームサーバーを変更し、改訂されたビルドをパッケージ化します。
3. ローカルコンピューティングでゲームサーバービルドを更新します。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。
4. 更新された認証トークンを取得します。[get-compute-auth-token](#) CLI コマンドを呼び出し、トークンをローカルコンピューティングに保存します。
5. ローカルコンピューティングで実行されている 1 つ以上のゲームサーバープロセスを開始します。ゲームサーバープロセスが `ProcessReady()` を呼び出すと、テストに使用する準備が整います。

ゲームを Amazon GameLift Servers マネージドフリートに移行する

開発テストを完了し、起動の準備が整ったら、Amazon GameLift Servers マネージドフリートに切り替える良いタイミングとなります。マネージドフリートを使用して、ゲームホスティングリソースをファインチューニングおよびテストします。ゲームセッションの配置ソリューション (キューとマッチメーカー) を実装し、最適なホスティングハードウェア (スポットフリートを含む) と場所を選択し、容量のスケーリング戦略を選択します。また、フリート、キュー、マッチメーカーなど、すべてのゲームホスティングリソースのライフサイクルをより効率的に管理 AWS CloudFormation するために、 の使用を開始することもできます。

ローカルテストフリートから Amazon GameLift Servers マネージドフリートに移行するには、いくつかの小さな修正を行う必要があります。同じキューとマッチメーカーは再利用できます。次のいずれかのタスクを実行します。

- ゲームサーバーのコード呼び出しを **InitSdk()** に変更します。サーバーパラメータを削除します。マネージドフリートの場合、Amazon GameLift Servers はこの情報を自動的に追跡します。
- Amazon GameLift Servers ビルドリソースを作成します。Anywhere テストフリートでは、ゲームサーバーのビルドと依存関係を各フリートコンピューティングに手動でデプロイする必要があります。

す。マネージドフリートでは、ゲームビルドパッケージを作成して Amazon GameLift Servers にアップロードします。これにより、すべてのフリートコンピューティングに自動的にデプロイされます。ゲームビルドファイルのパッケージ化と、Amazon S3 バケット内のファイルを使用したビルドリソース作成の詳細については、「[Amazon GameLift Servers のゲームサーバービルドを作成する](#)」を参照してください。Amazon GameLift Servers はマネージドフリートでこれらのタスクを自動的に処理するため、コンピューティングを登録して認証トークンを取得するスクリプトを含めないでください。

- マネージドフリートを作成します。EC2 マネージドフリートを指定して、コンソールまたは AWS CLI を使用してフリートを作成します。このタイプのフリートには、ビルドリソースとインスタンスタイプの指定など、追加の設定が必要です。フリートコンピューティングごとにゲームサーバーライフサイクルを管理するには、すべてのユーザーがランタイム設定をセットアップする必要があります。マネージドフリートの作成の詳細については、「[Amazon GameLift Servers マネージド EC2 フリートを作成する](#)」を参照してください。
- フリートエイリアスをリダイレクトします (オプション)。Anywhere フリートで使用するエイリアスを設定すると、マネージドフリートに同じエイリアスを再利用できます。エイリアスの作成または更新の詳細については、「[Amazon GameLift Servers エイリアスの作成](#)」を参照してください。

Amazon GameLift Servers Local を使用して統合をテストする

Note

このトピックでは、サーバー SDK for Amazon GameLift Servers バージョン 3.x および 4.x と統合されたゲームのテストについて説明します。サーバー SDK パッケージには、Amazon GameLift Servers 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 Servers Anywhere フリートを用了ローカルテストについては「[GameLift Server SDK 5.x でのローカルテスト](#)」を参照してください。SDK 4.x から 5.x への移行に関するガイダンスについては、「[GameLift Server SDK 5.x への移行](#)」を参照してください。

Amazon GameLift Servers Local を使用して、ローカルデバイスでマネージド Amazon GameLift Servers 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。代替の方法 - 新しい各ビルドを Amazon GameLift Servers 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、毎回数回以上かかる場合があります。

Amazon GameLift Servers Local では、以下の点を確認できます。

- ゲームサーバーが サーバー SDK と正常に統合されており、新しいゲームセッションの開始、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 Servers 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 Servers 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開始、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 Servers Local は、マネージド Amazon GameLift Servers 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 Servers Amazon GameLift Servers 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。Local は、Amazon GameLift Servers 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 Servers 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 Servers Local への接続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::ProcessEnding` を呼び出して、プロセスを終了します。Amazon GameLift Servers 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 Servers 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 Servers 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 Servers Local のセットアップ

Amazon GameLift Servers Local は、[サーバー SDK](#) にバンドルされた実行可能ファイル `.jar` として提供されます。Windows または Linux で実行でき、Amazon GameLift Servers でサポートされる任

意の言語で使用できます。SDK バージョン 4.0.2 の場合は、[GitHub の公式リリース](#)からダウンロードできます。GameLiftLocal.jar ファイルは GameLift-CSharp-ServerSDK-4.0.2.zip アーティファクトに含まれています。

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。

- サーバー SDK for Amazon GameLift Servers バージョン 3.1.5 から 4.x までのビルド
- 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 Servers 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、サーバー SDK for Amazon GameLift Servers を初期化します。
- 起動プロセスの一環として、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 Servers に通知する。
- 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 Servers に送信する。
- 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 Servers Local を起動します。

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 Servers が統合されたゲームサーバーをローカルで起動します。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 Servers 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 Server SDK for Amazon GameLift Servers を、InitSDK() を使って正常に初期化したことを意味します。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 Servers はゲームサーバーから各ヘルスステータスレポートをログに記録します。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。

- エラー: "pID: *<process ID>* の ProcessReady が見つかりません! InitSDK() は呼び出されましたか?"
- 警告: "プロセスのステータスは pID: *<process ID>* で既にプロセスに存在します! ProcessReady(...) は複数回呼び出されましたか?"

3. AWS CLI を開始します。

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

では、AWS CLI は Amazon GameLift Servers ウェブサービスエンドポイントを使用します。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 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 Servers 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: gameId:
```

```
"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 Servers 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 Servers 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、AWS Local サービスへの Amazon GameLift Servers 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 Servers Local を起動します。

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。

```
./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 Servers が統合されたゲームサーバーをローカルで起動します。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 Servers Local サービスでゲームクライアントを使用するには、「[Amazon GameLift Servers API をセットアップします。](#)」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。

- ClientConfiguration などの Local エンドポイントをポイントするように、`http://localhost:9080` 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-\S+` などの任意の有効な文字列 (`fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d`) 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起動すると、GameLiftClient を初期化し、Amazon GameLift Servers 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ます。

4. Amazon GameLift Servers 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 コール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 Servers Local がゲームサーバーにゲームセッションの開始を求め (`onStartGameSession` コールバック)、ゲームサーバーがゲームセッションを呼び出したときに `ActivateGameSession` 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 コール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ログメッセージに表示されます。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 Servers サービスにレポートしていることを確認します。

Amazon GameLift Servers 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、ゲームサーバーも各種ステータスを Amazon GameLift Servers 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する - Amazon GameLift Servers 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ます。AWS CLI への `DescribePlayerSessions()` 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する - Amazon GameLift Servers 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

以前のガイドンスでは、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 Servers サーバー SDK v4.0.1 で非推奨です。「[ゲームセッションを終了する](#)」を参照してください。

- サーバードキュメントが終了した - Amazon GameLift Servers 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。AWS CLI への `DescribeGameSessions()` 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 Servers Local を使用する場合は、次の点に留意してください。

- Amazon GameLift Servers 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 Servers フリートの問題をデバッグする

このトピックでは、Amazon GameLift Servers マネージド EC2 フリートの問題を解決する方法を説明します。

フリート作成の問題

マネージド EC2 フリートを作成すると、Amazon GameLift Servers サービスはフリートを作成するワークフローを開始し、ゲームサーバービルドをインストールして EC2 インスタンスをデプロイし、各インスタンスでゲームサーバープロセスを開始します。詳細な説明については、次を参照してください。フリートは、[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。

問題が発生したフリート作成フェーズを特定し、フリート作成イベントとログを調査することで、フリートがアクティブにならない原因となっている問題をデバッグできます。ログから有用な情報を得られなし場合、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、ゲームビルドを再アップロードして、ファイル破損の可能性を解消します。Amazon GameLift Servers サポートに問い合わせるか、フォーラムで質問を投稿することもできます。

ビルドのダウンロードと検証

このフェーズでは、Amazon GameLift Servers はアップロードされたゲームサーバービルドを取得し、ファイルを抽出し、インストールスクリプトを実行します。これらのフェーズでフリートの作成に失敗した場合は、フリートイベントとログを調べて問題を特定します。エラーの原因として以下が考えられます。

- Amazon GameLift Servers が圧縮されたビルドファイルを取得できない (イベント `FLEET_BINARY_DOWNLOAD_FAILED`)。ビルドのストレージロケーションにアクセスできること、AWS リージョン ビルドと同じにフリートを作成すること、およびそれにアクセスするための正しいアクセス許可 Amazon GameLift Servers があることを確認します。
- Amazon GameLift Servers がビルドファイルを抽出できない (イベント `FLEET_CREATION_EXTRACTING_BUILD`)。
- ビルドファイルのインストールスクリプトが正常に完了しなかった (イベント `FLEET_CREATION_FAILED_INSTALLER`)。

フリートリソースの構築

このフェーズでは通常、フリートリソースの割り当てとデプロイに関する問題が発生します。エラーの原因として以下が考えられます。

- リクエストされたインスタンスタイプが利用できない。
- リクエストされたフリートタイプ (スポットまたはオンデマンド) が利用できない。

ゲームサーバープロセスのアクティブ化

このフェーズでは、Amazon GameLift Servers は多数のタスクを試行し、ゲームサーバーの実行可能性、ランタイム設定、サーバー SDK を使用した Amazon GameLift Servers サービスへのゲームサーバーの接続能力など、主要な要素をテストしています。

Note

このフェーズでは、フリートインスタンスにリモートアクセスして問題をさらに調査できます。「[Amazon GameLift Servers 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。

考えられる問題には、次のようなものがあります。

- サーバープロセスが実行を開始しない。これは、フリートのランタイム設定 (イベント `FLEET_VALIDATION_LAUNCH_PATH_NOT_FOUND` または `FLEET_VALIDATION_EXECUTABLE_RUNTIME_FAILURE`) に問題があることを示唆しています。起動パスとオプションの起動パラメータが正しく設定されていることを確認してください。
- サーバープロセスは実行を開始するが、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 Amazon GameLift Servers サービスと通信できていない可能性があります。ゲームサーバーが以下の適切なサーバー SDK 呼び出しを行っていることを確認してください (「[サーバープロセスを初期化する](#)」を参照)。
- サーバープロセスが初期化に失敗する (イベント `SERVER_PROCESS_SDK_INITIALIZATION_TIMEOUT`)。サーバープロセスが `InitSdk()` を正常に呼び出していません。
- サーバープロセスが、ゲームセッションをホストできる状態になったことを Amazon GameLift Servers に通知できていない (イベント `SERVER_PROCESS_PROCESS_READY_TIMEOUT`)。サーバープロセスは初期化されましたが、`ProcessReady()` が時間内に呼び出されませんでした。
- VPC ピアリング接続が失敗する。作成されたフリートで VPC ピアリング接続が使用される場合 (「[新しいフリートとの VPC ピアリング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 [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 ピアリング認証 ([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 Servers コンソール (フリートの容量割り当て設定の編集) を使用するか、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、自動スケーリング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 ピアリング接続を特別に作成したフリートにのみ適用されます (「」を参照してください [Amazon GameLift Servers に対する VPC ピアリングのセットアップ](#))。このシナリオは、フリートを削除するプロセスに、フリートの VPC と VPC ピアリング接続の削除が含まれるために発生します。まず、Amazon GameLift Servers [CreateVpcPeeringAuthorization\(\)](#) のサービス API を呼び出すか AWS、CLI コマンド を使用して認可を取得する必要があります `create-vpc-peering-authorization`。承認を取得したら、フリートを削除できます。

Amazon GameLift Servers 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 Servers マネージド EC2 フリートまたはコンテナフリート内の任意のインスタンスに接続できます。インスタンスにリモート接続する一般的な理由には、次のようなものがあります。

- ゲームサーバーの統合に関する問題をトラブルシューティングする。
- ランタイム設定などフリート固有の設定をファインチューニングする。
- ログの追跡など、リアルタイムのゲームサーバーアクティビティを取得する。
- 実際のプレー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する。
- ゲームセッションまたはゲームサーバープロセスに関する特定の問題を調査する。

インスタンスに接続する際に考慮すべき問題として、次のものが考えられます。

- アクティブなフリートのインスタンスの任意のインスタンスに接続できます。通常、アクティブセッション中またはエラー状態にあるフリートなど、非アクティブなフリートには接続できません。(このようなフリートの可用性は短期間制限される場合があります)。フリートのアクティブ化に関する問題のヘルプは、「[Amazon GameLift Servers 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引き続きサーバープロセスの起動と停止を行い、ゲームセッションをアクティブ化して実行します。インスタンスは、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される可能性があります。
- インスタンスに加えた変更は、インスタンスのアクティブなゲームセッションおよび接続されたプレーヤーに影響を与える可能性があります。

コンソールからのリモートアクセス

Amazon EC2 Systems Manager (SSM) を使用して、Amazon GameLift Servers コンソールから直接フリートインスタンスに接続できます。このメソッドは、追加のセットアップや認証情報管理を必要とせずに、安全なアクセスを提供します。

1. Amazon GameLift Servers コンソールで、ナビゲーションペインから [マネージド EC2] または [マネージドコンテナ] を選択し、次に [フリート] を選択します。

2. アクセスするインスタンスを含むフリート ID を選択します。
3. フリートの詳細ページで、インスタスタブを選択して、フリートのすべてのコンピューティングインスタンスを表示します。
4. 接続するインスタンスを選択し、[接続] を選択します。これにより、接続の詳細を通知するインスタンスへの接続ダイアログが表示され、インスタンスへの接続に使用されるスクリプトを表示できます。[接続] を再度選択して確認します。
5. 接続ダイアログで、[実行] を選択して新しい SSM セッションを作成します。システムは AWS Key Management Service (AWS KMS) を介してセッションを認証し、ブラウザでターミナルを開きます。

Note

コンソールベースのリモートアクセスは、サーバー SDK バージョン 5.x を実行しているフリートで使用できます。以前の SDK バージョンを実行しているフリートの場合は、次のセクションで説明する AWS CLI メソッドを使用します。

CLI AWS によるリモートアクセス

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を説明します。[のサービス API リファレンスAmazon GameLift Servers](#)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を収集する

Amazon GameLift Servers マネージド EC2 フリートインスタンスに接続するには、次の情報が必要です。

- 接続するインスタンスの ID。インスタンスの ID または ARN のどちらでも使用できます。
- インスタンスで使用されているサーバー SDK for Amazon GameLift Servers 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。

次の手順では、CLI AWS を使用してこれらのタスクを完了する方法について説明します。接続するインスタンスのフリート ID を知っておく必要があります。

1. コンピューティング名を取得します。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。フリート ID または ARN を使用して [list-compute](#) を呼び出します。単一

ロケーションフリートの場合は、フリート ID のみを指定します。マルチロケーションフリートの場合は、フリート ID とロケーションを指定します。マネージド EC2 フリートでは、`list-compute` はフリートインスタンスのリストを返します。プロパティ `ComputeName` はインスタンス ID です。アクセスするコンピューティングを検索します。

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":
        "b08444ki909kvqu6zpw3is24x5pyz4b6m05i3jbxvpk9craztu0lqrbbrbrnbkks.uwp57060n1k6dn1nw49b78hg1west-2.amazongamelift.com",
      "ComputeStatus": "Active",
      "Location": "sa-east-1",
      "CreationTime": "2023-07-09T22:51:45.931000-07:00",
      "OperatingSystem": "AMAZON_LINUX_2023",
      "Type": "c4.large"
    }
  ]
}
```

2. サーバー SDK バージョンを確認します。この情報を得るには、フリートにデプロイされているビルドを調べる必要があります。サーバー SDK バージョンはビルドのプロパティです。
 - a. [describe-fleet-attributes](#) をフリートの ID または ARN で呼び出して、フリートのビルドの ID と ARN を取得します。
 - b. ビルドの 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。

例えば、次のようになります。

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_2023",  
  "ServerSdkVersion": "5.1.1",  
  . . .  
}
```

インスタンスに接続する (サーバー SDK 5)

接続する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、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。

開始する前に:

SSM のセットアップステップを完了し、ローカルマシンに SSM プラグインをインストールします。詳細については、「Amazon EC2 Systems Manager ユーザーガイド」の「[SSM のセットアップ](#)」および「CLI 用の Session Manager プラグインのインストール」を参照してください。 [AWS](#) Amazon EC2 Systems Manager

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フリート ID とコンピューティング名を使用して [get-compute-access](#) を呼び出します。Amazon GameLift Servers 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を返します。例えば、次のようになります。

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 (省略可)。認証情報を環境変数にエクスポートし、それら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、「[AWS Command Line Interface ユーザーガイド](#)」の「[CLI AWS を設定する環境変数](#)」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。接続するインスタンスで 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。SSM と SSM プラグインの設定方法などの詳細については、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。

start-session リクエストは、ステップ 1 で取得した認証情報を自動的に使用します。

```
aws ssm start-session \
--target i-11111111a222b333c \
--region us-west-2 \
```

Note

アクセス拒否エラーが発生した場合、AWS_PROFILE 環境変数が AWS プロファイルに設定されている可能性があります。これにより、CLI AWS はリモートアクセスに間違った認証情報を使用します。解決するには、AWS_PROFILE 環境変数の設定を一時的に解除します。または、リモートアクセス認証情報のカスタム AWS プロファイルを作成し、--profile コマンドラインパラメータを start-session リクエストに追加することもできます。

インスタンスに接続する (サーバー SDK 4.x 以前)

接続するインスタンスがサーバー SDK バージョン 4 以前のゲームビルドを実行している場合は、次の手順を実行します。Windows または Linux を実行しているリモートインスタンスに接続できます。Windows インスタンスには、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て接続します。Linux インスタンスには、SSH クライアントを使用して接続します。

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 [get-instance-access](#) を使用してアクセス認証情報をリクエストします。成功すると、Amazon GameLift Servers 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) 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 はユーザー名とプライベートキーを必要とします。Amazon GameLift Servers は、RSA プライベートキーを発行し、これらを 1 つの文字列として返します。改行文字 (\n) は改行を示します。プライベートキーを使用可能にするには、(1) 文字列を .pem ファイルに変換し、(2) この新しいファイルへ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/akH7i05dSrvC7dQkW2duV5QuUdE0QW
\nZ/aNxMniGQE6XAgfwlnXVBwrerrQo+ZWQeqiUwwMkuEbLeJFLhMCvYURpUMSC1oehm449ilx9X1F
\nG50TCFe0zf18dqqCP6GzbPaIjiU19xX/az0R9V+tpU0zEL+wmXnZt3/nHPQ5xvD20JH67km6SuPW
\nnoPzev/D8V+x4+bHthfSjR9Y7DvQFjfbVwHXigBdtZcU2/wei8D/HYwIDAQABaoIBAGZ1kaEvnrrqu
\n/uler7vgIn5m7lN5LKw4hJLAIW6tUT/fzvtcHK0SkbQCQXuriHmQ2MQyJX/0kn2NfjLV/
ufGxbL1\nmb5qwmGUNepJaZD6QSSs3kICLwUyUiGfc0uiSbmJoap/
GTLU0W5Mfcv36PaBUNy5p53V6G7hXb2\nnbahyWyJNfjLe4M86yd2YK3V2CmK+X/
B0sShnJ36+hjrXPPWmV3N9zEmCdJjA+K15DYmhm/
tJWSD9\n81oGk9TopEp7CkIfatEATyyZiVqoRq6k64iuM9JkA30zdXzMqexXVJ1TLZVEH0E7bh1Y9d801ozR
\noQs/FiZNAx2iijCWyv0lpjE73+kCgYEA9mZtyhkhkFDpwrSM1APaL8oNAbbjwEy7Z5Mqfq1
+1Ip1\nYkriL0DbLXlvRAH+yHPRit2hH0jtUNZh4Axv+cpG09qbUI3+43eEy24B7G/Uh
+GTfbjsXs0xQx/x\np9otyVwc7hsQ5TA5PZb
+mvkJ50BEKzet9XcKw0NBVELGhnEPe7cCgYEA06Vgov6YHleHui9kHuws
\nayav0elc5zkxjF9nfHFJRry21R1trw2Vdpn+9g481URrpzWV0Eihvm+xTtmaZ1Sp//1kq75XDwnU
\nWA8gkn603QE3fq2yN98BURsAKdJfJ5RL1HvGQvTe10HLYYXpJnEkHv+Unl2ajLivWUt5pbBrKbUC
\nngYBjb0+0Zk0sCcpZ29sbzjYjpIddErySIyRX5gV2uNQwAjLdp9PfN295yQ+BxMBXiIycWVQiw0bH
-----"
```

```

\noMo7yykABY70zd5wQewBQ4AdSlWSX4nGDtsiFxiI5sKuAAe0CbTosy1s8w8fxoJ5Tz1sdoxNeGs
\nArq6Wv/G16zQuAE9zK9vwwKBgF+09VI/1wJBirsDGz9whVWfFPrTkJNvJZzYt69qezx1sjgFKshy
\nWBhd4xHZtmCqpBP1AymEjr/T01bxyARmXMnIOWIAnNXMGB4KGSyl1mzSVAoQ+fQr+cJ3d0dyP11j
\njjb0Ed/NY8frlNDxAVHE8BSkdsx2f6ELEyBKJSRr9snRAoGAMrTwYneXzvTskF/S5Fyu0i0egLda
\nNWUH38v/nDCgEpIXD5Hn3qAEcju1IjmbwlvtW+nY2jVhv7UGd8MjwUTNGItdb6nsYqM2asrnF3qS
\nVRkAKKKYeGjKpUfVTrW0YFjXkfcrR/V+QFL50ndHAKJXjW7a4ejJLncTzmZSpYzwApc=\n-----END
RSA PRIVATE KEY-----",
  "UserName": "gl-user-remote"
}

```

AWS CLI を使用する場合、`--query` および `--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
```

- リモート接続でポートを開きます。Amazon GameLift Servers フリート内にあるインスタンスは、フリート設定で承認されている任意のポートを通じてアクセスできます。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、フリートが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。問題が発生した場合は、ポート設定を開いた状態でフリートを再作成してください。

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上のファイルを表示する

コンソールまたは CLI AWS を介してリモートでインスタンスに接続したかどうかにかかわらず、インスタンスへのフルユーザーおよび管理アクセス権があります。つまり、ゲームホスティングに対しても、誤操作などによりエラーや障害を引き起こし得る権限を持っていることを意味します。インスタンスがアクティブなプレイヤーを含むゲームをホストしている場合、ゲームセッションのクラッシュやプレイヤーの切断を招いたり、ゲームのシャットダウン処理を妨げて、保存済みのゲームデータやログにエラーを発生させてしまうおそれがあります。

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 GameLift Serversにアップロードしたゲームビルドです。これには、1 つまたは複数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、game という名前のルートディレクトリにあります。
 - 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したログファイルを、指定した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 Servers ホスティングリソース。Whitewaterという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 Serversサービスで使われるファイルがあります。これらのファイルは、いかなる理由があっても変更しないでください。
- ランタイム設定。個別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 (「SDK オペレーション [UpdateRuntimeConfiguration](#)」または AWS CLI [update-runtime-configuration](#)」を参照してください AWS)。

- フリートデータ。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できるように、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。この JSON ファイルは、次の場所にあります。
 - Windows の場合: `C:\GameMetadata\gamelift-metadata.json`
 - 歳Linux の場合: `/local/gamemetadata/gamelift-metadata.json`
- TLS 証明書。インスタンスがあるフリートで TLS 証明書の生成が有効になっている場合、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は次の場所にあります。
 - Windows の場合: `c:\GameMetadata\Certificates`
 - 歳Linux の場合: `/local/gamemetadata/certificates/`

Amazon GameLift Servers を使用したゲームホスティング容量のスケールリング

ホスティング容量は、Amazon GameLift Servers が同時にホストできるゲームセッションの数と、これらのゲームセッションに参加できる同時プレイヤーの数を表し、インスタンス単位で測定されます。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ルリングすることです。

容量は、フリートのロ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、フリートのホーム AWS リージョンという少なくとも 1 つの場所があります。容量を表示またはスケールリングすると、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ル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ル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ルリングの目標は、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ルリングは非常に効果的なスケールリングソリューションです。

ほとんどのフリートスケールリングアクティビティは、Amazon GameLift Servers コンソールを使用して行うことができます。のサービス 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。 [Amazon GameLift Servers](#)

トピック

- [コンソールでフリートの容量を管理するには](#)
- [Amazon GameLift Servers の容量制限の設定](#)
- [Amazon GameLift Servers フリートの容量を手動で設定する](#)
- [Amazon GameLift Servers でのフリート容量の自動スケールリング](#)
- [ゼロとの間でAmazon GameLift Serversフリートのスケールリングを管理する](#)
- [Amazon GameLift Servers コンテナフリートのスケールリング](#)

コンソールでフリートの容量を管理するには

1. [Amazon GameLift Servers 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。

Amazon GameLift Servers の容量制限の設定

Amazon GameLift Servers フリートロケーションのホスティング容量を手動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てください。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 Servers 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。容量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。(AWS コストをモニタリングするための[請求アラームを作成](#)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。

フリートの場所の容量制限は、[Amazon GameLift Servers コンソール](#) または AWS Command Line Interface () を使用して設定できますAWS CLI。

容量制限を設定するには

Console

1. [Amazon GameLift Servers 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 [スケーリング容量] 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. [スケーリング容量を編集] 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. 現在の容量設定を確認します。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](#) 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 Servers がフリート ID を戻します。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 Servers はエラーを返します。

Amazon GameLift Servers フリートの容量を手動で設定する

新しいフリートを作成すると、Amazon GameLift Servers は必要なインスタンス数をフリートロケーションごとに 1 インスタンスに自動的に設定します。次に、Amazon GameLift Servers 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。

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。

コンソールまたは AWS Command Line Interface () Amazon GameLift Serversを使用して、容量を手動で設定できますAWS CLI。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 Servers 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. [スケーリング容量] 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. [スケーリング容量を編集] 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 Servers に伝えます。

Amazon GameLift Servers 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 Servers の処理が完了すると、ロケーション内の Active インスタンス数が、更新された目標インスタンス数に合わせて変更されます。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](#) 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、新しい値で使います。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 Servers がフリート ID を戻します。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 Servers はエラーを返します。

Amazon GameLift Servers でのフリート容量の自動スケーリング

自動スケーリング Amazon GameLift Servers を使用して、ゲームサーバーの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 Servers 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 Servers 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。ルールベースの自動スケーリングは、AWS CLI または AWS SDK を使用してのみ管理できますが、コンソールでルールベースのスケーリングポリシーを表示できます。

トピック

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

ターゲットベースの自動スケーリング

Amazon GameLift Servers 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容量の割合となります。Amazon GameLift Servers 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 Servers 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. [スケーリング容量] 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 Servers 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 Servers の容量制限の設定](#)」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

ルールベースのスケーリングポリシー (Amazon GameLift Servers) を使用すると、フリートキャパシティの自動スケーリングを詳細に制御して、プレイヤーのアクティビティに対応できます。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、特別な状況进行处理するために[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

コンストラクト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 AWS Command Line Interface または (AWS CLI) と [のサービス API for Amazon GameLift Servers](#) を使用して、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 Servers 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 Servers の容量制限の設定](#)」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構築するには、6 つの変数を指定します。

`<operator>` が `<metric>` にわたって `<threshold>` `<action>` である場合は、`<metric>` を使用してフリートの容量を `<threshold>` 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 Servers 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)

この変数は、Amazon GameLift Servers にメトリクスデータをしきい値と比較する方法を示します。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 Servers 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。

- 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。
- 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。

- 正確な容量 – 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。

フリートの容量は、フリートのインスタンスタイプの制限と のサービスクォータによっても制限されます AWS アカウント。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容量を変更した後、Amazon GameLift Servers 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 Servers 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

タ収集の開始を行う時間を確保できます。この間、Amazon GameLift Servers 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。

ゼロとの間でAmazon GameLift Serversフリートのスケーリングを管理する

Amazon GameLift Servers は、ゲームセッションアクティビティに基づいて、ゼロインスタンスとの間の自動スケーリングをサポートします。このマネージドキャパシティオプションを使用すると、ゲームセッションアクティビティがない定義された期間後にフリートロケーションをゼロインスタンスにスケールインし、ゲームセッションがリクエストされたときに自動的にスケールアウトできます。

ゼロインスタンスとのスケーリングには、いくつかの利点があります。

- コストの最適化 – ゲームセッションアクティビティがないときにインスタンスをゼロにすることで、非アクティブ期間中のコンピューティングコストを排除します。
- 自動リアクティブスケールアウト – ゲームセッションがリクエストされると、フリートのロケーションが 1 つのインスタンスに自動的にスケールアウトされるため、手動による介入は必要ありません。
- 管理の簡素化 – 予想されるプレイヤーの需要や開発ニーズに基づいて、フリートの容量をゼロに手動で調整する必要はありません。

フリートでゼロへのスケール/ゼロからのスケールを有効にすると、Amazon GameLift Servers はゲームセッションアクティビティをモニタリングし、フリートの容量を自動的に調整します。

- ゼロへのスケールイン – ゲームセッションアクティビティのない設定された期間後、はフリートの場所をゼロインスタンスにAmazon GameLift Serversスケールインします。
- ゼロからのスケールアウト – ゲームセッション作成リクエストを受信すると、はフリートの場所を 1 つのインスタンスにAmazon GameLift Serversスケールアウトし、自動スケーリングを再開できるようにします。
- 継続的なスケーリング – スケールアウト後、フリートは設定された自動スケーリングポリシーを使用して容量を管理します。

スケールイン動作

Amazon GameLift Servers は、設定された非アクティブ期間が経過し、ゲームセッションアクティビティがない場合、フリートの場所のスケールインプロセスを開始します。これは、次の期間として定義されます。

- フリートの場所にアクティブなゲームセッションはありません。
- フリートロケーションに新しいゲームセッションを作成するリクエストは行われていません。

スケールイン中、Amazon GameLift Serversはフリートロケーションの最小容量と希望する容量をゼロに設定し、コスト削減のために迅速にスケールインします。

スケールアウト動作

フリートの場所が 0 インスタンスのときにゲームセッション作成リクエストを受信した場合:

- Amazon GameLift Servers は 1 つのインスタンスのスケールアウトをすぐに開始します。
- ゲームセッションの配置試行は、キューの設定 (使用する場合) に応じて、他のフリートまたはフリートのロケーションで続行できます。

Note

ゼロからスケールアウトするには、インスタンスのプロビジョニングと初期化に時間がかかります。プレイヤーは、非アクティブ状態が続くと、最初のゲームセッションの待機時間が長くなることがあります。このため、この機能はマルチロケーションフリートやキューと組み合わせるのが最適です。

ゼロとの間でスケールを設定する

ゼロとのスケールリングは、既存のフリートを更新することによって設定されます。

Console

1. [Amazon GameLift Servers 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」で、ゼロとの間でスケールを設定する各場所を選択し、「編集」を選択します。
5. スケーリング容量の編集ダイアログボックスで、最小容量戦略に「自動」を選択し、最小容量を 0 に設定してから「確認」を選択します。

Amazon GameLift Servers その後、設定された期間にゲームセッションアクティビティがない場合、は選択したフリートロケーションを 0 インスタンスにスケールインします。その後、この場所でゲームセッションのリクエストが行われると、Amazon GameLift Servers はできるだけ早く 1 つのインスタンスをスケールアウトします。このプロセスには少し時間がかかります。

AWS CLI

- ゼロとの間でスケールを設定します。コマンドラインウィンドウで、[update-fleet-capacity](#) コマンドをフリート ID、ロケーション、マネージドキャパシティ設定とともに使用して、ゼロへのスケール/ゼロからのスケールを設定します。

```
aws gamelift update-fleet-capacity \
    --fleet-id <fleet identifier> \
    --location <location name> \
    --managed-capacity-configuration
ScaleInAfterInactivityMinutes=60,ZeroCapacityStrategy=SCALE_TO_AND_FROM_ZERO
```

例:

```
aws gamelift update-fleet-capacity \
    --fleet-id
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
    --location us-west-2 \
    --desired-instances 5 \
    --max-size 10 \
    --managed-capacity-configuration
ScaleInAfterInactivityMinutes=60,ZeroCapacityStrategy=SCALE_TO_AND_FROM_ZERO
```

リクエストが成功すると、は更新された容量設定Amazon GameLift Serversを返します。
Amazon GameLift Serversは、設定された期間にゲームセッションアクティビティがない場合、選択したフリートロケーションを 0 インスタンスにスケールインします。その後、この場所でゲームセッションのリクエストが行われると、Amazon GameLift Servers はできるだけ早く 1 つのインスタンスをスケールアウトします。このプロセスには少し時間がかかります。

ベストプラクティス

Scale To/From Zero を使用する場合は、次の推奨事項を考慮してください。

- 適切な非アクティブ期間を設定する – コスト削減とスケールイン/スケールアウトサイクルの頻度のバランスを取ります。ゼロにスケールするまでの待機時間が短いほど、節約が最大化されますが、コールドスタートの頻度は高くなります。
- 予測可能なワークロードでの使用 – ゼロへのスケーリング/ゼロからのスケーリングは、開発/テスト環境やオフピーク時間が異なるゲームなど、非アクティブ期間が明確なゲームに最適です。
- CloudWatch メトリクスのモニタリング – フリートスケーリングイベントとゲームセッションの配置時間を追跡して、設定を最適化します。
- スケーリングポリシーとの組み合わせ – 包括的なキャパシティ管理のために、ターゲットベースまたはルールベースの自動スケーリングとともにゼロへのスケーリング/ゼロからのスケーリングを使用します。

Amazon GameLift Servers コンテナフリートのスケーリング

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。マネージド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して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 GameLift Servers はフリートの希望する容量を 1 つのインスタンスに設定し、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 GameLift Servers 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デプロイします。フリートのステータスが ACTIVE に達したら、希望する容量を増やして、希望する容量を増減してスケールダウンできます。

Amazon GameLift Servers スケーリング機能を使用して容量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 Servers フリートの容量を手動で設定する](#)」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、インスタンスの追加または削除が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。
 - この式を使用して、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します。

```
[Game sessions per instance] = [# of game server processes per game server container] * [# of game server container groups per instance]
```

コンテナアーキテクチャがゲームサーバーコンテナで1つのゲームサーバープロセスを同時に実行する場合、インスタンスあたりのゲームセッション数は、インスタンスあたりのゲームサーバーコンテナグループの数と等しくなります。

- インスタンスあたりのゲームサーバーコンテナグループの場合、[DescribeContainerFleet](#) を呼び出して `GameServerContainerGroupsPerInstance` または `MaximumGameServerContainerGroupsPerInstance` 値を取得します。
- 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の配置を処理する方法に応じて、マッチメイ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、ゲームセッションの配置を開始できます。
 - 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するには、次の式を使用します。

```
[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](#) **キャパシティ** を呼び出して、フリート内のゲームサーバーコンテナグループ

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できています。これらの値をゲームサーバーコンテナグループあたりのサーバープロセスの数で乗算します。

Amazon GameLift Servers のモニタリング

モニタリングは、およびその他の Amazon GameLift Servers 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な部分です。Amazon GameLift Servers でのメトリクスの主な用途は 3 つあります。システムの状態をモニタリングしてアラームを設定すること、ゲームサーバーのパフォーマンスと使用状況を追跡すること、および手動/自動スケーリングを使用して容量を管理することです。

AWS には、を監視し Amazon GameLift Servers、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 Servers コンソール – グラフィカルインターフェイスを使用して Amazon GameLift Servers リソースを管理し、ゲームホスティングアクティビティを追跡します。
- サーバーテレメトリメトリクス – SDK とプラグインを統合することで、Amazon GameLift Servers はゲームサーバーから直接カスタムメトリクスを出力できます。組み込みのパフォーマンス、ネットワーク、メモリ、タイミングデータとともに、独自のゲーム固有のメトリクスを定義および追跡できます。すべてのメトリクスを [Amazon Managed Service for Prometheus](#) に発行し、完全にカスタマイズ可能なダッシュボードを使用して Amazon Grafana でモニタリングできます。Amazon CloudWatch に公開して、他の AWS サービスと統合することもできます。テレメトリシステムは完全にカスタマイズ可能です。カスタム Prometheus クエリを作成して、組み込みメトリクスに加えて追加のメトリクスを取得できます。テクノロジースタックに固有の実装ガイドについては、「[the section called “サーバーテレメトリメトリクスによるモニタリング”](#)」を参照してください。
- Amazon CloudWatch – Amazon GameLift Servers メトリクスと、AWS 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。Amazon CloudWatch 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、メトリクスが指定のしきい値に達したときに通知または処置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を提供します。
- AWS CloudTrail – は、およびその他の AWS サービスに対して、AWS アカウントによって、またはアカウントに代わって行われたすべての API コール Amazon GameLift Servers 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。呼び出し元のユーザーとアカウント AWS、呼び出し元の送信元 IP アドレス、呼び出しの発生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。

モニタリングソース間のメトリクスの比較

Amazon GameLift Servers は、Amazon GameLift Servers コンソールフリートアクティビティメトリクス、サーバーテレメトリメトリクス、Amazon CloudWatch Amazon GameLift Servers メトリクスの 3 つの主要なソースを通じてメトリクスを提供します。各ソースの重複と固有の機能を理解することで、ニーズに合った適切なモニタリングアプローチを選択できます。

ソース別のメトリクスの可用性

次の表は、メトリクスカテゴリ別に整理された、さまざまなモニタリングソースで利用できるメトリクスを示しています。

インスタンスメトリクス

フリート容量とヘルスマニタリングのインスタンスレベルのメトリクス:

インスタンスメトリクスの可用性

メトリクス	コンソール	CloudWatch	テレメトリ
アクティブなインスタンス	✓	✓	✓
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	✓	✓	✓*
必要なインスタンス	✓	✓	
最大インスタンス	✓	✓	
最小インスタンス	✓	✓	
CPU 使用率		✓	✓
ネットワーク入出力		✓	✓
ディスク/ストレージの読み取り/書き込み		✓	✓
インスタンスの中断	✓	✓	
リサイクルされたインスタンス	✓	✓	

メトリクス	コンソール	CloudWatch	テレメトリ
置き換えられた異常なインスタンス		✓	

* テレメトリデータを使用したカスタム Prometheus クエリで使用できます。

ゲームサーバーのメトリクス

サーバープロセスとゲームセッションのメトリクス:

ゲームサーバーのメトリクスの可用性

メトリクス	コンソール	CloudWatch	テレメトリ
アクティブなゲームセッション	✓	✓	✓*
ゲームセッションのアクティブ化	✓	✓	
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	✓	✓	
同時にアクティブ化可能なゲームセッション	✓	✓	
ゲームセッションの中断	✓	✓	
アクティブなサーバープロセス			✓
正常なゲームサーバー			✓
クラッシュしたゲームセッション			✓

* テレメトリデータを使用したカスタム Prometheus クエリで使用できます。

プレイヤー使用状況メトリクス

プレイヤーセッションと同時接続ユーザーメトリクス:

プレイヤー使用状況メトリクスの可用性

メトリクス	コンソール	CloudWatch	テレメトリ
現在のプレイヤーセッション	✓		
プレイヤーセッションのアクティベーション	✓		
グローバル同時接続ユーザー (CCU)			✓
ロケーション同時接続ユーザー (CCU)			✓
ロケーションキャパシティ			✓

コンテナフリートのメトリクス

コンテナ固有のメトリクス (コンテナフリートの CloudWatch でのみ使用可能):

コンテナフリートメトリクスの可用性

メトリクス	コンソール	CloudWatch	テレメトリ
アクティブなコンテナグループ		✓	✓
アイドル状態のコンテナグループ		✓	✓
コンテナ CPU/メモリ使用率		✓	✓
コンテナネットワークトラフィック		✓	✓
コンテナストレージオペレーション		✓	✓

適切なモニタリングソースの選択

特定のニーズに基づいてモニタリングアプローチを選択します。

- サーバーテレメトリメトリクスを使用して、ゲームサーバーから包括的なパフォーマンスデータをモニタリングし、ゲームに固有のカスタムメトリクスを出力します。これらのメトリクスは、すべてのゲームエンジンとサーバー SDK のサーバーパフォーマンス、ネットワークアクティビティ、メモリ使用量、タイミングデータに関する詳細なインサイトを提供します。ゲームプレイ

イベント、ビジネスロジックのパフォーマンス、アプリケーション固有のデータポイントのカスタムメトリクスを定義できます。すべてのダッシュボードは完全にカスタマイズ可能で、カスタム Prometheus クエリを作成して、収集されたデータから追加のメトリクスを取得できます。詳細については、「[the section called “サーバーテレメトリメトリクスによるモニタリング”](#)」を参照してください。

- フリート管理、容量プランニング、一般的な運用上の監視には、Amazon GameLift Servers コンソールを使用します。コンソールには、フリートの正常性とプレイヤーアクティビティの統合ビューが表示されます。詳細については、「[the section called “ホスティングリソースの管理”](#)」を参照してください。
- Amazon CloudWatch を使用して、自動モニタリング、アラート、他の AWS サービスとの統合を行います。CloudWatch は、カスタムダッシュボードとアラームベースの自動化を有効にします。詳細については、「[the section called “CloudWatch を使用して監視する”](#)」を参照してください。
- [Amazon Managed Service for Prometheus](#) を使用して、PromQL クエリ機能を備えた高性能メトリクスの収集とストレージを実現します。Prometheus は、サーバーテレメトリメトリクスのスケーラブルな時系列データストレージを提供します。
- [Amazon Managed Grafana](#) を使用すると、高度な視覚化と完全にカスタマイズ可能なダッシュボードを利用できます。Grafana は、カスタマイズと拡張が可能な構築済みの GameLift ダッシュボードを提供し、複数のデータソースを持つ完全にカスタムのダッシュボードの作成をサポートします。カスタムクエリとビジュアライゼーションを構築して、ゲームにとって重要なメトリクスを追跡できます。
- 包括的なモニタリングには複数のソースを使用します。コンソールの監視、サーバーテレメトリメトリクス、自動アラートを組み合わせて、ゲームホスティングインフラストラクチャを完全に可視化します。

追加のモニタリング機能については、以下を使用することもできます。

- [the section called “API コールのログ作成”](#) – 監査およびコンプライアンスのための API コールおよび関連イベントを追跡します。
- [the section called “サーバーアクセスのログ記録”](#) – カスタムサーバーメッセージとゲームセッションログをキャプチャします。

トピック

- [the section called “ホスティングリソースの管理”](#)
- [the section called “CloudWatch を使用して監視する”](#)

- [the section called “サーバーテレメトリメトリクスによるモニタリング”](#)
- [the section called “API コールのログ作成”](#)
- [the section called “サーバーアクセスのログ記録”](#)

Amazon CloudWatch で Amazon GameLift Servers を監視する

Amazon CloudWatch Amazon GameLift Serversを使用してモニタリングできます。Amazon CloudWatch 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処理する AWS サービスです。Amazon CloudWatch これらの統計は 15 か月間保持され、Amazon GameLift Servers 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。

詳細なテレメトリメトリクスでサーバー側のオプザバビリティを強化するために、Amazon GameLift Servers は [Amazon Managed Service for Prometheus](#) と [Amazon Managed Grafana](#) によるモニタリングもサポートしています。これらのテレメトリメトリクスは、次のような詳細なサーバーパフォーマンスデータを提供します。

- サーバーのタイミング: サーバー差分時間 (サーバーのティックレートの整合性)、サーバーのティックレート (1 秒あたりの更新)、サーバーティック時間 (ティックあたりの処理時間)、サーバーワールドティック時間 (ゲームワールドの更新時間)
- ネットワークメトリクス: 接続 (合計ネットワーク接続数)、バイトとパケットのネットワーク I/O (データ転送統計)、パケットロス(送信失敗率)
- メモリメトリクス: 単位で表されるメモリ使用率 (合計メモリ消費量) とパーセンテージとしての物理メモリ使用量 (メモリ使用率)
- CPU 使用率: CPU 使用率 (ゲームサーバープロセス別のリソース使用率)

SDK またはプラグインに固有の実装ガイドについては、「[the section called “サーバーテレメトリメトリクスによるモニタリング”](#)」を参照してください。

以下の表は、Amazon GameLift Servers 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 Servers 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 マネジメントコンソール、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 Servers メトリクスのディメンション

Amazon GameLift Servers 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つのマッチメイ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イキング設定のメトリクスでのみ使用されます。
ConfigurationName-RuleName	マッチメイキング設定とマッチメイ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イキングルールのメトリクスでのみ使用されます。

ディメンション	説明
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処理します。
ContainerGroupType	コンテナフリートメトリクスをフィルタ処理して、フリートロケーションの GameServer Container Group メトリクスとフリートロケーションの PerInstance ContainerGroup メトリクスを区別します。このディメンションは、特定のコンテナフリートメトリクス(ContainerNetworkIn、ContainerStorageReadBytes など) に含まれ、さまざまなコンテナグループタイプのメトリクスを区別できます。

フリートの Amazon GameLift Servers メトリクス

Amazon 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 Servers ソリューションで使用されます。Amazon GameLift Servers 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。このメトリクスは自動スケーリングで使用できます。 単位: 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 Servers 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。自動スケーリング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。自動スケーリング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: 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。このメトリクスは自動スケーリングで使用できます。 単位: 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: 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	フリート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: 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 ($\text{IdleInstances} / \text{ActiveInstances}$ で計算)。このメトリクスは自動スケーリングで使用できます。 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	リサイクルおよび交換されたスポットインスタンスの数。Amazon GameLift Servers 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。このメトリクスは、マネージド EC2 フリートでのみ使用できます。 単位: 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
RecycledSpotInstances	リサイクルおよび交換されたスポットインスタンスの数。Amazon GameLift Servers 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。このメトリクスは、コンテナフリートでのみ使用できます。 単位: 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
InstanceSpotInterruptions	中断されたスポットインスタンス数。 単位: 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 Servers 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わた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 Servers 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わた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

メトリクス	説明
NetworkOut	EC2 メトリクス。Amazon GameLift Servers 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わた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
DiskReadBytes	EC2 メトリクス。Amazon GameLift Servers 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わた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 Servers 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わた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

メトリクス	説明
DiskReadOps	EC2 メトリクス。Amazon GameLift Servers 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わた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: カウント
DiskWriteOps	EC2 メトリクス。Amazon GameLift Servers 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わた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: 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。

メトリクス	説明
	単位: 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: 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
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プロセスのパーセント ($\text{HealthyServerProcesses} / \text{ActiveServerProcesses}$ 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

メトリクス	説明
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 Servers サービス主導の終了が含まれます。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: 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
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: 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
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: 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ます。このメトリクスは自動スケーリングで使用できます。 単位: 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

メトリクス	説明
	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。このメトリクスは自動スケーリングで使用できます。 単位: 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
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。このメトリクスは自動スケーリングで使用できます。  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 <code>ConcurrentActivatableGameSessions</code> 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: 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
<code>ConcurrentActivatableGameSessions</code>	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 <code>AvailableGameSessions</code> 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 <code>MaxConcurrentGameSessionActivations</code> 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 <code>AvailableGameSessions</code> と同一です。 単位: 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
<code>PercentAvailableGameSessions</code>	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 ($\text{AvailableGameSessions} / [\text{ActiveGameSessions} + \text{AvailableGameSessions} + \text{unhealthy server processes}]$ で計算)。このメトリクスは自動スケーリングで使用できます。 単位: パーセント 関連する CloudWatch 統計: 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: 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ポ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。このメトリクスは自動スケーリングで使用できます。 単位: 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: カウント 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum

コンテナフリートのメトリクス

コンテナフリートメトリクスは、コンテナグループのステータス、リソース使用率、パフォーマンスメトリクスなど、コンテナベースのゲームホスティングのモニタリングデータを提供します。

Note

テレメトリメトリクスソリューションを使用する場合、[Amazon Managed Grafana](#) を通じてコンテナフリートモニタリング用の拡張ダッシュボードを利用できます。セットアップ手順については、「[the section called “サーバーテレメトリメトリクスによるモニタリング”](#)」を参照してください。

メトリクス	説明
ActiveGameServerContainerGroups	現在アクティブで、ゲームセッションをホストする準備ができているコンテナグループ。 単位: 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
IdleGameServerContainerGroups	アクティブだが、現在ゲームセッションをホストしていないコンテナグループ。 単位: 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
PendingGameServerContainerGroups	起動プロセス中のコンテナグループ。 単位: 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
TerminatingGameServerContainerGroups	シャットダウン中のコンテナグループ。 単位: 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
UnhealthyGameServerContainerGroupsReplaced	ヘルスチェックの失敗によって置き換えられたコンテナグループ。 単位: 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
ContainerCPUUtilizationPerInstance	インスタンス上のすべてのコンテナの平均 CPU 利用率。 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

メトリクス	説明
ContainerCPUReservation	コンテナによって予約された CPU リソースの割合。 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
ContainerMemoryReservation	コンテナによって予約されたメモリリソースの割合。 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
ContainerMemoryUtilization	コンテナで使用されている予約メモリの割合。 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
ContainerNetworkIn	コンテナによって受信されたネットワークバイト数。 単位: バイト 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
ContainerNetworkOut	コンテナによって送信されたネットワークバイト数。 単位: バイト 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: ロケーション
ContainerStorageReadBytes	コンテナによってストレージから読み取られたバイト数。 単位: バイト 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: ロケーション
ContainerStorageWriteBytes	コンテナによってストレージに書き込まれたバイト数。 単位: バイト 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: ロケーション

DDoS 保護 (プレイヤーゲートウェイ) メトリクス

Amazon GameLift Servers プレイヤーゲートウェイメトリクス `PlayerGatewayMode` は、 が `ENABLED` または に設定されている場合、DDoS 保護レイヤーを通過する UDP トラフィックを追跡します `REQUIRED`。これらのメトリクスは、トラフィックパターンのモニタリング、潜在的な DDoS 攻撃の特定、リレーのパフォーマンスの追跡に役立ちます。

メトリクス	説明
PlayerGatewayPacketsIn	プレイヤーから受信され、DDoS 保護レイヤーを介してゲームサーバーに転送された UDP パケットの数。 単位: カウント 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: FleetId、ロケーション
PlayerGatewayBytesIn	プレイヤーから受信され、DDoS 保護レイヤーを介してゲームサーバーに転送されたバイト数。 単位: バイト 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: FleetId、ロケーション
PlayerGatewayPacketsOut	ゲームサーバーから受信し、DDoS 保護レイヤーを介してプレイヤーに転送された UDP パケットの数。 単位: カウント 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: FleetId、ロケーション
PlayerGatewayBytesOut	ゲームサーバーから受信し、DDoS 保護レイヤーを介してプレイヤーに転送されたバイト数。 単位: バイト 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum

メトリクス	説明
	ディメンション: FleetId、ロケーション
PlayerGatewayPacketsThrottled	DDoS 保護レイヤーによるレート制限のためにドロップされたインバウンドパケットの数。 単位: カウント 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: FleetId、ロケーション
PlayerGatewayBytesThrottled	DDoS 保護レイヤーによるレート制限によりドロップされたインバウンドバイト数。 単位: バイト 関連する CloudWatch 統計: Sum、Average、Minimum、Maximum ディメンション: FleetId、ロケーション
PlayerGatewayPlayerSessions	DDoS 保護レイヤーによって現在保護されているプレイヤーセッション。 単位: カウント 関連する CloudWatch 統計: Average、Minimum、Maximum ディメンション: FleetId、ロケーション

キューの Amazon GameLift Servers 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 Servers ソリューションで使用されます。Amazon GameLift Servers 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 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: カウント 関連する CloudWatch 統計: Average、Minimum、Maximum、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: カウント

メトリクス	説明
	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: 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: カウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum
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: カウント 関連する CloudWatch 統計: Sum

メトリクス	説明
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: 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: 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: カウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum
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: カウント 関連する CloudWatch 統計: Average、Minimum、Maximum、Sum

メトリクス	説明
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: 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: 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

マッチメイキングの Amazon GameLift Servers メトリクス

Amazon GameLift 名前空間には、マッチメイキング設定とマッチメイキングルールの FlexMatch アクティビティに関するメトリクスが含まれます。FlexMatch マッチメイキングは、マネージド Amazon GameLift Servers ソリューションで使用されます。Amazon GameLift Servers サービスは CloudWatch に毎分メトリクスを送信します。

マッチメイ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 Servers FlexMatch の仕組み」](#)を参照してください。

マッチメイキング設定

メトリクス	説明
CurrentTickets	マッチメイ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: カウント

メトリクス	説明
	関連する CloudWatch 統計: Average、Minimum、Maximum、Sum
MatchAcceptancesTimedOut	承諾を要求するマッチメイキング設定の場合、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: カウント 関連する CloudWatch 統計: Sum
MatchesAccepted	承諾を要求するマッチメイキング設定の場合、前回のレポート後に承諾されたマッチング案の数。 単位: 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: 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: カウント 関連する CloudWatch 統計: Sum
MatchesRejected	承諾を要求するマッチメイキング設定の場合、前回のレポート後に少なくとも 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: カウント 関連する CloudWatch 統計: Sum

メトリクス	説明
PlayersStarted	前回のレポート後に追加されたマッチメイキングチケットのプレイヤーの数。 単位: カウント 関連する CloudWatch 統計: Sum
TicketsFailed	前回のレポート後に、マッチメイキングリクエストが失敗した数。 単位: 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イキングリクエスト。 単位: 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イキングリクエストの数。 単位: カウント 関連する CloudWatch 統計: Sum
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イ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

メトリクス	説明
TimeToTicketCancel	前回のレポート前にキャンセルされたマッチメイ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イ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

マッチメイキングルール

メトリクス	説明
RuleEvaluationsPassed	前回のレポート後に、マッチメイ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: カウント 関連する CloudWatch 統計: Sum
RuleEvaluationsFailed	前回のレポート後に、マッチメイ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: カウント 関連する CloudWatch 統計: Sum

Amazon GameLift Servers の FleetIQ 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一部として、FleetIQ 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 Servers 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。

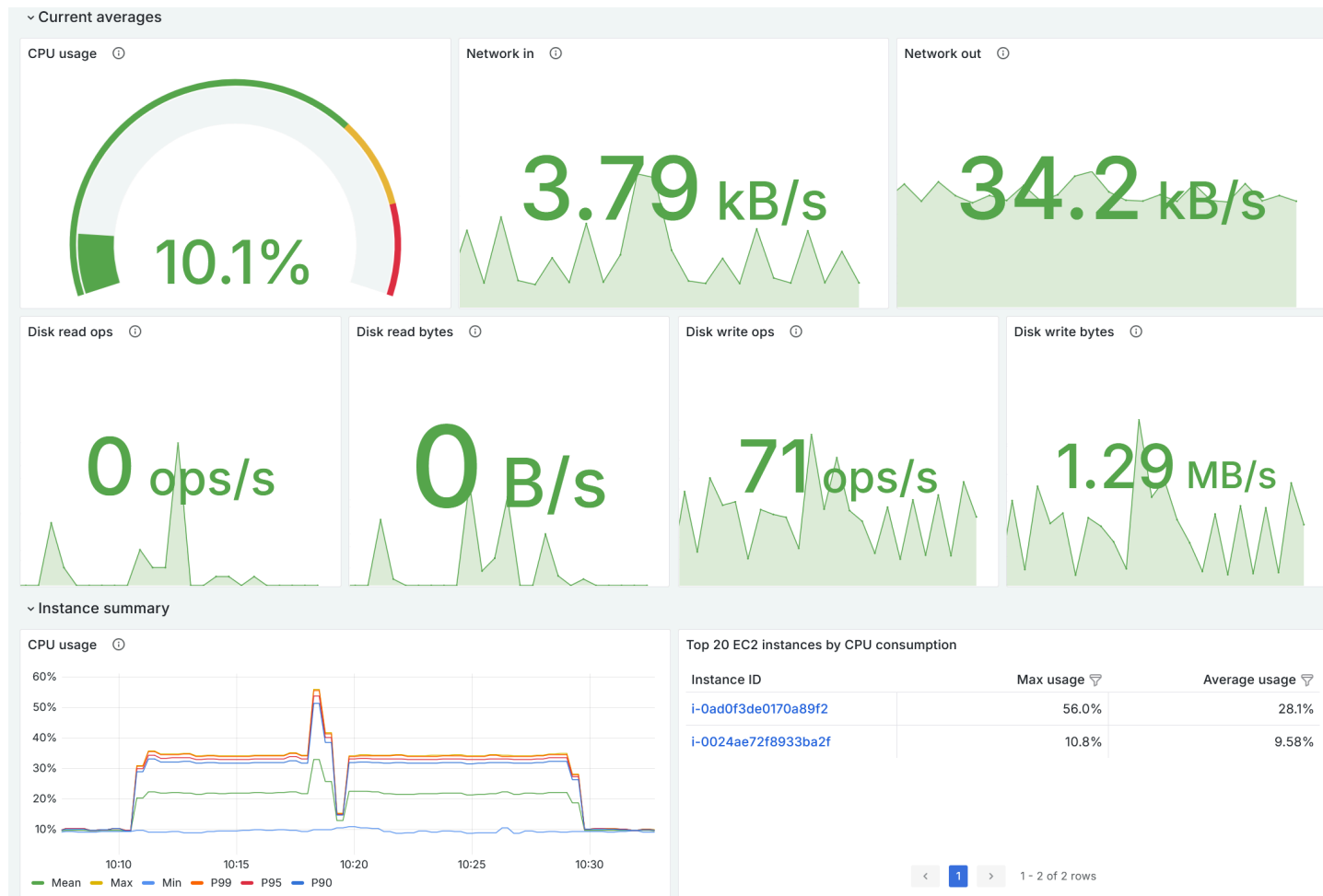
メトリクス	説明
AvailableGameServers	ゲーム実行に使用可能で、ゲームプレイに現在使用されていないゲームサーバーの数。この数には、クレーム済みであるが、まだ [利用可能] ステータスにあるゲームサーバーが含まれます。 単位: カウント 関連する Amazon CloudWatch 統計: Sum ディメンション: GameServerGroup
UtilizedGameServers	ゲームプレイに現在使用されているゲームサーバー。この数字には、[使用中] ステータスのゲームサーバーも含まれます。 単位: カウント 関連する Amazon CloudWatch 統計: Sum 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: カウント 関連する Amazon CloudWatch 統計: Sum

メトリクス	説明
	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: カウント 関連する Amazon 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、ゲームサーバー容量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 Amazon CloudWatch 統計: Average、Minimum、Maximum ディメンション: GameServerGroup
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: カウント 関連する Amazon CloudWatch 統計: Sum ディメンション: GameServerGroup、InstanceType

メトリクス	説明
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: カウント 関連する Amazon CloudWatch 統計: Sum ディメンション: GameServerGroup、InstanceType

サーバーテレメトリメトリクスによるモニタリング

Amazon GameLift Servers は、マネージド Amazon EC2 およびコンテナフリートで実行されているゲームサーバーのテレメトリメトリクスを収集して公開するように設定できます。これらのメトリクスは、サーバービルドでテレメトリコレクターをデプロイした後に使用可能になります。メトリクスシステムは、すべての SDK (C++、C#、Go)、すべてのプラグイン (Unreal、Unity)、および Amazon GameLift Servers Game Server Wrapper をサポートしています。 [Amazon Managed Service for Prometheus](#)、[the section called “CloudWatch を使用して監視する”](#)、および [Amazon Managed Grafana](#) ダッシュボードへのメトリクスデータフロー (視覚化に推奨)。



Amazon Managed Grafana または Amazon CloudWatch で構築済みのダッシュボードを使用して、ゲームサーバー全体のパフォーマンスをモニタリングします。

テレメトリメトリクスの利点

テレメトリメトリクスシステムには、次の 5 つの主な利点があります。

- ゲームエンジン固有のメトリクス - ゲームエンジンプラグイン(Unreal、Unity)は、ゲームパフォーマンスの最適化に不可欠なサーバーティック時間、フレームレート、エンジンレベルのリソース使用率などのエンジン固有のパフォーマンスメトリクスとネイティブに統合します。
- カスタムメトリクスのサポート - サーバー SDK 関数呼び出しを使用して独自のゲーム固有のメトリクスを定義および追跡し、使用するゲームに重要なカスタムゲームプレイイベント、ビジネスロジックのパフォーマンス、アプリケーション固有のデータポイントをモニタリングします。
- 自動収集 - メトリクスはテレメトリコレクターのデプロイ後に自動的に流れ、追加の計測は不要で、[Amazon Managed Service for Prometheus](#) および Amazon CloudWatch と直接統合されます。

- マルチレベルモニタリング - 容量とスケーリングのフリートレベルのメトリクス、リソース使用率のインスタンスレベルのメトリクス、パフォーマンス追跡用のゲームセッションメトリクス。
- ユニバーサル互換性 - Amazon GameLift Servers がサポートするすべての開発環境で動作し、すべてのサーバー SDK と統合され、ゲームエンジンプラグインでネイティブサポートされます。

Note

テレメトリメトリクスは、Amazon Linux 2023 または Windows を実行している Amazon GameLift Servers マネージド Amazon EC2 またはコンテナフリートで使用できます。

[開始する前に]

必要な AWS リソース

- AWS 用に設定された アカウント Amazon GameLift Servers。
- 以下の場所で実行されているマネージドフリート:
 - 対応するオペレーティングシステムを使用した Amazon EC2、または
 - Amazon Linux 2023 を使用したコンテナ
- 適切な IAM アクセス許可

IAM の要件

次の IAM アクセス許可は、対応するサービスを使用する場合にのみ必要です。

- [Amazon Managed Service for Prometheus](#) (メトリクスを Prometheus に発行する場合にのみ必要)
 - `aps:RemoteWrite` アクセス許可
 - Prometheus ワークスペースへのアクセス
- Amazon CloudWatch (Amazon CloudWatch にメトリクスを発行する場合にのみ必要)
 - `cloudwatch:PutMetricData` アクセス許可
 - メトリクス名前空間へのアクセス
- [Amazon Managed Grafana](#) (Grafana ダッシュボードを使用する場合にのみ必要)
 - `grafana:Read` アクセス許可
 - ダッシュボードアクセス用の SSO 設定

実装

開発環境に基づいて実装パスを選択します。

SDK 実装

SDK タイプ	SDK のセットアップ	カスタムメトリクス	API リファレンス
Go SDK	セットアップガイドを完了する	Go メトリクス API	Go アクションとデータ型
C# SDK	セットアップガイドを完了する	C# メトリクス API	C# アクションとデータ型
C++ SDK	セットアップガイドを完了する	C++ メトリクス API	C++ アクションとデータ型

プラグインの実装

プラグイン	プラグインのセットアップ	カスタムメトリクス	API リファレンス
非現実的	セットアップガイドを完了する	Unreal メトリクス API	Unreal アクションとデータ型
Unity	セットアップガイドを完了する	Unity メトリクス API	C# アクションとデータ型

実装ワークフロー

各実装は 2 ステップのプロセスに従います。

1. セットアップガイド (METRICS.md) を完了する — インフラストラクチャのデプロイ、AWS インフラストラクチャ設定、フリート設定、Grafana ダッシュボード設定。
2. API 実装ガイド (CUSTOM_METRICS.md) - 言語固有の SDK の使用、メトリクスタイプ、カスタムメトリクスの作成、高度な設定。

検証

1. [Amazon Managed Service for Prometheus](#) ワークスペースまたは Amazon CloudWatch コンソールで受信テレメトリデータをチェックして、メトリクスフローを検証します。
2. 構築済みのダッシュボードを使用して、[Amazon Managed Grafana](#) のダッシュボードの可視性を確認します。
3. モニタリングダッシュボードに表示されることを確認することで、カスタムメトリクスをテストします。

Note

実装が完了したら、このページに戻り、[使用可能なメトリクス](#) ページに移動します。

使用可能なメトリクス

メトリクスは、3 つのカテゴリに分類されます。

- 自動的に収集されるメトリクス
- SDK が提供するメトリクス
- カスタムメトリクス

自動メトリクス収集

これらのメトリクスにはコードの変更は必要ありません。

インスタンスメトリクス

メトリクスタイプ	説明	ユースケース
CPU	インスタンスあたりの使用率	リソースのモニタリング
メモリ	物理メモリの使用量と使用率	容量プランニング
ネットワーク I/O	送受信されたバイトとパケット	接続の正常性

メトリクスタイプ	説明	ユースケース
ディスク I/O	読み取り/書き込みオペレーションとスループット	ストレージパフォーマンス

フリートメトリクス

メトリクスタイプ	説明	ユースケース
アクティブなインスタンス	実行中のインスタンスの数	フリートスケーリング
ゲームセッション	アクティブセッションと使用可能なセッション	容量管理
クラッシュしたゲームセッション	クラッシュしたゲームセッション	エラーモニタリング

SDK が提供するメトリクス

コードに SDK 関数呼び出しが必要です。

サーバータイミングメトリクス

メトリクス	説明	実装
サーバー差分時間	現在のサーバーティックと以前のサーバーティックの間の時間の差。サーバーのティックレートの整合性を測定します。	GetDeltaTime() を呼び出す
サーバーのティックレート	サーバーが更新を処理する回数 (1 秒あたり) を示します	自動計算
サーバーのティック時間	サーバーが 1 つのティックまたは更新を処理するのにかかる時間	GetTickTime() を呼び出す

メトリクス	説明	実装
サーバーワールド ティック時間	サーバーがティックごとにゲームワールドを更新するのにかかる時間	GetWorldUpdateTime() を呼び出す

実装: エンジンに依存しない SDK (C++、C#、Go) の場合、計算されたタイミング値を使用してゲームループから SDK 関数を呼び出すことで、これらのメトリクスを実装します。エンジンプラグイン (Unreal、Unity) の場合、これらのメトリクスはエンジン統合によって自動的にキャプチャされます。

ネットワークメトリクス

メトリクス	説明	実装
Connections	サーバーで確立されたネットワーク接続の合計数	InitMetrics() の後で自動
ネットワーク I/O (バイト)	サーバーによって、ネットワーク経由で送受信されている合計バイト数。	InitMetrics() の後で自動
ネットワーク I/O (パケット)	サーバーによって送受信されるネットワークパケットの合計数	InitMetrics() の後で自動
パケットロス	送信中に失われているネットワークパケットの割合	InitMetrics() の後で自動

実装: SDK 関数呼び出しをネットワークライブラリと統合します。SDK は、さまざまなネットワーク実装のガイダンスを提供します。

プロセスメトリクス

メトリクス	説明	実装
CPU 使用率 (%)	ゲームサーバープロセスで使用されている CPU リソースの割合	InitMetrics() の後で自動

メトリクス	説明	実装
メモリ使用量 (単位)	サーバープロセスによって消費されるメモリの合計量	InitMetrics() の後で自動
物理メモリ使用量 (%)	現在使用されているサーバーの合計物理メモリの割合	InitMetrics() の後で自動
サーバーステータス	ゲームサーバーのヘルス状態	InitMetrics() の後で自動

実装: これらのメトリクスは、ゲームセッションプロセスごとに SDK によって自動的に収集されます。

プロセスごとのダッシュボード組織

プロセスごとのメトリクスは、次の 2 つの特殊なダッシュボードで使用できます。

- [サーバーパフォーマンス] ダッシュボード - サーバータイミング(デルタタイム、ティックレート、ティック時間、ワールドティック時間)、ネットワークメトリクス(接続、I/O バイト/パケット、パケット損失)、メモリ使用率、個々のゲームセッションの CPU 使用率が含まれます。
- [インスタンスパフォーマンス] ダッシュボード - インスタンスリソースの消費に最も寄与するプロセスを特定するのに役立つ [メモリ消費量が多い上位 N のゲームセッション] テーブルと [CPU 消費量が多い上位 N のゲームセッション] テーブルを備えています。ゲームセッションリンクをクリックすると、詳細なメトリクスをより詳細に調査できます。

プロセス毎のメトリクスのユースケース

プロセス/ゲームセッション毎のメトリクスは、次のモニタリングシナリオをサポートします。

- 詳細なパフォーマンス調査の詳細 - 特定のプロセスまたはゲームセッションが原因でホスト/インスタンスのパフォーマンスが低下した場合、プロセス毎のメトリクスは、CPU/メモリ消費が高い上位Nのゲームセッションテーブルを通じて問題の原因となったプロセスを特定するのに役立ちます。
- ゲームサーバーのクラッシュ調査 - ゲームセッションがクラッシュすると、これらのメトリクスは、クラッシュの原因がメモリ不足、CPU 過負荷、またはネットワーク帯域幅の問題であるかどうかを判断するのに役立ちます。

- プレイヤーが報告した問題の調査 - プレイヤーがゲームプレイ中に遅延や中断を報告すると、プロセス毎のメトリクスは CPU、メモリ、ネットワーク、ティック時間、またはワールド更新時間のボトルネックを特定するのに役立ちます。
- さまざまなビルドでのパフォーマンスの変化を特定する - ティック時間、ティックレート、ワールド更新時間メトリクスにより、開発者はさまざまなサーバービルドにおけるゲームパフォーマンスの変化を測定できます。
- ゲームプレイの遅延と速度を検出する - ティック時間、ティックレート、ワールド更新時間のメトリクスは、サーバーがゲームを更新する速度を反映し、カスタマーエクスペリエンスに直接影響します。
- ベンチマーク - プレイヤー数、ゲームモード、その他の変数などの要因に基づいて、さまざまなゲームシナリオがサーバーのパフォーマンスにどのように影響するかを特定します。

ダッシュボードの整理

メトリクスは、さまざまなモニタリングシナリオ用に [Amazon Managed Grafana](#) の特殊なダッシュボードに整理されています。使用可能なダッシュボードは、フリートタイプによって異なります。

EC2 フリートダッシュボード

- [EC2 フリースの概要] ダッシュボード - 高レベルのフリート容量、スケーリングインサイト、同時接続プレイヤー (CCU)、インスタンス、プレイヤー容量、クラッシュしたゲームセッション。
- [インスタンス概要] ダッシュボード - 平均 CPU、メモリ、ネットワーク、ディスク使用率など、すべてのインスタンスで集約されたホストレベルのメトリクス。
- [インスタンスパフォーマンス] ダッシュボード - リソース集約型プロセスを識別するための [メモリ消費量が多い上位 N のゲームセッション] テーブルと [CPU 消費量が多い上位 N のゲームセッション] テーブルを持つ個々のインスタンスの詳細なメトリクス。
- [サーバーパフォーマンス] ダッシュボード (EC2) - EC2 インスタンス上の個々のゲームセッションのゲームループタイミング、ネットワークパフォーマンス、メモリ、CPU メトリクス。

コンテナフリートダッシュボード

- [コンテナフリートの概要] ダッシュボード - CPU 予約、メモリ使用率、コンテナグループのステータスなど、コンテナフリートのリソース使用率の高レベルの概要。
- コンテナパフォーマンスダッシュボード - CPU 使用率、メモリ使用率、ネットワーク I/O、ストレージパフォーマンスなど、特定の ECS タスク内の個々のコンテナの詳細なメトリクス。

- [サーバーパフォーマンス] ダッシュボード (コンテナ) - コンテナ内の個々のゲームセッションのゲームループタイミング、ネットワークパフォーマンス、メモリ、CPU メトリクス。

ダッシュボードの詳細な情報と使用手順については、「[the section called “ダッシュボードの整理と使用状況”](#)」を参照してください。

仕組み

テレメトリメトリクスシステムは、ゲームサーバーから視覚化ダッシュボードへのシンプルな 4 段階のデータフローに従います。

コレクション: GameLift Server SDK またはプラグインと統合されたゲームサーバーは、同じインスタンスで実行されているローカルテレメトリコレクターにメトリクスを自動的に出力します。SDK は、コードで定義した自動メトリクス (サーバーライフサイクル、リソース使用量) とカスタムメトリクスの両方をキャプチャします。

処理: テレメトリコレクターは、ゲームサーバーからメトリクスを集約し、インスタンスレベルのパフォーマンスデータ (CPU、メモリ、ネットワーク、ディスク使用量) と組み合わせます。これにより、ゲームのパフォーマンスと基盤となるインフラストラクチャの両方を完全に把握できます。

ストレージ: 処理されたメトリクスは、選択したメトリクスウェアハウス - 高性能時系列ストレージ用の [Amazon Managed Service for Prometheus](#)、AWS サービス統合用の Amazon CloudWatch、またはその両方にエクスポートされます。すべてのデータ転送は認証され、暗号化されます。

視覚化: [Amazon Managed Grafana](#) はメトリクスウェアハウスに接続して、構築済みの GameLift ダッシュボードを表示します。これらのダッシュボードには、ゲームホスティングインフラストラクチャのモニタリングとトラブルシューティングに役立つフリートの概要、サーバーパフォーマンスの詳細、コンテナモニタリングビューが表示されます。

Note

ゲームサーバーとテレメトリコレクター間のすべてのメトリクス送信は、セキュリティのためにインスタンス上でローカルに行われます。コレクターのみが、適切な認証を使用して AWS サービスと通信します。

ダッシュボードの整理と使用状況

[Amazon Managed Grafana](#) の包括的なダッシュボードでメトリクスを表示します。使用可能なダッシュボードは、フリートタイプによって異なります。

フリートタイプ別のダッシュボードの可用性

次の表は、各フリートタイプで利用できるダッシュボードを示しています。

ダッシュボード	フリートタイプ	説明
EC2 フリートの概要	EC2 Fleet	同時接続プレイヤー (CCU)、インスタンス、プレイヤー容量に関する情報を表示します。
インスタンスの概要	EC2 Fleet	すべてのフリートインスタンスの平均 CPU、メモリ、ネットワーク使用率を表示します。
インスタンスのパフォーマンス	EC2 Fleet	個々のインスタンスの詳細なメトリクス (CPU、メモリ、ディスク、ネットワーク) を表示します。
コンテナフリートの概要	コンテナフリート	マネージドコンテナフリート内のすべてのコンテナの平均リソース使用率を表示します。
コンテナパフォーマンス	コンテナフリート	特定の ECS タスク内の個々のコンテナの詳細なメトリクスを表示します
サーバーパフォーマンス	[Both] (両方)	指定されたゲームサーバープロセス (EC2 フリートとコンテナフリートの個別のバージョン) のネットワーク、メモリ、ランタイムパフォーマンスを表示します。

マネージド EC2 フリート:

- [EC2 フリートの概要] は、高レベルのフリート容量とスケーリングに関するインサイトを提供します。
- [インスタンスの概要] および [インスタンスパフォーマンス] ダッシュボードをホストレベルのモニタリングに使用します。
- システムレベルの可視性のためにホストメトリクスレシーバーを介して収集されるメトリクス。
- EC2 インスタンスリソースの使用率とパフォーマンスに焦点を当てます。
- [サーバーパフォーマンス (EC2)] は、基盤となるインフラストラクチャとは無関係にゲームサーバーアプリケーションのメトリクスをモニタリングします。

マネージドコンテナフリート:

- ECS タスクとコンテナレベルのモニタリングには、[コンテナフリートの概要] ダッシュボードと [コンテナパフォーマンス] ダッシュボードを使用します。
- コンテナ化されたワークロードを可視化するために ECS Container Receiver を介して収集されたメトリクス。
- タスクレベルの集約とコンテナリソースの分離に焦点を当てます。
- [サーバーパフォーマンス (コンテナ)] は、基盤となるインフラストラクチャとは無関係にゲームサーバーアプリケーションのメトリクスをモニタリングします。

[EC2 フリートの概要] ダッシュボード

このダッシュボードでは、フリートの使用率と容量についてグローバルおよびロケーション別に高レベルの概要を説明します。ゲームサーバーの停止、起動、クラッシュの数と、正常なゲームサーバーの割合を示すグラフが含まれています。FleetID とロケーションでフィルタリングできます。

フリートの概要メトリクス

次の表は、[フリート概要] ダッシュボードで使用できるメトリクスを示しています。

ダッシュボード	メトリクス名	定義
グローバル CCU と容量	CCU	すべてのロケーションのすべてのインスタンスで、すべてのゲームサーバーの同時接続ユーザー数を表示します。

ダッシュボード	メトリクス名	定義
	グローバル CCU	すべてのインスタンスのすべてのゲームサーバーの同時接続ユーザー数をグローバルに表示します。
	アクティブなインスタンス	フリート内でアクティブなインスタンスの総数を表示します
	アクティブなプロセス	ゲームセッションをホストする準備ができているアクティブなゲームサーバープロセスの数を表示します。
	ゲームサーバーの起動	フリート全体で開始されたゲームセッションの数を表示します。
グローバル CCU と容量	正常なゲームサーバー	フリート全体で Amazon GameLift Servers に正常を報告するゲームサーバーの平均パーセンテージを示します。
	クラッシュしたゲームセッション	クラッシュしたゲームセッションのゲームセッション ID を表示します。ゲームセッション ARN のリンクをクリックして、[サーバーパフォーマンス] ダッシュボードに移動します。
ロケーション CCU	ロケーションプレイヤー	ロケーション内のすべてのインスタンスのすべてのゲームサーバーを含む、ロケーション内の同時接続プレイヤーの数を表示します。
	ロケーションキャパシティ	ロケーションの容量使用率 (%) と、そのロケーションで使用されているゲームサーバーの割合を表示します。

 Note

CCU メトリクスには、ゲームサーバーコードへの実装が必要です。これらのメトリクスは自動的に収集されないため、アプリケーションによって実装および報告する必要があります。

[インスタンスの概要] ダッシュボード

このダッシュボードは、フリート内のすべてのインスタンスに集約されたホストレベルのメトリクスを提供します。現在の平均は、インスタンスの全体的な状態を示します。パフォーマンスが低下したら、CPU 使用率、メモリ消費量、ネットワークとディスクの消費量にボトルネックがないか確認します。FleetID とロケーションでフィルタリングできます。

インスタンスの概要メトリクス

次の表は、[インスタンスの概要] ダッシュボードで利用できるメトリクスを示しています。

ダッシュボード	メトリクス名	定義
インスタンスの概要	平均 CPU 使用率	インスタンス概要は、インスタンスごとの平均リソース使用量を追跡します。すべてのフリートインスタンスで使用されている CPU リソースの平均割合。使用率の高いパターンでは、個々のインスタンスメトリクスとリソース分散をチェックする必要があります
	ピーク CPU 使用率	期間中にフリート内の任意のインスタンスで記録された CPU 使用率の最大値
	インスタンス別の CPU 使用率	フリート内の各インスタンスの個々の CPU 使用率。リソースホットスポットの特定を可能にします。
インスタンスの概要	平均メモリ使用量	すべてのフリートインスタンスで使用されている RAM の平均パーセンテージ。使用率の高いパターンでは、個々のインスタンスメトリクスとメモリ消費量などのリソース分散をチェックする必要があります。
	ピークメモリ使用量	期間中にフリート内の任意のインスタンスで記録されたメモリ使用率の最大値
	インスタンス別のメモリ使用率	フリート内の各インスタンスの個々のメモリ使用率。メモリを大量に消費するワークロードの特定に役立ちます。

ダッシュボード	メトリクス名	定義
ネットワークの概要	ネットワーク I/O (バイト)	ネットワーク概要には、平均インスタンス接続が反映されます。すべてのフリートインスタンスの平均ネットワークトラフィック量(送受信)。レイテンシーや接続の問題については、ネットワーク I/O やパケットレートなどの個々のインスタンスメトリクスを確認してください。
	ネットワーク I/O(パケット)	すべてのフリートインスタンスの平均ネットワークパケットレート(送受信)
	インスタンス別のネットワーク使用状況	各インスタンスの個々のネットワーク使用率。ネットワークのボトルネックを特定するのに役立ちます。
ディスクの概要	ディスク I/O オペレーション	ディスクの概要は、インスタンスの平均ディスクパフォーマンスを示します。すべてのフリートインスタンスの平均ディスク読み取り/書き込みオペレーション。応答時間が遅い場合は、ディスクオペレーション、I/O 待機時間、保留中オペレーションを個々のインスタンスレベルで調べることが推奨されます。
	ディスク I/O スループット	すべてのフリートインスタンスの平均ディスク読み取り/書き込みスループット
	インスタンス別のディスク使用率	各インスタンスの個々のディスク使用率。ストレージパフォーマンスの問題の特定に役立ちます。

Note

インスタンスレベルのメトリクスは、ホストメトリクスレシーバーを介して収集され、フリートのインフラストラクチャパフォーマンスをシステムレベルで可視化します。このダッ

シユボードを使用して、フリート全体のヘルス傾向を特定し、パフォーマンスの問題が検出されたときに個々のインスタンスにドリルダウンします。

[インスタンスパフォーマンス] ダッシュボード

このダッシュボードには、個々のインスタンスの詳細なパフォーマンスメトリクスが表示されます。現在の平均は、インスタンス全体の健全性を示します。パフォーマンスが低下したら、CPU 使用率、メモリ消費量、ファイルシステムの消費量にボトルネックがないか確認します。インスタンスリソースの消費に最も寄与するプロセスを特定するのに役立つ [メモリ消費量が多い上位 N のゲームセッション] テーブルと [CPU 消費量が多い上位 N のゲームセッション] テーブルを備えています。ゲームセッションリンクをクリックすると、詳細なメトリクスをより詳細に調査できます。特定のインスタンス ID によってフィルタリングできます。

インスタンスパフォーマンスメトリクス

次の表は、[インスタンスパフォーマンス] ダッシュボードで利用できるメトリクスを示しています。

ダッシュボード	メトリクス名	定義
ロードの概要	メモリ消費量が多い 上位 N のゲームセッション	ロードサマリーは、ゲームサーバーごとのリソース使用状況を追跡します。このインスタンスで最も多くのメモリリソースを消費するゲームセッションのランクリスト。使用率の高いパターンでは、個々のセッションメトリクスとリソース分散をチェックする必要があります。ゲームセッションリンクをクリックすると、詳細なメトリクスをより詳細に調査できます。
	CPU 消費量が多い 上位 N のゲームセッション	このインスタンスで最も CPU リソースを消費するゲームセッションのランクリスト。使用率の高いパターンでは、個々のセッションメトリクスと CPU やメモリなどのリソース分散をチェックする必要があります。ゲームセッションリンクをクリックすると、詳細なメトリクスをより詳細に調査できます。

ダッシュボード	メトリクス名	定義
現在の平均	ゲームセッションあたりの CPU 使用率	アクティブなゲームセッションごとのリソース消費量を示す個々の CPU 使用率の内訳
	ゲームセッションあたりのメモリ使用率	アクティブなゲームセッションごとのリソース消費量を示す個々のメモリ使用率の内訳
	インスタンス CPU 使用率	時間の経過に伴う選択したインスタンスの全体的な CPU 使用率
	インスタンスメモリの使用率	時間の経過に伴う選択したインスタンスの全体的なメモリ使用率
	インスタンスファイルシステムの使用率	選択したインスタンスのファイルシステムの消費量。ストレージ容量の問題を特定するのに役立ちます。
Network	インスタンスネットワーク I/O	ネットワークはインスタンス接続を反映します。選択したインスタンスのネットワークトラフィック量とパケットレート。レイテンシーまたは接続の問題については、ネットワーク I/O、パケットレート、エラー数を調査します。
	ネットワーク接続	選択したインスタンスへのアクティブなネットワーク接続の数
	ネットワークエラー数	接続の問題を特定するためのネットワークエラー統計
ディスク	ディスク I/O オペレーション	ディスクはディスクのパフォーマンスを示します。選択したインスタンスのディスク読み取り/書き込みアクティビティと使用率。応答時間が遅い場合は、ディスクオペレーション、I/O 待機時間、保留中オペレーションを調べるのが推奨されます。
	ディスク I/O 待機時間	ディスクオペレーションが完了するまでの待機にかかった時間

ダッシュボード	メトリクス名	定義
	保留中のディスクオペレーション	処理を待っているディスクオペレーションの数

Note

[メモリ消費が多い上位 N のゲームセッション] テーブルと [CPU 消費が高い上位 N のゲームセッション] テーブルは、インスタンス全体のパフォーマンスに影響を与える可能性のあるパフォーマンスのボトルネックとリソースを大量に消費するプロセスを特定する上で不可欠です。これらのランキングにより、問題のあるゲームセッションをすばやく特定して、さらに調査できます。

[コンテナフリートの概要] ダッシュボード

このダッシュボードには、コンテナフリートのリソース使用率と容量の高レベルの概要が表示されます。CPU 予約、メモリ使用率、コンテナグループのステータスなど、マネージドコンテナフリート内のすべてのコンテナの平均リソース使用率が表示されます。FleetID と ロケーションでフィルタリングできます。

コンテナフリートの概要メトリクス

次の表は、[コンテナフリートの概要] ダッシュボードで利用できるメトリクスを示しています。

ダッシュボード	メトリクス名	定義
コンテナグループのステータス	アクティブなゲームサーバーコンテナグループ	現在アクティブで、ゲームセッションをホストする準備ができているコンテナグループ
	アイドルゲームサーバーコンテナグループ	アクティブだが、現在ゲームセッションをホストしていないコンテナグループ
	保留中のゲームサーバーコンテナグループ	起動プロセス中のコンテナグループ

ダッシュボード	メトリクス名	定義
	終了中のゲームサーバーコンテナグループ	シャットダウン中のコンテナグループ
リソース使用率	コンテナ CPU 使用率	フリート内のすべてのコンテナの平均 CPU 使用率
	コンテナメモリ使用率	フリート内のすべてのコンテナの平均メモリ使用率
	コンテナ CPU 予約	フリート全体のコンテナによって予約された CPU リソースの割合
ネットワークアクティビティ	コンテナネットワーク入力	フリート全体のコンテナが受信した平均ネットワークバイト数
	コンテナネットワーク出力	フリート全体のコンテナが送信した平均ネットワークバイト数

Note

コンテナフリートメトリクスは ECS Container Receiver を介して収集され、タスクレベルの集約とコンテナリソースの分離に焦点を当て、コンテナ化されたワークロードの可視性を提供します。

コンテナパフォーマンスダッシュボード

このダッシュボードには、特定の ECS タスク内の個々のコンテナの詳細なパフォーマンスメトリクスが表示されます。CPU 使用率、メモリ使用率、ネットワーク I/O、ストレージパフォーマンスなど、個々のコンテナの詳細なメトリクスが表示されます。特定のコンテナ ID または ECS タスクでフィルタリングできます。

コンテナパフォーマンスメトリクス

次の表は、[コンテナパフォーマンス] ダッシュボードで利用できるメトリクスを示しています。

ダッシュボード	メトリクス名	定義
リソースパフォーマンス	コンテナ CPU 使用率	選択したコンテナの継続的な CPU 使用率
	コンテナメモリ使用率	選択したコンテナの継続的なメモリ使用率
	コンテナメモリの予約	選択したコンテナによって予約されたメモリリソースの割合
ネットワークパフォーマンス	コンテナネットワーク I/O	選択したコンテナのネットワークトラフィックボリューム (入力/出力バイト数)
	ネットワーク接続	選択したコンテナのアクティブなネットワーク接続の数
ストレージパフォーマンス	コンテナストレージの読み取り/書き込み	選択したコンテナのストレージ読み取りおよび書き込みアクティビティ
	コンテナストレージ I/O オペレーション	選択したコンテナによって実行されたストレージ I/O オペレーションの数

Note

コンテナパフォーマンスメトリクスは、ECS タスク内の個々のコンテナリソースの消費量とパフォーマンス特性を詳細に可視化します。

[サーバーパフォーマンス] ダッシュボード

[サーバーパフォーマンス] ダッシュボードには、個々のゲームセッションのサーバーのタイミング、ネットワークアクティビティ、メモリ、CPU 使用率に関連するメトリクスが表示されます。ゲームセッション ID でフィルタリングし、メトリクスを Amazon CloudWatch または [Amazon Managed Grafana](#) に直接エクスポートできます。

サーバーパフォーマンスメトリクス

次の表は、[サーバーパフォーマンス] ダッシュボードで利用できるメトリクスを示しています。

ダッシュボード	メトリクス名	定義
サーバーのタイミング	サーバー差分時間	現在のサーバーティックと以前のサーバーティックの間の時間の差。これは、サーバーのティックレートの整合性の尺度です。
	サーバーのティックレート	サーバーが更新を処理する回数 (1 秒あたり) を示します
	サーバーのティック時間	サーバーが 1 つのティックまたは更新を処理するのにかかる時間
	サーバーワールドティック時間	サーバーがティックごとにゲームワールドを更新するのにかかる時間
Network	Connections	サーバーで確立されたネットワーク接続の合計数。
	ネットワーク I/O (バイト)	サーバーによって、ネットワーク経由で送受信されている合計バイト数。
	ネットワーク I/O (パケット)	サーバーによって送受信されるネットワークパケットの合計数
	パケットロス	送信中に失われているネットワークパケットの割合
メモリ	メモリ使用量 (単位)	サーバープロセスによって消費されるメモリの合計量
	物理メモリ使用量 (%)	現在使用されているサーバーの合計物理メモリの割合
CPU 使用率	CPU 使用率 (%)	ゲームサーバープロセスで使用されている CPU リソースの割合

一般的なモニタリングシナリオ

詳細なパフォーマンス調査の詳細

シナリオ: 特定のプロセスまたはゲームセッションが原因でホスト/インスタンスのパフォーマンスが低下している

調査ステップ:

- [インスタンスパフォーマンス] ダッシュボードにアクセスします。
- [メモリ消費量が多い上位 N のゲームセッション] テーブルを確認して、インスタンスのメモリ消費量に最も寄与するプロセスを特定します。
- [CPU 消費量が多い上位 N のゲームセッション] テーブルを確認して、インスタンスの CPU 消費量に最も寄与するプロセスを特定します。
- ゲームセッションリンクをクリックすると、詳細なメトリクスをより詳細に調査できます。
- サーバーのタイミング(サーバーデルタタイム、サーバーティックレート、サーバーティック時間、サーバーワールドティック時間)を分析して、パフォーマンスのボトルネックを特定します。

ゲームサーバーのクラッシュ調査

シナリオ: ゲームセッションがクラッシュし、根本原因を特定する必要がある

調査ステップ:

- クラッシュしたゲームセッションの [サーバーパフォーマンス] ダッシュボードにアクセスします。
- [メモリ使用率](単位) と [物理メモリ使用率](%) をチェックして、クラッシュがメモリ不足によるものかどうかを判断します。
- [CPU 使用率](%) を確認して、CPU オーバーロードがクラッシュの原因となったかどうかを特定します。
- [ネットワーク I/O] (バイト) と [ネットワーク I/O] (パケット) を分析して、ネットワーク帯域幅の問題がクラッシュの原因となったかどうかを確認します。
- [パケットロス] の割合を調べて、ネットワーク関連の問題を特定します。

プレイヤーが報告した問題を調査する

シナリオ: ゲームプレイ中にプレイヤーが遅延または中断を報告する

調査ステップ:

- 影響を受けたゲームセッションの [サーバーパフォーマンス] ダッシュボードにアクセスします。
- [サーバーティック時間] と [サーバーワールドティック時間] を確認して、ゲーム更新の遅延を特定します。
- [サーバーティックレート] をチェックして、サーバーの更新頻度が一貫していることを確認します。
- [CPU 使用率 (%)] を分析して、処理のボトルネックを特定します。
- [メモリ使用状況] メトリクスを確認して、メモリ関連のパフォーマンスの問題を特定します。
- [ネットワーク I/O] メトリクスと [パケットロス] をチェックして、ネットワークのボトルネックを特定します。

さまざまなゲームサーバービルドのパフォーマンスの変化を特定する

シナリオ: 異なるサーバービルド間でゲームパフォーマンスがどのように変化するかを測定する場合

調査ステップ:

- 異なるビルド間で [サーバーティック時間] メトリクスを比較して、処理効率の変化を測定します。
- ビルド間で [サーバーティックレート] の一貫性を分析して、パフォーマンスのリグレッションを特定します。
- [サーバーワールドティック時間] を確認して、ゲームワールドの更新パフォーマンスの変化を測定します。
- ビルド間で [メモリ使用率] パターンを比較して、メモリ最適化の改善またはリグレッションを特定します。
- [CPU 使用率] の傾向をモニタリングして、計算効率の変化を評価します。

ゲームプレイの遅延と速度を検出する

シナリオ: サーバーの応答性とゲーム更新速度をモニタリングする必要がある

調査ステップ:

- [サーバーティック時間] をモニタリングして、サーバーが各更新サイクルを処理する速度を測定します。

- [サーバーティックレート] を追跡して、1 秒あたりの一貫したゲーム状態の更新を確保します。
- [サーバーワールドティック時間] を分析して、カスタマーエクスペリエンスに直接影響するゲームワールドの更新速度を測定します。
- [サーバー差分時間] のバリエーションのアラートを設定して、サーバーのパフォーマンスの不整合を検出します。

さまざまなゲームシナリオのベンチマーク

シナリオ: さまざまなゲームシナリオがサーバーのパフォーマンスにどのように影響するかを特定したい

調査ステップ:

- 異なるプレイヤー数間でサーバーパフォーマンスメトリクスを比較して、スケーリングの影響を理解します。
- [サーバーティック時間] と [CPU 使用率] メトリクスを使用して、ゲームモード間のパフォーマンスの違いを分析します。
- さまざまなゲームシナリオで [メモリ使用率] パターンをモニタリングして、リソースを大量に消費する機能を特定します。
- [ネットワーク I/O] メトリクスを追跡して、さまざまなゲームプレイシナリオの帯域幅要件を理解します。
- [インスタンスパフォーマンス] ダッシュボードを使用して、リソースを最も多く消費するゲームセッションを生成するゲームシナリオを特定します。

リソース使用率の高いレスポンス

シナリオ: 異常なリソースの急増 (CPU >85%、メモリ >90%)

調査ステップ:

影響を受けるリソースを特定する

- DescribeGameSessionDetails API を使用します。
- 必要に応じてステータスでフィルタリングします。
- 影響を受けるインスタンスを文書化します。

リソースの使用状況を分析する

- [インスタンスの概要] ダッシュボードを確認します。
- フリート全体の使用率を比較します。
- 履歴パターンを確認します。

ゲームサーバーの影響をモニタリングする

- [サーバーパフォーマンス] メトリクスを確認します。
- ティック時間とパケットロスを確認します。
- メモリリークをモニタリングします。

解決の手順

- セッションログをダウンロードします。
- ビルドの問題に対処します。
- 改善をモニタリングします。

ゲームサーバーのクラッシュ分析

シナリオ: フリート間の複数のエラーステータスゲームセッション

調査ステップ:

初期評価

- [フリートの概要] ダッシュボードにアクセスします。
- クラッシュしたセッションテーブルを確認します。
- タイミング/場所のパターンを書き留めます。

分析の実行

- サーバーのタイミングメトリクスを確認します。
- リソース使用率を確認します。
- ネットワークパフォーマンスをモニタリングします。

インフラストラクチャの確認

- フリートの容量を確認します。
- インスタンスヘルスを確認します。
- スケーリングポリシーを確認します。

解決パス

- サーバーログを分析します。
- コードの最適化を確認します。
- 修正を実装します。

フリート容量の最適化

シナリオ: ゲームの起動またはベンチマーク調査

分析ステップ:

リソース使用率

- 場所によるフィルタリングを行います。
- P50/P95/P99 メトリクスを確認します。
- 使用率パターンを分析します。

インスタンスタイプの分析

- タイプ別にパフォーマンスを比較します。
- スケーリング候補を特定します。
- 使用率パターンを文書化します。

最適化アクション

- スケーリングポリシーを調整します。
- インスタンスタイプを変更します。
- フリート設定を更新します。

トラブルシューティングガイド

一般的な問題と解決手順

欠落しているまたは不完全なメトリクス

症状

- ダッシュボードにメトリクスが表示されない。
- メトリクスが部分的にしか収集されない。
- メトリクスの更新が遅延している。

解決の手順

A. コレクターのステータスを確認する

systemd サービスを確認します。

```
sudo systemctl status gamelift-telemetry-collector
```

コレクターログを確認します。

```
sudo journalctl -u gamelift-telemetry-collector
```

- コレクター設定を確認します。

B. IAM アクセス許可の検証

- インスタンスロールのアクセス許可を確認します。
- 必要なポリシーを確認します。
 - `aps:RemoteWrite`
 - `cloudwatch:PutMetricData`
- ロールの信頼関係を検証します。

C. ネットワーク接続

- エンドポイントアクセスを確認します。

- セキュリティグループルールを確認します。
- ネットワーク ACL を確認します。

認証エラー

症状

- SigV4 認証の失敗。
- 「アクセスが拒否されました」のメッセージ。
- 認証情報の更新の問題。

解決の手順

A. SigV4 認証

- 一時的な認証情報を確認します。
- 認証情報のローテーションを確認します。
- インスタンスプロファイルを検証します。

B. AMP アクセス

- ワークスペース設定を確認します。
- リモート書き込み URL を確認します。
- IAM ロールのバインディングを確認します。

ダッシュボードの問題

症状

- ダッシュボードが空である。
- データポイントが欠落している。
- 認証の失敗。

解決の手順

A. データソースの設定

- Prometheus 接続を確認します。
- Amazon CloudWatch との統合を確認します。
- データソースの権限機能をテストします。

B. Grafana アクセス

- SSO 設定を確認します。
- 必要に応じて 2FA セットアップを確認します。
- ユーザー権限を確認します。

Windows 固有の問題

症状

- サービススタートアップの失敗。
- メトリクス収集のギャップ。
- アクセス許可エラー。

解決の手順

- Windows サービスのステータスを確認します。
- Windows イベントログを確認します。
- コレクター設定を確認します。
- Windows 固有のパスを検証します。

を使用した Amazon GameLift Servers API コールのログ記録 AWS CloudTrail

Amazon GameLift Servers は、 のユーザー AWS CloudTrail、ロール、または のサービスによって実行されたアクションを記録する AWS サービスである と統合されていますAmazon GameLift

Servers。CloudTrail は、Amazon GameLift Servers 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。キャプチャされたコールには、Amazon GameLift Servers コンソールのコールと、Amazon GameLift Servers API オペレーションへのコードのコールが含まれます。証跡を作成する場合は、Amazon GameLift Servers 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。証跡を設定しない場合でも、「CloudTrail」コンソールの「イベント履歴」で最新のイベントを表示できます。CloudTrail が収集した情報を使用して、Amazon GameLift Servers に対して行われた要求、要求が行われた IP アドレス、要求を行った人、要求が行われた日時、および追加の詳細を判別できます。

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 での Amazon GameLift Servers 情報

CloudTrail は、アカウントの作成 AWS アカウント 時に 有効になります。でアクティビティが発生すると Amazon GameLift Servers、そのアクティビティはイベント履歴の他の AWS サービスイベントとともに CloudTrail イベントに記録されます。で最近のイベントを表示、検索、ダウンロードできます AWS アカウント。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 Servers のイベントなど、AWS アカウントのイベントの継続的な記録に対して、追跡を作成します。証跡により、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 パーティション内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 ログで収集されたイベントデータをさらに分析して処理するように他の AWS 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の CloudTrail ログファイルの受け取り](#)および[複数のアカウントからの CloudTrail ログファイルの受け取り](#)

すべての Amazon GameLift Servers アクションは CloudTrail でログに記録されます。これらのアクションは[Amazon GameLift Servers API リファレンス](#)で説明されています。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、誰がリクエストを生成したかという情報が含まれます。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または AWS Identity and Access Management (IAM) ユーザー認証情報を使用して行われたかどうか。
- リクエストがロールまたはフェデレーション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 サービスによって行われたかどうか。

詳細については、「[CloudTrail userIdentity 要素](#)」を参照してください。

Amazon GameLift Servers ログファイルエントリの理解

追跡は、指定した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 のログファイルは、単一か複数のログエントリを含みます。イベントは、任意の出典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、リクエストパラメータなどに関する情報が含まれます。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 Servers サーバーアクセスのログ記録

Amazon GameLift Servers 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、Amazon GameLift Servers Realtime、マネージド EC2 フリート、コンテナフリートのどれを使用するかによって異なります (この章の適切なサブセクションを参照してください)。

トピック

- [ログ記録の違い: マネージド EC2 フリートとコンテナフリート](#)
- [サーバーメッセージのログ記録 \(カスタムサーバー\)](#)

ログ記録の違い: マネージド EC2 フリートとコンテナフリート

ログ記録の動作は、マネージド EC2 フリートとコンテナフリートで大きく異なります。これらの違いを理解することは、EC2 からコンテナに移行する場合や、ログ記録要件に適したフリートタイプを選択する場合に重要です。

主な違いの概要

機能	マネージド EC2 フリート	コンテナフリート
ストレージのログ	ゲームセッション終了後に Amazon S3 に自動的にアップロードされる	Amazon CloudWatch Logs または カスタムログ記録ソリューションを設定する必要があります
GetGameSessionLogUrl API	使用可能 - ログのダウンロード用に Amazon S3 URL を返します	使用不可 - ログは Amazon S3 に自動的に保存されません
ログの保持	Amazon S3 で 14 日間 (自動)	ログ記録設定によって異なります
リアルタイムモニタリング	制限あり - ゲームセッションの終了後にのみ使用できるログ	Amazon CloudWatch Logs との統合
セットアップの複雑さ	自動 - 追加の設定は必要ありません	明示的なログ記録設定が必要です

マネージド EC2 フリートのログ記録

マネージド EC2 フリートの場合、Amazon GameLift Servers は自動ログ管理を提供します。

- 自動アップロード: ゲームセッションが終了すると、サーバーログが Amazon S3 に自動的にアップロードされます。
- GetGameSessionLogUrl API: この API を使用して、Amazon S3 からログをダウンロードするための署名付き URL を取得します。
- 14 日間の保持: ログは自動削除されるまで 14 日間 Amazon S3 に保持されます
- サイズ制限: ログファイルにはゲームセッションあたりのサイズ制限があります ([Amazon GameLift Servers「エンドポイントとクォータ」](#)を参照)

詳細については、「[the section called “カスタムサーバーのログ記録”](#)」および「[GetGameSessionLogUrl API リファレンス](#)」を参照してください。

コンテナフリートのログ記録

コンテナフリートでは、ログ記録を明示的に設定する必要があります。

- Amazon S3 の自動アップロードなし: ログは Amazon S3 に自動的にアップロードされません
- GetGameSessionLogUrl は使用できません: この API はコンテナフリートでは機能しません
- Amazon CloudWatch Logs の統合: 一元化されたログ記録のために Amazon CloudWatch Logs にログを送信するようにコンテナを設定する
- カスタムログ記録ソリューション: ログドライバーまたはサイドカーコンテナを使用して独自のログ記録インフラストラクチャを実装する
- リアルタイムアクセス: 適切な設定により、ゲームセッション中にログにリアルタイムでアクセスできます。

詳細なコンテナログ記録オプションについては、「[コンテナフリートの仕組み](#)」を参照してください。

移行に関する考慮事項

マネージド EC2 からコンテナフリートに移行するときは、次のログ記録の変更を検討してください。

- ログ取得コードの更新: GetGameSessionLogUrl API コールを Amazon CloudWatch Logs クエリまたはカスタムログ記録ソリューションに置き換える
- ログ保持の設定: Amazon CloudWatch Logs またはロギングシステムで適切な保持ポリシーを設定する
- リアルタイムモニタリングの実装: リアルタイムログアクセスを活用してオブザーバビリティを向上させる
- ログの量とコストを確認する: Amazon CloudWatch Logs の料金は、マネージド EC2 フリートに含まれる Amazon S3 ストレージとは異なります

推奨されるコンテナログ設定

コンテナフリートの場合は、次のことをお勧めします。

1. Amazon CloudWatch Logs の統合: awslogs ログドライバーを使用するようにコンテナ定義を設定する
2. 構造化ログ記録: 構造化ログ形式 (JSON) を使用して検索と分析を改善
3. ログレベル: 詳細度とコストを制御する適切なログレベルを実装する
4. 保持ポリシー: コンプライアンスとデバッグのニーズに基づいて保持期間を設定する

サーバーメッセージのログ記録 (カスタムサーバー)

Amazon GameLift Servers カスタム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。

Important

ゲームセッションごとのログファイルのサイズには制限があります (「AWS 全般のリファレンス」の「[Amazon GameLift Servers エンドポイントとクォータ](#)」を参照)。ゲームセッションが終了すると、Amazon GameLift Servers はサーバーログを Amazon Simple Storage Service (Amazon S3) にアップロードします。Amazon GameLift Servers 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 Servers カスタムサーバーでは、ログ記録を実行する独自のコードを記述します。このコードをサーバープロセス設定の一部として設定します。Amazon GameLift Servers 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 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);
```

- ベクトルを [ProcessParameters](#) オブジェクトの [LogParameters](#) 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));
```

- [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#)

- 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
```

- リストを [ProcessParameters](#) オブジェクトの [LogParameters](#) として指定します。

```
var processReadyParameter = new ProcessParameters(  
    this.OnGameSession,  
    this.OnProcessTerminate,  
    this.OnHealthCheck,  
    this.OnGameSessionUpdate,  
    port,  
    new LogParameters(logPaths));
```

- [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

ログアクセスはフリートタイプによって異なります。

マネージド EC2 フリート

ゲームセッションが終了すると、Amazon GameLift Servers は自動的に S3 バケットにログを保存し、14 日間保持します。ゲームセッションのログのロケーションを取得するには、[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。

または、任意のログ記録サービスまたはストレージロケーションにログを直接送信するようにゲームサーバーを設定することで、独自のログ記録ソリューションを設定できます。詳細については、「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」を参照してください。

コンテナフリート

コンテナフリートは、すべてのコンテナから標準出力とエラーストリームをキャプチャします。ログ記録は、次のいずれかのオプションを使用して設定できます。

- コンテナ出力を CloudWatch ログストリームとして指定されたロググループに保存する
- コンテナ出力を S3 ストレージバケットに保存
- ログ記録をオフにする (コンテナ出力は保存されません)

コンテナフリートの作成時にログ記録オプションを設定する方法の詳細については、Amazon GameLift Servers API リファレンスの「[LogConfiguration](#)」を参照してください。

Amazon GameLift Servers でのセキュリティ

Amazon GameLift Servers FleetIQ を Amazon EC2 と組み合わせてスタンドアロン機能として使用している場合は、「Amazon 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。

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 お客様とお客様の間の責任共有です。[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 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 AWS 責任があります。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。サードパーティーの監査者は、[AWS 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、当社のセキュリティの有効性を定期的にテストおよび検証します。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Amazon GameLift Servers、「コンプライアンス[AWS プログラムによる対象範囲内のサービスコンプライアンス](#)」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、会社の要件、適用される I 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 GameLift Servers を使用する際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。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ために Amazon GameLift Servers を設定する方法を示します。また、Amazon GameLift Servers リソースのモニタリングや保護に役立つ他の AWS サービスの使用方法についても説明します。

トピック

- [Amazon GameLift Servers でのデータ保護](#)
- [Amazon GameLift Servers のためのアイデンティティおよびアクセス管理](#)
- [Amazon GameLift Servers でのログ記録とモニタリング](#)
- [のコンプライアンス検証Amazon GameLift Servers](#)
- [Amazon GameLift Servers での耐障害性](#)
- [Amazon GameLift Servers 内のインフラストラクチャセキュリティ](#)

- [Amazon GameLift Servers での構成と脆弱性の分析](#)
- [Amazon GameLift Servers のセキュリティに関するベストプラクティス](#)

Amazon GameLift Servers でのデータ保護

Amazon GameLift Servers FleetIQ を Amazon EC2 と組み合わせてスタンドアロン機能として使用している場合は、「Amazon 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。

責任 AWS [共有モデル](#)、でのデータ保護に適用されます Amazon GameLift Servers。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。ユーザーは、このインフラストラクチャでホストされ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に関するよくある質問](#)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「[AWS 責任共有モデルおよび GDPR](#)」のブログ記事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 アイデンティティセンターまたは 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 は必須ですが、TLS 1.3 を推奨します。
- 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。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「AWS CloudTrail ユーザーガイド」の[CloudTrail 証跡の使用](#)」を参照してください。
- 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 などの高度な管理された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 140-3 検証済み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 \(FIPS\) 140-3](#)」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を、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含めないことを強くお勧めします。これは、コンソール Amazon GameLift Servers、API、または SDK を使用して 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 タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

Amazon GameLift Servers 固有のデータは、次のように処理されます。

- Amazon GameLift Servers 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 Servers 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 Servers コンソール内のリンクまたはサービス API を呼び出して、ログデータにアクセスできます。
- メトリクスとイベントのデータは Amazon GameLift Servers に保存され、Amazon GameLift Servers コンソールを介して、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、マッチメイキングチケット、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 を通じてアクセスできます。
- 顧客提供のデータは Amazon GameLift Servers に保存されます。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イヤー ID を指定する場合、これらの値は匿名化された UUID であり、識別されるプレイ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 Servers 固有のデータの保存時の暗号化は、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。
- お客様が指定するデータは、暗号化された形式で Amazon GameLift Servers に保存されます。

転送中の暗号化

Amazon GameLift Servers 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 (CLI または AWS SDK AWS 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 Servers リソース上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 Servers サービスが含まれません。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 対応フリートの証明書はフリートと同時に作成され、有効期限はフリート作成日に基づきます。
- TLS 証明書生成が有効な Amazon GameLift Servers Realtime では、クライアント SDK for Realtime を使用するゲームクライアントと Realtime サーバー間のトラフィックは、転送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。

Note

TLS 証明書の認証機関要件を満たすために、Amazon GameLift Servers は証明書を生成するように設定されたフリートの最大証明書有効期間を調整します。証明書の有効期間はフリートの作成時に開始され、次のスケジュールで変更されます。

- 2026 年 3 月 11 日まで、発行された TLS 証明書の最大有効期間は 398 日です。
- 2026 年 3 月 1 日現在、発行された TLS 証明書の最大有効期間は 200 日です。
- 2027 年 3 月 1 日現在、発行された TLS 証明書の最大有効期間は 100 日です。

- 2029 年 3 月 1 日現在、発行された TLS 証明書の最大有効期間は 47 日間です。

証明書が更新され、up-to-dateゲームサーバーランタイム環境を維持するために、ではゲームサーバーフリートを定期的に置き換えることAmazon GameLift Serversを推奨しています。

ネットワーク間のトラフィックのプライバシー

リモートで Amazon GameLift Servers 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。ではAmazon GameLift Servers FleetIQ、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、接続を作成するリクエストは SigV4 を使用して署名されます。マネージド Amazon GameLift Servers インスタンスへの接続に関するヘルプについては、「[Amazon GameLift Servers 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。

Amazon GameLift Servers のためのアイデンティティおよびアクセス管理

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 AWS のサービス するのに役立つです。IAM 管理者は、誰を認証 (サインイン) し、誰に Amazon GameLift Servers リソースの使用を許可する (権限を持たせる) かを制御します。IAM は、追加料金なしで使用できる AWS のサービス です。

トピック

- [オーディエンス](#)
- [アイデンティティを使用した認証](#)
- [ポリシーを使用したアクセスの管理](#)
- [Amazon GameLift Servers と IAM の連携方法](#)
- [Amazon GameLift Servers のアイデンティティベースのポリシーの例](#)
- [Amazon GameLift Servers ID とアクセスのトラブルシューティング](#)
- [AWS の 管理ポリシー Amazon GameLift Servers](#)

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「[Amazon GameLift Servers ID とアクセスのトラブルシューティング](#)」を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「[Amazon GameLift Servers と IAM の連携方法](#)」を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「[Amazon GameLift Servers のアイデンティティベースのポリシーの例](#)」を参照)

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

AWS IAM アイデンティティセンター (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーテッド ID としてサインインできます。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。

プログラムによるアクセスの場合、は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「[API リクエストに対するAWS 署名バージョン 4](#)」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント ルートユーザーと呼ばれる 1 つのサインインアイデンティティから始めます。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。ルートユーザー認証情報を必要とするタスクについては、「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。

フェデレーテッドアイデンティティ

ベストプラクティスとして、人間のユーザーが一時的な認証情報 AWS のサービス を使用して にアクセスするには、ID プロバイダーとのフェデレーションを使用する必要があります。

フェデレーテッド ID は、エンタープライズディレクトリ、ウェブ ID プロバイダー、または ID Directory Service ソースの認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、AWS IAM アイデンティティセンターをお勧めします。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「[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、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの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。詳細については、IAM ユーザーガイドの「[ID プロバイダーとのフェデレーションを使用して にアクセスする必要がある AWS](#)」を参照してください。

[IAM グループ](#)は、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「[IAM ユーザーに関するユースケース](#)」を参照してください。

IAM ロール

[IAM ロール](#)は、特定のアクセス許可を持つアイデンティティであり、一時的な認証情報を提供します。ユーザーから [IAM ロール \(コンソール\) に切り替えるか、または API オペレーションを呼び出すことで、ロール](#)を引き受けることができます。AWS CLI AWS 詳細については、「IAM ユーザーガイド」の「[ロールを引き受けるための各種方法](#)」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。詳細については、IAM ユーザーガイドの [IAM でのクロスアカウントリソースアクセス](#) を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられている場合のアクセス許可を定義します。は、プリンシパルがリクエストを行うときにこれらのポリシー AWS を評価します。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの詳細については、「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、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

アイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティ (ユーザー、グループ、またはロール) 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、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイドの [カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する](#) 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、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーと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「[管理ポリシーとインラインポリシーのいずれかを選択する](#)」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。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。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。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。

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を使用できません。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の上限を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「[IAM エンティティのアクセス許可境界](#)」を参照してください。

- サービスコントロールポリシー (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「[サービスコントロールポリシー](#)」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「[リソースコントロールポリシー \(RCP\)](#)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。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](#)」を参照してください。

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アクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうかわからない AWS を決定する方法については、「IAM ユーザーガイド」の「[ポリシー評価ロジック](#)」を参照してください。

Amazon GameLift Servers と IAM の連携方法

IAM を使用して Amazon GameLift Servers へのアクセスを管理する前に、Amazon GameLift Servers で利用できる IAM の機能について学びます。

Amazon GameLift Servers で使用できる IAM の機能

IAM 機能	Amazon GameLift Servers 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	なし
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)	あり

IAM 機能	Amazon GameLift Servers サポート
一時的な認証情報	あり
プリンシパルアクセス権限	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 Servers およびその他の AWS のサービスがほとんどの IAM 機能とどのように連携するかの概要については、「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 GameLift Servers のアイデンティティベースのポリシー

アイデンティティベースのポリシーのサポート: あり

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 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、および条件をコントロールします。アイデンティティベースポリシーの作成方法については、「IAM ユーザーガイド」の「[カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する](#)」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。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 GameLift Servers のアイデンティティベースのポリシーの例

Amazon GameLift Servers 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、[Amazon GameLift Servers のアイデンティティベースのポリシーの例](#)を参照してください。

Amazon GameLift Servers 内のリソースベースのポリシー

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には例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげ

られます。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コントロールできます。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、[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、フェデレーティッドユーザー、またはを含めることができます AWS のサービス。

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。詳細については、IAM ユーザーガイドの[IAM でのクロスアカウントリソースアクセス](#)を参照してください。

Amazon GameLift Servers のポリシーアクション

ポリシーアクションのサポート: あり

管理者は JSON AWS 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。

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。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アクセス許可を付与するポリシーで使用されます。

Amazon GameLift Servers アクションのリストを確認するには、「サービス認可リファレンス」の「[Amazon GameLift Servers で定義されるアクション](#)」を参照してください。

Amazon GameLift Servers のポリシーアクションは、アクションの前に以下のプレフィックスを使用します。

```
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 Servers 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、[Amazon GameLift Servers のアイデンティティベースのポリシーの例](#) を参照してください。

Amazon GameLift Servers のポリシーリソース

ポリシーリソースのサポート: あり

管理者は JSON AWS 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。

Resource JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ベストプラクティスとして、[Amazon リソースネーム \(ARN\)](#) を使用してリソースを指定します。リソースレベルのアクセス許可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 Servers リソースのタイプとその ARN のリストを確認するには、サービス認可リファレンスの「[Amazon GameLift Servers で定義されるリソース](#)」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 GameLift Servers で定義されるアクション](#)」を参照してください。

一部の Amazon GameLift Servers リソースには ARN 値があり、これによりリソースは IAM ポリシーを使用してアクセスを管理できます。Amazon GameLift Servers 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 Servers 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、「[Amazon GameLift Servers のアイデンティティベースのポリシーの例](#)」を参照してください。

Amazon GameLift Servers 向けのポリシー条件キー

サービス固有のポリシー条件キーのサポート: あり

管理者は JSON AWS 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。

Condition 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。イコールや未満などの[条件演算子](#)を使用して条件式を作成して、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。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 Servers の条件キーのリストを確認するには、「サービス認可リファレンス」の「[Amazon GameLift Servers の条件キー](#)」を参照してください。どのアクションおよびリソースと条件キーを使用できるかについては、「[Amazon GameLift Servers で定義されるアクション](#)」を参照してください。

Amazon GameLift Servers 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、[Amazon GameLift Servers のアイデンティティベースのポリシーの例](#)を参照してください。

Amazon GameLift Servers の ACL

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アクセス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 GameLift Servers を備えた ABAC

ABAC (ポリシー内のタグ) のサポート: あり

属性ベースのアクセス制御 (ABAC) は、タグと呼ばれる属性に基づいてアクセス許可を定義する認可戦略です。IAM エンティティと AWS リソースにタグをアタッチし、プリンシパルのタグがリソースのタグと一致するときにオペレーションを許可するように ABAC ポリシーを設計できます。

タグに基づいてアクセスを管理するには、`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ありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、「IAM ユーザーガイド」の「[ABAC 認可でアクセス許可を定義する](#)」を参照してください。ABAC をセットアップする手順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「[属性ベースの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 Servers フリートを表示する](#)」を参照してください。

Amazon GameLift Servers での一時的な認証情報の使用

一時的な認証情報のサポート: あり

一時的な認証情報は、AWS リソースへの短期的なアクセスを提供し、フェデレーションまたはスイッチロールの使用時に自動的に作成されます。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 AWS 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「[IAM の一時的な認証情報](#)」および「[AWS のサービス と IAM との連携](#)」を参照してください。

Amazon GameLift Servers のクロスサービスプリンシパル権限

転送アクセスセッション (FAS) のサポート: あり

転送アクセスセッション (FAS) は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストをリクエストする を使用します。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 GameLift Servers のサービスロール

サービスロールのサポート: 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 のサービスに許可を委任するロールを作成する](#) を参照してください。

Warning

サービスロールの権限を変更すると、Amazon GameLift Servers の機能が破損する可能性があります。Amazon GameLift Servers が指示する場合以外は、サービスロールを編集しないでください。

Amazon GameLift Servers 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 GameLift Servers 管理するフリートでホストされるため、他の AWS リソースへのアクセス Amazon GameLift Servers を制限するサービスロールが必要です。詳細については、「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」を参照してください。

Amazon GameLift Servers の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のサポート: なし

サービスにリンクされたロールは、にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント、サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

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 GameLift Servers のアイデンティティベースのポリシーの例

デフォルトでは、ユーザーおよびロールには、Amazon GameLift Servers リソースを作成または変更する権限はありません。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権限をユーザーに付与する IAM ポリシーを作成できます。

これらのサンプルの 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 ポリシーを作成する \(コンソール\)](#)」を参照してください。

Amazon GameLift Servers が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式など) の詳細については、「サービス認可リファレンス」の「[Amazon GameLift Servers のアクション、リソース、および条件キー](#)」を参照してください。

トピック

- [ポリシーに関するベストプラクティス](#)
- [Amazon GameLift Servers コンソールを使用する](#)
- [自分の権限の表示をユーザーに許可する](#)
- [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 Servers キューへのアクセスを許可する](#)
- [タグに基づいて Amazon GameLift Servers 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](#)

ポリシーに関する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 Servers リソースを作成、アクセス、または削除できるか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 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与する AWS 管理ポリシーを使用します。これらは使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス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 [ジョブ機能の AWS マネージドポリシー](#) を参照してください。

- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、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、最小特権アクセス許可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 でのポリシーとアクセス許可](#) 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。たとえば、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。条件を使用して、サービスアクションがなどの特定の を通じて使用されている場合に AWS のサービス、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 CloudFormation。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、安全で機能的な権限を確保する – 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 Access Analyzer でポリシーを検証する](#) を参照してください。
- 多要素認証 (MFA) を要求する – IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、MFA をオンにしてセキュリティを強化します。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 を使用した安全な API アクセス](#) 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 Servers コンソールを使用する

Amazon GameLift Servers コンソールにアクセスするには、最小限の許可セットが必要です。これらのアクセス許可により、の Amazon GameLift Servers リソースの詳細を一覧表示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 Servers コンソールを使用できるようにするには、次の例と [管理アクセス許可の例](#) の構文を使用して、ユーザーおよびグループにアクセス許可を追加します。詳細については、「[Amazon GameLift Servers にユーザーの権限を設定する](#)」を参照してください。

AWS CLI または AWS API オペレーション Amazon GameLift Servers を使用して 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。

Amazon GameLift Servers コンソールのすべての機能を使用するために必要なアクセス許可の詳細については、[管理アクセス許可の例](#) の管理者用のアクセス許可の構文を参照してください。

自分の権限の表示をユーザーに許可する

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 または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
  "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 Servers キューへのアクセスを許可する

次の例では、ユーザーに特定の Amazon GameLift Servers キューへのアクセスを提供します。

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。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewSpecificQueueInfo",
      "Effect": "Allow",
      "Action": [
        "gamelift:DescribeGameSessionQueues"
      ],
      "Resource": "arn:aws:gamelift:us-east-1:555555555555:gamesessionqueue/examplequeue123"
    },
    {
      "Sid": "ManageSpecificQueue",
      "Effect": "Allow",
      "Action": [
```

```

        "gamelift:UpdateGameSessionQueue",
        "gamelift>DeleteGameSessionQueue"
    ],
    "Resource": "arn:aws:gamelift:us-east-1:111122223333:gamesessionqueue/
examplequeue123"
    }
}
]
}

```

タグに基づいて Amazon GameLift Servers 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 Servers リソースへのアクセスをコントロールできます。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。

JSON

```

{
  "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": {"aws:ResourceAccount": "${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 GameLift Servers と統合したら、ビルドファイルを Amazon S3 にアップロードします。ビルドファイルにアクセスするには、Amazon GameLift Servers で次のポリシーを使用してください。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": "arn:aws:s3:::bucket-name/object-name"
    }
  ]
}
```

Amazon GameLift Servers ゲームファイルのアップロードの詳細については、「[Amazon GameLift Servers のゲームサーバービルドを作成する](#)」を参照してください。

Amazon GameLift Servers ID とアクセスのトラブルシューティング

次の情報は、Amazon GameLift Servers および AWS Identity and Access Management (IAM) の使用時に発生する可能性がある一般的な問題の診断と修正に役立ちます。

トピック

- [Amazon GameLift Servers でアクションを実行する権限がない](#)
- [iam:PassRole を実行する権限がない](#)
- [自分の 以外のユーザーに自分の Amazon GameLift Servers リソース AWS アカウント へのアクセスを許可したい](#)

Amazon GameLift Servers でアクションを実行する権限がない

にアクションを実行する権限がないと AWS マネジメントコンソール 通知された場合は、AWS アカウント管理者にお問い合わせください。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:PassRole を実行する権限がない

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 GameLift Servers 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。

一部の AWS のサービスでは、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、既存のロールをその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、marymajor という IAM ユーザーがコンソールを使用して Amazon GameLift Servers でアクションを実行しようとする場合に発生します。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アクセス許可がありません。

```
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メアリーに 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 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに自分のAmazon GameLift Servers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、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コントロールリスト (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 GameLift Servers がこれらの機能をサポートしているかどうかを確認するには、「[Amazon GameLift Servers と IAM の連携方法](#)」をご参照ください。
- 所有 AWS アカウント している のリソースへの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「[所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する](#)」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「[サードパーティー AWS アカウント が所有する へのアクセスを提供する](#)」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におけるロールとリソースベースのポリシーの使用法の違いについては、「IAM ユーザーガイド」の「[IAM でのクロスアカウントのリソースへのアクセス](#)」を参照してください。

AWS の 管理ポリシー Amazon GameLift Servers

AWS 管理ポリシーは、によって作成および管理されるスタンドアロンポリシーです AWS。AWS 管理ポリシーは、ユーザー、グループ、ロールにアクセス許可の割り当てを開始できるように、多くの一般的なユースケースにアクセス許可を付与するように設計されています。

AWS 管理ポリシーは、すべての AWS お客様が使用できるため、特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることに注意してください。ユースケースに固有の[カスタマー管理ポリシー](#)を定義して、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義されているアクセス許可は変更できません。が AWS マネージドポリシーで定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID (ユーザー、グループ、ロール) に影響します。AWS は、新しい が起動されるか、新しい API

オペレーション AWS のサービス が既存のサービスで使えるようになったときに、AWS マネージドポリシーを更新する可能性が最も高くなります。

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](#)」を参照してください。

AWS マネージドポリシー: GameLiftContainerFleetPolicy

IAM ロールに GameLiftContainerFleetPolicy をアタッチできます。

このポリシーは、Amazon GameLift Servers コンテナフリート内のコンピューティングアクションのアクセス許可を付与します。コンテナフリートは、が Amazon GameLift Servers 管理する一連のホスティングリソースです。には、ユーザーに代わって Amazon GameLift Servers サービスや他の AWS サービスに接続するためのアクセス許可 Amazon GameLift Servers が必要です。

Amazon GameLift Servers でコンテナフリートを作成するときは、GameLiftContainerFleetPolicy 管理ポリシーがアタッチされた IAM サービスロールを指定します。サービスロールを作成する手順については、「[Amazon GameLift Servers 用に IAM サービスロールをセットアップする](#)」を参照してください。

詳細については、「[GameLiftContainerFleetPolicy](#)」を参照してください。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `cloudwatch` – Amazon GameLift Servers アカウントの Amazon CloudWatch Events ログストリームにゲームセッションログを書き込むことを許可します AWS。
- `cloudwatch` – Amazon GameLift Servers に対して、CloudWatch ロググループを作成して、ログストリーム内のゲームセッションデータを整理できるようにします。
- `s3` – Amazon GameLift Servers アカウント内の Amazon Simple Storage Service バケットにゲームセッションログを書き込むことを許可します AWS。
- `s3` – API アクション を使用して、AWS リージョン 指定された Amazon S3 バケットが存在するの取得 Amazon GameLift Servers を許可します `s3:GetBucketLocation`。
- `gamelift` – Amazon GameLift Servers に対して、ホストされたゲームサーバーが AWS アカウントを介して Amazon GameLift Servers サービスと通信できるようにする認証トークンの取得を許可します。

Amazon GameLift Servers AWS 管理ポリシーの更新

このサービスがこれらの変更の追跡を開始Amazon GameLift Serversしてからの AWS の管理ポリシーの更新に関する詳細を表示します。このページの変更に関する自動通知については、「[Amazon GameLift Servers リリース](#)」ページの RSS フィードを購読してください。

変更	説明	日付
GameLiftContainerFleetPolicy – 変更	Amazon GameLift Servers は、Amazon S3 バケット AWS リージョン の を取得するための新しいアクセス許可を追加しました。	2024 年 2 月 5 日
GameLiftContainerFleetPolicy – 新しいポリシー	Amazon GameLift Servers は、ゲームサーバーコンテナを Amazon GameLift Servers マネージドフリートで実行できるようにする新しいアクセス許可を追加しました。	2024 年 11 月 12 日
Amazon GameLift Servers は変更の追跡を開始しました	Amazon GameLift Servers は、AWS 管理ポリシーの変更の追跡を開始しました。	2024 年 11 月 12 日

Amazon GameLift Servers でのログ記録とモニタリング

モニタリングは、Amazon GameLift Servers と 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な部分です。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 Servers 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、単一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、特定の状態になるまでではなく、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 Servers を監視する](#)」を参照してください。

AWS CloudTrail ログ

CloudTrail は、ユーザー、ロール、または Amazon GameLift Servers の AWS のサービスによって実行されたアクションの記録を提供します。CloudTrail で収集した情報を使用して、Amazon GameLift Servers に対して行ったリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。詳細については、「[を使用した Amazon GameLift Servers API コールのログ記録 AWS CloudTrail](#)」を参照してください。

のコンプライアンス検証Amazon GameLift Servers

Amazon GameLift Servers は AWS コンプライアンスプログラムの対象範囲外です。

AWS のサービス が特定のコンプライアンスプログラムの対象で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による対象範囲内の AWS のサービス](#)」をご覧ください、関心のあるコンプライアンスプログラムを選択してください。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。

AWS Artifact 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。詳細については、「[AWS Artifact でレポートをダウンロードする](#)」を参照してください。

AWS のサービス を使用する際のお客様のコンプライアンス責任は、お客様のデータの機密性や貴社のコンプライアンス目的、適用可能な法律および規制によって決定されます。AWS のサービスを使用する際のコンプライアンス責任の詳細については、「[AWS セキュリティドキュメント](#)」を参照してください。

Amazon GameLift Servers での耐障害性

Amazon GameLift Servers FleetIQ を Amazon EC2 と組み合わせてスタンドアロン機能として使用している場合は、「Amazon 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。

AWS の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ゾーンを中心として構築されています。AWSリージョンには、低レイテンシー、高いスループット、そして高

度の冗長ネットワークで接続されている複数の物理的に独立および隔離されたアベイラビリティゾーンがあります。アベイラビリティゾーンでは、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、耐障害性、および拡張性が優れています。

AWS リージョンとアベイラビリティ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。

AWS では、Amazon GameLift Servers グローバルインフラストラクチャに加えて、データの耐障害性をサポートする次の機能を提供しています。

- マルチリージョンキュー – Amazon GameLift Servers 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で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をカスタマイズする](#)」を参照してください。
- 自動容量スケーリング – Amazon GameLift Servers 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 Servers を使用したゲームホスティング容量のスケーリング](#)」を参照してください。
- インスタンス全体の配信 – Amazon GameLift Servers では、フリートサイズに応じて、着信トラフィックを複数のインスタンスに配信します。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 GameLift Servers 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は、スタンダードストレージクラスを使用して Amazon S3 に格納されます。ここでは、複数のデータセンターのレプリケーションを使用して耐障害性を高めます。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 GameLift Servers 内のインフラストラクチャセキュリティ

Amazon GameLift Servers FleetIQ を Amazon EC2 と組み合わせてスタンドアロン機能として使用している場合は、「Amazon 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。

マネージドサービスである Amazon GameLift Servers は、ホワイトペーパー「[Amazon Web Services の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。

AWS 公開版 API コールを使用して、ネットワーク経由で Amazon GameLift Servers にアクセスします。クライアントで Transport Layer Security (TLS) 1.2 以降がサポートされている必要があります。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 Servers サービスでは、すべてのフリートが Amazon 仮想プライベートクラウド (VPC) に配置され、各フリートが AWS クラウド内の論理的に分離された領域に存在します。Amazon GameLift Servers ポリシーを使用して、特定の VPC エンドポイントや特定の VPC からアクセスをコントロールできます。これにより、実質的に Amazon GameLift Servers ネットワーク内の特定の VPC からの特定の AWS リソースへの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 GameLift Servers での構成と脆弱性の分析

Amazon GameLift Servers FleetIQ を Amazon EC2 と組み合わせてスタンドアロン機能として使用している場合は、「Amazon 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。

設定および 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。AWS 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。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。

以下のセキュリティのベストプラクティスも Amazon GameLift Servers での設定と脆弱性の分析に
対処します。

- お客様は、ゲームホスティング用に Amazon GameLift Servers 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があります。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、新しいビルドを Amazon GameLift Servers にアップロードするか、新しいコンテナイメージ内に提供して、新しいフリートを作成し、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。少なくとも、本番稼働用フリートは少なくとも 1 年に 1 回交換する必要があります。詳細については、「[フリートランタイム環境を最新の状態に保つ](#)」を参照してください。
- お客様は、SDK、AWS Amazon GameLift Servers サーバー SDK、リアルタイムサーバー用 Amazon GameLift Servers クライアント SDK 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 GameLift Servers のセキュリティに関するベストプラクティス

Amazon GameLift Servers FleetIQ を Amazon EC2 と組み合わせてスタンドアロン機能として使用している場合は、「Amazon 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 GameLift Servers 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る必要のあるいくつかの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。

フリートランタイム環境を最新の状態に保つ

Amazon GameLift Servers では、ゲームサーバーの安全なランタイム環境を維持するために、マネージドフリート (マネージド EC2 およびマネージドコンテナフリートを含む) を定期的に置き換

えることを強くお勧めします。ランタイム更新なしで長期間実行されるフリートには、ゲームサーバーを侵害する可能性のある古い依存関係やセキュリティの脆弱性が含まれている可能性があります。Amazon GameLift Servers フリートにデプロイされたソフトウェアの責任分担の詳細については、「[Amazon GameLift Servers での構成と脆弱性の分析](#)」を参照してください。

マネージドフリートのランタイム環境は、Amazon マシンイメージ (AMI) のバージョンによって決まります。新しいフリートが作成されると、Amazon GameLift Servers は利用可能な最新の AMI バージョンをフリートに割り当て、そのフリート内のすべてのコンピューティングインスタンスがそのバージョンでデプロイされます。AMI バージョンを更新するには、新しいフリートを作成する必要があります。現在の AMI バージョンの詳細については、「[Amazon GameLift Servers AMI バージョン](#)」を参照してください。

推奨ベストプラクティス:

- フリートの有効期間をモニタリングし、30 日以上経過したフリートを置き換える – Amazon GameLift Servers コンソールでフリートの作成日を追跡するか、CLI を使用してフリート属性を取得できます。Amazon GameLift Servers は、90 日以上経過したフリートの警告をコンソールに表示し、1 年以上経過したフリートのアカウント所有者に E メールで通知します。

Note

フリートの更新 ([UpdateFleetAttributes](#) や [UpdateContainerFleet](#) の使用など) では、AMI バージョンは変更されません。新しいフリートを作成する必要があります。

- セキュリティ状態に基づいてフリートを定期的に置き換える – 新しいフリートを作成し、古いフリートを廃止するための定期的なスケジュールを設定します。[Amazon Q](#) などのサービスを使用して、現在の AMI バージョンでゲームコードを確認し、セキュリティの問題を検出し、修復手順を提案することを検討してください。
- デプロイ前に最新の AMI バージョンを使用してサーバービルドをテストする – 新しいフリートを作成する Amazon GameLift Servers 前に、サーバービルドを変更してアップロードする必要がある場合があります。
- AWS アカウントのフリートクォータを管理する – 必要に応じて制限の引き上げをリクエストして、代替フリートを作成できます。詳細については、「[Amazon GameLift Servers エンドポイントとクォータ](#)」を参照してください。
- フリート交換の自動化を検討する – プロセスを自動化して新しいフリートを作成し、古いフリートからプレイヤートラフィックを移行できます。例えば、次のようになります。

- AWS CloudFormation を使用して、フリートの作成と管理を自動化します。フリート設定を CloudFormation テンプレートとして維持し、それらを使用してリソーススタックを起動します。
- エイリアス機能を活用して、特定のフリート ID Amazon GameLift Servers を抽象化します。フリートエイリアスを使用すると、進行中のゲームセッションを中断することなく、プレイヤートラフィックを既存のフリートから新しいフリートに簡単に切り替えることができます。詳細については、「[エイリアスを使用して Amazon GameLift Servers フリートの指定を抽象化する](#)」を参照してください。
- ブルー/グリーンデプロイ戦略を使用して、移行リスクを軽減し、ダウンタイムをゼロに維持します。2 つの同一の本番稼働環境では、本番稼働のような完全なテスト環境を活用し、移行プロセスをより細かく制御し、即時のロールバックを確保できます。

ポート設定を保護します。

インターネットへのポートの開放は、セキュリティ上のリスクがあるため、開放しないことを強くお勧めします。たとえば、次の設定では、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できるようにするリモートデスクトップポートが開きます。

```
{
  "FleetId": "<fleet identifier>",
  "InboundPermissionAuthorizations": [
    {
      "FromPort": 3389,
      "IpRange": "0.0.0.0/0",
      "Protocol": "RDP",
      "ToPort": 3389
    }
  ]
}
```

代わりに、[UpdateFleetPortSettings](#) を使用して、次の例に示すように、特定の IP アドレスまたはアドレス範囲を持つポートを開きます。

```
{
  "FleetId": "<fleet identifier>",
  "InboundPermissionAuthorizations": [
    {
      "FromPort": 3389,
      "IpRange": "54.186.139.221/32",
```



```
        "Protocol": "TCP",  
        "ToPort": 3389  
    }  
]  
}
```

追加のセキュリティリソース

Amazon GameLift Servers をより安全に使用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 Servers 料金とコスト計画

ゲームホスティングソリューションの Amazon GameLift Servers 料金構造を理解し、費用対効果の高い戦略を実装することは、ゲームインフラストラクチャの予算を最適化するために不可欠です。料金シナリオの例や料金ツールなど、サービスの詳細については、「[Amazon GameLift Servers 料金表](#)」を参照してください。

新規の AWS お客様は、最大 12 か月間、無料利用枠の料金 Amazon GameLift Servers なしで 使用できます。料金が発生しないように、無料利用枠の使用制限内にとどまります。

料金モデル

Amazon GameLift Servers には、さまざまな使用シナリオやビジネスニーズに合わせて設計された複数の料金モデルが用意されています。各モデルには異なるコストベースがあります。各モデルのコスト要因を調整することで、コストを制御できます。

マネージドホスティング

マネージド EC2 およびマネージドコンテナフリートは、Amazon GameLift Servers が管理する Amazon EC2 インスタンスを使用します。ゲームホスティングを設定するときは、使用する Amazon EC2 インスタンスのタイプを選択します。

- コンピューティングコスト - ゲームセッションをホストするための EC2 インスタンスの時間単位の使用料を支払います。インスタンスのコスト要因は次のとおりです。
 - AWS リージョン
 - インスタンスファミリー、タイプ、サイズ
 - 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 (Linux でのみ実行されるマネージドコンテナフリート)
- データ転送 - ゲームクライアントとホストされたゲームサーバー間のトラフィックに対する支払い

Anywhere ホスティング

Anywhere ホスティングは、ユーザーが提供および管理しているゲームホスティングリソースで実行されるため、コンピューティングコストは発生しません。代わりに、コストは以下に基づいています。

- ゲームセッション - Anywhere コンピューティングに配置されたゲームセッションの数に基づいて支払います。
- 接続 (分) - サーバードプロセス接続分の支払い

Amazon GameLift Servers 料金計算ツールを使用して、さまざまな Amazon GameLift Servers 設定のコストを見積もります。ゲームホスティングソリューションを設計するとき、または設定変更のコストを予測するときは、計算ツールを使用します。「[Amazon GameLift Servers 料金見積もりの生成](#)」を参照してください。

Amazon GameLift Servers 料金見積りを生成する

を使用すると AWS 料金見積りツール、[の料金見積りを作成できます Amazon GameLift Servers](#)。計算ツール AWS を使用するには、AWS アカウント または の詳細な知識は必要ありません。

AWS 料金見積りツール 計算ツールは、サービスコストに影響する決定をガイドし、ゲームプロジェクトの Amazon GameLift Servers コストを把握できるようにします。Amazon GameLift Servers 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。

Amazon GameLift Servers マネージド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 Servers 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。Amazon GameLift Servers マネージドホスティングを使用する場合、FlexMatch マッチメイキングに追加のコストはかかりません。

複数の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トしている場合 AWS、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ます。

Amazon GameLift Servers インスタンス

このセクションでは、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コンピューティングリソースのタイプと数を推定するのに役立ちます。Amazon GameLift Servers 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 Servers 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、[の設定Amazon GameLift Serversページ](#)を開きます AWS 料金見積りツール。[説明]を追加し、[リージョン]を選択してから、[Amazon GameLift Servers 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 Servers 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イヤーの最大数です。このフィールドは、Amazon GameLift Servers 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 時間あたりのプレイヤーの平均数です。この値を使用して、Amazon GameLift Servers がプレイヤーのために維持する必要が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 Servers 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、組み合わせ使用することをお勧めします。Amazon GameLift Servers がどのようにスポットインスタンスを使用するかの詳細については、「[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。

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 Servers 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 Servers 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 Servers 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ます。Amazon GameLift Servers でインスタンスタイプを選択する方法については、「[マネージドフリートのコンピューティングリソースを選択する](#)」を参照してください。

Amazon GameLift Servers 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。

[の設定Amazon GameLift Serversページ](#)で AWS 料金見積りツール、データ転送アウト (DTO) を展開し、次のフィールドに入力します。

- 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イヤー時間数に基づいてプレイヤー 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 Servers 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。での見積りの作成と管理の詳細については AWS 料金見積りツール、AWS 料金見積りツール「ユーザーガイド」の [「見積りの作成」](#)、[「サービスの設定」](#)、[「サービスの追加」](#) を参照してください。

コスト最適化戦略

マネージドホスティングソリューションでこれらの戦略を使用すると、高いパフォーマンスとプレイヤーエクスペリエンスを維持しながら、クラウドホスティングコストを削減できます。

ベストプラクティス

ゲームホスティングのコスト最適化プラクティスを組み込むには、以下のヒントに従ってください。少なくとも、Amazon GameLift Servers コストと使用パターンの定期的なレビューをスケジュールすることをお勧めします。料金に関する質問がある場合は、[お問い合わせ](#)してください。

フリート設定

- 料金計算ツールを使用する - このツールをゲームホスティングデータと共に使用して、オプションを調べ、コスト削減のための潜在的な設定シナリオをテストします。「[AWS 料金計算ツール Amazon GameLift Servers](#)」を参照してください。
- 開発中の保存 - 開発とテストには、より小さく安価なインスタンスタイプを使用し、必要な場合にのみ実行します。
- インスタンスタイプをゲームに一致 - ゲームの要件に最適なフリートの Amazon EC2 インスタンスファミリー、タイプ、サイズを選択します。「[マネージドフリートのコンピューティングリソースの選択](#)」の「インスタンスタイプ」セクションを参照してください。
- Graviton インスタンスタイプを使用する - Graviton インスタンスは ARM ベースのプロセッサを搭載しています。オンデマンド料金モデルと同等の x86 ベースのインスタンスよりも、優れた料金パフォーマンス、高いエネルギー効率、低コストを提供します。
- スポットインスタンスの使用 - スポットインスタンスは通常、オンデマンドよりも低コストですが、常に利用できるとは限りません。スポットフリートとオンデマンドフリートを組み合わせて、低コストと高可用性のバランスを取ります (50~85% のスポット使用量を推奨)。重要ではないゲームモードやオフピーク時にはスポットインスタンスを使用します。スポットインスタンスの実行可能性をモニタリングするには、InstanceInterruptions や GameServerInterruptions などのメトリクスを追跡します。「[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。
- Linux でサーバーを実行する - Linux 用のゲームサーバーランタイムを構築します。Linux でデプロイされたインスタンスは、一般的に Windows の場合よりもコスト効率が高くなります。
- リソース使用率の最適化 - パフォーマンスを維持しながら、可能な限り多くの同時ゲームサーバープロセスを実行することで、フリートを最大効率に設定します。「[マネージド Amazon GameLift Servers でのゲームサーバーランタイム設定の最適化](#)」を参照してください。
- マルチリージョン戦略を作成する - ゲームホスティングを複数の場所にデプロイすると、プレイヤーのレイテンシーを短縮し、バックアップオプションを改善できます。最大のプレイヤー市場にホスティングを最初にデプロイし、ピーク時にオーバーフロー容量のセカンダリロケーションを使用することで、リージョンのカバレッジとコスト効率のバランスを取ります。「[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。

フリートの容量のスケーリング

- ゼロにスケール — フリートが使用されていない場合は、不要な料金が発生しないよう、フリート容量を手動でゼロに設定します。「[Amazon GameLift Servers フリートの容量を手動で設定する](#)」を参照してください。
- 自動スケーリングの追加 - 自動スケーリングを使用してゲームホスティング容量を調整することで、ホスティングリソースの過剰プロビジョニングを回避します。容量を変動するプレイヤーの需要やその他の主要なメトリクスに一致させます。「[Amazon GameLift Servers でのフリート容量の自動スケーリング](#)」を参照してください。
- バッファの維持 - プレイヤーを待たずにプレイヤーの需要の急増に対処するには、ターゲット追跡を使用してアイドル状態のゲームサーバーのバッファを維持します。プレイヤーベースのサイズと使用パターンに基づいてバッファをカスタマイズします。「[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。

ゲームセッションの配置

- 配置キューの使用 - Amazon GameLift Servers キューはアルゴリズムに依存して、リソースのホスティングコストやその他の要因に基づいて、ゲームセッションの「可能な限り最適な」ホスティング場所を決定します。「[ゲームセッションの配置を設定する](#)」を参照してください。
- キューの優先順位のカスタマイズ - ゲームセッションを配置するときに、キューがホスティングコストを優先する方法を変更できます。「[ゲームセッション配置に優先順位を付ける](#)」を参照してください。
- 配置メトリクスのモニタリング - キューがゲームセッションの場所をどのように選択しているかを追跡して、最適化の機会を特定します。「[キューの Amazon GameLift Servers メトリクス](#)」を参照してください。

データ転送

- データ転送の追跡 - ゲームクライアントとサーバー間でのデータ転送方法をモニタリングし、アクティビティを最適化する手順を実行します。
- データ圧縮の使用 - ゲームクライアントとサーバー間のデータ転送にデータ圧縮手法を実装することを検討してください。圧縮は、帯域幅の使用量を減らし、ゲームプレイのパフォーマンスとレイテンシーを向上させる上で重要な役割を果たします。

リソースコストと使用率ツール

ツールを使用して、でゲームホスティングコスト AWS をモニタリングおよび最適化する方法について説明します Amazon GameLift Servers。その他のツールの詳細については、「[AWS Billing and Cost Management](#)」を参照してください。

AWS コスト管理ツール

- 請求コンソール - AWS 請求書と使用状況を確認します。「[請求の設定](#)」を参照してください。
- 無料利用枠使用状況アラート - 無料利用枠の上限に近づいたときに通知を設定します。[AWS 「無料利用枠の使用状況の追跡」](#)を参照してください。
- Amazon CloudWatch 請求アラート - 使用量がカスタムしきい値に達したときにアラートを設定します。「[AWS の推測料金をモニタリングする請求アラームの作成](#)」を参照してください。

コスト配分と追跡

- コスト配分タグ - フリートやその他のリソースにタグ付けして、ホスティングコストを整理および追跡します。「[コスト配分タグを使用した AWS コストの整理と追跡](#)」を参照してください。
- コストレポート - 割り当てられたタグ別に分類されたレポートを作成します。「[毎月のコスト配分レポートの使用](#)」を参照してください。
- AWS Cost Explorer - フィルタリングとカスタマイズ可能なビューを使用して、コスト、傾向、予測を分析します。「[を使用したコストと使用状況の分析 AWS Cost Explorer](#)」を参照してください。
- AWS Budgets — コストと使用量を追跡し、AWS に基づいてアクションを実行します。「[でコストを管理する AWS Budgets](#)」を参照してください。

Amazon GameLift Servers と Amazon CloudWatch によるパフォーマンスモニタリング

これらの主要なメトリクスをモニタリングして、リソース使用率を最適化します。Amazon GameLift Servers コンソールでメトリクスを確認するか、Amazon CloudWatch ダッシュボードを使用します。Amazon GameLift Servers で使用できるすべてのメトリクスの詳細については、「[Amazon CloudWatch で Amazon GameLift Servers を監視する](#)」を参照してください。

- インスタンスメトリクス - ActiveInstances、IdleInstances、PercentIdleInstances
- サーバードプロセスマトリクス - ActiveServerProcesses、HealthyServerProcesses

- ゲームセッションメトリクス - ActiveGameSessions、AvailableGameSessions
- プレイヤーセッションメトリクス - CurrentPlayerSessions
- キューメトリクス - AverageWaitTime、QueueDepth
- マッチメイキングメトリクス - CurrentTickets、MatchesPlaced
- ハードウェアパフォーマンス -
CPUUtilization、NetworkIn/NetworkOut、DiskReadBytes/DiskWriteBytes
- リソース使用率メトリクス
 - PercentIdleInstances を使用して最適なフリートサイズを決定します。
 - PercentAvailableGameSessions を追跡して、十分なゲームホスティング容量を確保します。
 - InstanceInterruptions とGameServerInterruptions をモニタリングして、スポットインスタンスの実行可能性を判断します。

Amazon GameLift Servers リファレンスガイド

このセクションには、Amazon GameLift Servers 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 Servers 用サービス API](#)
- [サーバー SDK 5.x for Amazon GameLift Servers](#)
- [バージョン 4 以前のサーバー SDK for Amazon GameLift Servers の使用](#)
- [ゲームセッション配置イベント](#)
- [Amazon GameLift Servers AMI バージョン](#)
- [Amazon GameLift Servers サービスロケーション](#)
- [Amazon GameLift Servers エンドポイントとクォータ](#)
- [Amazon GameLift Servers API の制限](#)
- [Amazon GameLift Servers SDK API の制限](#)
- [Amazon GameLift Servers UDP ping ビーコン](#)

Amazon GameLift Servers 用サービス API

Amazon GameLift Servers ゲームホスティングソリューションやその他機能を構築する際に、このタスクベースのリストを使用して、API オペレーションを検索します。AWS SDK には、これらのオペレーションが `aws.gamelift` 名前空間に含まれます。[AWS SDK をダウンロードするか、Amazon GameLift Servers API リファレンスドキュメントを参照してください。](#)[AWS CLI コマンドリファレンス](#)に記載されているように、AWS コマンドラインインターフェイス (AWS CLI) で API を使用することもできます。

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 Servers ホスティングリソースの管理](#)
- [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 Servers サービス API には、他の Amazon GameLift Servers 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 オペレーションのリストに

については、「[FleetIQ API オペレーション](#)」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 オペレーション](#)」を参照してください。

Amazon GameLift Servers ホスティングリソースの管理

これらのオペレーションを呼び出すことで、ゲームサーバーのホスティングリソースを設定したり、プレイヤーの需要に合わせた容量をスケールしたり、パフォーマンスと使用状況のメトリクスを取得したりすることができます。ゲームサーバーを Amazon GameLift Servers (Amazon GameLift Servers Realtime を含む) にホスティングする際は、これらの API オペレーションを使用します。ほとんどのリソース管理タスクは、[Amazon GameLift Servers コンソール](#)で作業できますが、AWS Command Line Interface (AWS CLI) ツールを使用して呼び出すこともできます。

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。

カスタムゲームサーバービルドの管理

- [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 Servers ビルドリソースを作成します。AWS CLI コマンドとして利用できるこのオペレーションは、ゲームサーバービルドをアップロードする最も一般的な方法です。
- [CreateBuild](#) - 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ビルドを作成します。
- [ListBuilds](#) - Amazon GameLift Servers 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](#) - ビルドに関連付けられた情報を取得します。
- [UpdateBuild](#) - ビルド名とバージョンを含むビルドメタデータを変更します。
- [DeleteBuild](#) - Amazon GameLift Servers からビルドを削除します。

Amazon GameLift Servers Realtime 設定スクリプトを管理する

- [CreateScript](#) - JavaScript ファイルをアップロードし、新しい Amazon GameLift Servers スクリプトリソースを作成します。
- [ListScripts](#) - Amazon GameLift Servers リージョンにアップロードされたすべての Realtime スクリプトのリストを取得します。

- [DescribeScript](#) - Realtime スクリプトに関連付けられた情報を取得します。
- [UpdateScript](#) - 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 Servers から Realtime 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、ゲームサーバービルドまたは Realtime 設定スクリプトでデプロイします。

[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 Servers 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 Servers 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](#) - ゲームサーバーの実行やプレイヤーをホスティングしなくなったフリートを削除します。
- フリートロケーションの表示/更新。
 - [CreateFleetLocations](#) - 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](#) - 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](#) - 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/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/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/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](#) - 現在の AWS アカウントと現在の使用レベルで許可されるインスタンスの最大数を取得します。
- [DescribeFleetCapacity](#) - 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](#) - 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](#) - フリートの容量設定を手動で調整します。
- 設定:
 - [PutScalingPolicy](#) - ターゲットベースの自動スケーリングを有効にするか、カスタム自動スケーリング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](#) - 既存の自動スケーリングポリシーを取得します。
 - [DeleteScalingPolicy](#) - 自動スケーリングポリシーを削除し、フリートの容量に影響しないようにします。
 - [StartFleetActions](#) - フリートの自動スケーリングポリシーを再起動します。
 - [StopFleetActions](#) - フリートアクションの停 - フリートの自動スケーリング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](#) - 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](#) - 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](#) - 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](#) - 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用キューを設定する

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](#) - 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 Servers リージョンで定義されたゲームセッションキューを取得します。
- [UpdateGameSessionQueue](#) – ゲームセッションキューの設定を変更します。
- [DeleteGameSessionQueue](#) – 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、代替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](#) – 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 Servers リージョンで定義されたすべてのフリートエイリアスを取得します。
- [DescribeAlias](#) – 既存のエイリアスに関する情報を取得します。
- [UpdateAlias](#) – 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](#) – リージョン からエイリアスを削除します。
- [ResolveAlias](#) – 指定されたエイリアスが指すフリート ID を取得します。

マネージドホスティングインスタンスに接続する

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](#) – 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](#) – 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 Servers ホスティングリソースと他のリソース間の VPC ピアリング接続を作成および管理します AWS。

- [CreateVpcPeeringAuthorization](#) – いずれかの VPC へのピアリング接続を許可します。

- [DescribeVpcPeeringAuthorizations](#) - 有効なピアリング接続の承認を取得します。
- [DeleteVpcPeeringAuthorization](#) - ピアリング接続の認可を削除します。
- [CreateVpcPeeringConnection](#) - Amazon GameLift Serversフリートの VPC と VPCs の 1 つの間にピア接続を確立します。
- [DescribeVpcPeeringConnections](#) - Amazon GameLift Servers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 Servers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

これらのオペレーションをバックエンドサービスから呼び出して、新しいゲームセッションを開始し、既存のゲームセッションに関する情報を取得し、ゲームセッションにプレイヤーを参加させます。以下のオペレーションは、Amazon GameLift Servers 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。Amazon GameLift Servers Realtime を使用している場合は、[Amazon GameLift Servers Realtime クライアント API \(C#\)](#) を使用して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 Servers 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](#) - 配置リクエストの詳細とステータスを取得します。
 - [StopGameSessionPlacement](#) - 配置リクエストをキャンセルします。
 - [CreateGameSession](#) - 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](#) - 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](#) - 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](#) - 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](#) - 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](#) - 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](#) - 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](#) - ゲームセッションのログが保存された場所を取得します。

サーバー SDK 5.x for Amazon GameLift Servers

このセクションには、SDK 5.x for Amazon GameLift Servers のリファレンスドキュメントが含まれます。サーバー SDK には、ゲームサーバーが Amazon GameLift Servers サービスとやり取りするために使用するコア機能があります。例えば、ゲームサーバーはサービスからプロンプトを受け取り、ゲームセッションを開始および停止し、定期的にゲームセッションステータスをサービスに更新します。ホスティング用にゲームサーバーをデプロイする前に、サーバー SDK と統合します。

このサーバー SDK リファレンスを使用して、ホスティング向けのカスタムマルチプレイヤーゲームサーバーと Amazon GameLift Servers を統合します。統合プロセスのガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。

サーバー SDK for Amazon GameLift Servers の最新メジャーバージョンは 5.x です。以下のホスティング機能では、バージョン 5.x を使用する必要があります。

- Amazon GameLift Servers Anywhere
- Unreal Engine と Unity 用の Amazon GameLift Servers プラグイン

Note

サーバー SDK バージョン 4.x 以前を使用する必要がある場合は、[バージョン 4 以前のサーバー SDK for Amazon GameLift Servers の使用](#) のドキュメントとダウンロード情報を参照してください。

トピック

- [サーバー SDK 5 for Amazon GameLift Servers の更新](#)
- [サーバー SDK 5.x への移行 Amazon GameLift Servers](#)
- [C++ サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)
- [C# サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)
- [Go サーバー SDK for Amazon GameLift Servers -- アクション](#)
- [C++ \(Unreal\) サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)

サーバー SDK 5 for Amazon GameLift Servers の更新

ホストされたゲームサーバーは、サーバー SDK for Amazon GameLift Servers を使用して Amazon GameLift Servers サービスと通信し、プレイヤーのゲームセッションを開始および管理します。最新バージョンの Amazon GameLift Servers サーバー SDK 5 では、Amazon GameLift Servers の新しい機能が改善されサポートされています。ゲームサーバービルドで現在 Amazon GameLift Servers サーバー SDK 4 以前を使用している場合は、このトピックのガイダンスに従ってゲームを更新します。

Amazon GameLift Servers サーバー SDK バージョン 5.0.0 以降には、以下の更新が含まれています。

- 拡張言語 – ライブラリは、C++、C#、Go の言語で使用できます。Unreal Engine で使用する C++ ライブラリを構築できます。
- ゲームエンジンプラグインサポート – Unreal Engine および Unity 用の Amazon GameLift Servers スタンドアロンプラグインには、Amazon GameLift Servers サーバー SDK 5 ライブラリが必要です。これらのプラグインは、ゲームをホスティング用に Amazon GameLift Servers に統合、テスト、デプロイするためのガイド付きワークフローを提供します。「[Unity 用 Amazon GameLift Servers プラグイン \(サーバー SDK 5.x\)](#)」および「[Unreal Engine 用 Amazon GameLift Servers プラグイン](#)」のドキュメントを参照してください。

- Amazon GameLift Servers Anywhere サポート – Anywhere フリートでは、Amazon GameLift Servers 機能 (マッチメイキングを含む) を使用するように独自のホスティングリソースを設定できます。Amazon GameLift Servers エージェントを追加して、ゲームセッションのライフサイクル管理を自動化します。オンプレミスハードウェアでの本番ホスティングには Anywhere フリートを使用するか、高速反復ゲーム開発用のテスト環境を設定します。「[Anywhere ホスティング](#)」および「[Amazon GameLift Servers エージェント](#)」を参照してください。
- テストツールの更新 – Amazon GameLift Servers Anywhere 機能を使用すると、ゲームに対してローカル環境またはクラウドベースのテスト環境を設定できます。Amazon GameLift Servers エージェントの有無にかかわらず、テストをセットアップします。これらのツールは Amazon GameLift Servers Local を置き換えます。「[Amazon GameLift Servers Anywhereでローカルテストを設定する](#)」を参照してください。
- C# 用の統合.NET ソリューション – C# サーバー SDK 5.1+ 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をサポートします。.NET 標準 2.1 は Unity で構築したライブラリで利用可能です。
- 新しい Compute リソース – この新しいリソースは、さまざまなタイプのホスティングリソースを組み合わせています。これには、クラウドベースのホスティングリソース (マネージド EC2 またはコンテナフリート) と、顧客が管理するホスティングリソース (Anywhere フリート) が含まれます。更新は以下のとおりです。
- Compute リソースの新しい API コールには、[ListCompute\(\)](#)、[DescribeCompute\(\)](#) および [GetComputeAccess\(\)](#) が含まれます。これらのアクションは、あらゆる種類の Amazon GameLift Servers フリー트의ホスティングリソース情報を返します。一般的に、サーバー SDK 5.x を使用するゲームサーバーを持つフリートの場合、コンピューティング固有のアクションを使用してインスタンス固有のアクションを置き換えます。さらに、これらのアクションは、Amazon GameLift Servers エージェント: [RegisterCompute \(\)](#)、[DeregisterCompute \(\)](#)、および [GetComputeAuthToken \(\)](#) を使用しない Anywhere フリートで使用されます。
- CloudWatch デイメンション FleetId、Location、ComputeType を持つ新しいメトリック ActiveCompute。このメトリックスは以前のメトリクス ActiveInstances に取って代わります。
- リモートアクセス用の Amazon EC2 Systems Manager (SSM) – セキュリティを強化するには、Amazon GameLift Servers マネージドフリートのインスタンスに接続するときに SSH の代わりに SSM を使用します。「[Amazon GameLift Servers 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。

サーバー SDK 5.x への移行 Amazon GameLift Servers

サーバー SDK バージョン 5.x を使用するようにゲームプロジェクトを更新するには、以下の変更を行います。

Note

サーバー SDK バージョン 4.x 以前を引き続き使用する必要がある場合は、[バージョン 4 以前のサーバー SDK for Amazon GameLift Servers の使用](#) のドキュメントとダウンロード情報を参照してください。SDK バージョン 4.0.2 の場合は、[公式の GitHub リリース](#) からダウンロードできます。

1. 開発環境の最新のサーバー SDK for Amazon GameLift Servers パッケージを取得します [[サイトをダウンロード](#)]。ダウンロードしたパッケージとバージョンに対応する Readme ファイル内のインストール手順を実行します。ゲームプロジェクトでサーバー SDK を使用方法については、以下の手順を参照してください。

- [C++、C#、または Go を使用する開発環境の場合](#)
- [Unreal Engine プロジェクトの場合 \(C++ サーバー SDK for Unreal ライブラリのみ\)](#)
- [Unity プロジェクトの場合 \(C# サーバー SDK for Unity ライブラリのみ\)](#)
- [Unreal Engine 用 Amazon GameLift Servers プラグインで使用する場合](#)
- [Unity 用 Amazon GameLift Servers プラグインで使用する場合](#)

2. 次のようにサーバーコードを更新します。

- サーバーコードコールバック関数を `onCreateGameSession()` から `onStartGameSession()` に変更します。
- 必要に応じて `InitSDK()` 入力を更新します。
 - ゲームサーバービルドを Amazon GameLift Servers マネージド EC2 フリートまたは Amazon GameLift Servers エージェントを搭載した Anywhere フリートにデプロイする場合:

パラメータがない `InitSDK()` の呼び出し ([C++](#)) ([C#](#)) ([Unreal](#))。この呼び出しは、コンピューティング環境と Amazon GameLift Servers サービスへの WebSocket 接続を設定します。

- ゲームサーバービルドを Amazon GameLift Servers エージェントなしで Anywhere フリートにデプロイする予定がある場合:

サーバーパラメータ ([C++](#)) ([C#](#)) ([Unreal](#)) を使用して `InitSDK()` を呼び出します。ゲームサーバープロセスは、これらのパラメータを使用して Amazon GameLift Servers サービスとの接続を確立します。

3. ゲームサーバービルドやその他のホストされたアプリケーションが、実行中に他の AWS リソースと通信する場合は、アプリケーションがそれらのリソースにアクセスする方法を変更する必要があります。AssumeRoleCredentials の使用を、新しいサーバー SDK アクション `GetFleetRoleCredentials()` (ゲームサーバー用) に置き換えるか、または共有認証情報 (その他のアプリケーション用) を使用してください。この変更の実装方法の詳細については、「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」を参照してください。
4. プロジェクトがサーバー SDK アクション `GetInstanceCertificate()` を呼び出して TLS 証明書を取得した場合は、代わりに新しい `GetComputeCertificate()` ([C++](#)) ([C#](#)) ([Unreal](#)) を使用するようにコードを変更します。
5. ゲームビルドを Amazon GameLift Servers にアップロードする場合 ([upload-build](#) や [CreateBuild\(\)](#) など)、`ServerSdkVersion` パラメータを使用中の 5.x バージョンに設定します (このパラメータは現在デフォルトは 4.0.2 です)。このパラメータは、ゲームサーバービルドの実際のサーバー SDK ライブラリと一致する必要があります。アップロードされたゲームサーバービルドに間違ったバージョンを指定すると、そのビルドで作成するフリートは失敗します。「[Amazon GameLift Servers のゲームサーバービルドを作成する](#)」を参照してください。

次の例は、サーバー SDK バージョンを指定する方法を示しています。

```
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
```

6. 「[Amazon GameLift Servers フリートインスタンスにリモート接続する](#)」で説明されているとおり、スクリプトを使用してマネージドフリートにリモート接続する場合は、新しいプロセスを使用するようにスクリプトを更新します。

C++ サーバー SDK 5.x for Amazon GameLift Servers -- アクション

サーバー SDK 5.x リファレンスを使用して、Amazon GameLift Servers でホスティングするマルチプレイヤーゲームを統合します。統合プロセスのガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」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利用できる Amazon GameLift Servers C++ API について説明します。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。

C++ サーバー SDK 5.x for Amazon GameLift Servers -- データ型

Amazon GameLift Servers C++ サーバー SDK 5.x リファレンスを使用して、Amazon GameLift Servers でホスティングするマルチプレイヤーゲームを統合します。統合プロセスのガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」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利用できる Amazon GameLift Servers C++ API について説明します。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。

[C++ サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)

データ型

- [LogParameters](#)
- [MetricsParameters](#)
- [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するオブジェクトで、ゲームセッション終了時に Amazon GameLift Servers でアップロードして保存します。ゲームサーバーは、[ProcessReady\(\)](#) 呼び出しで ProcessParameters オブジェクトの一部として LogParameters を Amazon GameLift Servers に提供します。

プロパティ	説明

LogPaths	Amazon GameLift Servers 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\MyGame\sessionLogs (Windows インスタンスの場合) となります。 タイプ: <code>std::vector<std::string></code> 必須: いいえ

MetricsParameters

このデータ型を使用して、ゲームサーバーのメトリクス収集とクラッシュレポートを設定します。ゲームサーバーは、[InitMetrics\(\)](#) 呼び出しで MetricsParameters を Amazon GameLift Servers に送信します。

プロパティ	説明
StatsdHost	メトリクス収集用の StatsD サーバーのホスト名または IP アドレス。 タイプ: <code>std::string</code> 必須: いいえ
StatsdPort	メトリクス収集用の StatsD サーバーのポート番号です。 タイプ: <code>int</code>

	必須: いいえ
CrashReporterHost	クラッシュレポーターサーバーのホスト名または IP アドレスです。 タイプ: std::string 必須: いいえ
CrashReporterPort	クラッシュレポーターサーバーのポート番号です。 タイプ: int 必須: いいえ
FlushIntervalMs	メトリクスデータをサーバーにフラッシュする際のミリ秒単位の間隔です。 タイプ: int 必須: いいえ
MaxPacketSize	サーバーに送信されるメトリクスパケットの最大サイズ (バイト) です。 タイプ: int 必須: いいえ

ProcessParameters

このデータ型には、[ProcessReady\(\)](#) の中で Amazon GameLift Servers に送信されるパラメータのセットが含まれます。

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持つオブジェクトです。Amazon GameLift Servers はこれらのファ

	イルをコピーして保存し、後で参照できるようにします。 タイプ: <code>Aws::GameLift::Server::</code> LogParameters 必須: いいえ
OnHealthCheck	Amazon GameLift Servers がサーバープロセスにヘルスステータスレポートを要求するために呼び出すコールバック関数です。Amazon GameLift Servers はこの関数を 60 秒ごとに呼び出し、応答を最大 60 秒間待機します。サーバープロセスは正常であれば <code>TRUE</code> を返し、正常でない場合は <code>FALSE</code> を返します。応答が返されない場合、Amazon GameLift Servers はサーバープロセスを正常ではないと記録します。 タイプ: <code>std::function<bool()></code> <code>onHealthCheck</code> 必須: いいえ
OnProcessTerminate	Amazon GameLift Servers がサーバープロセスを強制的にシャットダウンするために呼び出すコールバック関数です。この関数を呼び出すと、Amazon GameLift Servers はサーバープロセスがシャットダウンするまで最大 5 分間待機し、サーバープロセスがシャットダウンする前に ProcessEnding() を呼び出して応答するのを待ちます。 タイプ: <code>std::function<void()></code> <code>onProcessTerminate</code> 必須: はい

OnStartGameSession

Amazon GameLift Servers が新しいゲームセッションをアクティブ化するために呼び出すコールバック関数。Amazon GameLift Servers は、クライアントリクエスト [CreateGameSession](#) に応じて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 GameLift Servers が呼び出すコールバック関数。Amazon GameLift Servers は、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が処理された際にこの関数を呼び出します。これは、[GameSession](#) オブジェクト、ステータス更新 (`updateReason`) そして、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	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: <code>Aws::GameLift::Server::GameSession</code> 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: <code>Aws::GameLift::Server::UpdateReason</code> 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。

プロパティ	説明
	タイプ: <code>std::string</code> 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 <code>arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token></code> です。 タイプ: <code>std::string</code> 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: <code>std::string</code> 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: <code>std::string</code> 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: <code>int</code>

プロパティ	説明
	必須: いいえ
ポート	ゲームセッションのポート番号。Amazon GameLift Servers 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: <code>in</code> 必須: いいえ
IpAddress	ゲームセッションの IP アドレス。Amazon GameLift Servers 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: <code>std::string</code> 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: <code>std::string</code> 必須: いいえ
MatchmakerData	ゲームセッションの作成に使用されたマッチメイ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イ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: <code>std::string</code> 必須: いいえ

プロパティ	説明
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: <code>std :: vector < GameProperty ></code> 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: <code><unique identifier>.<region identifier>.amazon gamelift.com</code>。 • TLS 対応でないフリート: <code>ec2-<unique identifier>.compute.amazonaws.com</code>。 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: <code>std::string</code> 必須: いいえ

ServerParameters

ゲームサーバプロセスが、Amazon GameLift Servers サービスへの接続を確立するために使用する情報。ゲームサーバビルドが、Amazon GameLift Servers エージェントを使用しない Anywhere フリートまたはコンテナフリートにデプロイされる場合にのみ、[InitSDK\(\)](#) を呼び出す際にこれらのパ

ラメータを含めます。他のすべてのデプロイシナリオでは、パラメータなしで [InitSDK\(\)](#) が呼び出されます。

プロパティ	説明
webSocketUrl	Amazon GameLift Servers Anywhere コンピューティングリソース用に RegisterCompute を実行すると、GameLiftServerSdkEndpoint Amazon GameLift Servers が返されます。 タイプ: std::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: 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、「RegisterCompute」を参照してください。 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、「RegisterCompute」を参照してください。 タイプ: std::string 必須: はい

プロパティ	説明
authToken	Amazon GameLift Servers によって生成され、サーバーを Amazon GameLift Servers に対して認証するための認証トークン。詳細については、「GetComputeAuthToken」を参照してください。 タイプ: std::string 必須: はい

StartMatchBackfillRequest

マッチメイ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\(\)](#) 呼び出しで Amazon GameLift Servers に送信します。

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

プロパティ	説明
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: <code>std::vector<Player></code> 必須: はい
TicketId	マッチメイ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 GameLift Servers ソリューションが値を生成します。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。 タイプ: <code>std::string</code> 必須: いいえ

プレイヤー

このデータ型はマッチメイキングのプレイヤーを表します。マッチメイキングリクエストを開始すると、プレイヤーにはプレイヤー ID、属性、必要に応じてレイテンシーデータが含まれます。マッチが成立すると、Amazon GameLift Servers が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、プレイ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

プロパティ	説明
	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: <code>Dictionary<string,int></code> 必須: いいえ
PlayerAttributes	マッチメイ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、マッチメイキングルールセットで使用されている PlayerAttributes 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、「Attribute Value」を参照してください。 タイプ: <code>std::map<std::string, AttributeValue></code> 必須: いいえ
PlayerId	プレイヤーを表す一意の識別子。 タイプ: <code>std::string</code> 必須: いいえ
チーム	マッチでプレイ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イキングルールセットで定義します。 タイプ: <code>std::string</code> 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、この情報を [DescribePlayerSessions\(\)](#) 呼び出して Amazon GameLift Servers に提供します。

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 の形式は <code>arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string></code> です。GameSessionID はカスタム ID 文字列または タイプ: <code>std::string</code> 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: <code>std::string</code> 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: <code>std::string</code> 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: <code>std::string</code> 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: <code>std::string</code> 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

マッチメイ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 Servers サービスに送信します。

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います。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: <code>std::string</code> 必須: いいえ
いいえ	数値の属性値を表します。 タイプ: <code>double</code> 必須: いいえ
SL	文字列の属性値の配列を表します。 タイプ: <code>std::vector<std::string></code> 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。

プロパティ	説明
	タイプ: <code>std::map<std::string, double></code>
	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 GameLift Servers 用に IAM サービスロールをセットアップする](#) を参照してください。

プロパティ	説明
RoleArn	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 AWS リソースネーム (ARN)。 タイプ: <code>std::string</code> 必須: いいえ
RoleSessionName	AWS Security Token Service AssumeRole セッションを一意に識別するために使用できるロールセッション名。この名前は、CloudTrail などの監査ログに公開されます。 タイプ: <code>std::string</code> 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。

プロパティ	説明
	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
Success	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
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
Success	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。 タイプ: <code>Aws::GameLift::Server::Model::DescribePlayerSessionsResult&&</code> 必須: いいえ
Success	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

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: <code>std::string</code> 必須: はい
PlayerSessions	タイプ: <code>IList<the section called “PlayerSession”></code> 必須:
ResultWithOwnership	アクションの結果を <code>rvalue</code> 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: <code>std::string&&</code> 必須: いいえ
Success	アクションが成功したかどうか。 タイプ: <code>bool</code>

プロパティ	説明
	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
Success	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
Success	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

PlayerSession

このデータ型は、Amazon GameLift Servers がゲームサーバーに渡すプレイヤーセッションを表します。詳細については、「[PlayerSession](#)」を参照してください。

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
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

プロパティ	説明
	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 が提供された場合、PlayerSessionStatusFilter はレスポンスに影響を与えません。 型: PlayerSessionStatus enum。以下に示しているのは、可能な値です。 • アクティブ • COMPLETED • NOT_SET • RESERVED • TIMEDOUT 必須: いいえ
TerminationTime	タイプ: long 必須: いいえ
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。

プロパティ	説明
	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
Success	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	マッチメイキングチケットの一意的識別子。ここでチケット ID を指定しない場合、Amazon GameLift Servers が UUID 形式の ID を自動生成します。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: <code>Aws::GameLift::Server::Model::GetComputeCertificateResult&&</code> 必須: いいえ
Success	アクションが成功したかどうか。 タイプ: <code>bool</code> 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 Servers マネージドフリートを使用する場合、このパスには次の内容が含まれます。

プロパティ	説明
	• <code>certificate.pem</code> : 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 <code>certificateChain.pem</code> を組み合わせたものです。 • <code>certificateChain.pem</code> : ルート証明書と中間証明書を含む証明書チェーン。 • <code>rootCertificate.pem</code> : ルート証明書。 • <code>privateKey.pem</code> : エンドユーザー証明書のプライベートキー。 タイプ: <code>std::string</code> 必須: いいえ
<code>ComputeName</code>	コンピューティングリソースの名前。 タイプ: <code>std::string</code> 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
<code>ResultWithOwnership</code>	アクションの結果を <code>rvalue</code> 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: <code>Aws::GameLift::Server::Model::GetFleetRoleCredentialsResult</code>

プロパティ	説明
	必須: いいえ
Success	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

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。

プロパティ	説明
	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 必須: いいえ
SessionToken	AWS リソースとやり取りす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
Success	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
Success	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

GameLiftError

プロパティ	説明
ErrorType	エラーのタイプ。 型: GameLiftErrorType enum 。 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
ErrorName	エラータイプの名前。 タイプ: std::string 必須: いいえ

列挙型

サーバー SDK for Amazon GameLift Servers (C++) で定義されている列挙体は、次のとおりです。

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 GameLift Servers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。
- SERVICE_CALL_FAILED - AWS サービスへの呼び出しに失敗しました。
- WEBSOCKET_CONNECT_FAILURE
- WEBSOCKET_CONNECT_FAILURE_FORBIDDEN
- WEBSOCKET_CONNECT_FAILURE_INVALID_URL
- WEBSOCKET_CONNECT_FAILURE_TIMEOUT
- ALREADY_INITIALIZED - Amazon GameLift Servers 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 gameSession または playerSession のフリートと一致しません。
- GAMELIFT_CLIENT_NOT_INITIALIZED - Amazon GameLift Servers クライアントは初期化されていません。
- GAMELIFT_SERVER_NOT_INITIALIZED - Amazon GameLift Servers クライアントは初期化されていません。
- GAME_SESSION_ENDED_FAILED - サーバー SDK for Amazon GameLift Servers 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_NOT_READY - Amazon GameLift Servers 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 - サーバー SDK for Amazon GameLift Servers 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 Servers サーバーまたはクライアントは Initialize() で初期化されていません。

- NO_TARGET_ALIASID_SET - ターゲットの aliasId 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 - サーバー SDK for Amazon GameLift Servers 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 - サーバードキュメント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、PlayerSessions 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 - サーバードキュメントをまだアクティブ化する準備ができていません。
- PROCESS_READY_FAILED - サーバー SDK for Amazon GameLift Servers 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。

C++ サーバー SDK 5.x for Amazon GameLift Servers -- データ型

トピック

- [GetSdkVersion\(\)](#)
- [InitMetrics\(\)](#)
- [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。

構文

```
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();
```

InitMetrics()

サーバーパフォーマンスデータを収集およびレポートするためのメトリクスシステムを初期化します。最良の結果を得るには、[InitSDK\(\)](#) の前にこのメソッドを呼び出して、SDK の初期化中にメトリクス収集を有効にします。

構文

```
Aws::GameLift::GenericOutcome InitMetrics();  
Aws::GameLift::GenericOutcome InitMetrics(const  
    Aws::GameLift::Server::MetricsParameters &metricsParameters);
```

パラメータ

[MetricsParameters](#) (オプション)

メトリクス収集の設定パラメータ。指定しない場合は、環境変数で上書きできる既定値を使用します。

戻り値

成功すると、成功を示す [the section called “GenericOutcome”](#) オブジェクトが返されます。成功しなかった場合、エラーメッセージを返します。

例

```
// Initialize with default parameters (uses environment variables if available)  
// Defaults: localhost:8125 for StatsD, localhost:8126 for crash reporter  
// FlushInterval: 10000ms, MaxPacketSize: 512 bytes
```

```
Aws::GameLift::GenericOutcome outcome = Aws::GameLift::Server::InitMetrics();
if (outcome.IsSuccess()) {
    // Metrics system initialized successfully
}

// Initialize with custom parameters
Aws::GameLift::Server::MetricsParameters metricsParams("localhost", 8125, "crash-host",
    8126, 5000, 1024);
Aws::GameLift::GenericOutcome customOutcome =
    Aws::GameLift::Server::InitMetrics(metricsParams);
if (customOutcome.IsSuccess()) {
    // Metrics system initialized with custom parameters
}
```

InitSDK()

Amazon GameLift Servers SDK を初期化します。他のAmazon GameLift Servers 関連の初期化が実行される前に、起動時にこのメソッドを呼び出します。このアクションは、ホスト環境からサーバーパラメータを読み取り、ゲームサーバープロセスと Amazon GameLift Servers サービス間の通信を設定します。べき等性トークンを使用するため、失敗したときにこの呼び出しを安全に再試行できます。

ゲームサーバービルドを Amazon GameLift Servers エージェントなしで Amazon GameLift Servers Anywhere フリートまたはコンテナフリートにデプロイする場合は、[InitSDK\(\)](#) を呼び出し、サーバーパラメータのセットを指定します。

構文

```
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 Amazon GameLift Servers Agent
to enable further communication.
Aws::GameLift::Server::InitSDKOutcome initSdkOutcome =
    Aws::GameLift::Server::InitSDK();
```

InitSDK()

Amazon GameLift Servers SDK を初期化します。他のAmazon GameLift Servers 関連の初期化が実行される前に、起動時にこのメソッドを呼び出します。このアクションには、ゲームサーバープロセスと Amazon GameLift Servers サービス間の通信を設定するサーバーパラメータのセットが必要です。べき等性トークンを使用するため、失敗したときにこの呼び出しを安全に再試行できます。

ゲームサーバービルドを Amazon GameLift Servers マネージド EC2 フリート、または Amazon GameLift Servers エージェントを使用する Amazon GameLift Servers Anywhere フリート、または コンテナフリートにデプロイする場合は、サーバーパラメータなしで [InitSDK\(\)](#) を呼び出します。

構文

```
Server::InitSDKOutcome Server::initSdkOutcome = InitSDK(serverParameters);
```

パラメータ

[ServerParameters](#)

Amazon GameLift Servers Anywhere フリエットのゲームサーバーを初期化するには、次の情報を含む `ServerParameters` オブジェクトを作成します。

- ゲームサーバーへの接続に使用される WebSocket 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 Servers Anywhere コンピューティングを含む Amazon GameLift Servers フリエットの ID。
- Amazon GameLift Servers オペレーションによって生成された認可トークン。

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。

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 Servers 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 Amazon GameLift Servers Agent
to enable further communication.
Aws::GameLift::Server::InitSDKOutcome initSdkOutcome =
    Aws::GameLift::Server::InitSDK(serverParameters);
```

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 Servers 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。

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 Servers 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。

- サーバードキュメントがリッスンするポートの番号。
- Amazon GameLift Servers 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。

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 Servers 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なら、このメソッドを呼び出します。パラメータで、Amazon GameLift Servers が特定の状況で呼び出すコールバック関数の名前を指定します。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 Servers 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 Servers 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;
}
```

ProcessEnding()

サーバープロセスが終了していることを Amazon GameLift Servers に通知します。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その他すべてのクリーンアップタスクの後そして、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、フリートイベントが生成されます。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。

構文

```
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 Servers 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。

構文

```
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。

構文

```
GenericOutcome  
    UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCreationPolicy  
        newPlayerSessionPolicy);
```

パラメータ

playerCreationSessionPolicy

型: PlayerSessionCreationPolicy [enum](#) 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。

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 [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 GameLift Servers から `onProcessTerminate()` コール

バックを受信した後、アクションを実行します。Amazon GameLift Servers は、次の理由で `onProcessTerminate()` を呼び出します。

- サーバープロセスが健全性異常を報告したり、Amazon GameLift Servers に応答しなかった場合。
- 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。

構文

```
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

```
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、検証が必要であることを、Amazon GameLift Servers に通知します。Amazon GameLift Servers は、プレイヤーセッション ID が有効であることを検証します。検証できたら、Amazon GameLift Servers はプレイ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 playerId)
```

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 Servers 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ことを Amazon GameLift Servers に通知します。それに応じて、Amazon GameLift Servers はプレイヤースロットを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 Servers 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

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 [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。NextToken を省略し、Limit 値を 10 に設定すると、Amazon GameLift Servers 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
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。新規プレイヤーが正常にマッチングされると、Amazon GameLift Servers 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 Servers が ID を 1 つ生成します。
- 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イキングデータ。

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。

構文

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

パラメータ

[StopMatchBackfillRequest](#)

キャンセルするマッチメイ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 GameLift Servers Anywhere コンピューティングリソースと Amazon GameLift Servers 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。この証明書パスは、コンピューティングデバイスを Amazon GameLift Servers Anywhere フリートに登録するとき使用できます。詳細については、「[RegisterCompute](#)」を参照してください。

構文

```
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 GameLift Servers が他のとやり取りすることを許可する IAM ロール認証情報を取得します。詳細については、「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」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 Servers 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、ゲームセッションのたびにサーバープロセスを終了しない場合は、`Destroy()`、`InitSDK()` の順に呼び出して再初期化してから、`ProcessReady()` で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 Servers に通知します。

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

```
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);  
}
```

C# サーバー SDK 5.x for Amazon GameLift Servers -- アクション

サーバー SDK 5.x リファレンスを使用して、Amazon GameLift Servers でホスティングするマルチプレイヤーゲームを統合します。統合プロセスのガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。Unity 用 Amazon GameLift Servers プラグインを使用している場合は、「[Unity 用 Amazon GameLift Servers プラグイン \(サーバー SDK 5.x\)](#)」も参照してください。

C# サーバー SDK 5.x for Amazon GameLift Servers -- データ型

Amazon GameLift Servers C# サーバー SDK 5.x リファレンスを使用して、Amazon GameLift Servers でホスティングするマルチプレイヤーゲームを統合します。統合プロセスのガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。Unity 用 Amazon GameLift Servers プラグインを使用している場合は、「[Unity 用 Amazon GameLift Servers プラグイン \(サーバー SDK 5.x\)](#)」も参照してください。

[C# サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)

データ型

- [LogParameters](#)
- [MetricsParameters](#)

- [ProcessParameters](#)
- [UpdateGameSession](#)
- [GameSession](#)
- [ServerParameters](#)
- [StartMatchBackfillRequest](#)
- [プレイヤー](#)
- [DescribePlayerSessionsRequest](#)
- [StopMatchBackfillRequest](#)
- [GetFleetRoleCredentialsRequest](#)
- [AttributeValue](#)
- [AwsStringOutcome](#)
- [GenericOutcome](#)
- [MetricsManagerOutcome](#)
- [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 Servers 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。ゲームサーバーは、[ProcessReady\(\)](#) 呼び出しで LogParameters to Amazon GameLift Servers を送信します。

プロパティ	説明
LogPaths	Amazon GameLift Servers 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\MyGame\sessionLogs (Windows インスタンスの場合) となります。 タイプ: List<String> 必須: いいえ

MetricsParameters

このデータ型を使用して、ゲームサーバーのメトリクス収集とクラッシュレポートを設定します。ゲームサーバーは、[InitMetrics\(\)](#) 呼び出しで MetricsParameters を Amazon GameLift Servers に送信します。

プロパティ	説明
StatsdHost	メトリクス収集用の StatsD サーバーのホスト名または IP アドレス。 タイプ: String 必須: いいえ
StatsdPort	メトリクス収集用の StatsD サーバーのポート番号です。

	タイプ: Integer 必須: いいえ
CrashReporterHost	クラッシュレポーターサーバーのホスト名または IP アドレスです。 タイプ: String 必須: いいえ
CrashReporterPort	クラッシュレポーターサーバーのポート番号です。 タイプ: Integer 必須: いいえ
FlushIntervalMs	メトリクスデータをサーバーにフラッシュする際のミリ秒単位の間隔です。 タイプ: Integer 必須: いいえ
MaxPacketSize	サーバーに送信されるメトリクスパケットの最大サイズ (バイト) です。 タイプ: Integer 必須: いいえ

ProcessParameters

このデータ型には、[ProcessReady\(\)](#)呼び出しの中で Amazon GameLift Servers に送信されるパラメータのセットが含まれます。

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。

	タイプ: <code>Aws::GameLift::Server::</code> LogParameters 必須: はい
OnHealthCheck	Amazon GameLift Servers 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 Servers 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 Servers 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、Amazon GameLift Servers はサーバープロセスを異常と記録します。 タイプ: <code>void OnHealthCheckDelegate()</code> 必須: はい
OnProcessTerminate	Amazon GameLift Servers がサーバープロセスを強制的にシャットダウンするために呼び出すコールバック関数です。この関数を呼び出すと、Amazon GameLift Servers 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: <code>void OnProcessTerminateDelegate()</code> 必須: はい

OnStartGameSession

Amazon GameLift Servers サービスが新しいゲームセッションを開始するために呼び出すコールバック関数の名前。Amazon GameLift Servers は、クライアントリクエスト [CreateGameSession](#) に応じて、この関数を呼び出します。コールバック関数は [GameSession](#) オブジェクトを取得します。

タイプ: void OnStartGameSession
Delegate([GameSession](#))

必須: はい

OnUpdateGameSession

Amazon GameLift Servers 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の名前。Amazon GameLift Servers は、マッチバックフィルリクエストが処理されると、この関数を呼び出します。これは、[GameSession](#) オブジェクト、ステータス更新 (updateReason) そして、マッチバックフィルチケット ID を渡します。

型: 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。

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、このデータ型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	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

プロパティ	説明
	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 <code>arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token></code> です。 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。 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

プロパティ	説明
ポート	ゲームセッションのポート番号。Amazon GameLift Servers 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 Servers 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
MatchmakerData	ゲームセッションの作成に使用されたマッチメイ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イ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

プロパティ	説明
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: Dictionary<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>.amazon gamelift.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 Servers Anywhere サーバーと Amazon GameLift Servers 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\(\)](#) で新しいサーバードプロセスを起動するときに使用されます。Amazon GameLift Servers 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 Servers Anywhere の一部として RegisterCompute を実行したときに、GameLiftServerSdkEndpoint が返されます。 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	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に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。 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。 タイプ: String 必須: はい
AuthToken	Amazon GameLift Servers によって生成され、サーバーを Amazon GameLift Servers に対して認証するための認証トークン。詳細について

プロパティ	説明
	は、「 GetComputeAuthToken 」を参照してください。
	タイプ: String
	必須: はい

StartMatchBackfillRequest

マッチメイ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\(\)](#) 呼び出しで Amazon GameLift Servers に送信します。

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

プロパティ	説明
	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: <code>List<Player></code> 必須: はい
<code>TicketId</code>	マッチメイ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 GameLift Servers が値を生成します。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。 タイプ: <code>String</code> 必須: いいえ

プレイヤー

マッチメイキングのプレイヤーを表します。マッチメイキングリクエストを開始すると、プレイヤーにはプレイヤー ID、属性、必要に応じてレイテンシーデータが含まれます。マッチが成立すると、Amazon GameLift Servers がチーム情報を追加します。

プロパティ	説明
<code>LatencyInMS</code>	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、プレイ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。

プロパティ	説明
	タイプ: Dictionary<string, int> 必須: いいえ
PlayerAttributes	マッチメイ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、マッチメイキングルールセットで使用されている PlayerAttributes 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、「Attribute Value」を参照してください。 タイプ: Dictionary<string, Attribute Value> 必須: いいえ
PlayerId	プレイヤーを表す一意の識別子。 タイプ: String 必須: いいえ
チーム	マッチでプレイ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イ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

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。 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。 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。 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

プロパティ	説明
	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。 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
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

マッチメイ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 Servers サービスに送信します。

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: 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: 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。 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 GameLift Servers 用に IAM サービスロールをセットアップする](#) を参照してください。

プロパティ	説明
RoleArn	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 AWS リソースネーム (ARN)。

プロパティ	説明
	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 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います。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。 タイプ: string 必須: はい
いいえ	数値の属性値を表します。 タイプ: double 必須: はい
SL	文字列の属性値の配列を表します。

プロパティ	説明
	タイプ: <code>string[]</code> 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。 タイプ: <code>Dictionary<string, double></code> 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。 タイプ: <code>string</code> 必須: いいえ
Success	アクションが成功したかどうか。 タイプ: <code>bool</code> 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
Success	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

MetricsManagerOutcome

[InitMetrics\(\)](#) 呼び出しの結果を表します。成功時の MetricsManager インスタンスまたは失敗時のエラー情報が含まれます。

結果	メトリクスを収集およびレポートするための MetricsManager インスタンス。 タイプ: MetricsManager 必須: いいえ
Success	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
Success	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

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。 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。

プロパティ	説明
	タイプ: IList< the section called “PlayerSession” > 必須:
Success	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

プロパティ	説明
CreationTime	タイプ: long 必須: はい
FleetId	タイプ: string 必須: はい
GameSessionId	タイプ: string 必須: はい
IpAddress	タイプ: string 必須: はい
PlayerData	タイプ: string

プロパティ	説明
	必須: はい
PlayerId	タイプ: string 必須: はい
PlayerSessionId	タイプ: string 必須: はい
ポート	タイプ: int 必須: はい
ステータス	型: PlayerSessionStatus enum 。 必須: はい
TerminationTime	タイプ: long 必須: はい
DnsName	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。 タイプ: the section called “StartMatchBackfillResult” 必須: いいえ
Success	アクションが成功したかどうか。

プロパティ	説明
	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

プロパティ	説明
TicketId	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。 タイプ: the section called “GetComputeCertificateResult” 必須: いいえ
Success	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

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	タイプ: string 必須: はい
ComputeName	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。 タイプ: the section called “GetFleetRoleCredentialsResult” 必須: いいえ
Success	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 必須: いいえ

プロパティ	説明
SessionToken	AWS リソースとやり取りす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
Success	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

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。 タイプ: DateTime 必須: いいえ
Success	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: GameLiftErrorType enum 。 必須: いいえ
ErrorMessage	エラータイプの名前。 タイプ: string 必須: いいえ
ErrorName	エラーメッセージです。 タイプ: string 必須: いいえ

列挙型

サーバー SDK for Amazon GameLift Servers (C#) で定義されている列挙体は、次のとおりです。

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に失敗しました。
- LOCAL_CONNECTION_FAILED - Amazon GameLift Servers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 Servers 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 gameSession または playerSession のフリートと一致しません。
- GAMELIFT_CLIENT_NOT_INITIALIZED - Amazon GameLift Servers クライアントは初期化されていません。
- GAMELIFT_SERVER_NOT_INITIALIZED - Amazon GameLift Servers クライアントは初期化されていません。
- GAME_SESSION_ENDED_FAILED - サーバー SDK for Amazon GameLift Servers 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_NOT_READY - Amazon GameLift Servers 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 - サーバー SDK for Amazon GameLift Servers 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 Servers サーバーまたはクライアントは Initialize() で初期化されていません。
- NO_TARGET_ALIASID_SET - ターゲットの aliasId 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 - サーバー SDK for Amazon GameLift Servers 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` - サーバード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、`PlayerSessions` 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` - サーバード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` - サーバード SDK for Amazon GameLift Servers 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 サーバード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 サーバード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ド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

- `アクティブ`
- `COMPLETED`

- NOT_SET
- RESERVED
- TIMEDOUT

C# サーバー SDK 5.x for Amazon GameLift Servers -- データ型

トピック

- [GetSdkVersion\(\)](#)
- [InitMetrics\(\)](#)
- [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。

構文

```
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();
```

InitMetrics()

サーバーパフォーマンスデータを収集およびレポートするためのメトリクスシステムを初期化します。最良の結果を得るには、[InitSDK\(\)](#) の前にこのメソッドを呼び出して、SDK の初期化中にメトリクス収集を有効にします。

構文

```
MetricsManagerOutcome InitMetrics();  
MetricsManagerOutcome InitMetrics(MetricsParameters metricsParameters);
```

パラメータ

[MetricsParameters](#) (オプション)

メトリクス収集の設定パラメータ。指定しない場合は、環境変数で上書きできる既定値を使用します。

戻り値

成功すると、MetricsManager インスタンスを含む [the section called “MetricsManagerOutcome”](#) オブジェクトを返します。成功しなかった場合、エラーメッセージを返します。

例

```
// Initialize with default parameters (uses environment variables if available)  
var outcome = GameLiftServerAPI.InitMetrics();  
if (outcome.Success) {  
    var metricsManager = outcome.Result;  
} else {  
    Console.WriteLine("Failed to initialize metrics: " + outcome.Error.ErrorMessage);  
}
```

```
}

// Initialize with custom parameters
var metricsParams = new MetricsParameters("localhost", 8125, "crash-host", 9999, 1000,
    1024);
var customOutcome = GameLiftServerAPI.InitMetrics(metricsParams);
if (customOutcome.Success) {
    var metricsManager = customOutcome.Result;
}
```

InitSDK()

マネージド EC2 フリート用の Amazon GameLift Servers SDK を初期化します。起動時に、他の Amazon GameLift Servers 関連の初期化が実行される前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 Servers サービス間の通信をセットアップします。べき等性トークンを使用するため、失敗したときにこの呼び出しを安全に再試行できます。

構文

```
GenericOutcome 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.
GenericOutcome initSDKOutcome = GameLiftServerAPI.InitSDK();
```

InitSDK()

Anywhere フリーットの Amazon GameLift Servers SDK を初期化します。起動時に、他の Amazon GameLift Servers 関連の初期化が実行される前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 Servers サービス間の通信をセットアップするための明示的なサーバーパラメータが必要です。べき等性トークンを使用するため、失敗したときにこの呼び出しを安全に再試行できます。

構文

```
GenericOutcome InitSDK(ServerParameters serverParameters);
```

パラメータ

[ServerParameters](#)

Amazon GameLift Servers Anywhere フリートのゲームサーバーを初期化するには、次の情報を含む `ServerParameters` オブジェクトを作成します。

- ゲームサーバーへの接続に使用される WebSocket 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 Servers Anywhere コンピューティングを含む Amazon GameLift Servers フリートの ID。
- Amazon GameLift Servers オペレーションによって生成された認可トークン。

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
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 Servers 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。

構文

```
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していることを Amazon GameLift Servers に通知します。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その他すべてのクリーンアップタスクの後そして、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、フリートイベントが生成されます。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。

構文

```
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 Servers 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。

構文

```
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。

構文

```
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。

構文

```
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 GameLift Servers 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 Servers は次の理由で `onProcessTerminate()` を呼び出します。

- サーバードプロセスが健全性異常を報告したり、Amazon GameLift Servers に応答しなかった場合。
- 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。

構文

```
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、6373559680000000000 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ドプロセスに接続し、検証が必要であることを、Amazon GameLift Servers に通知します。Amazon GameLift Servers は、プレイヤーセッション ID が有効であることを検証します。検証できたら、Amazon GameLift Servers はプレイヤーセッションの状態を 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 GameLift 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 playerId)
{
    GenericOutcome acceptPlayerSessionOutcome =
    GameLiftServerAPI.AcceptPlayerSession(playerId);
    if(acceptPlayerSessionOutcome.Success)
    {
        connectionToSessionMap.emplace(connection, playerId);
        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ことを Amazon GameLift Servers に通知します。それに応じて、Amazon GameLift Servers はプレイヤーセッションを利用可能に変更します。

構文

```
GenericOutcome RemovePlayerSession(String playerId)
```

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 Servers 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。

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 [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。NextToken を省略し、Limit 値を 10 に設定すると、Amazon GameLift Servers 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
```

```
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。新規プレイヤーが正常にマッチングされると、Amazon GameLift Servers 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](#)

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。

構文

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

パラメータ

[StopMatchBackfillRequest](#)

停止しているマッチメイ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。この証明書パスは、コンピューティングデバイスを Amazon GameLift Servers Anywhere フリートに登録するときに使用できます。詳細については、「[RegisterCompute](#)」を参照してください。

構文

```
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上の TLS 証明書へのパス。Amazon GameLift Servers マネージドフリートを使用する場合、このパスには次の内容が含まれます。

- `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 GameLift Servers が他のとやり取りすることを許可する IAM ロール認証情報を取得します。詳細については、「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」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 Servers 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、ゲームセッションのたびにサーバープロセスを終了しない場合は、Destroy()、InitSDK() の順に呼び出して再初期化してから、ProcessReady() で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 Servers に通知します。

構文

```
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);  
}
```

Go サーバー SDK for Amazon GameLift Servers – アクション

サーバー SDK 5.x リファレンスを使用して、Amazon GameLift Servers でホスティングするマルチプレイヤーゲームを統合します。統合プロセスのガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」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。

Go サーバー SDK for Amazon GameLift Servers -- データ型

サーバー SDK リファレンスを使用して、Amazon GameLift Servers でホスティングするマルチプレイヤーゲームを統合します。統合プロセスのガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」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。

[Go サーバー SDK for Amazon GameLift Servers – アクション](#)

データ型

- [LogParameters](#)
- [MetricsParameters](#)
- [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するオブジェクトで、ゲームセッション終了時に Amazon GameLift Servers でアップロードして保存します。ゲームサーバーは、[ProcessReady\(\)](#) 呼

び出しで `ProcessParameters` オブジェクトの一部として `LogParameters` を Amazon GameLift Servers に提供します。

プロパティ	説明
<code>LogPaths</code>	Amazon GameLift Servers 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 <code>MyGame\sessionLogs\</code> 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 <code>c:\game\MyGame\sessionLogs</code> (Windows インスタンスの場合) となります。 タイプ: <code>[]string</code> 必須: いいえ

MetricsParameters

メトリクスシステムを初期化するための設定パラメータを通信するオブジェクト。この設定は、StatsD レポート、クラッシュレポート、メトリクス処理の動作を設定するために使用されます。ゲームサーバーは、[InitMetrics\(\)](#) の呼び出しの一部として `MetricsParameters` を Amazon GameLift Servers に提供します。

プロパティ	説明
<code>StatsdHost</code>	メトリクスレポート用の StatsD サーバーホスト (「localhost」など)。 タイプ: <code>string</code> 必須: はい
<code>StatsdPort</code>	メトリクスレポートの StatsD サーバーポート (8125 など)。 タイプ: <code>int</code>

	必須: はい
CrashReporterHost	クラッシュ追跡とプロセスモニタリング用のクラッシュレポーターホスト。 タイプ: string 必須: はい
CrashReporterPort	クラッシュ追跡とプロセスモニタリング用のクラッシュレポーターポート。 タイプ: int 必須: はい
FlushIntervalMs	メトリクスのフラッシュ間隔はミリ秒単位です。メトリクスが StatsD に送信される頻度を制御します。 タイプ: int 必須: はい
MaxPacketSize	バイト単位のメトリクスの最大パケットサイズ。StatsD に送信される UDP パケットのサイズを制限します。 タイプ: int 必須: はい

ProcessParameters

サーバープロセスと Amazon GameLift Servers の間の通信を記述するオブジェクト。サーバープロセスは、この情報を Amazon GameLift Servers 呼び出しで [ProcessReady\(\)](#) に提供します。

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持つオブジェクトです。Amazon GameLift Servers はこれらのファイルをコピーして保存し、後で参照できるようにします。 タイプ: LogParameters

	必須: いいえ
OnHealthCheck	Amazon GameLift Servers がサーバープロセスにヘルスステータスレポートを要求するために呼び出すコールバック関数です。Amazon GameLift Servers はこの関数を 60 秒ごとに呼び出し、応答を最大 60 秒間待機します。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。応答が返されない場合、Amazon GameLift Servers はサーバープロセスを正常ではないと記録します。 タイプ: OnHealthCheck func() bool 必須: いいえ
OnProcessTerminate	Amazon GameLift Servers がサーバープロセスを強制的にシャットダウンするために呼び出すコールバック関数です。この関数を呼び出すと、Amazon GameLift Servers はサーバープロセスがシャットダウンするまで最大 5 分間待機し、サーバープロセスがシャットダウンする前に ProcessEnding() 呼び出しで応答するのを待ちます。 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 GameLift Servers が呼び出すコールバック関数。Amazon GameLift Servers は、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が処理された際にこの関数を呼び出します。これは、GameSession オブジェクト、ステータス更新 (updateReason) そして、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	Amazon GameLift Serversが呼び出して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コールバック関数。Amazon GameLift Servers 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	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()

プロパティ	説明
	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 <code>arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token></code> 形式があります。 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。 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

プロパティ	説明
	必須: いいえ
ポート	ゲームセッションのポート番号。Amazon GameLift Servers 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 Servers 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
MatchmakerData	ゲームセッションの作成に使用されたマッチメイ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イ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

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: String 必須: いいえ

ServerParameters

Amazon GameLift Servers Anywhere サーバーと Amazon GameLift Servers 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 Servers 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
WebSocket URL	Amazon GameLift Servers Anywhere コンピューティングリソース用に RegisterCompute を実行すると、GameLiftServerSdkEndpoint Amazon GameLift Servers が返されます。 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

プロパティ	説明
	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的識別子。 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。 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、「RegisterCompute」を参照してください。 タイプ: string 必須: はい
AuthToken	Amazon GameLift Servers によって生成され、サーバーを Amazon GameLift Servers に対して認証するための認証トークン。詳細については、「GetComputeAuthToken」を参照してください。 タイプ: string 必須: はい

StartMatchBackfillRequest

マッチメイ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\(\)](#) 呼び出しで Amazon GameLift Servers に送信します。

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

プロパティ	説明
	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 形式の一意的識別子。元のゲームセッションのマッチメーカー 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	マッチメイキングまたはバックフィルリクエストチケットの一意的識別子。値を指定しない場合、Amazon GameLift Servers が値を生成します。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。 タイプ: String 必須: いいえ

プレイヤー

マッチメイキングでプレイヤーを表すオブジェクト。マッチメイキングリクエストを開始すると、プレイヤーにはプレイヤー ID、属性、必要に応じてレイテンシーデータが含まれます。マッチが成立すると、Amazon GameLift Servers が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。

プロパティ	説明
	このプロパティを使用すると、プレイ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: <code>map[string] int</code> 必須: いいえ
PlayerAttributes	マッチメイ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、マッチメイキングルールセットで使用されている PlayerAttributes 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、「AttributeValue」を参照してください。 タイプ: <code>map[string] AttributeValue</code> 必須: いいえ
PlayerId	プレイヤーを表す一意の識別子。 タイプ: <code>String</code> 必須: いいえ
チーム	マッチでプレイ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イキングルールセットで定義します。 タイプ: <code>String</code> 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、この情報を [DescribePlayerSessions\(\)](#) 呼び出しで Amazon GameLift Servers に提供します。

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 の形式は <code>arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string></code> 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。 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) 内のプレイヤーの接続や検証は行われていません。

プロパティ	説明
NextToken	タイプ: String
	必須: いいえ
	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
Limit	必須: いいえ
	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

マッチメイ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 Servers サービスに送信します。

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。

プロパティ	説明
	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拡張するロール認証情報。詳細については、[Amazon GameLift Servers 用に IAM サービスロールをセットアップする](#) 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 必須: はい

[Go サーバー SDK for Amazon GameLift Servers -- データ型](#)

アクション

- [GetSdkVersion\(\)](#)
- [InitMetrics\(\)](#)
- [InitMetricsFromEnvironment\(\)](#)
- [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。

構文

```
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()
```

InitMetrics()

Amazon GameLift Servers SDK のメトリクス収集を初期化します。このメソッドは、サーバーのパフォーマンスとヘルスのモニタリングに役立つメトリクスレポートを設定します。このメソッドは、[InitSDK\(\)](#) の後、[ProcessReady\(\)](#) の前に呼び出します。

構文

```
func InitMetrics() error  
func InitMetrics(metricsParameters MetricsParameters) error
```

パラメータ

MetricsParameters (オプション)

メトリクスコレクションを設定する MetricsParameters オブジェクト。指定しない場合、デフォルトのメトリクス設定が使用されます。MetricsParameters 構造には、次のフィールドが含まれます。

- StatsdHost - StatsD サーバーのホスト名または IP アドレスを指定します。
- StatsdPort - StatsD サーバーのポート番号を指定します。
- CrashReporterHost - クラッシュレポーターサービスのホスト名または IP アドレスを指定します。
- CrashReporterPort - クラッシュレポーターサービスのポート番号を指定します。
- FlushIntervalMs - メトリクスデータをフラッシュする間隔 (ミリ秒) を指定します。
- MaxPacketSize - メトリクスパケットの最大サイズ (バイト単位) を指定します。

MetricsParameters 構造体の詳細については、「[サーバー SDK 5.x for C# のデータ型](#)」を参照してください。

戻り値

成功すると、nil はメトリクス収集が正常に初期化されたことを示すエラーを返します。

例

デフォルト設定でメトリクスを初期化します。

```
err := server.InitMetrics()
```

カスタム設定を使用してメトリクスを初期化します。

```
metricsParams := MetricsParameters{
    StatsdHost:      "localhost",
    StatsdPort:      8125,
    CrashReporterHost: "localhost",
    CrashReporterPort: 9125,
    FlushIntervalMs:  5000,
    MaxPacketSize:    1024,
}
```

```
err := server.InitMetrics(metricsParams)
```

InitMetricsFromEnvironment()

環境変数の設定を使用して Amazon GameLift Servers SDK のメトリクス収集を初期化します。このメソッドは、ランタイム環境から派生したデフォルト設定を使用してメトリクスレポートを設定します。

このメソッドは、[InitSDK\(\)](#) の後、[ProcessReady\(\)](#) の前に呼び出します。

構文

```
func InitMetricsFromEnvironment() error
```

戻り値

成功すると、nil エラーを返し、環境設定を使用してメトリクス収集が正常に初期化されたことを示します。

例

```
err := server.InitMetricsFromEnvironment()
```

InitSDK()

Amazon GameLift Servers SDK を初期化します。起動時に、他の Amazon GameLift Servers 関連の初期化が実行される前にこのメソッドを呼び出します。このメソッドで、サーバーと Amazon GameLift Servers サービスとの通信を確立します。

構文

```
func InitSDK(params ServerParameters) error
```

パラメータ

[ServerParameters](#)

Amazon GameLift Servers Anywhere フリートのゲームサーバーを初期化するには、次の情報を含む `ServerParameters` オブジェクトを作成します。

- ゲームサーバーへの接続に使用される WebSocket 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 Servers Anywhere コンピューティングを含む Amazon GameLift Servers フリートの ID。
- Amazon GameLift Servers オペレーションによって生成された認可トークン。

Amazon GameLift Servers マネージド EC2 フリートのゲームサーバーを初期化するには、パラメータなしで `ServerParameters` オブジェクトを作成します。この呼び出しで、Amazon GameLift Servers エージェント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 Servers サービスに自動的に接続します。

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 Servers 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 Amazon GameLift Servers Agent
to enable further communication.
```

```
err := server.InitSDK(serverParameters)
```

Amazon GameLift Servers マネージド EC2 の例

```
//Define the server parameters
serverParameters := ServerParameters {}

//Call InitSDK to establish a local connection with the Amazon GameLift Servers Agent
to enable further communication.
err := server.InitSDK(serverParameters)
```

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 Servers 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。

構文

```
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。

- サーバープロセスと通信するために Amazon GameLift Servers 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スンするポートの番号。
- Amazon GameLift Servers 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していることを Amazon GameLift Servers に通知します。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その他すべてのクリーンアップタスクの後そして、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、フリーイベントが生成されます。プロセスがエラーで終了した場合、生成されるフリーイベントは SERVER_PROCESS_TERMINATED_UNHEALTHY です。

構文

```
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 Servers 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。

構文

```
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。

構文

```
func UpdatePlayerSessionCreationPolicy(policy model.PlayerSessionCreationPolicy) error
```

パラメータ

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。

- **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。

構文

```
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 GameLift Servers 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 Servers は次の理由で `onProcessTerminate()` を呼び出します。

- サーバードプロセスが健全性異常を報告したり、Amazon GameLift Servers に応答しなかった場合。
- 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。

構文

```
func GetTerminationTime() (int64, error)
```

戻り値

成功すると、サーバード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、`6373559680000000000` 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ドプロセスに接続し、検証が必要であることを、Amazon GameLift Servers に通知します。Amazon GameLift Servers は、プレイヤーセッション ID が有効であることを検証します。検証できたら、Amazon GameLift Servers はプレイヤースロットの状態を `RESERVED` から `ACTIVE` に変更します。

構文

```
func AcceptPlayerSession(playerSessionID string) error
```

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 Servers 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ことを Amazon GameLift Servers に通知します。それに応じて、Amazon GameLift Servers はプレイヤースロットを利用可能に変更します。

構文

```
func RemovePlayerSession(playerSessionID string) error
```

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 Servers 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

構文

```
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。NextToken を省略し、Limit 値を 10 に設定すると、Amazon GameLift Servers 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。

```
// 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。新規プレイヤーが正常にマッチングされると、Amazon GameLift Servers 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

```
func StartMatchBackfill(req request.StartMatchBackfillRequest)
(result.StartMatchBackfillResult, error)
```

パラメータ

[StartMatchBackfillRequest](#)

StartMatchBackfillRequest オブジェクトは、次の情報を伝達します。

- 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は Amazon GameLift Servers が 1 つ生成します。
- 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カーデータ。

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。

構文

```
func StopMatchBackfill(req request.StopMatchBackfillRequest) error
```

パラメータ

[StopMatchBackfillRequest](#)

キャンセルするマッチメイ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。この証明書パスは、コンピューティングデバイスを Amazon GameLift Servers Anywhere フリートに登録するときに使用できます。詳細については、「[RegisterCompute](#)」を参照してください。

構文

```
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上の TLS 証明書へのパス。Amazon GameLift Servers マネージドフリートを使用する場合、このパスには次の内容が含まれます。
 - `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()

アクセス許可を他のに拡張するために作成したサービスロールの認証情報を取得します AWS のサービス Amazon GameLift Servers。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 GameLift Servers 用に IAM サービスロールをセットアップする](#)」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 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。
- `SecretAccessKey` - 認証のためのシークレットアクセスキー ID。
- `SessionToken` - AWS リソースとやり取りする現在の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 Servers 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、ゲームセッションのたびにサーバープロセスを終了しない場合は、Destroy()、InitSDK() の順に呼び出して再初期化してから、ProcessReady() で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 Servers に通知します。

構文

```
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)
    }
}
```

C++ (Unreal) サーバー SDK 5.x for Amazon GameLift Servers -- アクション

Amazon GameLift Servers Unreal サーバー SDK 5.x リファレンスは、Amazon GameLift Servers で使用するマルチプレイヤーゲームの準備を整えるのに役立ちます。統合プロセスの詳細については、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。Unreal 用 Amazon GameLift Servers プラグインを使用している場合は、「[Unreal Engine 用 Amazon GameLift Servers プラグイン](#)」も参照してください。

Note

このトピックでは、Unreal Engine を構築するときに使用できる Amazon GameLift Servers 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。

C++ (Unreal)サーバー SDK 5.x for Amazon GameLift Servers -- データ型

Amazon GameLift Servers Unreal サーバー SDK 5.x リファレンスは、Amazon GameLift Servers で使用するマルチプレイヤーゲームの準備を整えるのに役立ちます。統合プロセスの詳細については、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。Unreal 用 Amazon GameLift Servers プラグインを使用している場合は、「[Unreal Engine 用 Amazon GameLift Servers プラグイン](#)」も参照してください。

Note

このトピックでは、Unreal Engine を構築するときに使用できる Amazon GameLift Servers 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。

[C++ \(Unreal\) サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)

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](#)

FProcessParameters

このデータ型には、[ProcessReady\(\)](#) の中で Amazon GameLift Servers に送信されるパラメータのセットが含まれます。

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持つオブジェクトです。Amazon GameLift Servers はこれらのファイルをコピーして保存し、後で参照できるようにします。 タイプ: TArray<FString>

	必須: いいえ
OnHealthCheck	Amazon GameLift Servers がサーバープロセスにヘルスステータスレポートを要求するために呼び出すコールバック関数です。Amazon GameLift Servers はこの関数を 60 秒ごとに呼び出し、応答を最大 60 秒間待機します。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。応答が返されない場合、Amazon GameLift Servers はサーバープロセスを正常ではないと記録します。 このプロパティは、<code>DECLARE_DELEGATE_RetVal(bool, FOnHealthCheck)</code> として定義されるデリゲート関数です。 タイプ: <code>FOnHealthCheck</code> 必須: いいえ
OnProcessTerminate	Amazon GameLift Servers がサーバープロセスを強制的にシャットダウンするために呼び出すコールバック関数です。この関数を呼び出すと、Amazon GameLift Servers はサーバープロセスがシャットダウンするまで最大 5 分間待機し、サーバープロセスがシャットダウンする前に ProcessEnding() を呼び出して応答するのを待ちます。 タイプ: <code>FSimpleDelegate</code> 必須: はい

OnStartGameSession

Amazon GameLift Servers が新しいゲームセッションをアクティブ化するために呼び出すコールバック関数。Amazon GameLift Servers は、クライアントリクエスト [CreateGameSession](#) に応じて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 GameLift Servers が呼び出すコールバック関数。Amazon GameLift Servers は、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が処理された際にこの関数を呼び出します。これは、[GameSession](#) オブジェクト、ステータス更新 (`updateReason`) そして、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	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: <code>Aws::GameLift::Server::GameSession</code> 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: <code>enum class UpdateReason</code> • <code>MATCHMAKING_DATA_UPDATED</code> • <code>BACKFILL_FAILED</code> • <code>BACKFILL_TIMED_OUT</code> • <code>BACKFILL_CANCELLED</code>

プロパティ	説明
	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的識別子。ゲームセッション ARN の形式は <code>arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token></code> 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的識別子。 タイプ: char[] 必須: いいえ

プロパティ	説明
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 Servers 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 Servers 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イ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イ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: <code>char[]</code> 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: <code>GameProperty[]</code> 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>.amazon gamelift.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 Servers Anywhere サーバーと Amazon GameLift Servers 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 Servers 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 Servers Anywhere コンピューティングリソース用に RegisterCompute を実行すると、GameLiftServerSdkEndpoint Amazon GameLift Servers が返されます。 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、「RegisterCompute」を参照してください。 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、「RegisterCompute」を参照してください。 タイプ: char[] 必須: はい
authToken	Amazon GameLift Servers によって生成され、サーバーを Amazon GameLift Servers に対して認証するための認証トークン。詳細については、「GetComputeAuthToken」を参照してください。 タイプ: char[] 必須: はい

FStartMatchBackfillRequest

マッチメイ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\(\)](#) 呼び出しで Amazon GameLift Servers に送信します。

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	マッチメイ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 GameLift Servers ソリューションが値を生成します。この識別子を使用してマッ

プロパティ	説明
	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: <code>char[]</code> 必須: いいえ

FPlayer

このデータ型はマッチメイキングのプレイヤーを表します。マッチメイキングリクエストを開始すると、プレイヤーにはプレイヤー ID、属性、必要に応じてレイテンシーデータが含まれます。マッチが成立すると、Amazon GameLift Servers が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、プレイ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: <code>TMap>FString, int32<</code> 必須: いいえ
PlayerAttributes	マッチメイ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、マッチメイキングルールセットで使用されている <code>PlayerAttributes</code> 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。 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
チーム	マッチでプレイ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イ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、この情報を [DescribePlayerSessions\(\)](#) 呼び出しで Amazon GameLift Servers に提供します。

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: <code>std::string</code> 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: <code>FString</code> 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: <code>FString</code> 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: <code>int</code> 必須: いいえ

FStopMatchBackfillRequest

マッチメイ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 Servers サービスに送信します。

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: <code>FString</code> 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: <code>FString</code> 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: <code>FString</code> 必須: はい

FAttributeValue

これらの値を [FPlayer](#) 属性のキーと値のペアで使います。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 FAttributeValue 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
いいえ	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

このデータ型は、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 GameLift Servers 用に IAM サービスロールをセットアップする](#)」を参照してください。

プロパティ	説明
RoleArn	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 AWS 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
Success	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
Success	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 “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
Success	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

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
Success	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
Success	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

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 必須: はい
PlayerSessionId	タイプ: FString 必須: はい
ポート	タイプ: int

プロパティ	説明
	必須: はい
ステータス	型: 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。 タイプ: FGameLiftGetComputeCertificateResult && 必須: いいえ
Success	アクションが成功したかどうか。 タイプ: bool

プロパティ	説明
	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。 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。

プロパティ	説明
	タイプ: FGameLiftGetFleetRoleCredentialsResult<&& 必須: いいえ
Success	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。 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

プロパティ	説明
	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: <code>FDateTime</code> 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: <code>FString</code> 必須: いいえ
SessionToken	AWS リソースとやり取りす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: <code>FString</code> 必須: いいえ
Success	アクションが成功したかどうか。 タイプ: <code>bool</code> 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

FGameLiftError

プロパティ	説明
ErrorType	エラーのタイプ。

プロパティ	説明
	型: <code>GameLiftErrorType</code> enum。 必須: いいえ
ErrorName	エラータイプの名前。 タイプ: <code>std::string</code> 必須: いいえ
ErrorMessage	エラーメッセージです。 タイプ: <code>std::string</code> 必須: いいえ

列挙型

サーバー SDK for Amazon GameLift Servers (Unreal)で定義されている列挙体は、次のとおりです。

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に失敗しました。
- LOCAL_CONNECTION_FAILED - Amazon GameLift Servers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 Servers 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 gameSession または playerSession のフリートと一致しません。
- GAMELIFT_CLIENT_NOT_INITIALIZED - Amazon GameLift Servers クライアントは初期化されていません。
- GAMELIFT_SERVER_NOT_INITIALIZED - Amazon GameLift Servers クライアントは初期化されていません。
- GAME_SESSION_ENDED_FAILED - サーバー SDK for Amazon GameLift Servers 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_NOT_READY - Amazon GameLift Servers 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 - サーバー SDK for Amazon GameLift Servers 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 Servers サーバーまたはクライアントは Initialize() で初期化されていません。
- NO_TARGET_ALIASID_SET - ターゲットの aliasId 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 - サーバー SDK for Amazon GameLift Servers 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、PlayerSessions 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 - サーバー SDK for Amazon GameLift Servers 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。

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

- アクティブ
- COMPLETED
- NOT_SET
- RESERVED
- TIMEDOUT

[C++ \(Unreal\)サーバー SDK 5.x for Amazon GameLift Servers -- データ型](#)

トピック

- [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。

構文

```
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 フリート用の Amazon GameLift Servers SDK を初期化します。起動時に、他の Amazon GameLift Servers 関連の初期化が実行される前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 Servers サービス間の通信をセットアップします。べき等性トークンを使用するため、失敗したときにこの呼び出しを安全に再試行できます。

構文

```
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 Amazon GameLift Servers Agent  
to enable further communication.  
FGameLiftGenericOutcome initSdkOutcome = GameLiftSdkModule->InitSDK();
```

InitSDK()

Anywhere フリートまたはマネージドコンテナフリートの Amazon GameLift Servers SDK を初期化します。起動時に、他の Amazon GameLift Servers 関連の初期化が実行される前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 Servers サービス間の通信をセットアップするための明示的なサーバーパラメータが必要です。べき等性トークンを使用するため、失敗したときにこの呼び出しを安全に再試行できます。

構文

```
FGameLiftGenericOutcome InitSDK(serverParameters)
```

パラメータ

[FServerParameters](#)

Amazon GameLift Servers Anywhere フリートのゲームサーバーを初期化するには、次の情報を含む `ServerParameters` オブジェクトを作成します。

- ゲームサーバーへの接続に使用される WebSocket 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 Servers Anywhere コンピューティングを含む Amazon GameLift Servers フリートの ID。
- Amazon GameLift Servers オペレーションによって生成された認可トークン。

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 Amazon GameLift Servers Agent
to enable further communication.
FGameLiftGenericOutcome initSdkOutcome = GameLiftSdkModule->InitSDK(serverParameters);
```

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 Servers 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。

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 [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 Servers 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 Servers 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。

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 Amazon GameLift Servers 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していることを Amazon GameLift Servers に通知します。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その他すべてのクリーンアップタスクの後そして、プロセス

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、フリーイベントが生成されます。プロセスがエラーで終了した場合、生成されるフリーイベントは SERVER_PROCESS_TERMINATED_UNHEALTHY です。

構文

```
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. Amazon GameLift Servers 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 Amazon GameLift Servers 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 Servers 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。

構文

```
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, Amazon GameLift Servers 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。

構文

```
FGameLiftGenericOutcome UpdatePlayerSessionCreationPolicy(EPlayerSessionCreationPolicy
policy)
```

パラメータ

playerCreationSessionPolicy

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。

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 [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, Amazon GameLift Servers 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 GameLift Servers から `onProcessTerminate()` コールバックを受信した後、アクションを実行します。Amazon GameLift Servers は、次の理由で `onProcessTerminate()` を呼び出します。

- サーバープロセスが健全性異常を報告したり、Amazon GameLift Servers に応答しなかった場合。
- 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。

構文

```
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、6373559680000000000 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、検証が必要であることを、Amazon GameLift Servers に通知します。Amazon GameLift Servers は、プレイヤーセッション ID が有効であることを検証します。検証できたら、Amazon GameLift Servers はプレイ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& playerId)
```

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 Servers 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& playerId, const
FString& gameId)
{
    #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ことを Amazon GameLift Servers に通知します。それに応じて、Amazon GameLift Servers はプレイヤースロットを利用可能に変更します。

構文

```
FGameLiftGenericOutcome RemovePlayerSession(const FString& playerId)
```

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 Servers 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& playerId)
{
    #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

構文

```
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。NextToken を省略し、Limit 値を 10 に設定すると、Amazon GameLift Servers 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。

```
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。新規プレイヤーが正常にマッチングされると、Amazon GameLift Servers 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 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 Servers が ID を 1 つ生成します。
- 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イキングデータ。

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。

構文

```
FGameLiftGenericOutcome StopMatchBackfill (FStopMatchBackfillRequest  
&stopBackfillRequest);
```

パラメータ

[FStopMatchBackfillRequest](#)

キャンセルするマッチメイ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 GameLift Servers Anywhere コンピューティングリソースと Amazon GameLift Servers 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。この証明書パスは、コンピューティングデバイスを Amazon GameLift Servers Anywhere フリートに登録するときに使えます。詳細については、「[RegisterCompute](#)」を参照してください。

構文

```
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上の 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_T0_TCHAR(outres.GetCertificatePath());
        result.m_computeName = UTF8_T0_TCHAR(outres.GetComputeName());
        return FGameLiftGetComputeCertificateOutcome(result);
    }
    else {
        return FGameLiftGetComputeCertificateOutcome(FGameLiftError(outcome.GetError()));
    }
}
```

```
}  
#else  
return FGameLiftGetComputeCertificateOutcome(FGameLiftGetComputeCertificateResult());  
#endif  
}
```

GetFleetRoleCredentials()

Amazon GameLift Servers が他のとやり取りすることを許可する IAM ロール認証情報を取得します。詳細については、「[Amazon GameLift Servers ホストされたゲームサーバーを他の AWS リソースに接続する](#)」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_T0_TCHAR(outres.GetAssumedUserRoleArn());
result.m_assumedRoleId = UTF8_T0_TCHAR(outres.GetAssumedRoleId());
result.m_accessKeyId = UTF8_T0_TCHAR(outres.GetAccessKeyId());
result.m_secretAccessKey = UTF8_T0_TCHAR(outres.GetSecretAccessKey());
result.m_sessionToken = UTF8_T0_TCHAR(outres.GetSessionToken());
result.m_expiration = FDateTime::FromUnixTimestamp(outres.GetExpiration());
return FGameLiftGetFleetRoleCredentialsOutcome(result);
}
else {
    return FGameLiftGetFleetRoleCredentialsOutcome(FGameLiftError(outcome.GetError()));
}
#else
return
FGameLiftGetFleetRoleCredentialsOutcome(FGameLiftGetFleetRoleCredentialsResult());
#endif
}
```

Destroy()

Amazon GameLift Servers 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、ゲームセッションのたびにサーバープロセスを終了しない場合は、Destroy()、InitSDK() の順に呼び出して再初期化してから、ProcessReady() で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 Servers に通知します。

構文

```
FGameLiftGenericOutcome 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

```
// First call ProcessEnding()
FGameLiftGenericOutcome processEndingOutcome = GameLiftSdkModule->ProcessEnding();
```

```
// Then call Destroy() to free the SDK from memory
FGameLiftGenericOutcome destroyOutcome = GameLiftSdkModule->Destroy();

// Exit the process with success or failure
if (processEndingOutcome.IsSuccess() && destroyOutcome.IsSuccess())
{
    UE_LOG(GameServerLog, Log, TEXT("Server process ending successfully"));
}
else {
    if (!processEndingOutcome.IsSuccess()) {
        const FGameLiftError& error = processEndingOutcome.GetError();
        UE_LOG(GameServerLog, Error, TEXT("ProcessEnding() failed. Error: %s"),
            error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
            *error.m_errorMessage);
    }
    if (!destroyOutcome.IsSuccess()) {
        const FGameLiftError& error = destroyOutcome.GetError();
        UE_LOG(GameServerLog, Error, TEXT("Destroy() failed. Error: %s"),
            error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
            *error.m_errorMessage);
    }
}
```

バージョン 4 以前のサーバー SDK for Amazon GameLift Servers の使用

このリファレンスは、サーバー SDK for Amazon GameLift Servers、バージョン 4.x 以前を文書化しています。サーバー SDK は、ゲームサーバーが Amazon GameLift Servers サービスと通信するために使用するコア機能を提供します。例えば、ゲームサーバーはサービスからプロンプトを受け取り、ゲームセッションを開始および停止し、定期的にゲームセッションステータスをサービスに更新します。ホスティング用にゲームサーバーをデプロイする前に、サーバー SDK と統合します。

このサーバー SDK リファレンスを使用して、ホスティング向けのカスタムマルチプレイヤーゲームサーバーと Amazon GameLift Servers を統合します。統合プロセスのガイダンスについては、[「サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。」](#)を参照してください。

SDK バージョン 4.0.2 の場合は、[GitHub の公式リリース](#)からダウンロードできます。GameLiftLocal.jar を含む完全な SDK パッケージは、GameLift-CSharp-ServerSDK-4.0.2.zip アーティファクトで入手できます。

サーバー SDK for Amazon GameLift Servers の最新メジャーバージョンは 5.x です。以下のホスティング機能を使用するには、バージョン 5.x に更新する必要があります。

- Amazon GameLift Servers Anywhere
- Amazon GameLift Servers マネージドコンテナ
- Unreal Engine と Unity 用の Amazon GameLift Servers プラグイン

トピック

- [C++ サーバー SDK for Amazon GameLift Servers 4.x -- アクション](#)
- [C# サーバー SDK for Amazon GameLift Servers 4.x -- アクション](#)
- [サーバー SDK \(Unreal\) for Amazon GameLift Servers -- アクション](#)

C++ サーバー SDK for Amazon GameLift Servers 4.x -- アクション

サーバー SDK リファレンスを使用して、Amazon GameLift Servers でホスティングするマルチプレイヤーゲームを統合します。統合プロセスのガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。

Note

このリファレンスは、サーバー SDK for Amazon GameLift Servers の以前のバージョン用です。最新バージョンについては、「[C++ サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)」を参照してください。

C++ サーバー SDK for Amazon GameLift Servers 4.x -- データ型

サーバー SDK リファレンスを使用して、Amazon GameLift Servers でホスティングするマルチプレイヤーゲームを統合します。統合プロセスのガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。

Note

このリファレンスは、サーバー SDK for Amazon GameLift Servers の以前のバージョン用です。最新バージョンについては、「[C++ サーバー SDK 5.x for Amazon GameLift Servers -- データ型](#)」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。

[C++ サーバー SDK for Amazon GameLift Servers 4.x — アクション](#)

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。

- 特定のプレイ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。

内容

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: String

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せん。

型: Integer

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: string

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: 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。

型: 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: string

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 Servers 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 Servers サービスに伝えられます。

内容

logPaths

Amazon GameLift Servers 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。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 Servers サービスに送られたパラメータセットが含まれます。

内容

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: `Integer`

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: `Aws::GameLift::Server::LogParameters`

必須: いいえ

onStartGameSession

Amazon GameLift Servers サービスが新しいゲームセッションを有効化するために呼び出すコールバック関数の名前です。Amazon GameLift Servers は、クライアントリクエスト [CreateGameSession](#) に応答して、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 Servers サービス API リファレンスで定義) を渡します。

型: `const std::function<void(Aws::GameLift::Model::GameSession)>`
`onStartGameSession`

必須: はい

onProcessTerminate

Amazon GameLift Servers 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の名前。この関数を呼び出した後、Amazon GameLift Servers 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。

型: `std::function<void()>` `onProcessTerminate`

必須: いいえ

onHealthCheck

Amazon GameLift Servers 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の名前です。Amazon GameLift Servers 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 Servers 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: `std::function<bool()>` `onHealthCheck`

必須: いいえ

onUpdateGameSession

Amazon GameLift Servers 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サーバープロセスに渡すために呼び出すコールバック関数の名前です。Amazon GameLift Servers

は、[マッチバックフィル](#) リクエストが処理され、更新されたマッチメーカーデータを提供できる状態になると、この関数を呼び出します。[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。

型: `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イ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 Servers サービスに伝えられます。

内容

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: 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 形式の一意的識別子。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: 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。プレイヤーオブジェクトの形式の説明については、Amazon GameLift Servers 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

マッチメイ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 Servers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: `string`

必須: いいえ

StopMatchBackfillRequest

このデータ型はマッチメイ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 Servers サービスに伝えられます。

内容

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: `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。

型: `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。

型: string

必須: はい

C++ サーバー SDK for Amazon GameLift Servers 4.x -- データ型

トピック

- [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 を持つプレイヤーがサーバープロセスに接続し、検証が必要であることを Amazon GameLift Servers サービスに通知します。Amazon GameLift Servers は、プレイヤーセッション ID が有効であること、つまりプレイヤー ID がゲームセッションでプレイヤース

ロットを予約していることを確認します。検証できたら、Amazon GameLift Servers 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 Servers API アクション [CreatePlayerSession](#) の呼び出しに応答して Amazon GameLift Servers 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。

型: 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 Servers 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の構造は AWS SDK Amazon GameLift Servers API [PlayerSession](#) データ型と同じです。

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 Servers 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` として `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。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 AWS Certificate Manager 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 Response Elements](#)」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 Servers サービスから onProcessTerminate() コールバックを受け取った後にこのアクションを実行します。Amazon GameLift Servers は、次の理由により onProcessTerminate() を呼び出す場合があります。(1) サーバープロセスが異常を報告した場合、または Amazon GameLift Servers 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させる場合 (3) [スポット中断](#)によりインスタンスが終了処理中の場合。

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。例えば、日付時刻の値 2020-09-13 12:26:40 -000Z は、6373559680000000000 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 Servers SDK を初期化します。このメソッドは起動時に他の Amazon GameLift Servers 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 Servers 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 Servers 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 Servers 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 Servers 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 Servers 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なら、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 Servers 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 Servers 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 Servers 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 Servers サービスに通知します。それに応じて、Amazon GameLift Servers 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& playerId);
```

パラメータ

playerSessionId

AWS SDK Amazon GameLift Servers API アクション [CreatePlayerSession](#) の呼び出しに回答して Amazon GameLift Servers 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。

型: 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\(\)](#) も参照してください。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 のバックフィル機能](#)について。

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 Servers サービスはコールバック関数 `OnUpdateGameSession()` を呼び出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 Servers 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イキングデータ。

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 のバックフィル機能](#) について。

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イ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 Servers 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 Servers 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);  
}
```

C# サーバー SDK for Amazon GameLift Servers 4.x -- アクション

サーバー SDK リファレンスを使用して、Amazon GameLift Servers でホスティングするマルチプレイヤーゲームを統合します。統合プロセスのガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。

Note

このリファレンスは、サーバー SDK for Amazon GameLift Servers の以前のバージョン用です。最新バージョンについては、「[C# サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)」を参照してください。

C# サーバー SDK for Amazon GameLift Servers 4.x -- データ型

サーバー SDK リファレンスを使用して、Amazon GameLift Servers でホスティングするマルチプレイヤーゲームを統合します。統合プロセスのガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。

Note

このリファレンスは、サーバー SDK for Amazon GameLift Servers の以前のバージョン用です。最新バージョンについては、「[C# サーバー SDK 5.x for Amazon GameLift Servers -- データ型](#)」を参照してください。

[C# サーバー SDK for Amazon GameLift Servers 4.x -- アクション](#)

トピック

- [LogParameters](#)
- [DescribePlayerSessionsRequest](#)
- [ProcessParameters](#)
- [StopMatchBackfillRequest](#)

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 Servers 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 Servers サービスに伝えられます。

内容

logPaths

Amazon GameLift Servers 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。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。

内容

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。

タイプ: 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: Integer

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: string

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: 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。

型: 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: string

必須: いいえ

ProcessParameters

このデータ型には、[ProcessReady\(\)](#) 呼び出しで Amazon GameLift Servers サービスに送られたパラメータセットが含まれます。

内容

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: Integer

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: `Aws::GameLift::Server::LogParameters`

必須: はい

onStartGameSession

Amazon GameLift Servers サービスが新しいゲームセッションを有効化するために呼び出すコールバック関数の名前です。Amazon GameLift Servers は、クライアントリクエスト [CreateGameSession](#) に応答して、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 Servers サービス API リファレンスで定義) を取得します。

型: void OnStartGameSessionDelegate(GameSession gameSession)

必須: はい

onProcessTerminate

Amazon GameLift Servers 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の名前。この関数を呼び出すと、Amazon GameLift Servers 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: void OnProcessTerminateDelegate()

必須: はい

onHealthCheck

Amazon GameLift Servers 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の名前です。Amazon GameLift Servers 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 Servers 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: bool OnHealthCheckDelegate()

必須: はい

onUpdateGameSession

Amazon GameLift Servers 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サーバープロセスに渡すために呼び出すコールバック関数の名前です。Amazon GameLift Servers は、[マッチバックフィル](#) リクエストが処理され、更新されたマッチメーカーデータを提供できる状態になると、この関数を呼び出します。[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。

型: void OnUpdateGameSessionDelegate (UpdateGameSession
updateGameSession)

必須: いいえ

StartMatchBackfillRequest

このデータ型はマッチメイ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 Servers サービスに伝えられます。

内容

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。

タイプ: 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。

タイプ: 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 Servers 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

マッチメイ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 Servers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: string

必須: いいえ

StopMatchBackfillRequest

このデータ型はマッチメイ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 Servers サービスに伝えられます。

内容

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: 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的識別子。

型: 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的識別子。

型: string

必須: はい

[C# サーバー SDK for Amazon GameLift Servers 4.x -- データ型](#)

トピック

- [AcceptPlayerSession\(\)](#)
- [ActivateGameSession\(\)](#)
- [DescribePlayerSessions\(\)](#)

- [GetGameSessionId\(\)](#)
- [GetInstanceCertificate\(\)](#)
- [GetSdkVersion\(\)](#)
- [GetTerminationTime\(\)](#)
- [InitSDK\(\)](#)
- [ProcessEnding\(\)](#)
- [ProcessReady\(\)](#)
- [RemovePlayerSession\(\)](#)
- [StartMatchBackfill\(\)](#)
- [StopMatchBackfill\(\)](#)
- [TerminateGameSession\(\)](#)
- [UpdatePlayerSessionCreationPolicy\(\)](#)

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、検証が必要であることを Amazon GameLift Servers サービスに通知します。Amazon GameLift Servers は、プレイヤーセッション ID が有効であること、つまりプレイヤー ID がゲームセッションでプレイヤーズロットを予約していることを確認します。検証できたら、Amazon GameLift Servers はプレイヤーズロットの状態を 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 Servers 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: 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, String playerId){
    var acceptPlayerSessionOutcome =
        GameLiftServerAPI.AcceptPlayerSession(playerId);
    if(acceptPlayerSessionOutcome.Success)
    {
        connectionToSessionMap.emplace(connection, playerId);
        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 Servers 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の構造は AWS SDK Amazon GameLift Servers API [PlayerSession](#) データ型と同じです。

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 Servers 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 gameIdOutcome = 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。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 AWS Certificate Manager 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 Response Elements](#)」を参照してください。

例

```
var certOutcome = 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 Servers 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 Servers が onProcessTerminate() を呼び出す理由は次のとおりです。(1) サーバープロセスがポートヘルスを報告した、または Amazon GameLift Servers に応答しなかった場合 (2) スケールダウンイベント中に、インスタンスが終了処理中である場合 (3) [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されつつある場合。

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。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

```
var getTerminationTimeOutcome = GameLiftServerAPI.GetTerminationTime();
```

InitSDK()

Amazon GameLift Servers SDK を初期化します。このメソッドは起動時に他の Amazon GameLift Servers 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 Servers 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 Servers 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 Servers 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 Servers 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 Servers サービスに通知します。それに応じて、Amazon GameLift Servers 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 playerId)
```

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 Servers 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 응답して返されます。

型: 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\(\)](#) も参照してください。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 のバックフィル機能](#)について。

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 Servers 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 Servers 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イキングデータ。

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 のバックフィル機能](#) について。

構文

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

パラメータ

StopMatchBackfillRequest

キャンセルするマッチメイ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 Servers 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 Servers 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 Servers 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](#) 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。

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 updatePlayerSessionCreationPolicyOutcome =  
  
    GameLiftServerAPI.UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy.ACCEPT_ALL);
```

サーバー SDK (Unreal) for Amazon GameLift Servers -- アクション

サーバー SDK for Unreal リファレンスを使用して、Amazon GameLift Servers でホスティングするマルチプレイヤーゲームを統合します。統合プロセスのガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。

Note

このリファレンスは、サーバー SDK for Amazon GameLift Servers の以前のバージョン用です。最新バージョンについては、「[C++ \(Unreal\) サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)」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 Servers を Unreal Engine プロジェクトに統合する](#)」を参照してください。

Amazon GameLift Servers データ型用のサーバー SDK (Unreal)

マルチプレイヤーゲームを Amazon GameLift Servers でホスティングできるように統合する際は、Amazon GameLift Servers サーバー SDK for Unreal リファレンスを使用します。統合プロセス

のガイダンスについては、「[サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)」を参照してください。

 Note

このリファレンスは、サーバー SDK for Amazon GameLift Servers の以前のバージョン用です。最新バージョンについては、「[C++ \(Unreal\)サーバー SDK 5.x for Amazon GameLift Servers -- データ型](#)」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 Servers を Unreal Engine プロジェクトに統合する](#)」を参照してください。

[サーバー SDK \(Unreal\) for Amazon GameLift Servers -- アクション](#)

トピック

- [FDescribePlayerSessionsRequest](#)
- [FProcessParameters](#)
- [FStartMatchBackfillRequest](#)
- [FStopMatchBackfillRequest](#)

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。

- 特定のプレイ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。

内容

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: String

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: Integer

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: string

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: 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。

型: 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: string

必須: いいえ

FProcessParameters

このデータ型には、[ProcessReady\(\)](#) 呼び出しで Amazon GameLift Servers サービスに送られたパラメータセットが含まれます。

内容

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: Integer

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 Servers サービスが新しいゲームセッションを有効化するために呼び出すコールバック関数の名前です。Amazon GameLift Servers は、クライアントリクエスト [CreateGameSession](#) に応答して、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 Servers 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 Servers 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の名前。この関数を呼び出すと、Amazon GameLift Servers 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 Servers 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の名前です。Amazon GameLift Servers 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 Servers 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 Servers 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サーバープロセスに渡すために呼び出すコールバック関数の名前です。Amazon GameLift Servers は、[マッチバックフィル](#) リクエストが処理され、更新されたマッチメーカーデータを提供できる状態になると、この関数を呼び出します。[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。

型: FOnUpdateGameSession

必須: いいえ

FStartMatchBackfillRequest

このデータ型はマッチメイ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 Servers サービスに伝えられます。

内容

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: 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。

型: 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 Servers 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

マッチメイ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 Servers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: FString

必須: いいえ

FStopMatchBackfillRequest

このデータ型はマッチメイ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 Servers サービスに伝えられます。

内容

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: 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。

型: 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。

型: FString

必須: はい

[Amazon GameLift Servers データ型用のサーバー SDK \(Unreal\)](#)

トピック

- [AcceptPlayerSession\(\)](#)
- [ActivateGameSession\(\)](#)
- [DescribePlayerSessions\(\)](#)
- [GetGameSessionId\(\)](#)
- [GetInstanceCertificate\(\)](#)
- [GetSdkVersion\(\)](#)
- [InitSDK\(\)](#)
- [ProcessEnding\(\)](#)
- [ProcessReady\(\)](#)
- [RemovePlayerSession\(\)](#)
- [StartMatchBackfill\(\)](#)
- [StopMatchBackfill\(\)](#)
- [TerminateGameSession\(\)](#)
- [UpdatePlayerSessionCreationPolicy\(\)](#)

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、検証が必要であることを Amazon GameLift Servers サービスに通知します。Amazon GameLift Servers は、プレイヤーセッション ID が有効であること、つまりプレイヤー ID がゲームセッションでプレイヤースロットを予約していることを確認します。検証できたら、Amazon GameLift Servers はプレイ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& playerId)
```

パラメータ

playerSessionId

AWS SDK Amazon GameLift Servers API アクション [CreatePlayerSession](#) の呼び出しに応答して Amazon GameLift Servers 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。

型: 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 Servers 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の構造は AWS SDK Amazon GameLift Servers API [PlayerSession](#) データ型と同じです。

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。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 AWS Certificate Manager 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 Response Elements](#)」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 Servers SDK を初期化します。このメソッドは起動時に他の Amazon GameLift Servers 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 Servers 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 Servers 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。

- サーバードキュメントと通信するために Amazon GameLift Servers サービスが呼び出す、ゲームサーバードキュメントで実装されたコールバックメソッドの名前。
- サーバードキュメントがリッスンするポートの番号。
- Amazon GameLift Servers 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ドキュメントから切断されたことを Amazon GameLift Servers サービスに通知します。それに応じて、Amazon GameLift Servers はプレイヤーセッション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 Servers API アクション [CreatePlayerSession](#) の呼び出しに回答して Amazon GameLift Servers 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、サーバードキュメントに接続するときにこの ID をリファレンスします。

型: 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\(\)](#) も参照してください。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 のバックフィル機能](#)について。

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 Servers 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 Servers 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イキングデータ。

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 のバックフィル機能](#)について。

構文

```
FGameLiftGenericOutcome StopMatchBackfill (FStopMatchBackfillRequest  
&stopBackfillRequest);
```

パラメータ

StopMatchBackfillRequest

キャンセルするマッチメイ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 Servers 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 Servers 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 Servers 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

```
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。

型: 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。

ゲームセッション配置イベント

Amazon GameLift Servers は、各ゲームセッション配置リクエストの処理時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ト通知を設定](#)」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 にも発行されます。

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については、Amazon GameLift Servers 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 CloudWatch Events パターンに準拠しており、同様のトップレベルフィールドとサービス固有の詳細を備えています。

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「[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 GameLift Servers の管理に使用されている AWS アカウント。

時間

イベントのタイムスタンプ。

リージョン

プレイメントリクエストが処理されている 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。

リソース

プレイ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。プレイヤーの接続情報が表示されます。

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

ゲームセッションをホスティングしているフリートの AWS リージョン。これは、GameSessionArn のリージョントークンに対応しています。

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-1232131232324124123123"
  }
]
}
}
```

PlacementCancelled

GameLift サービスへの呼び出し [StopGameSessionPlacement](#) でプレイメントリクエストがキャンセルされました。

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。プレイメントリクエストは、必要に応じて再送信できます。

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 Servers 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。プレイメントリクエストは、必要に応じて再送信できます。

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 Servers AMI バージョン

次の表は、Amazon GameLift Servers がマネージド EC2 ホスティングに使用する最新の Amazon マシンイメージ (AMI) を示しています。[Amazon GameLift Servers での構成と脆弱性の分析](#) で説明されているように、新しい Amazon GameLift Servers マネージド EC2 フリートを定期的に作成し最新の AMI バージョン更新をデプロイします。

サーバー SDK for Amazon GameLift Servers 5+ で使用する AMI

Amazon GameLift Servers は、以下の AMI を使用して、バージョン 5 用のサーバー SDK for Amazon GameLift Servers と統合されたゲームサーバーをホストします。

Amazon GameLift Servers イメージ	アーキテクチャ	最新のパッチ	ベース AWS イメージ
Amazon Linux 2023 BASE_AMI_LINUX_2023	x86	2025-12-16	Amazon Linux 2023 AMI 2023.9.20 251208.0 x86_64 HVM kernel-6.1 (リリースノート)
Amazon Linux 2 BASE_AMI_LINUX_2_SDK_5	x86	2025-12-16	Amazon Linux 2 カーネル 5.10 AMI 2.0.20251208.0 x86_64 HVM gp2 (リリースノート)
Windows 2016 BASE_AMI_WINDOWS_2016_SDK_5	x86	2025-12-16	Windows_Server-2016-English-Full-Base-2025.12.10 (リリースノート)
Amazon Linux 2023 BASE_AMI_LINUX_2023_ARM	ARM64	2025-12-16	Amazon Linux 2023 AMI 2023.9.20 251208.0 arm64 HVM kernel-6.1 (リリースノート)
Amazon Linux 2	ARM64	2025-12-16	Amazon Linux 2 LTS Arm64 カーネル 5.10 AMI 2.0.20251208.0 arm64 HVM gp2 (リリースノート)

Amazon GameLift Servers イメージ	アーキテクチャ	最新のパッチ	ベース AWS イメージ
BASE_AMI_LINUX_2_ARM			
Windows 2022 BASE_AMI_WINDOWS_2022	x86	2025-12-16	Windows_Server-2022-English-Full-Base-2025.12.10 (リリースノート)

サーバー SDK for Amazon GameLift Servers 4 で使用する AMI

Amazon GameLift Servers は、以下の AMI を使用して、バージョン 4 以前のサーバー SDK for Amazon GameLift Servers と統合されたゲームサーバーをホストします。

Amazon GameLift Servers イメージ	アーキテクチャ	最新のパッチ	ベース AWS イメージ
Amazon Linux 2 BASE_AMI_LINUX_2_SDK_4	x86	2025-12-16	Amazon Linux 2 AMI 2.0.20251208.0 x86_64 HVM gp2 (リリースノート)
Amazon Linux 2023 BASE_AMI_LINUX_2023_SDK_4 中国 (北京) および中国 (寧夏) リージョンでのみ使用できます。	x86	2025-12-16	Amazon Linux 2023 AMI 2023.9.20251208.0 x86_64 HVM kernel-6.1 (リリースノート)
Windows 2016 BASE_AMI_WINDOWS_2016_SDK_4	x86	2025-12-16	Windows_Server-2016-English-Full-Base-2025.12.10 (リリースノート)

詳細については、以下のリソースを参照してください。

- [Amazon Linux 2023 リリースノート](#)
- [Amazon Linux 2 リリースノート](#)
- [AWS Windows AMI のバージョン履歴](#)
- [2024 年のコンテンツにおけるソフトウェア更新サービスと Windows Server 更新サービスの変更の説明](#)

Amazon GameLift Servers サービスロケーション

Amazon GameLift Servers 機能は、複数の AWS リージョン および Local Zones で使用できます。プレイヤーのいる場所にゲームサーバーを配置するホスティングソリューションを設計できます。

サポートされている AWS ロケーション

次の表は、Amazon GameLift Serversリソースをサポートする AWS リージョン および Local Zones のリストです。各場所で作成できるリソースのタイプを示します。

地理的場所	ロケー ション コード	マネー ジドフ リート のホー ムリー ジョン (単一 ロケー ション)	マネー ジドフ リート のホー ムリー ジョン (マルチ ロケー ション)	マネー ジドフ リート のリ モート ロケー ション (マルチ ロケー ション)	Anywhere フリー ト	ゲーム セッ ション キュー	FlexMatch マッ チメイ カーと ルール セット	DDoS 保護 (プレ イヤー ゲート ウェイ)
米国東部 (バージニア 北部)	us- east-1	はい	はい	はい	はい	はい	はい	はい [†]
米国東部 (オ ハイオ)	us- east-2	あり		はい	はい	あり		
米国西部 (北 カリフォル ニア)	us- west-1	あり		はい	はい	あり		

地理的場所	ロケーションコード	マネージドフリートのホームリージョン (単一ロケーション)	マネージドフリートのホームリージョン (マルチロケーション)	マネージドフリートのリモートロケーション (マルチロケーション)	Anywhere フリー	ゲームセッションキュー	FlexMatch マッチメーカーとルールセット	DDoS 保護 (プレイヤーゲートウェイ)
米国西部 (オレゴン)	us-west-2	あり	はい	はい	はい	はい	はい	はい [†]
アフリカ (ケープタウン)	af-south-1			あり				
アジアパシフィック (タイ)	ap-southeast-7			あり				
アジアパシフィック (香港)	ap-east-1			あり				
アジアパシフィック (マレーシア)	ap-southeast-5			あり				
アジアパシフィック (ムンバイ)	ap-south-1	あり		はい	はい	あり		
アジアパシフィック (大阪)	ap-northeast-3			はい				

地理的場所	ロケーションコード	マネージドフリートのホームリジョン (単一ロケーション)	マネージドフリートのホームリジョン (マルチロケーション)	マネージドフリートのリモートロケーション (マルチロケーション)	Anywhere フリート	ゲームセッションキュー	FlexMatch マッチメーカーとルールセット	DDoS 保護 (プレイヤーゲートウェイ)
アジアパシフィック (ソウル)	ap-northeast-2	あり	はい	はい	はい	はい	はい	はい [†]
アジアパシフィック (シンガポール)	ap-southeast-1	はい		はい	はい	はい		
アジアパシフィック (シドニー)	ap-southeast-2	はい	はい	はい	はい	はい	はい	はい [†]
アジアパシフィック (東京)	ap-northeast-1	あり	はい	はい	はい	はい	はい	はい [†]
カナダ (中部)	ca-central-1	あり		はい	はい	あり		
欧州 (フランクフルト)	eu-central-1	あり	はい	はい	はい	はい	はい	はい [†]
欧州 (アイルランド)	eu-west-1	あり	はい	はい	はい	はい	はい	はい [†]

地理的場所	ロケーションコード	マネージドフリートのホームリジョン (単一ロケーション)	マネージドフリートのホームリジョン (マルチロケーション)	マネージドフリートのリモートロケーション (マルチロケーション)	Anywhere フリート	ゲームセッションキュー	FlexMatch マッチメーカーとルールセット	DDoS 保護 (プレイヤーゲートウェイ)
欧州 (ロンドン)	eu-west-2	はい		はい	はい	はい		
欧州 (ミラノ)	eu-south-1			はい				
欧州 (パリ)	eu-west-3			はい				
欧州 (ストックホルム)	eu-north-1			はい				
中東 (バーレーン)	me-south-1			はい				
南米 (サンパウロ)	sa-east-1	はい		はい	はい	あり		
アトランタローカルゾーン	us-east-1-atl-1			あり				
シカゴのローカルゾーン	us-east-1-chi-1			あり				

地理的場所	ロケーションコード	マネージドフリートのホームリジョン (単一ロケーション)	マネージドフリートのホームリジョン (マルチロケーション)	マネージドフリートのリモートロケーション (マルチロケーション)	Anywhere フリート	ゲームセッションキュー	FlexMatch マッチメーカーとルールセット	DDoS 保護 (プレイヤーゲートウェイ)
ダラスローカルゾーン*	us-east-1-dfw-1			あり				
ダラスローカルゾーン 2	us-east-1-dfw-2			あり				
デンバーローカルゾーン	us-west-2-den-1			あり				
ヒューストンローカルゾーン	us-east-1-iah-1			あり				
カンザスシティのローカルゾーン	us-east-1-mci-1			あり				
ロサンゼルス (ローカルゾーン)	us-west-2-lax-1			あり				
Phoenix ローカルゾーン	us-west-2-phx-1			あり				

地理的場所	ロケーションコード	マネージドフリースピーチのホームリージョン (単一ロケーション)	マネージドフリースピーチのホームリージョン (マルチロケーション)	マネージドフリースピーチのホームリージョン (マルチロケーション)	Anywhere フリート	ゲームセッションキュー	FlexMatch マッチメーカーとルールセット	DDoS 保護 (プレイヤーゲートウェイ)
ラゴス、ナイジェリ アローカルゾーン	af-south-1			はい				

* すでにオプトイン AWS アカウント している で使用できます。

† DDoS 保護 (プレイヤーゲートウェイ) に有効なフリートホームリージョン。

Note

の一部 AWS リージョン およびすべてのローカルゾーンは、に対してデフォルトで有効になっていません AWS アカウント。ゲームサーバーをそれらの場所にデプロイする前に、まずこれらのリージョンまたはローカルゾーンにオプトイン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。詳細については、「AWS Local Zones ユーザーガイド」の「[Local Zones の開始方法](#)」を参照してください。(2022 年 2 月 28 日より前に作成されたフリートは、この要件の影響を受けません)。
また、Amazon GameLift Servers 管理者ポリシーで `ec2:DescribeRegions` アクションを許可するように更新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のポリシーの例については、「[管理アクセス許可の例](#)」を参照してください。

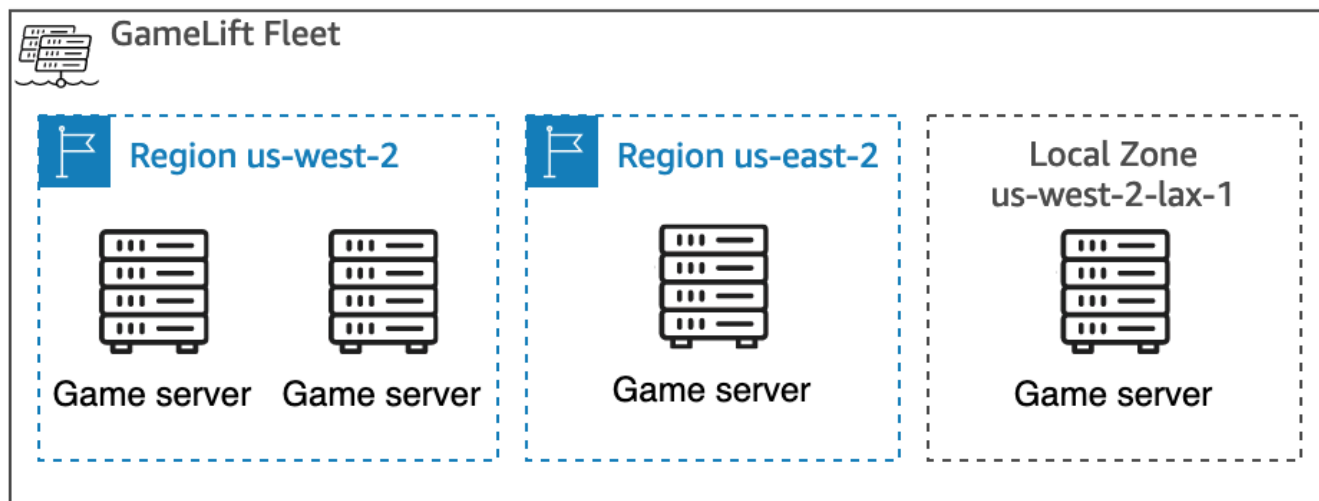
マネージドホスティングのロケーション

Amazon GameLift Servers マネージドホスティングでは、ゲームサーバーリソースのフリートをデプロイします。各フリートは AWS リージョン、フリートのホームリージョンである に作成されます。フリートのホームリージョンは、フリートの Amazon リソース番号 (ARN) で参照されます。

単一リージョンフリートでホスティングリソースをデプロイし、ホームリージョンのみにホスティングリソースを配置できます。または、複数の地理的場所にリソースをホスティングするマルチロケーションフリートをデプロイすることもできます。マルチロケーションフリートには、ホームリージョンに加え、1 つまたは複数のリモートロケーションがあります。フリート内でロケーションをグループ化する方法に制限はありません。マルチロケーションフリートのホスティング容量を管理する場合、ロケーションごとに容量を個別に設定できます。

マルチロケーションフリートのリモートロケーションは、他の AWS リージョン またはローカルゾーンにすることができます。ローカルゾーンは AWS リージョンの拡張です。これにより、コンピューティングリソースをユーザーの近くに配置して、低レイテンシーのゲームプレイを提供できます。詳細については、「[AWS ローカルゾーン](#)」を参照してください。Local Zone のロケーションコードは、親リージョンのコードの後に物理ロケーション識別子が続く形式になっています。例えば、ロサンゼルススのローカルゾーンのコードは us-west-2-lax-1 です。

次の図は、2 つの AWS リージョン と 1 つのローカルゾー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。フリートのホームリージョンは us-west-2、us-east-2 リージョンと us-west-2-lax-1 ローカルゾーンの 2 つのリモートロケーションがあります。



フリートリソースに加えて、Amazon GameLift Servers によるマネージドホスティングでは、次のタイプのリソースを使用することもできます。これらのリソースは、リソースタイプ AWS リージョンをサポートする特定の で作成します。

- **ビルド** - これは、マネージド EC2 フリートでホストされるゲームサーバービルドです。デプロイ先のフリートと同じリージョンにビルドリソースを作成します。
- **スクリプト** - これは、Amazon GameLift Servers Realtime でゲームをホストするための設定スクリプトです。デプロイ先のフリートと同じリージョンにスクリプトリソースを作成します。
- **コンテナグループ定義** - これは、マネージドコンテナフリートでコンテナを実行するための設定です。ゲームサーバーソフトウェアを使用して 1 つ以上のコンテナイメージを識別し、コンテナフリートにデプロイします。デプロイ先のフリートと同じリージョンにコンテナグループ定義を作成します。
- **ゲームセッションキュー** - このリソースは、ゲームセッションのリクエストを処理し、新しいゲームセッションを開始します。処理は、キュー AWS リージョン があるで行われます。ゲームセッション配置プロセスのレイテンシーを減らすには、それを使用するプレイヤーの近くに地理的にキューを作成します。

Amazon GameLift Servers Anywhere の場所

Amazon GameLift Servers Anywhere フリートは、提供するホスティングハードウェアのコレクションです。ゲームサーバーソフトウェアのデプロイ、更新の維持、サーバープロセスの起動など、ホスティングリソースに対するすべてのアクティビティを管理します。Anywhere フリートを作成して、Amazon GameLift Servers サービスをセルフマネージドホスティングリソースに接続します。Amazon GameLift Servers は、ゲームセッションのプレイメント処理プレイヤー参加リクエストを管理し、利用可能なホスティングリソースを見つけ、新しいゲームセッションを開始し、ゲームクライアントに接続情報を提供します。Anywhere フリートは、それらをサポートする任意の AWS リージョン で作成できます。

Anywhere フリートに登録することで、ホスティングハードウェアのインスタンスを追加します。登録された各インスタンスには、カスタムロケーションが関連付けられている必要があります。カスタムロケーションは、AWS リージョン または Local Zones とは関係ありません。これらは、ハードウェアの物理的な場所を表すために使用されます。

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 Servers Anywhere フリートを作成する](#)」と「[Amazon GameLift Servers Anywhereでローカルテストを設定する](#)」を参照してください。

Amazon GameLift Servers FlexMatch の場所

FlexMatch リソースは、マッチメイキングのプレイヤーリクエストを処理するために使用されます。これには、マッチメイキング設定リソースとルールセットリソースが含まれます。処理は、FlexMatch リソース AWS リージョン が配置されているで行われます。マッチメイキングプロセスのレイテンシーを減らすには、それを使用するプレイヤーの近くに地理的にリソースを作成します。マッチメイキング設定とそれを使用するルールセットは、同じ AWS リージョンに配置する必要があります。FlexMatch リソース AWS リージョン は、それらをサポートする任意の で作成できます。

ホスティングソリューション用の FlexMatch の設定の詳細については、「[Amazon GameLift Servers FlexMatch デベロッパーガイド](#)」を参照してください。

Amazon GameLift Servers (中国)

Sinnet が中国 (北京) リージョンで運用しているリソース、または NWCD が中国 (寧夏) リージョンで運営しているリソースで Amazon GameLift Servers を使用する場合は、AWS (中国) アカウントが別途必要になります。中国リージョンでは、一部の機能が使用できません。これらのリージョンで Amazon GameLift Servers を使用方法については、次のリソースを参照してください。

- [中国の Amazon Web Services](#)
- [Amazon GameLift Servers](#) (中国での Amazon Web Services の開始)

Amazon GameLift Servers エンドポイントとクォータ

- サービスとプログラムで接続するために使用できるサービスエンドポイントとビーコンエンドポイントについては、[Amazon GameLift Servers 「サービスエンドポイント」](#)を参照してください。
- AWS アカウントごとのサービスリソースまたはオペレーションの使用に関する Amazon GameLift Servers クォータについては、「[Amazon GameLift Servers サービスクォータ](#)」を参照してください。クォータおよびクォータの引き上げをリクエストする方法については、「[AWS](#)」を参照してください。Amazon GameLift Servers コンソールを使用して、クォータの引き上げをリクエストできます。

Amazon GameLift Servers API の制限

このセクションの表は、Amazon GameLift Servers API オペレーションのデフォルトのレート制限を示しています。これらの制限は、サービスの安定性を維持し、公平な使用を確保するのに役立ちま

す。各 API オペレーションには、リソースへの影響と使用パターンに基づいて特定の制限があります。

デフォルトでは、各カスタマーアカウントのスロットル制限は、バーストリクエストの場合は 500、通常のトラフィックの場合は 500 に設定されます。さらに、以下に説明するように、各 API には独自のデフォルトの制限ペアがあります。

API 制限リファレンス

次の表に、Amazon GameLift Servers API オペレーションのデフォルトのレート制限を示します。リソースレベルのスロットリングでマークされた APIs は、データベースホットキーの問題を防ぐために追加の制限の対象となる場合があります。

API 制限リファレンス

API アクション	バースト制限	[Rate limit] (レート制限)	リソースレベルのスロットリング	注意事項
AcceptMatch	100	100	いいえ	制限はアカウントごとです。
ClaimGameServer	40	20	いいえ	制限はアカウントごとです。
CreateAlias	4	1	いいえ	制限はアカウントごとです。
CreateBuild	10	1	いいえ	制限はアカウントごとです。
CreateContainerFleet	4	1	いいえ	制限はアカウントごとです。
CreateContainerGroupDefinition	10	1	いいえ	制限はアカウントごとです。
CreateFleet	4	1	いいえ	制限はアカウントごとです。
CreateFleetLocations	4	1	いいえ	制限はアカウントごとです。
CreateGameServerGroup	5	0.1	いいえ	制限はアカウントごとです。
CreateGameSession	50	50	いいえ	制限はアカウントごとです。

API アクション	バースト制限	[Rate limit] (レート制限)	リソースレベルのスロットリング	注意事項
CreateGameSessionQueue	4	1	いいえ	制限はアカウントごとです。
CreateLocation	4	1	いいえ	制限はアカウントごとです。
CreateMatchmakingConfiguration	4	1	いいえ	制限はアカウントごとです。
CreateMatchmakingRuleSet	4	1	いいえ	制限はアカウントごとです。
CreatePlayerSession	200	200	いいえ	制限はアカウントごとです。
CreatePlayerSessions	200	200	いいえ	制限はアカウントごとです。
CreateScript	10	1	いいえ	制限はアカウントごとです。
CreateVpcPeeringAuthorization	1	0.1	いいえ	制限はアカウントごとです。
CreateVpcPeeringConnection	1	0.1	いいえ	制限はアカウントごとです。
DeleteAlias	4	1	いいえ	制限はアカウントごとです。
DeleteBuild	10	1	いいえ	制限はアカウントごとです。
DeleteContainerFleet	4	1	いいえ	制限はアカウントごとです。
DeleteContainerGroupDefinition	10	1	いいえ	制限はアカウントごとです。
DeleteFleet	4	1	いいえ	制限はアカウントごとです。
DeleteFleetLocations	4	1	いいえ	制限はアカウントごとです。

API アクション	バースト制限	[Rate limit] (レート制限)	リソースレベルのスロットリング	注意事項
DeleteGameServerGroup	5	0.1	いいえ	制限はアカウントごとです。
DeleteGameSessionQueue	4	1	いいえ	制限はアカウントごとです。
DeleteLocation	4	1	いいえ	制限はアカウントごとです。
DeleteMatchmakingConfiguration	4	1	いいえ	制限はアカウントごとです。
DeleteMatchmakingRuleSet	4	1	いいえ	制限はアカウントごとです。
DeleteScalingPolicy	4	1	いいえ	制限はアカウントごとです。
DeleteScript	10	1	いいえ	制限はアカウントごとです。
DeleteVpcPeeringAuthorization	1	0.1	いいえ	制限はアカウントごとです。
DeleteVpcPeeringConnection	1	0.1	いいえ	制限はアカウントごとです。
DeregisterGameServer	40	20	いいえ	制限はアカウントごとです。
DeregisterCompute	10	2	いいえ	制限はアカウントごとです。
DescribeAlias	10	2	いいえ	制限はアカウントごとです。
DescribeBuild	20	5	いいえ	制限はアカウントごとです。
DescribeCompute	10	2	いいえ	制限はアカウントごとです。

API アクション	バースト制限	[Rate limit] (レート制限)	リソースレベルのスロットリング	注意事項
DescribeContainerFleet	20	5	いいえ	制限はアカウントごとです。
DescribeContainerGroupDefinition	20	5	いいえ	制限はアカウントごとです。
DescribeEC2InstanceLimits	40	15	いいえ	制限はアカウントごとです。
DescribeFleetAttributes	10	2	いいえ	制限はアカウントごとです。
DescribeFleetCapacity	4	1	いいえ	制限はアカウントごとです。
DescribeFleetDeployment	20	5	いいえ	制限はアカウントごとです。
DescribeFleetEvents	10	5	いいえ	制限はアカウントごとです。
DescribeFleetLocationAttributes	10	2	いいえ	制限はアカウントごとです。
DescribeFleetLocationCapacity	10	5	いいえ	制限はアカウントごとです。
DescribeFleetLocationUtilization	10	2	いいえ	制限はアカウントごとです。
DescribeFleetPortSettings	10	2	いいえ	制限はアカウントごとです。
DescribeFleetUtilization	10	2	いいえ	制限はアカウントごとです。

API アクション	バースト制限	[Rate limit] (レート制限)	リソースレベルのスロットリング	注意事項
DescribeGameServer	40	20	いいえ	制限はアカウントごとです。
DescribeGameServer Group	10	2	いいえ	制限はアカウントごとです。
DescribeGameServer Instances	10	2	いいえ	制限はアカウントごとです。
DescribeGameSessionDetails	10	5	いいえ	制限はアカウントごとです。デフォルトの制限値が高く、リソースレベルの制限にも対応しているため、DescribeGameSessions の使用を検討してください。
DescribeGameSessionPlacement	75	75	はい	制限はアカウントごとです。リソースレベルのスロットリングの対象となります。
DescribeGameSessionQueues	10	2	いいえ	制限はアカウントごとです。
DescribeGameSessions	20	20	はい	制限はアカウントごとです。リソースレベルのスロットリングの対象となります。
DescribeInstances	10	2	いいえ	制限はアカウントごとです。
DescribeMatchmaking	200	200	はい	制限はアカウントごとです。リソースレベルのスロットリングの対象となります。
DescribeMatchmakingConfigurations	15	7	いいえ	制限はアカウントごとです。

API アクション	バースト制限	[Rate limit] (レート制限)	リソースレベルのスロットリング	注意事項
DescribeMatchmakingRuleSets	20	10	いいえ	制限はアカウントごとです。
DescribePlayerSessions	200	200	はい	制限はアカウントごとです。リソースレベルのスロットリングの対象となります。
DescribeRuntimeConfiguration	4	1	いいえ	制限はアカウントごとです。
DescribeScalingPolicies	20	20	いいえ	制限はアカウントごとです。
DescribeScript	20	5	いいえ	制限はアカウントごとです。
DescribeVpcPeeringAuthorizations	1	0.1	いいえ	制限はアカウントごとです。
「DescribeVpcPeeringConnections」	1	0.1	いいえ	制限はアカウントごとです。
GetComputeAccess	4	1	いいえ	制限はアカウントごとです。
GetComputeAuthToken	10	5	いいえ	制限はアカウントごとです。
GetGameSessionLogUrl	4	1	いいえ	制限はアカウントごとです。
GetInstanceAccess	4	1	いいえ	制限はアカウントごとです。
ListAliases	10	2	いいえ	制限はアカウントごとです。
ListBuilds	20	5	いいえ	制限はアカウントごとです。

API アクション	バースト制限	[Rate limit] (レート制限)	リソースレベルのスロットリング	注意事項
ListContainerFleets	20	5	いいえ	制限はアカウントごとです。
ListCompute	10	2	いいえ	制限はアカウントごとです。
ListContainerGroup Definitions	20	5	いいえ	制限はアカウントごとです。
ListContainerGroup DefinitionVersions	20	5	いいえ	制限はアカウントごとです。
ListFleets	10	2	いいえ	制限はアカウントごとです。
ListFleetDeployments	20	5	いいえ	制限はアカウントごとです。
ListGameServerGroups	10	2	いいえ	制限はアカウントごとです。
ListGameServers	10	2	いいえ	制限はアカウントごとです。
ListLocations	10	2	いいえ	制限はアカウントごとです。
ListScripts	20	5	いいえ	制限はアカウントごとです。
ListTagsForResource	20	5	いいえ	制限はアカウントごとです。
PutScalingPolicy	4	1	いいえ	制限はアカウントごとです。
RegisterCompute	10	5	いいえ	制限はアカウントごとです。
RegisterGameServer	40	20	いいえ	制限はアカウントごとです。
RequestUploadCredentials	4	1	いいえ	制限はアカウントごとです。
ResolveAlias	4	1	いいえ	制限はアカウントごとです。

API アクション	バースト制限	[Rate limit] (レート制限)	リソースレベルのスロットリング	注意事項
ResumeGameServerGroup	1	0.1	いいえ	制限はアカウントごとです。
SearchGameSessions	10	5	いいえ	制限はアカウントごとです。
StartFleetActions	1	0.1	いいえ	制限はアカウントごとです。
StartGameSessionPlacement	75	75	いいえ	制限はアカウントごとです。
StartMatchBackfill	70	70	いいえ	制限はアカウントごとです。
StartMatchmaking	100	100	いいえ	制限はアカウントごとです。
StopFleetActions	1	0.1	いいえ	制限はアカウントごとです。
StopGameSessionPlacement	20	20	いいえ	制限はアカウントごとです。
StopMatchmaking	100	100	いいえ	制限はアカウントごとです。
SuspendGameServerGroup	1	0.1	いいえ	制限はアカウントごとです。
TagResource	20	5	いいえ	制限はアカウントごとです。
TerminateGameSession	10	2	いいえ	制限はアカウントごとです。
UntagResource	20	5	いいえ	制限はアカウントごとです。
UpdateAlias	4	1	いいえ	制限はアカウントごとです。
UpdateBuild	10	1	いいえ	制限はアカウントごとです。

API アクション	バースト制限	[Rate limit] (レート制限)	リソースレベルのスロットリング	注意事項
UpdateContainerGroupDefinition	10	1	いいえ	制限はアカウントごとです。
UpdateContainerFleet	4	1	いいえ	制限はアカウントごとです。
UpdateFleetAttributes	4	1	いいえ	制限はアカウントごとです。
UpdateFleetCapacity	4	1	いいえ	制限はアカウントごとです。
UpdateFleetPortSettings	4	1	いいえ	制限はアカウントごとです。
UpdateGameServer	40	20	いいえ	制限はアカウントごとです。
UpdateGameServerGroup	5	0.1	いいえ	制限はアカウントごとです。
UpdateGameSession	10	2	いいえ	制限はアカウントごとです。
UpdateGameSessionQueue	4	1	いいえ	制限はアカウントごとです。
UpdateMatchmakingConfiguration	4	1	いいえ	制限はアカウントごとです。
UpdateRuntimeConfiguration	4	1	いいえ	制限はアカウントごとです。
UpdateScript	10	1	いいえ	制限はアカウントごとです。
ValidateMatchmakingRuleSet	4	1	いいえ	制限はアカウントごとです。

リソースレベルのロットリング

一部の API オペレーションは、データベースのホットキー問題を防ぐため、リソースレベルのロットリングが適用されます。これらのオペレーションの API レベルの制限を引き上げても、リソースレベルでロットリングされる可能性があります。

次の API オペレーションは、リソースレベルのロットリングの対象となります。

- DescribeGameSessionPlacement
- DescribeGameSessions
- DescribeMatchmaking
- DescribePlayerSessions

リソースレベルのロットリングは、リソース識別子を含む特定のロットリングキーを使用して適用されます。例えば、DescribeGameSessionPlacement はキー「Operation:GameLift/DescribeGameSessionPlacement,aws-account-placement-id:」を使用して、アカウントとプレイセッション ID の組み合わせごとにロットリングを適用します。これにより、顧客が特定のリソースに対するリクエストでシステムに負担をかけるのを防ぐことができます。

Amazon GameLift Servers SDK API の制限

次の表に、Amazon GameLift Servers SDK API オペレーションのデフォルトのレート制限を示します。これらの制限により、パフォーマンスが最適化され、ゲームサーバー環境でのリソースの競合が防止されます。これらの制限を理解することは、Amazon GameLift Servers とのサーバー側の効率的な統合にとって重要です。

サーバー SDK の制限は、アカウント識別子とプロセス識別子を含む特定のロットリングキーを使用して適用されます。個々のプレイヤーによる不正使用を防ぐため、一部の制限はプレイヤーごとです。

Note

この表は変更される可能性があります。現在の制限の信頼できるソースは、内部 SDC 設定です。必要に応じて、AWS サポートに最新情報をお問い合わせください。

Server SDK API 制限リファレンス

API アクション	バースト制限	レート制限	スロットリングキー	メモ
AcceptPlayerSession	3	1	aws-account-and-process-id-and-player-id	制限はプレイヤーごととプロセスごとです。プレイヤーごとに制限を引き上げる必要はありません。
RemovePlayerSession	3	1	aws-account-and-process-id-and-player-id	制限はプレイヤーごととプロセスごとです。プレイヤーごとに制限を引き上げる必要はありません。
ActivateGameSession()	3	1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
ActivateHostProcessV2	3	1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
StartMatchBackfill	3	1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
DescribePlayerSessions	10	5	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
GetComputeCertificate	3	0.1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。

API アクション	バースト制限	レート制限	スロットリングキー	メモ
GetCustomerRoleCredentials	3	0.1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
InitSDK	3	0.1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
RefreshHostProcess	3	0.1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
StopMatchmaking	3	1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
ProcessEnding	3	1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
ProcessReady	3	1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
UpdateGameSession	3	1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。

API アクション	バースト制限	レート制限	スロットリングキー	メモ
UpdateHostProcess	3	1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
GetGameSessionId	10	5	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
GetTerminationTime	3	1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。
UpdatePlayerSessionCreationPolicy	3	1	aws-account-and-process-id	制限はサーバープロセスごとです。プロセスごとに制限を引き上げる必要はありません。

Note

これらはデフォルトの制限です。これらの APIs でスロットリングが発生している場合は、アカウントレベルのスロットリングが原因である可能性があります。制限緩和をリクエストするには、AWS サポートまでお問い合わせください。

プロセスごとおよびプレイヤーごとの制限は、通常の使用パターンに対応するように設計されており、調整はほとんど必要ありません。これらの APIs でスロットリングが発生している場合は、制限の引き上げをリクエストする前に、潜在的な最適化の機会について実装を確認してください。

アカウントレベルのスロットリングは、次の追加ルールを通じても適用される場合があります。

- SdkWebSocket/AccountLevelHighUsage、aws-account

- SdkWebSocket/AccountLevelThrottle、aws-account

Amazon GameLift Servers UDP ping ビーコン

UDP ping ビーコンは、プレイヤーデバイスと Amazon GameLift Servers ホスティングロケーション間のネットワークレイテンシーを測定する方法を提供します。ping ビーコンを使用すると、正確なレイテンシーデータを収集して、ゲームサーバーの配置について情報に基づいた意思決定を行い、レイテンシー要件に基づいてプレイヤーのマッチメイキングを改善できます。

UDP ping ビーコンの仕組み

Amazon GameLift Servers は、ゲームサーバーをデプロイできる各ホスティング場所に固定 UDP エンドポイント (ping ビーコン) を提供します。ほとんどのゲームサーバーは UDP を使用して通信するため、UDP ping ビーコンでレイテンシーを測定すると、ICMP ping を使用するよりも正確な結果が得られます。ネットワークデバイスは、多くの場合、UDP パケットとは異なる方法で ICMP パケットを処理するため、プレイヤーが経験する実際のパフォーマンスを反映しないレイテンシー測定値につながる可能性があります。

UDP ping ビーコンを使用すると、ゲームクライアントは UDP メッセージをこれらのエンドポイントに送信し、非同期レスポンスを受け取ることができます。これにより、プレイヤーのデバイスと潜在的なホスティング場所間の実際のゲームトラフィック条件をより適切に表すレイテンシー測定値が得られます。エンドポイントは永続的であり、Amazon GameLift Servers がその場所でゲームホスティングをサポートしている限り、引き続き使用できます。

UDP ping ビーコンの一般的なユースケース

UDP ping ビーコンをいくつかの方法で使用して、ゲームのネットワークエクスペリエンスを最適化できます。

最適なホスティングロケーションの選択

さまざまな地理的リージョンでレイテンシーデータを収集して、プレイヤーベースのゲームサーバーをホストするための最適なプライマリロケーションとバックアップロケーションを特定します。

プレイヤーのレイテンシーに基づくゲームセッションの配置

新しいゲームセッションをリクエストするときにプレイヤーのレイテンシーデータを含め、レイテンシーが最も低い場所を選択するのに役立ちます。

レイテンシーに基づくマッチメイキングの最適化

同様のレイテンシープロファイルを持つプレイヤーをマッチングし、マッチングされたプレイヤーに最適な場所にゲームセッションを配置できるように、マッチメイキングをリクエストするときにプレイヤーレイテンシーデータを提供します。

Note

マッチメイキングリクエストを作成するときは、フリートがない場所にレイテンシー情報を提供しないでください。その場合、Amazon GameLift Servers はフリート容量がない場所にゲームセッションを配置しようとし、マッチメイキングリクエストが失敗する可能性があります。

ビーコンエンドポイントの取得

Amazon GameLift Servers ロケーションの ping ビーコンドメインとポート情報を取得するには、[ListLocations](#) API オペレーションを使用します。この API によって返される場所のセットは、呼び出し時に AWS リージョン 指定する (指定しない場合はデフォルトのリージョン) によって異なります。から を呼び出す場合:

- マルチロケーションをサポートするフリートのホームリージョン: API はすべてのホスティングロケーションの情報を返します
- 1つのロケーションをサポートするフリートのホームリージョン: API は、そのロケーションの情報を返します。

マルチロケーションフリートのリモートロケーションのみが可能なロケーションを使用してこの API を呼び出すと、そのタイプのロケーションにはサービスエンドポイントがないため、API はエラーを返します。

[サポートされている AWS ロケーション](#) でサポートされているロケーションの表を参照して、単一口ケーションフリートとマルチロケーションフリートをサポートするホームリージョンを特定します。

例

```
aws gamelift list-locations --region ap-northeast-2
```

これにより、マルチロケーションフリート AWS リージョン がサポートされるため、複数のロケーションが返されます。戻り値の 1 つの例を次に示します。

```
[...]
{
  "LocationName": "ap-northeast-1",
  "PingBeacon": {
    "UDPEndpoint": {
      "Domain": "gamelift-ping.ap-northeast-1.api.aws",
      "Port": 7770
    }
  }
}
```

Important

各レイテンシー測定の前に `ListLocations` を呼び出すのではなく、ping ビーコン情報をキャッシュします。ドメインとポートの情報は静的であり、API は大量のリクエスト用に設計されていません。

レイテンシー測定の実装

UDP ping ビーコンを使用してレイテンシー測定を実装する場合は、次のベストプラクティスに従ってください。

1. 次のいずれかのアプローチを使用して、ping ビーコン情報を保存します。
 - ゲームクライアントのエンドポイントをハードコード化します。
 - ゲームバックエンドの情報をキャッシュします。
 - 定期的な更新メカニズム (毎日/毎週) を実装して、情報を更新します。
2. UDP ping メッセージの送信:
 - メッセージ本文に必要なものを空でなく、メッセージの最大サイズを 300 バイトに保ちます。
 - ロケーションごとに次のレート制限を確認します。
 - 一意の送信者 IP アドレスとポートの組み合わせごとに 3 トランザクション/秒 (TPS)
 - 一意の送信者 IP アドレスあたり 1000 TPS
3. レイテンシーを計算します。
 - 各ロケーションに複数の ping を送信して、平均レイテンシーを計算します。

- より迅速な結果を得るには、複数の場所に同時 ping を送信することを検討してください。
- UDP には 100% の保証された配信がないため、必要に応じて再試行ロジックを使用して、短期間 (通常は 1~3 秒) に返されなかったパケットの新しいパケットを送信します。
- メッセージの送信とレスポンスの受信の時間差を計算します。
- ロケーションへの UDP ping の大部分が一貫してレスポンスを返さない場合は、標準[Amazon GameLift Servers サービスエンドポイント](#)への ICMP ping をフォールバックとして使用してレイテンシーを計算します。

Tip

プレイヤーがローカルネットワークで開いている必要があるポートのリストを文書化する場合は、ポート 7770 を含めることをお勧めします。これは、このポートがブロックされた場合に (ICMP などを使用して) レイテンシーを測定するフォールバックが必要なもう 1 つの理由です。

コードの例

UDP ping を送信してレイテンシーを計算する方法を示す簡単な例をいくつか示します。

C++

```
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <chrono>

int main() {
    // Replace with Amazon GameLift Servers UDP ping beacon domain for your desired
    location
    const char* domain = "gamelift-ping.ap-south-1.api.aws";
    const int port = 7770;
    const char* message = "Ping";           // Your message
    const int num_pings = 3;                // Number of pings to send
```

```
// Create socket
int sock = socket(AF_INET, SOCK_DGRAM, 0);
if (sock < 0) {
    std::cerr << "Error creating socket" << std::endl;
    return 1;
}

// Resolve domain name to IP address
struct hostent* host = gethostbyname(domain);
if (host == nullptr) {
    std::cerr << "Error resolving hostname" << std::endl;
    close(sock);
    return 1;
}

// Set up the server address structure
struct sockaddr_in server_addr;
std::memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(port);
std::memcpy(&server_addr.sin_addr, host->h_addr, host->h_length);

// Set socket timeout
struct timeval tv;
tv.tv_sec = 1; // 1 second timeout
tv.tv_usec = 0;
if (setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv)) < 0) {
    std::cerr << "Error setting socket timeout" << std::endl;
    close(sock);
    return 1;
}

double total_latency = 0;
int successful_pings = 0;

for (int i = 0; i < num_pings; ++i) {
    auto start = std::chrono::high_resolution_clock::now();

    // Send the message
    ssize_t bytes_sent = sendto(sock, message, std::strlen(message), 0,
                                (struct sockaddr*)&server_addr,
                                sizeof(server_addr));

    if (bytes_sent < 0) {
```

```
        std::cerr << "Error sending message" << std::endl;
        continue;
    }

    // Receive response
    char buffer[1024];
    socklen_t server_addr_len = sizeof(server_addr);
    ssize_t bytes_received = recvfrom(sock, buffer, sizeof(buffer), 0,
                                      (struct sockaddr*)&server_addr,
    &server_addr_len);

    auto end = std::chrono::high_resolution_clock::now();

    if (bytes_received < 0) {
        std::cerr << "Error receiving response or timeout" << std::endl;
    } else {
        std::chrono::duration<double, std::milli> latency = end - start;
        total_latency += latency.count();
        successful_pings++;

        std::cout << "Received response, latency: " << latency.count() << " ms"
    << std::endl;
    }

    // Wait a bit before next ping
    usleep(1000000); // 1s
}

// Close the socket
close(sock);

if (successful_pings > 0) {
    double avg_latency = total_latency / successful_pings;
    std::cout << "Average latency: " << avg_latency << " ms" << std::endl;
} else {
    std::cout << "No successful pings" << std::endl;
}

return 0;
}
```

C#

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Diagnostics;
using System.Threading.Tasks;

class UdpLatencyTest
{
    static async Task Main()
    {
        // Replace with Amazon GameLift Servers UDP ping beacon domain for your
        // desired location
        string domain = "gamelift-ping.ap-south-1.api.aws";
        int port = 7770;
        string message = "Ping";           // Your message
        int numPings = 3;                  // Number of pings to send
        int timeoutMs = 1000;              // Timeout in milliseconds

        await MeasureLatency(domain, port, message, numPings, timeoutMs);
    }

    static async Task MeasureLatency(string domain, int port, string message, int
numPings, int timeoutMs)
    {
        using (var udpClient = new UdpClient())
        {
            try
            {
                // Resolve domain name to IP address
                IPAddress[] addresses = await Dns.GetHostAddressesAsync(domain);
                if (addresses.Length == 0)
                {
                    Console.WriteLine("Could not resolve domain name.");
                    return;
                }

                IPEndPoint endPoint = new IPEndPoint(addresses[0], port);
                byte[] messageBytes = Encoding.UTF8.GetBytes(message);

                // Set receive timeout
                udpClient.Client.ReceiveTimeout = timeoutMs;
```

```
double totalLatency = 0;
int successfulPings = 0;
var stopwatch = new Stopwatch();

for (int i = 0; i < numPings; i++)
{
    try
    {
        stopwatch.Restart();

        // Send message
        await udpClient.SendAsync(messageBytes, messageBytes.Length,
endPoint);

        // Wait for response
        UdpReceiveResult result = await
ReceiveWithTimeoutAsync(udpClient, timeoutMs);

        stopwatch.Stop();
        double latency = stopwatch.Elapsed.TotalMilliseconds;

        totalLatency += latency;
        successfulPings++;

        string response = Encoding.UTF8.GetString(result.Buffer);
        Console.WriteLine($"Ping {i + 1}: {latency:F2}ms - Response:
{response}");
    }
    catch (SocketException ex)
    {
        Console.WriteLine($"Ping {i + 1}: Failed - {ex.Message}");
    }
    catch (TimeoutException)
    {
        Console.WriteLine($"Ping {i + 1}: Timeout");
    }

    // Wait before next ping
    await Task.Delay(1000); // 1s between pings
}

if (successfulPings > 0)
{
```

```

        double averageLatency = totalLatency / successfulPings;
        Console.WriteLine($"\\nSummary:");
        Console.WriteLine($"Successful pings: {successfulPings}/
{numPings}");
        Console.WriteLine($"Average latency: {averageLatency:F2}ms");
    }
    else
    {
        Console.WriteLine("\\nNo successful pings");
    }
}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}
}

static async Task<UdpReceiveResult> ReceiveWithTimeoutAsync(UdpClient client,
int timeoutMs)
{
    using var cts = new System.Threading.CancellationTokenSource(timeoutMs);
    try
    {
        return await client.ReceiveAsync().WaitAsync(cts.Token);
    }
    catch (OperationCanceledException)
    {
        throw new TimeoutException("Receive operation timed out");
    }
}
}

```

Python

```

import socket
import time
import statistics
from datetime import datetime

def udp_ping(host, port, timeout=2):
    """
    Send a UDP ping and return the round trip time in milliseconds.
    """

```

```
Returns None if timeout occurs.
"""
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.settimeout(timeout)
message = b'ping'

try:
    # Resolve hostname first
    try:
        socket.gethostbyname(host)
    except socket.gaierror as e:
        print(f"Could not resolve hostname: {e}")
        return None

    start_time = time.time()
    sock.sendto(message, (host, port))

    # Wait for response
    data, server = sock.recvfrom(1024)
    end_time = time.time()

    # Calculate round trip time in milliseconds
    rtt = (end_time - start_time) * 1000
    return rtt

except socket.timeout:
    print(f"Request timed out")
    return None
except Exception as e:
    print(f"Error: {type(e).__name__}: {e}")
    return None
finally:
    sock.close()

def main():
    # Replace with Amazon GameLift Servers UDP ping beacon domain for your desired
    location
    host = "gamelift-ping.ap-south-1.api.aws"
    port = 7770
    num_pings = 3
    latencies = []

    print(f"\nPinging {host}:{port} {num_pings} times...")
    print(f"Start time: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
```

```
for i in range(num_pings):
    print(f"Ping {i+1}:")
    rtt = udp_ping(host, port)

    if rtt is not None:
        print(f"Response from {host}: time={rtt:.2f}ms")
        latencies.append(rtt)

    # Wait 1 second between pings
    if i < num_pings - 1:
        time.sleep(1)
    print()

# Calculate and display statistics
print("-" * 50)
print(f"Ping statistics for {host}:")
print(f"    Packets: Sent = {num_pings}, Received = {len(latencies)}, "
      f"    Lost = {num_pings - len(latencies)} "
      f"    f"({((num_pings - len(latencies)) / num_pings * 100):.1f}% loss)")

if latencies:
    print("\nRound-trip latency statistics:")
    print(f"    Minimum = {min(latencies):.2f}ms")
    print(f"    Maximum = {max(latencies):.2f}ms")
    print(f"    Average = {statistics.mean(latencies):.2f}ms")

print(f"\nEnd time: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")

if __name__ == "__main__":
    main()
```


Amazon GameLift Servers リリースノート

Amazon GameLift Servers リリースノートには、サービスに関する新機能、更新、修正の詳細が記載されています。

SDK のバージョン

次の表は、Amazon GameLift Servers のすべてのリリースを SDK バージョン情報とともに示しています。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があります。

Amazon GameLift Servers SDKsとプラグインはオープンソースです。「[Amazon GameLift Servers 開発用ツールを取得する](#)」を参照してください。最新バージョンを取得するには、[Amazon GameLift Servers GitHub 組織を参照してください](#)。

現在のバージョン

1.0.0	1.0.0
1.0.1	1.0.1
1.0.2	1.0.2
1.0.3	1.0.3
1.0.4	1.0.4
1.0.5	1.0.5
1.0.6	1.0.6
1.0.7	1.0.7
1.0.8	1.0.8
1.0.9	1.0.9
1.0.10	1.0.10
1.0.11	1.0.11
1.0.12	1.0.12
1.0.13	1.0.13
1.0.14	1.0.14
1.0.15	1.0.15
1.0.16	1.0.16
1.0.17	1.0.17
1.0.18	1.0.18
1.0.19	1.0.19
1.0.20	1.0.20
1.0.21	1.0.21
1.0.22	1.0.22
1.0.23	1.0.23
1.0.24	1.0.24
1.0.25	1.0.25
1.0.26	1.0.26
1.0.27	1.0.27
1.0.28	1.0.28
1.0.29	1.0.29
1.0.30	1.0.30
1.0.31	1.0.31
1.0.32	1.0.32
1.0.33	1.0.33
1.0.34	1.0.34
1.0.35	1.0.35
1.0.36	1.0.36
1.0.37	1.0.37
1.0.38	1.0.38
1.0.39	1.0.39
1.0.40	1.0.40
1.0.41	1.0.41
1.0.42	1.0.42
1.0.43	1.0.43
1.0.44	1.0.44
1.0.45	1.0.45
1.0.46	1.0.46
1.0.47	1.0.47
1.0.48	1.0.48
1.0.49	1.0.49
1.0.50	1.0.50
1.0.51	1.0.51
1.0.52	1.0.52
1.0.53	1.0.53
1.0.54	1.0.54
1.0.55	1.0.55
1.0.56	1.0.56
1.0.57	1.0.57
1.0.58	1.0.58
1.0.59	1.0.59
1.0.60	1.0.60
1.0.61	1.0.61
1.0.62	1.0.62
1.0.63	1.0.63
1.0.64	1.0.64
1.0.65	1.0.65
1.0.66	1.0.66
1.0.67	1.0.67
1.0.68	1.0.68
1.0.69	1.0.69
1.0.70	1.0.70
1.0.71	1.0.71
1.0.72	1.0.72
1.0.73	1.0.73
1.0.74	1.0.74
1.0.75	1.0.75
1.0.76	1.0.76
1.0.77	1.0.77
1.0.78	1.0.78
1.0.79	1.0.79
1.0.80	1.0.80
1.0.81	1.0.81
1.0.82	1.0.82
1.0.83	1.0.83
1.0.84	1.0.84
1.0.85	1.0.85
1.0.86	1.0.86
1.0.87	1.0.87
1.0.88	1.0.88
1.0.89	1.0.89
1.0.90	1.0.90
1.0.91	1.0.91
1.0.92	1.0.92
1.0.93	1.0.93
1.0.94	1.0.94
1.0.95	1.0.95
1.0.96	1.0.96
1.0.97	1.0.97
1.0.98	1.0.98
1.0.99	1.0.99
1.0.100	1.0.100

以前のバージョン

Realtime
SDK
ト
SDK

Unreal
City
++
Unreal

1052
人
降

1052
人
降

1050
人
降

1051
人
降

1052
人
降

	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal
	Unreal

	Realtime
	SDK
	SDK
	SDK
	SDK
	SDK
	SDK
	SDK
	Unreal
	Unity
	++
	Unreal
	051
	人
	译
	051
	人
	译
	051
	人
	译
	051
	人
	译
	051
	人
	译
	031
	人
	译

Unreal Engine

SDK

SDK

SDK

SDK

SDK

SDK

SDK

SDK

SDK

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

Unreal

	Realtime
	リアルタイム
	SDK
	ソフトウェア開発キット
	SDK
	Software Development Kit
	ト SDK
	Unreal
	虚幻引擎
	H+
	Jn)real
	5000-2
	人
	译
	5000-2
	人
	译
	5000-1
	人
	译
	5000-0
	人
	译
	5000-3
	人
	译

What's the time

職

STIK

二、

1431

身

✂

ト

SDK

Unreal

●●●●●

11

Local

Mean

3102210

NET

NET

6)

312200-2

以



3142200-2

11

38120-0

22

23

F

39428-2

100

1000

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine SDK

Unreal Engine

Unreal Engine SDK

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine 3-1

Unreal Engine

Unreal Engine

Unreal Engine 3-0

Unreal Engine

Unreal Engine

Unreal Engine 3-2

Unreal Engine

Unreal Engine

Unreal Engine 3-2

Unreal Engine

Unreal Engine

Unreal Engine 3-1

Unreal Engine

Unreal Engine

Unreal Engine 3-1

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine SDK

Unreal Engine

Unreal Engine SDK

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine 4.23.10-2

Unreal Engine

Unreal Engine

Unreal Engine 4.23.10-1

Unreal Engine

Unreal Engine

Unreal Engine 4.23.10-1

Unreal Engine

Unreal Engine

Unreal Engine 4.23.10-1

Unreal Engine

Unreal Engine

Unreal Engine 4.23.10-0

Unreal Engine

Unreal Engine

Unreal Engine 4.23.10-0

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine SDK

Unreal Engine

Unreal Engine SDK

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine

Unreal Engine 4.27.0-1

Unreal Engine

Unreal Engine

Unreal Engine 4.27.0-1

Unreal Engine

Unreal Engine

Unreal Engine 4.27.0-1

Unreal Engine

Unreal Engine

Unreal Engine 4.27.0-2

Unreal Engine

Unreal Engine

Unreal Engine 4.27.0-1

Unreal Engine

Unreal Engine

Unreal Engine 4.27.0-1

Unreal Engine

Unreal Engine

105	Realtime
106	Realtime
107	Realtime
108	Realtime
109	Realtime
110	Realtime
111	Realtime
112	Realtime
113	Realtime
114	Realtime
115	Realtime
116	Realtime
117	Realtime
118	Realtime
119	Realtime
120	Realtime
121	Realtime
122	Realtime
123	Realtime
124	Realtime
125	Realtime
126	Realtime
127	Realtime
128	Realtime
129	Realtime
130	Realtime
131	Realtime
132	Realtime
133	Realtime
134	Realtime
135	Realtime
136	Realtime
137	Realtime
138	Realtime
139	Realtime
140	Realtime
141	Realtime
142	Realtime
143	Realtime
144	Realtime
145	Realtime
146	Realtime
147	Realtime
148	Realtime
149	Realtime
150	Realtime
151	Realtime
152	Realtime
153	Realtime
154	Realtime
155	Realtime
156	Realtime
157	Realtime
158	Realtime
159	Realtime
160	Realtime
161	Realtime
162	Realtime
163	Realtime
164	Realtime
165	Realtime
166	Realtime
167	Realtime
168	Realtime
169	Realtime
170	Realtime
171	Realtime
172	Realtime
173	Realtime
174	Realtime
175	Realtime
176	Realtime
177	Realtime
178	Realtime
179	Realtime
180	Realtime
181	Realtime
182	Realtime
183	Realtime
184	Realtime
185	Realtime
186	Realtime
187	Realtime
188	Realtime
189	Realtime
190	Realtime
191	Realtime
192	Realtime
193	Realtime
194	Realtime
195	Realtime
196	Realtime
197	Realtime
198	Realtime
199	Realtime
200	Realtime
201	Realtime
202	Realtime
203	Realtime
204	Realtime
205	Realtime
206	Realtime
207	Realtime
208	Realtime
209	Realtime
210	Realtime
211	Realtime
212	Realtime
213	Realtime
214	Realtime
215	Realtime
216	Realtime
217	Realtime
218	Realtime
219	Realtime
220	Realtime
221	Realtime
222	Realtime
223	Realtime
224	Realtime
225	Realtime
226	Realtime
227	Realtime
228	Realtime
229	Realtime
230	Realtime
231	Realtime
232	Realtime
233	Realtime
234	Realtime
235	Realtime
236	Realtime
237	Realtime
238	Realtime
239	Realtime
240	Realtime
241	Realtime
242	Realtime
243	Realtime
244	Realtime
245	Realtime
246	Realtime
247	Realtime
248	Realtime
249	Realtime
250	Realtime
251	Realtime
252	Realtime
253	Realtime
254	Realtime
255	Realtime
256	Realtime
257	Realtime
258	Realtime
259	Realtime
260	Realtime
261	Realtime
262	Realtime
263	Realtime
264	Realtime
265	Realtime
266	Realtime
267	Realtime
268	Realtime
269	Realtime
270	Realtime
271	Realtime
272	Realtime
273	Realtime
274	Realtime
275	Realtime
276	Realtime
277	Realtime
278	Realtime
279	Realtime
280	Realtime
281	Realtime
282	Realtime
283	Realtime
284	Realtime
285	Realtime
286	Realtime
287	Realtime
288	Realtime
289	Realtime
290	Realtime
291	Realtime
292	Realtime
293	Realtime
294	Realtime
295	Realtime
296	Realtime
297	Realtime
298	

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 Servers 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](#)」でリリース目のリンクを参照してください。

2026 年 3 月 4 日: SDKv5 での Linux ベースの EC2 およびコンテナフリートの DDoS 保護の提供Amazon GameLift Serversを開始

Amazon GameLift Servers は、SDKv5 の Linux ベースの EC2 およびコンテナフリートに対して distributed-denial-of-service 分散 (DDoS) 保護を提供するようになりました。プレイヤーゲートウェイプロキシリレーネットワークはゲームサーバーインスタンスと同じ場所にあり、トラフィック検証、プレイヤーごとのレート制限、ゲームサーバーの IP アドレス難読化をすべて、わずかな追加レイテンシーで追加コストなしで提供します。この機能は当初、米国東部 (バージニア北部)、米

国西部 (オレゴン)、欧州 (フランクフルト)、欧州 (アイルランド)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、アジアパシフィック (ソウル) の各リージョンで利用できます。サービスの場所の詳細については、「」を参照してください[Amazon GameLift Servers サービスロケーション](#)。

詳細はこちら:

- [ブログ記事: DDoS Amazon GameLift Servers 保護の紹介](#)
- [による DDoS 保護 Amazon GameLift Servers](#) Amazon GameLift Servers デベロッパーガイド
- [プレイヤーゲートウェイの仕組み](#) Amazon GameLift Servers デベロッパーガイド
- [プレイヤーゲートウェイをゲームに統合する](#) Amazon GameLift Servers デベロッパーガイド
- [C++ 用クライアント SDK](#)
- [クライアント SDK for Unreal](#)。アクセスには、GitHub の Epic Games 組織にメンバーシップが必要です。詳細については、[GitHub の「Unreal Engine」](#)を参照してください。

2026 年 2 月 24 日: Amazon GameLift Servers プラグインが Unity 6.3 と Unreal Engine 5.7 をサポートするようになりました

更新されたプラグインバージョン:

- [Unity 用プラグイン](#)、バージョン 3.2.1 (サーバー SDK for Unity、バージョン 5.4.0 を含む)
- [Unreal 用プラグイン](#)、バージョン 3.1.1 (サーバー SDK for Unreal、バージョン 5.4.0 を含む)

Amazon GameLift Servers は、Unreal Engine と Unity 用の新しいバージョンのプラグインとサーバー SDK をリリースします。これらの最新バージョンには、次の更新が含まれています。

- Unreal Engine 5.7 および Unity 6.3 での使用をサポートするようになりました。

詳細はこちら:

- [Unreal Engine 用 Amazon GameLift Servers プラグインでのゲームの統合](#)、Amazon GameLift Servers デベロッパーガイド
- [Unreal Engine プロジェクトを Amazon GameLift Servers に統合する](#)、Amazon GameLift Servers デベロッパーガイド
- [ゲームと Unity 用 Amazon GameLift Servers プラグインの統合](#)、Amazon GameLift Servers デベロッパーガイド

- [Unity Engine プロジェクトAmazon GameLift Serversに統合する](#)、Amazon GameLift Servers デベロッパーガイド

2026 年 1 月 29 日: ゼロインスタンスとの間でフリートのスケーリングをサポートするAmazon GameLift Serversようになりました

Amazon GameLift Servers は、ゲームセッションアクティビティに基づいて、ゼロインスタンスとの間の自動スケーリングをサポートするようになりました。フリートは、ゲームセッションアクティビティがない定義された期間後にゼロにスケールダウンし、ゲームセッションがリクエストされたときにゼロからスケールアップして、コスト最適化のオプションを提供します。

Amazon GameLift Servers API リファレンスリンク:

- [AWS SDK アクション UpdateFleetCapacity](#)

詳細はこちら:

- [Amazon GameLift サーバーフリートをゼロとの間でスケーリングする](#)

2026 年 1 月 22 日: Amazon GameLift Servers Realtime が Amazon Linux 2023 で Node.js 24.x をサポートするようになりました

Amazon GameLift Servers Realtime Amazon Linux 2023 で Node.js 24.x をサポートするようになりました。この更新により、最新の Amazon Linux 2023 オペレーティングシステムでの実行中に、最新の Node.js 機能とセキュリティの改善にアクセスできます。

Realtime フリートの作成時にオプションのインストールスクリプトを含めることができ、特定の要件に応じてインスタンスの起動時にソフトウェアをインストールできるようになりました。新しいランタイムバージョンを使用するには、Realtimeスクリプトの作成時に Node.js 24.x を指定します。

Amazon GameLift Servers API リファレンスリンク:

- [AWS SDK アクション CreateScript](#)

詳細はこちら:

- [Node.js ランタイムバージョン](#)
- [インストールスクリプトを追加する](#)

2025 年 12 月 9 日: ウェブコンソールで AI を活用したサポートの提供Amazon GameLift Serversを開始

Amazon GameLift Servers は、Amazon Q Developer を活用してゲーム開発者向けにカスタマイズされたガイダンスを提供し、ウェブコンソールで AI を活用したサポートを提供するようになりました。この新機能は、お客様が複雑なワークフローをナビゲートし、問題をトラブルシューティングし、ゲームサーバーのデプロイをより効率的に最適化するのに役立つAmazon GameLift Servers専門知識を統合しています。

この新機能は、Amazon GameLift Serversコンソール内で直接、ゲームサーバー統合、フリート設定、パフォーマンス最適化に関する AI 支援のレコメンデーションを提供します。この機能強化は、意思決定プロセスの合理化、トラブルシューティング時間の短縮、全体的なリソース使用率の向上を目的としており、コスト削減とプレイヤーエクスペリエンスの向上につながります。

詳細はこちら:

- [Amazon GameLift Servers コンソール](#)

2025 年 11 月 6 日: Amazon GameLift Servers Windows Server 2022 オペレーティングシステムを使用するゲームビルドへのサポートを開始

Amazon GameLift Servers のお客様は、Windows Server 2022 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。Windows Server 2022 では、セキュリティなど、Windows Server 2016 よりも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、中国リージョンを除くすべての AWS リージョンで利用可能です。

顧客は、より新しい Windows オペレーティングシステムを使用することで、2027 年 1 月に Windows Server 2016 のサポートが終了した後も、重要なセキュリティアップデートを受け続けることができます。Windows Server 2022 のサポートは、2031 年 10 月 13 日まで継続されます。

Amazon GameLift Servers API リファレンスリンク:

- [AWS SDK アクション CreateBuild](#)
- [CLI コマンド upload-build](#)
- [CLI コマンド create-build](#)

2025 年 10 月 28 日: Amazon GameLift Servers が OpenTelemetry (OTel) メトリクスコレクターをすべてのサーバー SDK とプラグインに追加

Amazon GameLift Servers では、すべてのサーバー SDK とプラグインに OpenTelemetry (OTel) メトリクスコレクションが組み込まれ、ゲームサーバーのパフォーマンスモニタリングのための標準化されたオブザーバビリティを提供するようになりました。この統合により、業界標準のテレメトリ機能がゲームサーバーに直接提供され、追加の計測や設定を必要とせずに包括的なモニタリングとトラブルシューティングが可能になります。

OpenTelemetry 統合を使用すると、次のことができるようになりました。

- 標準化されたプロトコルを使用して、重要なゲームサーバーのメトリクス、ログ、トレースを自動的にキャプチャする
- 既存のモニタリングプラットフォームやオブザーバビリティツールとシームレスに統合する
- すべてのフリートインスタンスでゲームサーバーのパフォーマンスと状態をリアルタイムでモニタリングする
- さまざまなモニタリングソリューションで動作するテレメトリデータを使用して問題をより迅速にトラブルシューティングする

OpenTelemetry は、テレメトリデータの収集、処理、エクスポートの標準化された方法を提供する、標準化されたオープンソースのオブザーバビリティフレームワークです。

詳細はこちら:

- [サーバーテレメトリメトリクスによるモニタリング](#) Amazon GameLift Servers デベロッパーガイド

2025 年 10 月 2 日: Amazon GameLift Servers がフリートインスタンスにコンソールベースのリモートアクセスを追加

Amazon GameLift Servers は、Amazon GameLift Servers コンソールを介してマネージド EC2 およびコンテナフリートインスタンスへの直接リモートアクセスを提供するようになり、認証管理が不要になりました。この新機能では、Amazon EC2 Systems Manager (SSM) を使用して、追加のセットアップや認証情報管理を必要とせずに、ブラウザベースの安全なターミナルアクセスを提供します。

コンソールベースのリモートアクセスでは、次のことができるようになりました。

- ゲームサーバーの統合に関する問題をトラブルシューティングする

- アクティブなゲームセッション中にゲームサーバーアクティビティをモニタリングし、ログファイルにアクセスする
- 実際のプレー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する

この機能は、サーバー SDK バージョン 5.x を実行しているフリートで使用できます。以前の SDK バージョンを実行しているフリートの場合、リモートアクセスに既存の CLI AWS メソッドを引き続き使用できます。

フリートインスタンスにリモート接続するには、Amazon GameLift Servers コンソールを開き、アクセスするマネージド EC2 またはコンテナフリートを見つけます。フリートの詳細で、インスタンスタブを開き、リストからインスタンスを選択し、接続を選択します。

詳細はこちら:

- [Amazon GameLift Servers フリートインスタンスにリモート接続する](#) Amazon GameLift Servers デベロッパーガイド

2025 年 9 月 23 日: Amazon GameLift Servers がテキサス州ダラスの新しいローカルゾーンのサポートを追加

Amazon GameLift Servers マネージドホスティングでは、ゲームサーバーリソースを新しいダラス、TX、ローカルゾーン (us-east-1-dfw-2) にデプロイできるようになりました。Local Zones を使用して、ゲームサーバーを地理的にプレイヤーに近い位置に配置し、レイテンシーを軽減することでゲームプレイを大幅に改善します。この新しいローカルゾーンでは、C6gn、C6i、C6in、M6g、M6i、M6in、M8g、R6i の EC2 インスタンスタイプを使用できます。各インスタンスタイプの詳細については、「[AWS EC2 インスタンスタイプ](#)」をご参照ください。

新しいダラスローカルゾーンでゲームセッションのホスティングを開始するには、まず AWS アカウントのローカルゾーンにオプトインします。その後、新規または既存のマルチロケーションフリートにリモートロケーションとして追加できます。ゲームで Amazon GameLift Servers FlexMatch を使用している場合は、マッチメイキングキューのフリートを更新して、新しいローカルゾーンを含めます。マルチロケーションフリートを使用すると、各ロケーションのホスティング容量を直接管理できます。

新しいダラス、TX、ローカルゾーン AWS リージョン の親は us-east-1 (バージニア) です。元のダラス、TX、ローカルゾーン (us-east-1-dfw-1) は、新しいオプトインリクエストを受け入れなくなりました。

詳細はこちら:

- [「ユーザーガイド」の開始 AWS Local Zones](#) AWS Local Zones 方法
- [Amazon GameLift Servers サービスロケーション](#) Amazon GameLift Servers デベロッパーガイド
- [フリートのロケーションを更新する](#) Amazon GameLift Servers デベロッパーガイド

2025 年 8 月 12 日: Amazon GameLift Servers が Unreal Engine 用プラグインが UE バージョン 5.6 をへのサポートを開始

更新されたプラグインバージョン:

- [Unreal 用プラグイン](#)、バージョン 3.0.1 (サーバー SDK for Unreal、バージョン 5.3.1 を含む)

Amazon GameLift Servers は、Unreal Engine 用プラグインとサーバー SDK for Unreal Engine の新しいバージョンをリリースします。これらの最新バージョンには、次の更新が含まれています。

- Unreal Engine 5.6 での使用をサポートするようになりました。
- Android クライアントターゲットをパッケージ化する際のコンパイルの問題が解決されました。

詳細はこちら:

- [Unreal Engine 用 Amazon GameLift Servers プラグインでのゲームの統合](#)、Amazon GameLift Servers デベロッパーガイド
- [Unreal Engine プロジェクトを Amazon GameLift Servers に統合する](#)、Amazon GameLift Servers デベロッパーガイド

2025 年 6 月 24 日: Amazon GameLift Servers がマルチプレイヤーゲームのリアルタイムネットワークレイテンシーの測定に役立つ UDP ping ビーコンをローンチ

更新された SDK バージョン:

- AWS SDK 1.11.595

Amazon GameLift Servers は UDP ping ビーコンと呼ばれる一連の固定エンドポイントを一般提供としてリリースし、プレイヤーデバイスとゲームサーバーの場所間のレイテンシーを正確に測定するのに役立ちます。UDP ping ビーコンエンドポイントは、AWS 中国リージョンを除き、Amazon

GameLift Servers でサポートされているすべての AWS グローバルリージョンとローカルゾーンで使用できます。

ほとんどのマルチプレイヤーゲームでは、UDP (ユーザーデータグラムプロトコル) をプライマリパケット送信プロトコルとして使用します。これは、リアルタイムゲームのパフォーマンス上の利点があるためです。可能な限り最高のプレイヤーエクスペリエンスを提供するには、ネットワークレイテンシーを理解して最適化することが重要です。UDP ping ビーコンは、プレイヤーとゲームサーバー間の実際の UDP パケットレイテンシーを測定するための一貫した信頼性の高い方法を提供し、プレイヤーとサーバーのマッチングとゲームセッションの配置に関するより良い意思決定に役立ちます。

[ListLocations](#) API は、返される場所のリストの一部としてエンドポイントドメインとポート情報を含めるように拡張されているため、エンドポイントにプログラムで簡単にアクセスできます。

ゲームクライアントは、UDP メッセージをこれらのエンドポイントに送信し、同じデータを含む非同期レスポンスを受け取ることができます。これにより、プレイヤーのデバイスと潜在的なホスティングロケーション間の実際のゲームトラフィック条件をより適切に表すレイテンシー測定値が得られます。これらのエンドポイントは永続的であり、Amazon GameLift Servers がその場所でゲームホスティングをサポートしている限り使用できます。

詳細はこちら:

- [UDP ping ビーコン](#) Amazon GameLift Servers デベロッパーガイド
- [ListLocations](#)、Amazon GameLift Servers API リファレンス

2025 年 6 月 24 日: Amazon GameLift Servers がタイとマレーシアのロケーションサポートを追加

Amazon GameLift Servers マネージドホスティングを使用すると、タイのバンコクとマレーシアのクアラルンプールにゲームサーバーリソースをデプロイし、東南アジアのプレイヤーにゲームのリーチを拡張できるようになりました。これらの新しいリージョンは、レイテンシーを短縮し、それらの分野のプレイヤーのゲームプレイエクスペリエンスを向上させるのに役立ちます。

以下は AWS リージョン、マルチロケーションフリートのリモートロケーションとして使用できます。これらのロケーションでゲームセッションのホスティングを開始するには、それらをリモートロケーションとして、新規または既存のマルチロケーションフリートに追加します。マルチロケーションフリートを使用すると、各ロケーションのホスティング容量を直接管理できます。

- アジアパシフィック (タイ) (ap-southeast-7)

- アジアパシフィック (マレーシア) (ap-southeast-5)

これらは、アカウントに対してデフォルトで AWS リージョン は有効になっていません AWS 。 Amazon GameLift Servers リソースをデプロイする前に、各リージョンにオプトインします。

詳細はこちら:

- [Amazon GameLift Servers サービスロケーション](#)、Amazon GameLift Servers ホスティングデベロッパーガイド
- [フリートのロケーションを更新する](#)、Amazon GameLift Servers ホスティングデベロッパーガイド
- [アカウントの AWS リージョンを有効または無効にするには](#)、AWS 「アカウント管理リファレンスガイド」を参照してください。

2025 年 5 月 29 日: Amazon GameLift Servers が Unreal Engine と Unity 用のサーバー SDK とプラグインの新しいバージョンをリリース

更新された SDK バージョン:

- [C++ サーバー SDK](#)、バージョン 5.3.0
- [C# サーバー SDK](#)、バージョン 5.3.0
- [Go サーバー SDK](#)、バージョン 5.3.0
- [Server SDK for Unreal](#)、バージョン 5.3.0
- [Server SDK for Unity](#)、バージョン 5.3.0

更新されたプラグインバージョン:

- [Unreal プラグイン](#)バージョン 3.0.0
- [Unity 用プラグイン](#)バージョン 3.1.0

サーバー SDK for C++、C#、Go、Unreal、Unity の新しいバージョン、および Unreal Engine と Unity の新しいプラグインバージョンがオープンソースになりました。これらはすべて [Amazon GameLift Servers GitHub 組織](#)で利用できます。更新の詳細については、各リポジトリのリリースノートと readmes を参照してください。

キーサーバー SDK の更新:

- すべてのサーバー SDK。
- [OnProcessTerminate](#) コールバック関数に、ゲームサーバープロセスを終了するデフォルトロジックが追加されました。
- [InitSDK\(\)](#) 関数は、べき等性トークンを使用して複数の再試行をサポートするようになりました。
- [OnUpdateGameSession](#) コールバックが `autoBackfillMode` のデータを渡すことができるようになりました。

キープラグインの更新:

- Unreal Engine 用プラグインのインストールとセットアップのプロセスがより合理化され、自動化が増え、前提条件 (CMake、OpenSSL、Unreal クロスコンパイルツールチェーン) が少なくなりました。
- Unreal Engine 用プラグインは、ゲームクライアントとサーバービルドパスのスペースのサポートなど、マネージド EC2 ワークフローの UI エクスペリエンスを改善しました。さらに、エディタからゲームクライアントを起動するときにコマンドライン引数を追加できるようになりました。
- Unreal Engine 用プラグインは、UE5 での ARM サーバービルドをサポートするようになりました。

詳細はこちら:

- [Unreal Engine 用 Amazon GameLift Servers プラグイン](#) Amazon GameLift Servers デベロッパーガイド
- [Unity 用 Amazon GameLift Servers プラグイン \(サーバー SDK 5.x\)](#) Amazon GameLift Servers デベロッパーガイド
- [サーバー SDK 5.x for Amazon GameLift Servers](#) Amazon GameLift Servers デベロッパーガイド
- [Amazon GameLift Servers を Unreal Engine プロジェクトに統合する](#) Amazon GameLift Servers デベロッパーガイド
- [Amazon GameLift Servers を Unity プロジェクトに統合する](#) Amazon GameLift Servers デベロッパーガイド

2025 年 4 月 24 日: C# サーバー SDK 5 for Amazon GameLift Servers が .NET 8 をサポート

更新された SDK バージョン:

- C# Server SDK、バージョン 5.2.1

C# を使用するゲーム開発者向けに、Amazon GameLift Servers プロジェクトのターゲットフレームワークとして .NET 8 を使用できるようになりました。.NET 8 を使用すると、just-in-time (JIT) コンパイルの改善、メモリ使用量の最適化、スタートアップ時間の短縮など、パフォーマンスの向上を活用できます。現在 .NET 6 を使用している場合は、C# サーバー SDK を最新バージョンに更新するなど、.NET 8 への移行を計画することをお勧めします。Microsoft は、2026 年 11 月まで、セキュリティパッチと技術アップデートを含む .NET 8 サポートを発表しました。

[C# サーバー SDK for Amazon GameLift Servers の最新バージョンをダウンロード](#)

詳細はこちら:

- [Amazon GameLift Servers 開発用ツールを取得する](#) Amazon GameLift Servers デベロッパーガイド

2025 年 3 月 27 日: Amazon GameLift Servers が次世代 EC2 インスタンスファミリーのサポートを拡張

更新された SDK バージョン:

- AWS SDK 1.11.535

第 5 世代から第 8 世代のインスタンスファミリーの幅広い Amazon EC2 インスタンスから選択することで、Amazon GameLift Servers でゲームサーバーホスティングをファインチューニングできるようになりました。新しい EC2 世代はそれぞれ EC2 コンピューティング、メモリ、ネットワーキングの進歩を提供し、第 8 世代インスタンスは最先端の Graviton4 AWS および Intel Xeon ベースのインスタンスを提供します。次世代インスタンスは、次のインスタンスファミリーで使用できます。

- [汎用](#) (M シリーズ)
- [コンピューティング最適化](#) (C シリーズ)
- [メモリ最適化](#) (R シリーズ)

ローカルストレージ (d)、拡張ネットワーキング (n)、および特定のプロセッサアーキテクチャ (Intel/AMD/Graviton – i/a/g) を提供するバリエーションを選択することもできます。次世代インスタンスは、AWS 中国リージョンを除き Amazon GameLift Servers、で AWS リージョン サポートされているすべてので使用できます。詳細については、[Amazon GameLift Servers サービスロケーション](#)を参照してください。

これらの新しいインスタンスタイプは、Amazon GameLift Servers マネージド EC2 フリートと マネージドコンテナフリートで使⽤します。既存のゲームホスティングを新しいインスタンスタイプ (同じアーキテクチャ) に切り替える場合は、インスタンスタイプを除くすべての設定が変更されていない新しいフリートをデプロイします。

詳細はこちら:

- [Amazon GameLift Servers インスタンスの料金](#)
- [CreateFleet](#) Amazon GameLift Servers API リファレンス
- [CreateContainerFleet](#)、Amazon GameLift Servers API リファレンス

2025 年 3 月 11 日: Amazon GameLift Servers が迅速なオンボーディングのために新しい Game Server Wrapper を開始

Amazon GameLift Servers 用の新しいゲームサーバーラッパーは、Amazon GameLift Servers でホストされているゲームサーバーを取得するのに必要な時間を大幅に短縮します。コードを変更することなく、ラッパーを使用して基本的なゲームセッション管理機能をゲームに追加し、Amazon GameLift Servers Anywhere フリート、マネージド EC2 フリート、またはマネージドコンテナフリートにデプロイできます。このツールは、独自のゲームサーバーまたはサンプルゲームのサーバーを使用して、Amazon GameLift Servers 機能の実践的な評価を行うのに最適です。また、ラピッドプロトタイプングやテストなど、ゲームサーバーのイテレーションを迅速にデプロイするのにも役立ちます。

基本的なゲームセッション管理機能を使用すると、ゲームサーバーは Amazon GameLift Servers サービスとの接続を初期化し、プロンプトに⽿答してゲームセッションを開始および停止し、ゲームセッションが完了するとシャットダウンできます。

[GitHub からゲームサーバーラッパーをダウンロードします。](#) ゲームプロジェクトでラッパーをインストールして使⽤する方法については、GitHub の readme を参照してください。

2025 年 3 月 6 日: Amazon GameLift が Amazon GameLift Servers および Amazon GameLift Streams になりました

Amazon GameLift は、デベロッパーが世界でも最も要求の厳しいゲームを構築、スケーリング、配信できるように支援するフルマネージドサービスです。Amazon GameLift Streams の一般提供リリースにより、Amazon GameLift は高スケーリングゲームサーバーとスムーズゲームプレイストリーミング機能の両方を提供するようになりました。

[Amazon GameLift Servers](#) を使⽤すると、ゲーム開発者は専用のゲームサーバーをデプロイ、運用、スケーリングできます。高性能ゲームサーバーを数分でクラウドにデプロイし、プレイヤーの需要に

合わせてスケールアップ/ダウンできます。AWS 実証済みのコンピューティング環境上に構築された Amazon GameLift Servers は、1 つのゲームで 1 億人の同時プレイヤーをサポートし、1 秒あたり 100,000 人のプレイヤーを追加し、1 分あたり 9,000 の新しいコンピューティングインスタンスをサポートします。また、エンタープライズグレードのセキュリティ、最大の群集のマッチメイキング、柔軟な従量制料金により、新しいゲームアイデアに取り組んでいる場合でも、数百万のプレイヤーでゲームを実行している場合でも、開始できます。

[Amazon GameLift Streams](#) は、ゲーム開発者が最大 1080p の解像度と 60 frames-per-second (fps) のゲームストリーミングエクスペリエンスを、iOS、Android、FireOS、ゲーマー向け PC などのデバイス間で認識可能な遅延なしで配信するのに役立ちます。単一の AWS サービスを使用すると、パブリッシャーは変更なしでゲームコンテンツをフルマネージド型のクラウドベースの GPU インスタンスに数分でデプロイし、AWS Network Backbone を介してウェブブラウザを使用して任意のエンドユーザーデバイスに直接配信できます。プレイヤーはダウンロードやインストールを待たずにわずか数秒でゲームを開始でき、PC やゲームコンソールでローカルでゲームをプレイするのとはほぼ区別できないゲームプレイエクスペリエンスを提供します。

2025 年 1 月 14 日: Amazon GameLift Servers がゲームセッション配置の優先順位付けのサポートを拡張

更新された SDK バージョン:

- AWS SDK 1.11.485

お客様からのフィードバックに応じて、個々のゲームセッション配置リクエストの場所を優先できる新機能をリリースします。場所ごとに配置を優先するように設定されたキューについて、各配置リクエストで優先度の場所のカスタマイズされたリストを提供できるようになりました。

この新機能により、お客様は必要に応じてプレイメントリクエストごとにロケーションの優先順位を動的に変更できます。柔軟性が高まると、プレイヤーの場所、フリートの負荷、サーバーの状態など、変化する状況により適切に対応できます。また、配置場所の選択方法をさらにカスタマイズしたいお客様をサポートすることもできます。

詳細はこちら:

- [ゲームセッション配置の優先順位付け](#)、Amazon GameLift Servers デベロッパーガイド
- [StartGameSessionPlacement](#)、Amazon GameLift Servers API リファレンス

2025 年 1 月 2 日: Amazon GameLift Servers がゲームセッションの終了に関する新しいサポートをリリース

更新された SDK バージョン:

- AWS SDK 1.11.477

お客様からのフィードバックに応じて、個々のゲームセッションをより簡単に終了できる新機能をリリースします。このリリースでは、ゲームセッションをAmazon GameLift Serversコンソールで直接終了するか、AWS CLI または AWS SDK for を使用して終了できるようになりましたAmazon GameLift Servers。

この新機能は、アクティブのままで状態が悪いゲームセッションを解決する必要性に対処します。これにより、コンピューティングリソースが新しいゲームセッションをホストできなくなります。以前は、お客様はゲームセッションを手動で終了するためにコンピューティングにリモートアクセスする必要がありました。

選択できる終了方法は 2 つあります。最初の方法では、カスタムシャットダウンシーケンスを使用してゲームセッションを適切に終了しようとします。これには、プレイヤーに通知してゲームデータを解決するアクションが含まれる場合があります。2 番目のメソッドはサーバープロセスを強制的に停止し、ゲームセッションをすぐに終了します。この 2 番目の方法では、サーバープロセスが応答していない場合でもゲームセッションが確実に終了します。

詳細はこちら:

- [Amazon GameLift Servers コンソールを使用してゲームセッションをシャットダウンする](#)、Amazon GameLift Servers デベロッパーガイド
- [TerminateGameSession](#)、Amazon GameLift Servers API リファレンス

2024 年 12 月 19 日: Amazon GameLift Servers がマネージドコンテナのゲームエンジンプラグインサポートをリリース

更新されたプラグインバージョン:

Unreal Engine 用 Amazon GameLift Servers プラグインバージョン 2.0.0

- マネージドコンテナをサポートする C++ サーバー SDK 5.2.0 をサポートするようにアップグレードされました。

- Unreal Engine 5.4 および 5.5 のサポートが追加されました。

Unity 用 Amazon GameLift Servers プラグイン、バージョン 3.0.0

- マネージドコンテナをサポートする C++ サーバー SDK 5.2.0 をサポートするようにアップグレードされました。
- Windows および Mac OS 用の Unity 2021.3 LTS および 2022.3 LTS のサポート。

Unreal および Unity ゲームエンジンの Amazon GameLift Servers プラグインは、Amazon GameLift Servers でゲームを起動して実行するステップを合理化するツールとワークフローを提供します。Amazon GameLift Servers は、ゲーム開発者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するために使用できるフルマネージド型のクラウドホスティングサービスです。

最新のプラグインバージョンでは、次の機能が強化されています。

- マネージドコンテナでホスティングするためのガイド付きワークフロー。このワークフローでは、ゲームサーバーソフトウェアでコンテナイメージをセットアップし、ゲームサーバーにクラウドベースのホスティングソリューションをデプロイする手順について説明します。ワークフローには、シンプルなデプロイと、ゲームセッションプレイスメントキューとFlexMatchマッチメーカーを使用したより完全なデプロイの 2 つの異なるデプロイシナリオがあります。各シナリオでは、Amazon GameLift Servers コンテナフリートとサポート AWS リソースが生成されます。
- AWS ユーザープロファイルをセットアップし、プラグイン使用のための AWS アクセス認証情報を管理するプロセスを改善しました。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、リージョンを操作できます。
- 既存のコンテナフリートを更新する追加機能。新しいコンテナイメージ (ゲームサーバーバージョンの更新など) をデプロイし、フリート設定を最初から開始することなく変更できます。
- Amazon GameLift Servers Anywhere フリートと Managed EC2 によるホスティングのワークフローを改善しました。お客様からのフィードバックに基づく改善には、ヒントや役立つリソースへのリンクを含む、より良いガイダンスが含まれます。

マネージドコンテナとマネージド EC2 ソリューションのデプロイシナリオでは、AWS CloudFormation テンプレートを使用して各シナリオの AWS リソースを作成およびデプロイします。これらのテンプレートは Amazon GameLift Servers プラグインのダウンロードに含まれており、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 用プラグイン: ゲームをマネージドコンテナフリートにデプロイする](#)、Amazon GameLift Servers デベロッパーガイド
- [Unity 用プラグイン: ゲームをマネージドコンテナフリートにデプロイする](#)、Amazon GameLift Servers デベロッパーガイド
- [GitHub から Unreal 用プラグインをダウンロード](#)
- [GitHub から Unity 用プラグインをダウンロード](#)
- [Amazon GameLift Servers を使用したゲームのホスティング](#)
- [Amazon GameLift Servers フォーラム](#)

2024 年 11 月 12 日: Amazon GameLift Servers がマネージドコンテナでマルチプレイヤーゲームホスティングを開始

更新された SDK バージョン:

- AWS SDK 1.11.445
- サーバー SDK、バージョン 5.2.0 (すべての言語)

Amazon GameLift Servers は、コンテナ化されたゲームサーバーワークロード用の新しいホスティングソリューションを一般公開用にリリースしました。このリリースでは、ゲーム開発者は、一貫性のある安全な環境、シンプルなデプロイプロセス、最適化されたリソース使用率など、コンテナ化の利点を活用できるようになりました。

マネージドコンテナフリートは、ユーザーに代わって、ユーザーの設定に基づいて Amazon GameLift Servers によって管理される Amazon EC2 インスタンスを使用します。ゲーム用のカスタムコンテナアーキテクチャを構築し、Amazon Elastic Container Registry (Amazon ECR) リポジトリに保存してコンテナイメージを提供します。コンテナフリートは Linux ベースのゲームサーバーでのみ使用できます。ゲームサーバーは Server SDK 5.2.0 以降と統合する必要があります。

マネージドコンテナフリートを使用すると、マネージド EC2 フリートと同じ利点が得られます。これには、オンデマンドインスタンスタイプとスポットインスタンスタイプのサポート、インテリジェントキャパシティスケールリング、キューを使用したゲームセッション配置、マッチメイキングが含まれます。また、他のフリートタイプと同じメトリクスと、コンテナの新しいメトリクスも取得できます。コンテナフリートのその他の機能は次のとおりです。

- コンテナ化されたワークロードのサーバーレスエクスペリエンスとの整合性。コンテナごとに 1 つのゲームサーバープロセスを実行し、最適なリソース使用のために複数のコンテナを各フリートインスタンスにパックします。複数のゲームサーバープロセスを持つコンテナを使用する場合は、Amazon GameLift Servers エージェントを使用してホストの自動管理を行うことができます。
- フリート作成の合理化。コンテナフリートは、デプロイ設定を最小限に抑えるように設計されており、適切な推奨値/デフォルト値があります。作業フリートをすばやくデプロイし、必要に応じて個々の設定をカスタマイズできます。
- コンテナアーキテクチャのバージョンングツール。コンテナグループ定義 (コンテナ「タスク」に似ています) を更新し、複数のバージョンを維持し、フリートにデプロイするバージョンを指定できるようになりました。
- フリート更新ツール。コンテナフリートを使用すると、ゲームサーバーのバージョン更新をリリースするときに新しいフリートを作成する必要がなくなります。代わりに、コンテナイメージを更新し、既存のフリートに更新をデプロイできるようになりました。

がマルチロケーションフリートAmazon GameLift Serversをサポート AWS リージョン している任意の Amazon GameLift Serversコンテナフリートを構築し、サポートされている任意のリモートロケーションにコンテナフリートインスタンスをデプロイできます。詳細については、[Amazon GameLift Servers サービスロケーション](#)を参照してください。マネージドコンテナは現在、AWS 中国リージョンでは利用できません。

詳細はこちら:

- ブログ記事: [フルマネージドコンテナを活用して、Amazon GameLift Servers のグローバル規模でマルチプレイヤーゲームをホストする](#)
- [マネージドコンテナ](#) 概要、Amazon GameLift Serversデベロッパーガイド
- [Amazon GameLift Servers におけるコンテナの仕組み](#) Amazon GameLift Servers デベロッパーガイド
- [Amazon GameLift Servers マネージドコンテナでホスティングするための開発ロードマップ](#) Amazon GameLift Servers デベロッパーガイド
- [CreateContainerFleet](#)、Amazon GameLift Servers API リファレンス

2024 年 9 月 19 日: Amazon GameLift Servers が Unreal Engine 用 C++ サーバー SDK とプラグインの更新をリリース

更新された SDK バージョン:

C++ サーバー SDK、バージョン 5.1.3

- 新しいログ機能。SDK リクエストログにアクセスできるようになりました。
- SDK メッセージ送信の信頼性が向上しました。ネットワークの中断やランダムなメッセージ障害を復旧にするため、SDK はより堅牢な再接続メカニズムを使用するようになりました。

更新されたプラグインバージョン:

Unreal Engine 用 Amazon GameLift Servers プラグインバージョン 1.1.2

- C++ サーバー SDK 5.1.3 の最新バージョンをサポートするようにアップグレードされました。
- Unreal Engine 向け Amazon GameLift Servers プラグインで、フリートのサーバービルド実行可能ファイルを参照するときに、すべてのファイルを参照するオプションが導入されました。

Unreal 用 C++ サーバー SDK プラグイン、バージョン 5.1.2

- C++ サーバー SDK 5.1.3 の最新バージョンをサポートするようにアップグレードされました。

詳細はこちら:

- [Unreal Engine 用 Amazon GameLift Servers プラグインでのゲームの統合](#)、Amazon GameLift Servers デベロッパーガイド
- [Amazon GameLift Servers プラグインと SDK ダウンロード](#)

2024 年 9 月 5 日: Amazon GameLift Servers がフリート作成プロセスのオブザーバビリティを改善

お客様のフィードバックに基づいて、マネージド EC2 フリートを作成し、ゲームセッションをホストする準備を整えるための Amazon GameLift Servers ワークフローを明確にしました。改善には以下のものがあります。

- フリート作成プロセスの各フェーズについて、より具体的で正確な説明を記載しました。この可視性の向上により、問題の特定と解決がより簡単になります。
- 構築フェーズとアクティベーションフェーズでは、インスタンスデプロイタスク (構築) をタスクからより適切に分離して、ゲームサーバープロセスを開始し、Amazon GameLift Servers サービス (アクティベーション) に接続します。この変更により、問題に対して考えられる原因を簡単に認識できます。さらに、アクティベーションフェーズにあるフリートにリモート接続できるようになりました。
- 2 つの新しいフリート作成イベントは、ゲームサーバーのインストールスクリプトの成功または失敗を伝えます。ゲームサーバービルドにインストールスクリプトが含まれている場合、Amazon GameLift Servers はスクリプトの実行を試行し、次の新しいイベントのいずれかを出力します。
 - FLEET_CREATION_COMPLETED_INSTALLER
 - FLEET_CREATION_FAILED_INSTALLER

詳細はこちら:

- [Amazon GameLift Servers フリートの問題をデバッグする](#) Amazon GameLift Servers デベロッパーガイド
- [イベントデータ型](#)、Amazon GameLift Servers API リファレンス

2024 年 8 月 15 日: Amazon GameLift Servers のコンソールエクスペリエンスを改善

お客様のフィードバックに基づいて、[Amazon GameLift Servers コンソール](#)エクスペリエンスに以下の更新を行いました。

- ページの表示設定が AWS アカウントユーザーに自動的に保存され、ページに戻るたびに適用されます。表示優先設定では、フリートリストページなど、テーブル表示に含める情報を選択できます。テーブルの右上隅にある



アイコンを使用すると、表示優先設定をカスタマイズできます。

- マネージド EC2 フリートのフリート作成ワークフローは、フリートの場所とインスタンスタイプの選択を組み合わせるように合理化されました。ロケーションの選択を変更しても、フリートに適したインスタンスタイプを簡単に見つけることができます。

詳細はこちら:

- [Amazon GameLift Servers マネージド EC2 フリートを作成する](#) Amazon GameLift Servers デベロッパーガイド

2024 年 7 月 25 日: AWS ナイジェリアローカルゾーンのサポート Amazon GameLift Servers を追加

Amazon GameLift Servers マネージドホスティングを使用し、ナイジェリア、西アフリカにゲームサーバーリソースをデプロイすることで、アフリカ中のプレイヤーにゲームのリーチを広げることができます。AWS Local Zones を使用して、プレイヤーに地理的に近い場所にゲームサーバーを配置し、レイテンシーを減らし、ゲームプレイを大幅に改善します。

ナイジェリアでゲームセッションのホスティングをすぐに開始するには、新しいナイジェリアローカルゾーンをリモートロケーションとして、新規または既存のマルチロケーションフリートに追加します。ゲームで Amazon GameLift Servers FlexMatch を使用している場合は、マッチメイキングキューのフリートを更新して、新しいローカルゾーンを含めます。マルチロケーションフリートを使用すると、各ロケーションのホスティング容量を直接管理できます。

Lagos AWS リージョン の親であるナイジェリアローカルゾーンは、アフリカ (ケープタウン) リージョン (af-south-1) であり、リモートロケーションとして Amazon GameLift Servers もサポートしています。ナイジェリアローカルゾーン名は、af-south-1-los-1 です。

詳細はこちら:

- [Amazon GameLift Servers サービスロケーション](#) Amazon GameLift Servers デベロッパーガイド
- [フリートのロケーションを更新する](#) Amazon GameLift Servers デベロッパーガイド

2024 年 7 月 2 日: Amazon GameLift Servers がプレイヤーセッションデータを表示するための新しいコンソールツールをリリース

Amazon GameLift Servers コンソールに、ゲームセッション ID、プレイヤーセッション ID、またはプレイヤー ID でプレイヤーセッション情報を取得できるプレイヤーセッション検索ツールが追加されました。FlexMatch マッチメイキングを使用するゲームは、マッチしたプレイヤーごとにプレイヤーセッションを自動的に生成します。他のすべてのゲームでは、プレイヤーセッションはオプション機能です。

プレイヤーセッション検索ツールは、Amazon GameLift Servers コンソールのメインナビゲーションにあります。個々のプレイヤーセッションを表示したり、複数のプレイヤーセッション間でデータを比較したりできます。ゲームセッションの詳細ページを表示する際に、プレイヤーセッションデータを開くこともできます。

詳細はこちら:

- [Amazon GameLift Servers コンソールでのゲームセッションとプレイヤーセッション](#) Amazon GameLift Servers デベロッパーガイド

2024 年 4 月 24 日: コンテナフリートプレビューをAmazon GameLift Serversリリース

Amazon GameLift Servers で、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、ゲームサーバーと、依存関係や設定などゲームサーバーが必要とするすべてのものが含まれています。依存関係の例としては SDK やソフトウェアパッケージなどが挙げられます。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 GameLift Servers はフリートにコンテナを入力します。

コンテナフリートで機能させるには、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。コンテナフリートで、ホスティングリソースをファインチューニングすると、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。また、コンテナ内で複数のゲームサーバーをホストすると、リソースの使用を減らすこともできます。

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、マッチメイキングなど、他のタイプのフリートと同様の利点が得られます。また、他のフリートタイプと同じメトリクスと、コンテナの新しいメトリクスも取得できます。コンテナフリートは、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リーチを提供します。

- 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 Servers コンテナを使用したホスティングの管理](#)、Amazon GameLift Servers デベロッパーガイド
- [CreateContainerGroupDefinition](#)、Amazon GameLift Servers API リファレンス

2024 年 2 月 13 日: Amazon GameLift Servers が SDK の改善を導入し、Unreal Engine 用 Amazon GameLift Servers プラグインのインストールを簡素化

更新された SDK バージョン:

- Go サーバー SDK、バージョン 5.1.0
- C# サーバー SDK、バージョン 5.1.2
- C++ サーバー 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 Servers 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 Servers Anywhere フリートのゲームサーバーは、サーバーパラメータを使用して InitSDK() を呼び出す必要があります。

更新されたプラグインバージョン:

- Unreal Engine 用 Amazon GameLift Servers プラグインバージョン 1.1.0
- Unity 用 Amazon GameLift Servers プラグインバージョン 2.1.0
- Unreal 用 C++ サーバー SDK プラグイン、バージョン 5.1.1
- Unity 用 C# サーバー 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。

- [Unreal Engine 用 Amazon GameLift Servers プラグイン] インストール手順を更新し、パッケージ化を簡素化しました。このプラグインには現在、最新バージョンの C++ サーバー SDK for Unreal が含まれています。
- サーバー SDK for Amazon GameLift Servers 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 Servers プラグインでのゲームの統合](#)、Amazon GameLift Servers デベロッパーガイド
- [Amazon GameLift Servers プラグインと SDK ダウンロード](#)

2023 年 12 月 14 日: Amazon GameLift Servers に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を追加

これまでは、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たり、指定されたプロパティのゲームセッションを検索したりすることはできていましたが、現在は、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できます。ゲームクライアントが `UpdateGameSession` を呼び出して、`GameProperty` 値を `{"Key": "map", "Value": "jungle"}` に変更します。その後、ゲームはゲームセッションのプレイヤーに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、`ACTIVE` の `Status` 値およびゲームプロパティ `{"Key": "map", "Value": "desert"}` を持つゲームセッションを一覧表示できます。

詳細はこちら:

- [the section called “ゲームクライアント機能の統合”](#) Amazon GameLift Servers デベロッパーガイド
- [GameProperty](#)、Amazon GameLift Servers API リファレンス
- [UpdateGameSession](#)、Amazon GameLift Servers API リファレンス
- [SearchGameSessions](#)、Amazon GameLift Servers API リファレンス

2023 年 11 月 21 日: による Terraform や Pulumi などの Infrastructure as Code ツールのサポート Amazon GameLift Serversを開始 AWS クラウドコントロール API

Infrastructure as Code (IaC) ツールを使用して、Amazon GameLift Servers 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 や、Terraform や Pulumi などのサードパーティーツールも含まれます。この追加サポートにより、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 Servers 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 Servers APIs または の AWS CloudFormation テンプレートを使用して、リソースを引き続き使用できます Amazon GameLift Servers。

IaC で利用可能な Amazon GameLift Servers リソースの詳細については、「[Amazon GameLift Servers リソースタイプのリファレンス](#) Amazon GameLift Servers リソースタイプのリファレンス」を参照してください。

さらに、新しい Fleet 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 を使用すると、開発者は数百の 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でリソース (CRUDL) を作成、読み取り、更新、削除、一覧表示するための標準の APIs セットを利用できます。

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](#)
- [Pulumi](#)

2023 年 11 月 16 日: Amazon GameLift Servers が Unity 用スタンドアロンプラグインを更新

更新された SDK バージョン: Unity 用 Amazon GameLift Servers プラグイン、バージョン 2.0.0

Unity 用 Amazon GameLift Servers プラグインは、Amazon GameLift Servers を使用して Unity ゲームをクラウドホスティング用に起動および実行するステップを合理化するツールとワークフローを提供します。Amazon GameLift Servers は、ゲーム開発者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 Servers Anywhere による ローカルテストのサポートなど、最新の Amazon GameLift Servers 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。

主要なプラグイン機能には以下が含まれます。

- 次のシナリオに対する Unity エディタのガイド付き UI ワークフロー:
 - ローカルワークステーションをホストとして使用して、Amazon GameLift Servers とのゲームの統合をテストします。このワークフローは、ローカルマシン用の Amazon GameLift Servers 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 GameLift Servers によるゲームセッションのリクエスト、新しいゲームセッションへの参加に役立ちます。
 - Amazon GameLift Servers 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、リージョンで作業できます。
 - Amazon GameLift Servers の統合およびデプロイプロセスの効率化に役立つ次のような特別に便利な機能:
 - 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。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 Servers サービスの間にあり、Amazon GameLift Servers 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とコードを含む、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 フリートや 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 Servers 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:

- [Unity 用 Amazon GameLift Servers プラグイン \(サーバー SDK 5.x\)](#) Amazon GameLift Servers デベロッパーガイド
- [GitHub からプラグインをダウンロードする](#)
- [Amazon GameLift Servers ホスティングについて](#)
- [Amazon GameLift Servers フォーラム](#)

2023 年 11 月 2 日: Amazon GameLift Servers が共有認証情報のサポートを追加

SDK バージョンの更新: AWS SDK 1.11.193

新しい Amazon GameLift Servers 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一緒にバンドルして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アクセス許可が必要です。これを行うには、まず CloudWatch AWS Identity and Access Management リソースを使用するアクセス許可を持つ IAM) ロールを設定し、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 GameLift Servers が

ゲームサーバービルドを各 EC2 インスタンスにデプロイすると、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。は、インスタンスの存続期間中、一時的な認証情報 Amazon GameLift Servers 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 Servers コンソールのフリート作成ワークフローで。
- 新しいパラメータ `InstanceRoleCredentialsProvider` を使用してサービス API オペレーション `CreateFleet` を呼び出すとき。
- 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 Servers デベロッパーガイド
- [CreateFleet、InstanceRoleCredentialsProvider](#)、Amazon GameLift Servers API リファレンス
- [IAM サービスロールのセットアップ](#)、Amazon GameLift Servers デベロッパーガイド

2023 年 9 月 28 日: Amazon GameLift Servers が Unreal Engine 用の新しいスタンドアロンプラグインをリリース

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 Servers プラグイン

Unreal Engine 用 Amazon GameLift Servers プラグインは、クラウドホスティングに Amazon GameLift Servers を使用してゲームを起動および実行するステップを合理化するツールとワークフローを提供します。Amazon GameLift Servers は、ゲーム開発者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して、Amazon GameLift Servers とのゲームの統合をテストします。このワークフローは、ローカルマシン用の Amazon GameLift Servers 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 GameLift Servers によるゲームセッションのリクエスト、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、ゲームをクラウドホスティング用に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へのデプロイ、または FlexMatch マッチメーカーによるフリートセットへのデプロイ、という 3 つの異なるデプロイシナリオが用意されています。各デプロイシナリオのソリューションには、Amazon GameLift Servers 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ィールを設定する機能。複数のプロフィールを維持して、異なる AWS アカウント、アカウントユーザー、リージョンで作業できます。
- Amazon GameLift Servers の統合およびデプロイプロセスの効率化に役立つ次のような特別に便利な機能:
 - 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 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。
 - 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 Servers 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 Fleet 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 Servers リソースをデプロイする際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 Servers プラグインのダウンロードに含まれており、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 Servers プラグイン](#) Amazon GameLift Servers デベロッパーガイド

- [GitHub からプラグインをダウンロードする](#)
- [Amazon GameLift Servers ホスティングについて](#)
- [Amazon GameLift Servers フォーラム](#)

2023 年 8 月 17 日: Graviton AWS プロセッサを使用したゲームサーバーホスティング Amazon GameLift Serversを提供

SDK バージョンの更新: AWS SDK 1.11.144

Amazon GameLift Servers を使用すると、Graviton プロセッサを搭載した EC2 AWS 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。

ではAmazon GameLift Servers、Graviton AWS 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 Processor](#): 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、ランタイム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ついての分析情報を把握します。

Note

Graviton Arm インスタンスには、Linux AMI 用のサーバービルドが必要です。C++ および 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) および Amazon Linux 2 (AL2)での Mono インストールは標準ではサポートされません。

2023 年 7 月 27 日: Amazon GameLift Servers が Unity 開発のサポートを追加したサーバー SDK 5.1.0 をリリース

更新された SDK バージョン: サーバー SDK for C++、C#/Unity、Unreal 5.1.0

Amazon GameLift Servers 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ための新しいプラグインが提供されています。ゲーム開発者は Amazon GameLift Servers サーバー SDK をゲームサーバーに統合し、Amazon GameLift Servers でホスティング用にデプロイします。

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 Servers 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、UnityNuGet による必要な依存関係が含まれています。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 – サーバー SDK for C# 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 Servers 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 Servers Anywhere フリートのゲームサーバーは、サーバーパラメータを使用して InitSDK() 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# サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)
 - [C++ サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)
 - [C++ \(Unreal\) サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)
- 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
 - [Amazon GameLift Servers を Unity プロジェクトに統合する](#)
 - [Amazon GameLift Servers 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 Servers 更新された API オペレーション [GetComputeAccess](#) と [GetInstanceAccess](#) のドキュメントを改訂し、使用中の Amazon GameLift Servers 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。
 - [GameSessionPlacement](#) の説明を改訂し、プレイメントが「保留中」状態のときにゲームセッション情報がどのように一時的になるかについて文書化しました。

2023 年 7 月 13 日: Amazon GameLift Servers にフリートハードウェアメトリクスが追加されました

Amazon GameLift Servers 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 GameLift Servers の場合、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表します。これらのフリートハードウェアメトリクスは、AWS マネジメントコンソールの Amazon CloudWatch ダッシュボードを使用して表示できます。また、Amazon GameLift Servers コンソールでフリートの詳細を確認することもできます。

詳細はこちら:

- [Amazon CloudWatch で Amazon GameLift Servers を監視する](#) (フリートのメトリクス)、Amazon GameLift Servers デベロッパーガイド

2023 年 6 月 29 日: Amazon GameLift Servers が Amazon Linux 2023 のサポートを開始

SDK バージョンの更新: AWS SDK 1.11.111

Amazon GameLift Servers のお客様は、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出来ます。Amazon Linux 2 のサポートは、2026 年 6 月 30 日まで継続されます。

詳細はこちら:

- [Amazon GameLift Servers Linux サーバーに関するよくある質問](#)
- [Amazon Linux 2 と Amazon Linux 2023 の比較](#)
- Amazon GameLift Servers API リファレンスリンク:
 - [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: 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 Amazon GameLift Servers FleetIQ を追加

SDK バージョンの更新: AWS SDK 1.11.87

Amazon GameLift Servers FleetIQ をゲームホスティングに使用する場合、現在ドレイン状態になっているインスタンスでのゲームセッションプレイス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。これに応じて、Amazon GameLift Servers 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確認します。

詳細はこちら:

- Amazon GameLift Servers API リファレンスの [ClaimGameServer](#)
- 「Amazon GameLift Servers FleetIQ デベロッパーガイド」の「[FleetIQ の仕組み](#)」

2023 年 5 月 16 日: Amazon GameLift Servers はフリートのコスト配分タグ付けをサポートします

Amazon GameLift Servers のお客様は、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。コスト配分タグを個々の Amazon GameLift Servers EC2 フリートリソースに割り当てて、フリートがホスティングコスト全体にどのように貢献しているかを追跡できます。

詳細はこちら:

- [リソースコストと使用率ツール](#)
- 「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 GameLift Servers が Windows Server 2016 のサポートを開始

SDK バージョンの更新: AWS SDK 1.11.63

Amazon GameLift Servers のお客様は、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 AWS リージョンで使用できます。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 Servers 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るようになりました。

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 Servers 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、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 Servers API リファレンスリンク:
 - [CLI コマンド upload-build](#)
 - [CLI コマンド create-build](#)
 - [AWS SDK アクション CreateBuild](#)
- [Amazon GameLift Servers Windows 2012 に関するよくある質問](#)

2023 年 4 月 13 日: Amazon GameLift Servers がサーバー SDK 5.x for Unreal を開始

更新された SDK バージョン: サーバー SDK 5.0.0 for Unreal

Unreal Engine用の Amazon GameLift Servers 軽量プラグインの最新バージョンは、Amazon GameLift Servers サーバー SDK 5.x をベースにしています。Unreal Engine 環境と Amazon GameLift Servers の統合を開始するには、以下のリンクを参照してください。

詳細はこちら:

- [Amazon GameLift Servers を Unreal Engine プロジェクトに統合する](#)
- [サーバー SDK を使用して、Amazon GameLift Servers をゲームサーバーに追加します。](#)
- [C++ サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)

2023 年 3 月 14 日: Amazon GameLift Servers が新しいコンソールエクスペリエンスを開始

新しい Amazon GameLift Servers コンソールには以下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 Servers リソース間のナビゲーションが容易になりました。

- Amazon GameLift Servers 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 GameLift Servers の概要の表示、ドキュメント、よくある質問および AWS re:Post へのリンクによるサポートが提供されています。
- Amazon CloudWatch メトリクスの改善 – Amazon GameLift Servers メトリクスは Amazon GameLift Servers コンソールと CloudWatch ダッシュボードの両方で利用できるようになりました。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:

- [Amazon GameLift Servers を使用した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 GameLift Servers が Amazon SNS トピックのサーバー側の暗号化をサポートするようになりました

SNS トピックのサーバー側の暗号化 (SSE) は、保管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ト通知を設定](#)
- [FlexMatch マッチメイキングイベント](#)
- [保管時の暗号化](#)

2023 年 2 月 9 日: Amazon GameLift Servers 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 用サーバー 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している場合は、引き続き Amazon GameLift Servers サーバー SDK 5.0.0 と .NET 4.6 を使用してください。Unity は .NET 6 をサポートしていません。

詳細はこちら:

- Amazon GameLift Servers Server SDK の最新バージョンを「[Amazon GameLift Servers の使用開始](#)」からダウンロードしてください。
- [C# サーバー SDK 5.x for Amazon GameLift Servers -- アクション](#)

2023 年 1 月 31 日: Amazon GameLift Servers 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 用サーバー SDK 5.0.0

詳細はこちら:

- Amazon GameLift Servers Server SDK の最新バージョンを「[Amazon GameLift Servers の使用開始](#)」からダウンロードしてください。
- [Go サーバー SDK for Amazon GameLift Servers – アクション](#)

2022 年 12 月 1 日: Amazon GameLift Servers が Amazon GameLift Servers Anywhere
および Amazon GameLift Servers Server SDK 5.0 をローンチ

SDK バージョンの更新: SDK AWS 1.10.21、C++ および C# 用の Server SDK 5.0.0

Amazon GameLift Servers Anywhere はゲームサーバーリソースを使用して Amazon GameLift Servers ゲームサーバーをホストします。Amazon GameLift Servers Anywhere を使用すると、独自のコンピューティングリソースを Amazon GameLift Servers マネージド EC2 コンピューティングと統合して、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させることができます。Amazon GameLift Servers Anywhere を使用すると、イテレーションのたびにビルドを Amazon GameLift Servers にアップロードしなくても、ゲームサーバーを繰り返しテストできます。

ハイライト:

- 新しい Amazon GameLift Servers Anywhere フリートとコンピューティングタイプ
- Amazon GameLift Servers Anywhere コンピューティングリソースの登録
- テストのイテレーションサイクルの改善

Amazon GameLift Servers サーバー SDK 5.0.0 では、既存のサーバー SDK が改善され、新しいリソースタイプであるコンピューティングが導入されました。サーバー SDK 5.0.0 は Amazon GameLift Servers Anywhere をサポートし、ゲームサーバーをホスティングするための独自のコンピューティングリソースの使用もサポートしています。

詳細はこちら:

- [サーバー SDK 5.x for Amazon GameLift Servers](#)
- [地理的場所](#)
- [マネージドフリートのコンピューティングリソースを選択する](#)
- [Amazon GameLift Servers Anywhere フリートを作成する](#)

2022 年 8 月 25 日: Amazon GameLift Servers がローカルゾーンのサポートを開始

SDK バージョンの更新: AWS SDK 1.9.333

Amazon GameLift Servers は米国の 8 つのローカルゾーンで利用できるようになったため、フリートをプレイヤーの近くに配置できます。フリートにローカルゾーンを追加することで、マネージド Amazon GameLift Servers のすべての機能をローカルゾーンで使用できます。

Local Zones は、AWS リソースとサービスをクラウドのエッジ、大規模な人口、業界、情報技術 (IT) センターの近くに拡張します。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:

- [Amazon GameLift Servers ローカルゾーン](#)
- [地理的場所](#)
- [Amazon GameLift Servers マネージド EC2 フリートを作成する](#)

2022 年 6 月 28 日: Amazon GameLift Servers が新しいオプトインコンソールエクスペリエンスを開始

新しい Amazon GameLift Servers コンソールには以下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 Servers リソース間のナビゲーションが容易になりました。
- Amazon GameLift Servers 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 GameLift Servers の概要の表示、ドキュメント、よくある質問および AWS re:Post へのリンクによるサポートが提供されています。
- Amazon CloudWatch メトリクスの改善 – Amazon GameLift Servers メトリクスは Amazon GameLift Servers コンソールと CloudWatch ダッシュボードの両方で利用できるようになりました

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:

- [Amazon GameLift Servers を使用した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: FlexMatch が複合ルールを追加し、さらに改善を行う

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 Servers FlexMatch デベロッパーガイド](#):
 - [FlexMatch ルールタイプ](#)
 - [FlexMatch プロパティ式](#)
- [Attribute Value: SL](#)

2021 年 10 月 28 日: アジアパシフィック (大阪) リージョンのマルチリージョンフリートのサポート Amazon GameLift Servers を追加、AWS Graviton2 プロセッサのサポート Amazon GameLift Servers FleetIQ を追加

SDK バージョンの更新: AWS SDK [1.9.133](#)

Amazon GameLift Servers が、アジアパシフィック (大阪) リージョンで利用できるようになりました。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 Servers が、アジアパシフィック (大阪) リージョンで利用できるようになりました。
- Amazon GameLift Servers 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、および r6g を管理できるようになりました。

詳細はこちら:

- [Amazon GameLift Servers 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 GameLift Servers が Unity 用プラグインをリリース

Unity バージョン 1.0.0 の Amazon GameLift Servers プラグインには、Amazon GameLift Servers リソースへのアクセスや Amazon GameLift Servers を Unity ゲームに統合することが容易にな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 Servers プラグインを使用して Amazon GameLift Servers APIs、一般的なゲームシナリオの 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 Servers Local を使用して、ゲームクライアントとゲームサーバー間で渡されたメッセージを確認して、一般的なゲームが Amazon GameLift Servers とどのようにやり取りするかを知ることができます。

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、AWS CloudFormation 一般的なゲームシナリオのサンプルシナリオをデプロイしますFlexMatch。

詳細はこちら:

- [Unity 用 Amazon GameLift Servers プラグインでのゲームの統合](#)

2021 年 6 月 30 日: FlexMatch が batchDistance ルールを追加

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、各セグメントにホスト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 Servers リアルタイムクライアント SDK および Server 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 Servers サーバー SDK for Unreal Version 4.26 を構築することもできます。この更新プログラムには、Amazon GameLift Servers サーバー SDK の C++ および C# バージョン、Amazon GameLift Servers ローカル、Unreal Engine プラグインなど、Windows または Linux のゲーム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 Servers デベロッパーガイド](#):
 - [DescribePlayerSessions\(\)](#)
 - [GetInstanceCertificate\(\)](#)

2021 年 3 月 23 日: Amazon GameLift Servers がゲームセッション配置に通知を追加

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 Events を使用してイベントトラッキングを設定します。

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 Servers デベロッパーガイド](#):

- [ゲームセッション配置のイベント通知を設定](#) (新規)
- [ゲームセッション配置イベント](#) (新規)
- [AWS API リファレンス](#) (SDK)
 - 新しいゲームセッションキューパラメータ NotificationTarget と CustomEventData: [GameSessionQueue](#)、[CreateGameSessionQueue](#)、[UpdateGameSessionQueue](#)
- [Amazon GameLift Servers フォーラム](#)

2021 年 3 月 16 日: Amazon GameLift Servers がマルチリージョンのフリート、6 つの新しいリージョンを追加

SDK バージョンの更新: AWS SDK [1.8.163](#)

Amazon GameLift Servers 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 Servers マルチロケーションフリート機能で、Amazon GameLift Servers がサポートする20 のリージョンすべてで (北京リージョンを除く) ゲームサーバーをホストする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 Servers ホスティングリソースをグローバルに設定および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、必要に応じて単一口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。以前と同様に、Amazon GameLift Servers の価格は、使用するインスタンスのタイプ、ロケーション、ポリシーに基づきます。(詳細については、「[Amazon GameLift Servers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 CloudFormation のサポートは間もなく利用可能になります。

Note

マルチロケーションフリートは、中国リージョンでは使用できません。中国リージョンに存在する Amazon GameLift Servers リソースは、他のリージョンのリソースとやり取りしたり、他の Amazon GameLift Servers リージョン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 GameLift Servers 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。自動スケーリングポリシーは、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 Servers 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、各フリートロケーションごとに分割して集計します。

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](#)、[DescribeFleetLocationAttributes](#)、[DescribeFleetLocationCapacity](#)、[DescribeFleetLocations](#)
 - 新しいマルチロケーションサポートの追加でフリートオペレーションを更新しました:
[CreateFleet](#)、[UpdateFleetCapacity](#)、[DescribeEC2InstanceLimits](#)、[DescribeInstances](#)、[StopFleetAction](#)
 - 新しい優先度とフィルタリング機能の追加でゲームセッションのプレイスメントオペレーションを更新しました:
[CreateGameSessionQueue](#)、[DescribeGameSessionQueues](#)、[UpdateGameSessionQueue](#)
 - 新しいロケーションサポートの追加で、ゲームセッション作成オペレーションを更新しました:
[CreateGameSession](#)、[DescribeGameSessions](#)、[DescribeGameSessionDetails](#)、[SearchGameSessions](#)
- [Amazon GameLift Servers デベロッパーガイド](#):
 - [Amazon GameLift Servers サービスロケーション](#) (更新済み)
 - [ホスティングリソースのカスタマイズ](#) (新規)

[Amazon GameLift Servers を使用したゲームホスティング容量のスケーリング](#) (更新済み)
 - [ゲームセッションキューをカスタマイズする](#) (新規)
 - [Amazon GameLift Servers コンソールのフリートの詳細](#) (更新済み)

- [Amazon GameLift Servers フォーラム](#)

2021 年 2 月 9 日: Amazon GameLift Servers は AMD インスタンスとスタンドアロン FlexMatch のサポートを延長

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 Servers FleetIQ ゲームサーバーグループを、AMD インスタンスファミリー C5a、M5a、および R5a を管理するように構成できるようになりました。GameServerGroup [InstanceDefinition](#) にリストされている、サポートされている Amazon EC2 インスタンスタイプには、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.24xlarge
 - m5a.large、m5a.xlarge、m5a.2xlarge、m5a.4xlarge、m5a.8xlarge、m5a.12xlarge、m5a.16xlarge、m5a.24xlarge
 - r5a.large、r5a.xlarge、r5a.2xlarge、r5a.4xlarge、r5a.8xlarge、r5a.12xlarge、r5a.16xlarge、r5a.24xlarge

注: の AMD インスタンス FleetIQ 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 Servers 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でき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 [InstanceType](#) にリストされているサポートされている EC2 インスタンスタイプには、次のものが含まれます。
 - m5a.large、m5a.xlarge、m5a.2xlarge、m5a.4xlarge、m5a.8xlarge、m5a.12xlarge、m5a.16xlarge、m5a.24xlarge
 - r5a.large、r5a.xlarge、r5a.2xlarge、r5a.4xlarge、r5a.8xlarge、r5a.12xlarge、r5a.16xlarge、r5a.24xlarge
- Amazon GameLift Servers FlexMatch は、シンネットが運営する中国 (北京) リージョンで、スタンドアロンのマッチメイ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。お客様が北京リージョンで FlexMatch マッチメーカーを作成し、[FlexMatchMode](#) パラメータを STANDALONE に設定できるようになりました。Amazon GameLift Servers マネージドホスティング、または Amazon GameLift Servers 以外のホスティングソリューションを使用する FlexMatch の詳細については、「[Amazon GameLift Servers FlexMatch デベロッパーガイド](#)」を参照してください。
- Amazon GameLift Servers FlexMatch のイベント通知を設定するときに、Amazon SNS FIFO トピックを通知ターゲットに指定できるようになりました。詳細については、「[MatchmakingConfiguration NotificationTarget](#)」を参照してください。
- 「[MatchmakingConfiguration NotificationTarget](#)」、[Amazon GameLift Servers API リファレンス](#)

- 「[FlexMatch イベント通知の設定](#)」、「Amazon GameLift Servers FlexMatch デベロッパーガイド」
- [Amazon SNS FIFO の紹介 – 先入れ先出し 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 Servers Server SDK が Unreal Engine 4.25 および Unity 2020 に対応

更新された SDK バージョン: Amazon GameLift Servers Server SDK 4.0.2、Unreal プラグインバージョン 3.3.3

Amazon GameLift Servers Server SDK の最新バージョンには、次の更新された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 Servers Server SDK の最新バージョンを「[Amazon GameLift Servers の使用開始](#)」からダウンロードしてください。

2020 年 11 月 24 日: Amazon GameLift Servers FlexMatch がどこでホストされるゲームでも使用できるようになりました

SDK バージョンの更新: AWS SDK [1.8.95](#)

Amazon GameLift Servers 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イキングサービスです。当初、Amazon GameLift Servers マネージドホスティングのユーザー向けに設計された FlexMatch は、ピアツーピア、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など、他のホスティングシステムを使用するゲームに統合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 Servers FleetIQ を使用するゲームは、FlexMatch でのマッチメイキングを実装できるようになりました。

FlexMatch は、強力なマッチメイキングアルゴリズムとルール言語を提供し、マッチメイ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と一緒にマッチングされるようにします。さらに、FlexMatch は、プレイヤーパーティ、プレイヤーの受け入れ、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イキングリクエストワークフローを提供します。FlexMatch を Amazon GameLift Servers マネージド

ホスティングまたは Amazon GameLift Servers Realtime で使用すると、マッチメーカーは自動的に Amazon GameLift Servers を使用してホスティングリソースを見つけ、新しく形成されたマッチの新しいゲームセッションを開始します。FlexMatch をスタンドアロンサービスとして使用する場合、マッチメーカーはマッチ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 Servers 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、FlexMatchMode はマッチメーカーが Amazon GameLift Servers 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イキングとして使用されているかを示します。
 - 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 GameLift Servers で利用可能になりました

SDK バージョンの更新: AWS SDK [1.8.95](#)

Amazon GameLift Servers 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、R5a という 3 つの新しいインスタンスファミリーが含まれるようになりました。これらのファミリーは、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。GHz。AMD インスタンスは x86 互換です。Amazon GameLift Servers 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 (バージニア北部とオハイオ)、米国西部 (オレゴンと北カリフォルニア)、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 (ロンドンとアイルランド)、アジアパシフィック南 (ムンバイ)、アジアパシフィック北東部 (ソウルと東京)、アジアパシフィック南東部 (シンガポールとシドニー) 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.24xlarge
- m5a.large、m5a.xlarge、m5a.2xlarge、m5a.4xlarge、m5a.8xlarge、m5a.12xlarge、m5a.16xlarge、m5a.24xlarge
- 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 Servers 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 Servers フォーラム](#)

2020 年 11 月 11 日: Amazon GameLift Servers Server SDK のバージョン更新

SDK バージョンの更新: Amazon GameLift Servers 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 Servers Server SDK の最新バージョンを「[Amazon GameLift Servers の使用開始](#)」からダウンロードしてください。

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、マッチメイキングプロセスの次のデフォルト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、マッチメイキングルールセットで設定されます。Amazon GameLift Servers SDK 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 GameLift Servers がサーバー SDK を更新

SDK バージョンの更新: Amazon GameLift Servers 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。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。
 - オペレーション [GetTerminationTime\(\)](#) がデータ型 AwsDateTimeOutcome の値を返すようになりました。
- 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 Servers Server SDK の最新バージョンを「[Amazon GameLift Servers の使用開始](#)」からダウンロードしてください。

2020 年 8 月 27 日: Amazon EC2 でゲームホスティングのための Amazon GameLift Servers FleetIQ (一般公開)

SDK バージョンの更新: 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の Amazon GameLift Servers FleetIQ ソリューションが一般公開されました。Amazon GameLift Servers FleetIQ を使用すると、開発者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、新しいゲームで使用したり、既存のゲームの容量を補足するために Amazon GameLift Servers FleetIQ を使用できます。このソリューションは、Shield や Amazon Elastic Container Service (Amazon ECS) AWS などのコンテナやその他の AWS サービスの使用をサポートしています。

この一般リリースには、Amazon GameLift Servers FleetIQ ソリューションへの次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 Servers 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シング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 GameLift Servers で Unity および Unreal Engine 用の Server SDK を更新。

更新された SDK バージョン: Amazon GameLift Servers SDK 4.0.0、Amazon GameLift Servers Local 1.0.5

Amazon GameLift Servers Server SDK の最新バージョンには、次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。
- C# サーバー SDK バージョン 4.0.0 を使用する統合をテストするため、Amazon GameLift Servers ローカルバージョン 1.0.5 を更新。

Amazon GameLift Servers Server SDK の最新バージョンを「[Amazon GameLift Servers の使用開始](#)」からダウンロードしてください。

2020 年 4 月 2 日: Amazon GameLift Servers FleetIQ が EC2 でのゲームホスティングに利用可能に (パブリックプレビュー)

SDK バージョンの更新: AWS SDK [1.7.310](#)

Amazon GameLift Servers 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 Servers サービスを介さないで直接ホスティングリソースを管理するお客様向けに拡張されました。このソリューションは、Shield や Amazon Elastic Container Service (Amazon ECS) AWS などのコンテナやその他の AWS サービスの使用をサポートしています。

詳細はこちら:

Amazon GameLift Servers FleetIQ の [GameTech ブログ記事](#)

2019 年 12 月 19 日: AWS リソースの Amazon GameLift Servers リソース管理を改善

SDK バージョンの更新: AWS SDK [1.7.249](#)

AWS リソースで Amazon GameLift Servers 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、マッチメイキング設定、マッチメイキングルールセットなど、すべての主要な Amazon GameLift Servers リソースに Amazon リソースネーム (ARN) 値が割り当てられるようになりました。リソース ARN は、すべての AWS リージョンで一貫した識別子を提供します。これらを使用して、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 Servers 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 Servers リソースのタグを管理する場合は、Amazon GameLift Servers API アクション `TagResource()`、`UntagResource()`、および `ListTagsForResource()` を使用します。

詳細はこちら:

- Amazon GameLift Servers API リファレンスの [TagResource](#)
- 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: 中国 (北京) リージョンの新しい CloudFormation テンプレートと更新

SDK バージョンの更新: AWS SDK [1.7.210](#)

CloudFormation の テンプレート Amazon GameLift Servers

Amazon GameLift Servers リソースを作成および管理できるようになりました CloudFormation。既存の 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、スクリプト、キュー、マッチメイキング設定、マッチメイキングルールセットで新しいテンプレートが利用可能になりました。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、関連する 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 Servers リソースタイプのリファレンス](#)」
- 「Amazon GameLift Servers デベロッパーガイド」の「[を使用してAmazon GameLift Serversホスティングリソースを管理する CloudFormation](#)」

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。