aws

开发人员指南

Amazon GameLift

Amazon GameLift: 开发人员指南

Table of Contents

什么是 Amazon GameLift？ 1
Amazon GameLift 的用途 1
开始使用 Amazon GameLift 解决方案 1
适用于自定义服务器的 Amazon GameLift 托管 2
使用实时服务器托管 Amazon GameLift 2
Amazon GameLift FleetIQ 用于在 Amazon EC2 上托管 2
Amazon GameLift FlexMatch 用于对战 3
Amazon GameLift Anywhere 硬件托管 3
访问 Amazon GameLift 4
Amazon GameLift 的定价 4
Amazon GameLift 的工作原理 4
关键组件 5
托管游戏服务器 5
运行游戏会话 6
扩展实例集容量 6
监控 Amazon GameLift 7
使用其他 AWS 资源 7
如何将玩家接入游戏 7
使用托管 Amazon 的游戏架构 GameLift 8
设置 11
设置了账户 11
注册获取 AWS 账户 11
创建具有管理访问权限的用户 12
管理 Amazon 的用户权限 GameLift 13
为用户设置编程式访问权限 14
为游戏设置编程式访问权限 15
IAM 权限示例 15
设置 IAM 服务角色 19
开发 Linux 支持 22
对于自定义游戏服务器 22
用于自定义客户端服务 24
对于实时服务器 24
管理您的游戏托管成本 25
创建账单提醒以监控使用情况 25

追踪每个 Amazon GameLift 车队的成本 25
将未使用的实例集容量设置为零 25
亚马逊 GameLift 托管地点 26
亚马逊 GameLift 托管 26
Local Zones 27
Amazon GameLift Anywhere 28
Amazon GameLift FlexMatch 28
亚马逊 GameLift 在中国 29
开始使用 30
自定义游戏服务器示例 30
实时服务器示例游戏 30
托管托管资源路线图 32
选择托管选项 32
准备您的游戏 34
准备自定义游戏服务器 34
准备实时服务器 35
测试您的集成。 35
规划和部署您的资源 35
部署资源 36
设计您的后端服务 36
对您的玩家进行身份验证 37
无服务器后端 37
基于 WebSocket 的后端 39
设置指标和日志记录 41
启动清单 41
激活 42
测试 42
启动 43
启动后 43
为亚马逊准备游戏 GameLift 44
将游戏与自定义游戏服务器集成 44
Amazon GameLift 互动 45
集成游戏服务器 49
集成游戏客户端 58
游戏引擎和 Amazon GameLift 63
测试您的集成（服务器软件开发工具包 5） 87

测试您的集成（服务器软件开发工具包 4） 94
将游戏与实时服务器集成 101
什么是实时服务器？ 101
管理游戏会话。 102
客户端与服务器的交互 102
自定义服务器 103
部署和更新 103
集成游戏客户端 103
自定义实时脚本 109
将游戏与适用于 Unity 的插件集成 114
适用于 Unity 的插件指南（服务器 SDK 5.x） 115
适用于 Unity 的插件指南（服务器 SDK 4.x） 130
将游戏与适用于 Unreal 的插件集成 155
关于插件 156
插件工作流 156
安装适用于 Unreal 的插件 157
设置 AWS 用户配置文件 160
使用 Anywhere 设置您的游戏 162
使用托管的 Amazon EC2 实例集部署您的游戏 174
获取车数据 177
添加 FlexMatch 对战 178
使用容器管理托管 [预览] 179
主要特征 179
在公开预览期间使用集装箱舰队 179
容器的工作原理 180
集装箱船队组件 180
常见架构 181
核心概念 183
开发路线图 185
将您的游戏与 Amazon 集成 GameLift 187
集成工具 188
为 Linux 构建你的游戏服务器 189
测试与 Anywhere 队列的集成 189
准备容器镜像 191
创建工作目录 191
建立你的形象 192

推送你的图片 201
设计集装箱船队 202
设计您的舰队集装箱结构 202
设置资源限制 203
指定基本容器 205
配置网络连接 205
为容器设置运行状况检查 208
设置容器依赖关系 209
配置集装箱舰队 209
创建容器组定义 211
开始之前 211
克隆容器组定义 211
创建副本容器组定义 212
创建容器定义JSON文件 214
创建集装箱舰队 215
管理您的集装箱船队 220
查看 资源 220
更新资源 221
删除资源 221
扩展集装箱舰队 221
管理托管资源 223
上传构建和脚本 224
上传构建 224
上传脚本 232
设置实例集 236
实例集设计指南 237
创建新实例集 243
管理实例集 258
将别名添加到实例集 260
调试实例集问题 262
远程连接到舰队实例 265
扩展托管容量 272
在控制台中管理实例集容量 272
设置托管容量限制 273
手动设置实例集容量 274
自动扩缩实例集容量 276

设置 Queule 282
设计队列 282
最佳实操 289
创建队列 290
设置事件通知 293
教程：竞价型实例的队列 297
借助 AWS CloudFormation 管理资源 304
最佳实操 304
使用 AWS CloudFormation 堆栈 305
更新构建 309
VPC 对等连接 311
为现有实例集设置 VPC 对等连接 312
使用新实例集设置 VPC 对等连接 314
VPC 对等连接问题疑难解答 316
查看游戏数据 318
查看您的亚马逊 GameLift状态 318
查看您的构建 319
构建详细信息 320
查看您的脚本 321
脚本详细信息 321
查看您的实例集 321
查看实例集详细信息 322
详细信息 322
指标 323
事件 324
扩展 324
Locations 324
游戏会话 325
查看游戏和玩家信息 325
详细信息 325
玩家会话 326
玩家信息 327
查看您的别名 327
别名详细信息 327
查看队列 328
查看队列详细信息 328

监控亚马逊 GameLift 331
使用 CloudWatch 监控 GPU 331
指标维度 332
实例集指标 333
队列指标 342
FlexMatch 指标 346
FleetIQ 指标 349
记录 API 调用 350
CloudTrail 中的 Amazon GameLift 信息 351
了解 Amazon GameLift 日志文件条目 352
记录服务器消息 354
自定义服务器的日志记录 354
实时服务器的日志记录 357
安全性 361
数据保护 361
静态加密 363
传输中加密 363
互联网络流量隐私保护 363
Identity and Access Management 364
受众 364
使用身份进行身份验证 365
使用策略管理访问 367
亚马逊如何 GameLift 使用 IAM 369
基于身份的策略示例 376
故障排除 380
Amazon GameLift 的日志记录和监控 382
合规性验证 382
韧性 383
基础设施安全性 384
配置和漏洞分析 385
安全最佳实操 385
不要打开互联网端口 385
了解更多信息 386
Amazon GameLift 参考指南 387
服务 API 参考（AWS 软件开发工具包） 387
设置和管理 Amazon GameLift 托管资源 387

开始游戏会话并加入玩家行列 391
实时服务器参考 391
实时客户端 API (C#) 参考 392
实时服务器脚本参考 405
服务器软件开发工具包参考 412
适用于 C++ 的服务器 API 参考 412
C# 的服务器软件开发工具包 参考 481
适用于 Go 的服务器软件开发工具包 参考 539
适用于 Unreal Engine 的 服务器 API 参考 564
游戏会话放置事件 619
放置事件语法 619
PlacementFulfilled 620
PlacementCancelled 622
PlacementTimedOut 622
PlacementFailed 623
估算价格 625
估算 Amazon GameLift 托管 625
Amazon GameLift 实例 625
数据传出 (DTO) 627
估算 Amazon GameLift 独立 FlexMatch 628
限额和支持的区域 630
发行说明和软件开发工具包版本 631
开发工具包版本 631
发布说明 640
AWS 术语表 666
..... dclxvii

什么是 Amazon GameLift？

Amazon GameLift 使开发人员能够在云中，为基于会话的多人游戏部署、操作和扩展专用的低成本服务器。Amazon GameLift 基于 AWS 全球计算基础设施而构建，可帮助您交付具有高性能、高可靠性的游戏服务器，并动态扩展资源使用量以满足全球玩家需求。

Amazon GameLift 的用途

Amazon GameLift 支持以下用例以及更多用例：

- 使用您自己的自定义多人游戏服务器，或使用随时可用的实时服务器来托管您的游戏。
- 使用 Amazon Elastic Compute Cloud (Amazon EC2) 竞价型实例来运行低成本的托管资源。
- 根据使用情况自动调整游戏所需的托管资源量。
- 使用 Amazon GameLift FleetIQ 在一个地方管理您的 Amazon EC2 计算资源。
- 使用 Amazon GameLift FlexMatch 在多人游戏中匹配玩家。
- 使用 Amazon GameLift Anywhere 迭代测试您的游戏服务器和客户端构建。
- 使用您自己的硬件，同时使用 Amazon GameLift Anywhere 在一个地方管理所有硬件。

> **Tip**
>
> 要试用 Amazon GameLift 游戏服务器托管，请参阅Amazon VPC 入门。

开始使用 Amazon GameLift 解决方案

面向游戏开发人员的 Amazon GameLift 解决方案

- 适用于自定义服务器的 Amazon GameLift 托管
- 使用实时服务器托管 Amazon GameLift
- Amazon GameLift FleetIQ 用于在 Amazon EC2 上托管
- Amazon GameLift FlexMatch 用于对战
- Amazon GameLift Anywhere 硬件托管

适用于自定义服务器的 Amazon GameLift 托管

Amazon GameLift 取代了托管您自己的自定义游戏服务器所需的工作。自动扩缩功能可帮助您避免为超出需求的资源付费。自动扩缩还有助于确保您始终有游戏可供新玩家加入，而无需等待。

有关 Amazon GameLift 托管的更多信息，请参阅Amazon GameLift 的工作原理。

主要特征

- 使用 Amazon GameLift 管理特征，包括自动扩缩、多区域队列和游戏会话放置。
- 在 Amazon Linux 或 Windows Server 操作系统上部署游戏服务器。
- 管理游戏会话和玩家会话。
- 为服务器进程设置自定义的运行状况跟踪，以便发现问题并解决性能较差的进程。
- 使用适用于 Amazon GameLift 的 AWS CloudFormation 模板管理游戏资源。

使用实时服务器托管 Amazon GameLift

使用实时服务器启动游戏无需自定义构建游戏服务器。此轻量级服务器解决方案提供可以进行配置来适合您的游戏的游戏服务器。

有关使用实时服务器托管 Amazon GameLift 的更多信息，请参阅将游戏与 Amazon GameLift 实时服务器集成。

主要特征

- 使用 Amazon GameLift 管理特征，包括自动扩缩、多区域队列和游戏会话放置。
- 使用 Amazon GameLift 托管资源，为您的实例集选择 AWS 计算硬件的类型。
- 充分利用完整的网络堆栈进行游戏客户端和服务器交互。
- 通过可定制的服务器逻辑获取核心游戏服务器功能。
- 对实时配置和服务器逻辑进行实时更新。

Amazon GameLift FleetIQ 用于在 Amazon EC2 上托管

使用 Amazon GameLift FleetIQ 直接使用 Amazon EC2 和 Amazon EC2 Auto Scaling 中的托管资源。这为低成本、弹性的游戏托管提供了Amazon GameLift 优化的好处。该解决方案适用于需要比完全托管的 Amazon GameLift 解决方案更高的灵活性的游戏开发人员。

有关 Amazon GameLift FleetIQ 如何与 Amazon EC2 和 EC2 Auto Scaling 合作进行游戏托管的信息，请参阅 Amazon GameLift FleetIQ 开发人员指南。

主要特征

- 使用 FleetIQ 算法优化竞价型实例平衡。
- 使用玩家路由特征高效管理游戏服务器资源，为玩家加入游戏提供更好的体验。
- 根据玩家使用情况自动扩展托管容量。
- 直接通过 AWS 账户管理 Amazon EC2 实例。
- 可使用多种支持的游戏服务器可执行文件格式，包括 Windows、Linux、容器和 Kubernetes。

Amazon GameLift FlexMatch 用于对战

使用 FlexMatch 构建自定义规则集，为您的游戏定义多人对战。FlexMatch 使用规则集来比较每场对战的兼容玩家，为玩家提供理想的多人游戏体验。

有关 FlexMatch 的更多信息，请参阅什么是 Amazon GameLift FlexMatch？

主要特征

- 平衡对战创建速度和质量。
- 根据定义的特征匹配玩家或团队。
- 定义规则，根据延迟安排玩家进入对战。

Amazon GameLift Anywhere 硬件托管

使用 Amazon GameLift Anywhere 将环境中任何地方的硬件集成到 Amazon GameLift 游戏托管中。您可以将 Anywhere 实例集和 EC2 队列集成到对战构建器和游戏会话队列中，以管理硬件上的对战和游戏放置。

有关使用 Anywhere 进行测试的更多信息，请参阅使用 Amazon GameLift Anywhere 实例集测试您的集成。有关设置 Anywhere 实例集的更多信息，请参阅设置 Amazon GameLift 实例集。

主要特征

- 对游戏服务器和客户端构建进行快速迭代测试。
- 使用设置的 Amazon GameLift 工具将游戏部署到您自己的硬件上。

- 随时随地使用离玩家最近的硬件。

访问 Amazon GameLift

使用这些工具与 Amazon GameLift 配合使用。

Amazon GameLift 软件开发工具包

Amazon GameLift 软件开发工具包包含从您的游戏客户端、游戏服务器和游戏服务与 Amazon GameLift 通信时需要的库。有关更多信息，请参阅Amazon 为开发提供支持 GameLift。

Amazon GameLift 实时客户端软件开发工具包

实时客户端软件开发工具包使游戏客户端能够连接到实时服务器、加入游戏会话并与其他玩家保持同步。下载软件开发工具包，了解有关使用实时服务器客户端 API (C#)发起 API 调用的更多信息。

Amazon GameLift 控制台

使用适用于 Amazon GameLift 的 AWS Management Console来管理游戏部署、配置资源和跟踪玩家使用情况及性能指标。Amazon GameLift 控制台提供了通过 AWS Command Line Interface (AWS CLI) 以编程方式管理资源的 GUI 替代方式。

AWS CLI

使用此命令行工具调用 AWS 软件开发工具包，包括 Amazon GameLift API。有关使用 AWS CLI 的更多信息，请参阅《AWS Command Line Interface 用户指南》中的 AWS CLI 入门。

Amazon GameLift 的定价

Amazon GameLift 按使用时长对实例收费，按传输的数据量收取带宽费用。有关 Amazon GameLift 的费用和价格的完整列表，请参阅 Amazon GameLift 定价。

有关计算通过 Amazon GameLift 托管游戏或对战的成本的信息，请参阅生成 Amazon GameLift 定价估算值，其中介绍了如何使用 AWS Pricing Calculator。

Amazon GameLift 的工作原理

本主题涵盖了游戏托管的核心组件，并描述了 Amazon GameLift 如何向玩家提供您的多人游戏服务器。

准备好在 Amazon GameLift 上托管游戏了吗？ 签出Amazon GameLift 托管托管资源路线图。

关键组件

设置 Amazon GameLift 托管您的游戏需要使用以下组件。使用托管 Amazon 的游戏架构 GameLift中的图表可视化了这些组件之间的关系。

- 游戏服务器是指在实例集上运行的游戏服务器软件。将游戏服务器构建或脚本上传到 Amazon GameLift 然后告诉 Amazon GameLift。当使用 Amazon GameLift Anywhere 或 Amazon GameLift FleetIQ 时，可以将游戏服务器构建直接上传到计算资源。
- 游戏会话是与玩家一起玩的正在进行的游戏。您定义游戏会话的基本特征，例如它的生命期和玩家数。然后，玩家连接到游戏服务器以加入游戏会话。
- 游戏客户端是指玩家设备上运行的游戏软件。游戏客户端根据从 Amazon GameLift 收到的连接信息，通过后端服务连接到游戏服务器以加入游戏会话。
- 后端服务是额外的自定义服务，用于处理与 Amazon GameLift 相关的任务。作为最佳实操，您的后端服务应处理所有游戏客户端与 Amazon GameLift 的通信。

托管游戏服务器

使用 Amazon GameLift，您可以通过三种不同的方式托管游戏服务器：托管 Amazon GameLift、Amazon GameLift FleetIQ 和 Amazon GameLift Anywhere。有关 Amazon GameLift FleetIQ 的更多信息，请参阅什么是 Amazon GameLift FleetIQ？

您可以设计一个适合游戏需求的实例集。有关设计实例集的更多信息，请参阅Amazon GameLift 实例集设计指南。

托管 Amazon GameLift

借助托管 Amazon GameLift，您可以将游戏服务器托管在 Amazon GameLift 虚拟计算资源（称为实例）上。通过创建实例实例集并将其部署到运行游戏服务器来设置托管资源。

Amazon GameLift Anywhere

借助 Amazon GameLift Anywhere，您可以在自己管理的计算机上托管游戏服务器。通过创建引用您的计算的 Anywhere 实例集来设置您的托管资源。

实例集别名

别名是可以在实例集之间进行传输的称号，从而方便地泛化实例集位置。使用别名，您可将游戏客户端从一个实例集切换到另一个实例集，而无需更改游戏客户端。您也可以创建指向内容的终端别名。

运行游戏会话

将游戏服务器构建部署到实例集并且 Amazon GameLift 在每个实例上启动游戏服务器进程后，实例集就可以托管游戏会话。当您的游戏客户端服务向后端服务或 Amazon GameLift 发送放置请求时，Amazon GameLift 会启动新的游戏会话。

游戏会话放置和 FleetIQ 算法

队列使用 FleetIQ 算法选择可用的游戏服务器托管新的游戏会话。游戏会话放置的关键组件是 Amazon GameLift 游戏会话队列。您可以为游戏会话队列分配队列一个实例集列表，该列表决定了队列可以将游戏会话放置在何处。有关游戏会话队列以及如何为您的游戏设计游戏会话队列的更多信息，请参阅设计游戏会话队列。

玩家与游戏的联系

作为游戏会话置放过程的一部分，队列或游戏会话提示选定的游戏服务器启动新的游戏会话。游戏服务器会对提示做出响应，并在准备好接受玩家连接时向 Amazon GameLift 报告。然后，Amazon GameLift 会向后端服务或游戏客户端服务提供连接信息。然后，游戏客户端使用此信息直接连接到游戏会话并开始游戏。

扩展实例集容量

当某个实例集激活并准备好托管游戏会话后，您可以调整实例集容量以满足玩家需求。我们建议您在所有新玩家快速找到游戏和超支闲置资源之间找到平衡。

Amazon GameLift 提供了一种高效的自动扩缩工具，您也可以手动设置实例集容量。有关更多信息，请参阅扩展 Amazon GameLift 托管容量。

Auto Scaling

Amazon GameLift 提供了两种自动扩缩方法：

- 基于目标的自动扩缩
- 使用基于规则的策略自动扩缩

其他扩展功能

- 游戏会话保护 – 防止托管活动的玩家的游戏会话在缩减事件期间被 Amazon GameLift 终止。
- 扩展限制 – 通过对实例集中的实例数设置最小和最大限制，控制总体实例使用情况。

- 暂停自动扩缩 – 在不更改或删除自动扩缩策略的情况下，在实例集位置级别暂停自动扩缩。
- 扩展指标 – 跟踪实例集的容量和扩展事件的历史记录。

监控 Amazon GameLift

在实例集设置完毕并开始运行后，Amazon GameLift 会收集各种信息，以帮助您监控已部署的游戏服务器的性能。此信息可用于优化资源使用、排除问题以及深入了解玩家在游戏中的活动情况。Amazon GameLift 收集以下信息：

- 实例集、位置、游戏会话和玩家会话详情
- 使用情况指标
- 服务器进程运行状况
- 游戏会话日志

有关在 Amazon GameLift 中监控的更多信息，请参阅监控亚马逊 GameLift。

使用其他 AWS 资源

您的游戏服务器和应用程序可以与其他 AWS 资源通信。例如，您可能将一组 Web 服务用于玩家身份验证或社交网络。要让您的游戏服务器访问 AWS 账户管理的 AWS 资源，请明确允许 Amazon GameLift 访问您的 AWS 资源。

Amazon GameLift 提供了几个用于管理此类访问的选项。有关更多信息，请参阅与您的实例集中的其他 AWS 资源进行通信。

如何将玩家接入游戏

游戏会话是在 Amazon GameLift 上运行的游戏实例。要玩游戏，玩家可以查找并加入一个现有的游戏会话或创建一个新的游戏会话并加入。玩家通过为游戏会话创建玩家会话来加入游戏。如果游戏会话对玩家开放，则 Amazon GameLift 会为玩家保留一个空位并提供连接信息。玩家随后可以连接到游戏会话并认领预留的位置。

有关创建和管理游戏会话和玩家会话的更多信息，请参阅将 Amazon GameLift 添加到您的游戏客户端。有关将玩家连接到实时服务器的信息，请参阅为实时服务器集成游戏客户端。

Amazon GameLift 提供了与游戏和玩家会话相关的多种特征。

在多个区域中的最佳可用资源上托管游戏会话

配置 Amazon GameLift 如何选择资源来托管新游戏会话时，可从多个选项中选择。如果您在多个位置运行实例集，则可以设计游戏会话队列，在任何实例集上放置新的游戏会话，无论其位置如何。

控制玩家访问游戏会话

不考虑当前已连接的玩家数量，将游戏会话配置为允许或拒绝新玩家加入请求。

使用自定义游戏和玩家数据

向游戏会话对象和玩家会话对象添加自定义数据。启动新游戏会话时，Amazon GameLift 将游戏会话数据传递到游戏服务器。当玩家连接到游戏会话时，Amazon GameLift 会将玩家会话数据传递给游戏服务器。

对游戏会话进行筛选和排序

使用会话搜索和排序为潜在玩家查找最佳匹配，或让玩家从可用游戏会话的列表中进行选择。使用会话搜索和排序，根据会话特征查找游戏会话。您也可以根据您自己的自定义游戏数据搜索和排序。

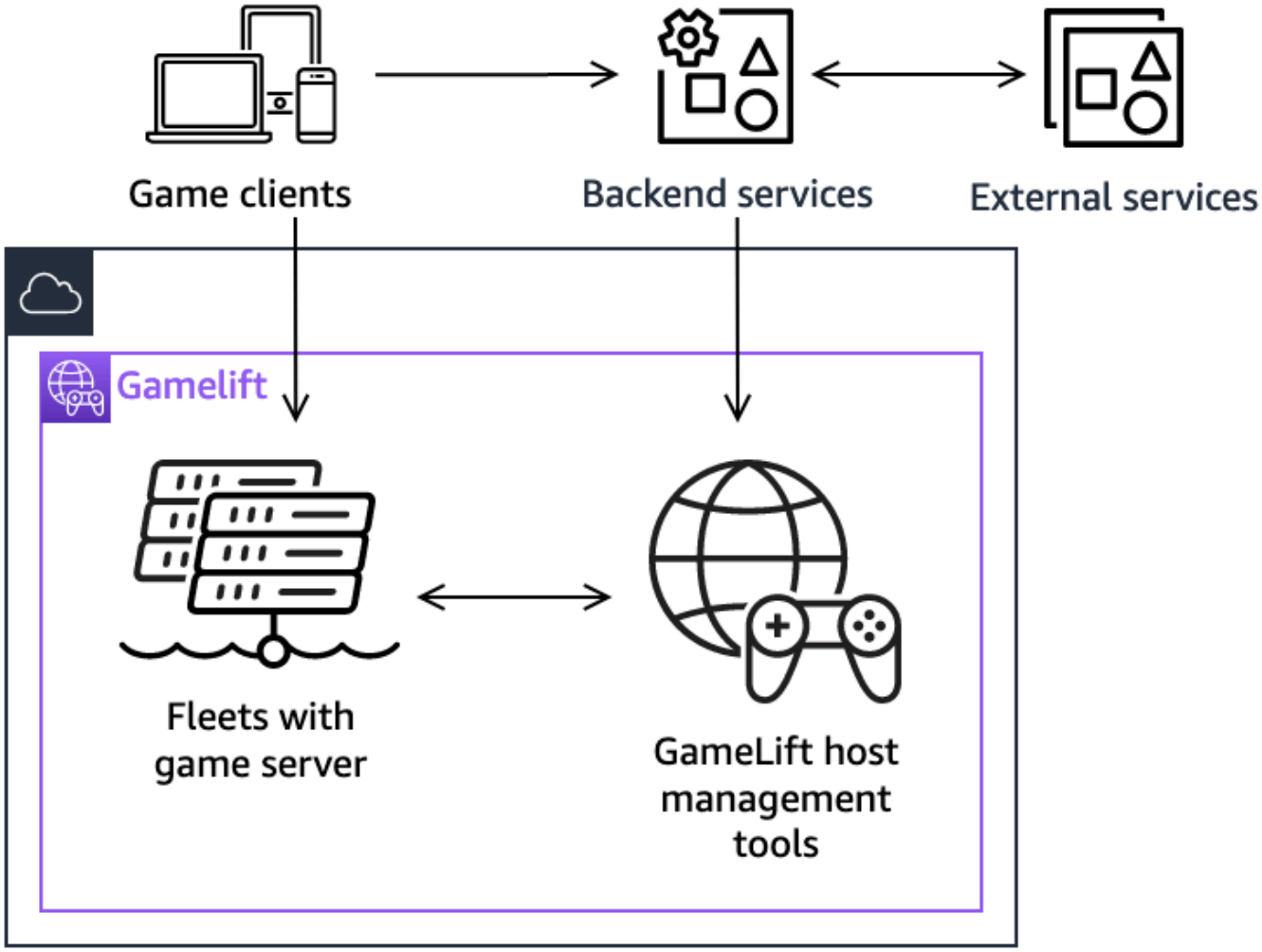
跟踪游戏和玩家使用数据

自动存储已完成的游戏会话的日志。在将 Amazon GameLift 集成到游戏服务器时设置日志存储。有关更多信息，请参阅在 Amazon GameLift 中记录服务器消息。

使用 Amazon GameLift 控制台查看游戏会话的详细信息，包括会话元数据、设置以及玩家会话数据。有关更多信息，请参阅 查看游戏和玩家会话中的数据 和 指标：

使用托管 Amazon 的游戏架构 GameLift

下图说明了使用托管 Amazon GameLift 解决方案托管的游戏架构的关键组件。



架构包含以下关键组件：

游戏客户端

要加入 Amazon 上托管的游戏 GameLift，您的游戏客户端必须先找到可用的游戏会话。游戏客户端通过后端服务与 Amazon 通信来搜索现有游戏会话、请求配对，或者 GameLift 通过后端服务与 Amazon 通信来启动新的游戏会话。后端服务向 Amazon 发出请求 GameLift，作为响应，该服务接收游戏会话信息，然后将其中继回游戏客户端。然后，游戏客户端连接到游戏服务器。有关更多信息，请参阅 为亚马逊准备游戏 GameLift。

后端服务

后端服务 GameLift 通过调用 AWS 软件开发工具包中的亚马逊 GameLift 服务 API 操作来处理游戏客户端与亚马逊之间的通信。后端服务也可用于其他游戏特定任务，例如玩家身份验证和授权、库存或货币控制。有关更多信息，请参阅 设计您的游戏客户端服务。

外部服务

您的游戏可以依赖外部服务，如用于验证订阅成员资格。外部服务可以通过后端服务和 Amazon 将信息传递给您的游戏服务器 GameLift。

游戏服务器

您将游戏服务器软件上传到亚马逊 GameLift， GameLift 然后 Amazon 将其部署到托管计算机上，以托管游戏会话并接受玩家连接。游戏服务器与 Amazon 通信 GameLift 以启动游戏会话、验证新连接的玩家，并报告游戏会话的状态、玩家连接和可用资源。

自定义游戏服务器使用亚马逊 GameLift GameLift 服务器 SDK 与亚马逊通信。游戏客户端 GameLift 通过后端服务从 Amazon 接收连接详情后，直接连接到游戏服务器。有关更多信息，请参阅 将游戏与自定义游戏服务器集成。

实时服务器是运行您的自定义脚本的游戏服务器。加入游戏时，游戏客户端使用实时客户端软件开发工具包直接连接到实时服务器。有关更多信息，请参阅 将游戏与 Amazon GameLift 实时服务器集成。

托管管理工具

在设置和管理托管资源时，游戏所有者使用托管管理工具来管理游戏服务器构建或脚本、实例集、对战和队列。软件开发 GameLift 工具 AWS 包和控制台中设置的 Amazon 工具为您提供了多种管理托管资源的方式。您可以远程访问任一游戏服务器以进行问题排查。

设置

获取有关设置 AWS 账户 使用 Amazon GameLift 托管多人游戏的帮助。

> **Tip**
>
> 要试用 Amazon GameLift 游戏服务器托管，请参阅Amazon VPC 入门。

主题

- 设置一个 AWS 账户
- Amazon 为开发提供支持 GameLift
- 管理您的游戏托管成本
- 亚马逊 GameLift 托管地点

设置一个 AWS 账户

要开始使用亚马逊 GameLift，请创建并设置您的 AWS 账户。创建 AWS 账户不收取任何费用。本部分将引导您完成创建账户、设置用户和配置权限的过程。

主题

- 注册获取 AWS 账户
- 创建具有管理访问权限的用户
- 管理 Amazon 的用户权限 GameLift
- 为用户设置编程式访问权限
- 为游戏设置编程式访问权限
- Amazon GameLift 的 IAM 权限示例
- 为亚马逊设置 IAM 服务角色 GameLift

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 https://aws.amazon.com/ 并选择我的账户来查看当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的以根用户身份登录。
2. 为您的根用户启用多重身份验证 (MFA)。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的创建权限集。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的添加组。

管理 Amazon 的用户权限 GameLift

根据需要创建更多用户或将访问权限扩展到现有用户，以访问您的 Amazon GameLift 资源。作为最佳实操（IAM 中的安全最佳实操），请为所有用户应用最低权限。有关权限语法的指导，请参阅Amazon GameLift 的 IAM 权限示例。

根据您管理 AWS 账户中用户的方式，按照以下说明设置用户权限。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。
- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。
- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

在与 IAM 用户合作时，最佳实操是始终向角色或用户组授予权限，而不是向个人用户授予权限。

为用户设置编程式访问权限

如果用户想在 AWS 外部进行交互，则需要编程访问权限 AWS Management Console。授予编程访问权限的方式取决于正在访问的用户类型 AWS。

要向用户授予编程式访问权限，请选择以下选项之一。

哪个用户需要编程式访问权限？	目的	方式
人力身份 （在 IAM Identity Center 中管理的用户）	使用临时证书签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 • 有关的 AWS CLI，请参阅《AWS Command Line Interface 用户指南》 AWS IAM Identity Center中的 “配置 AWS CLI 要使用”。 • 有关 AWS 软件开发工具包、工具和 AWS API，请参阅《软件开发工具包和AWS工具参考指南》中的 IAM 身份中心身份验证。
IAM	使用临时证书签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照 IAM 用户指南中的将临时证书与 AWS 资源配合使用中的说明进行操作。
IAM	（不推荐使用） 使用长期凭证签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 • 有关信息 AWS CLI，请参阅用户指南中的使用 IAM 用户证书进行身份验证。AWS Command Line Interface • 有关 AWS SDK 和工具，请参阅 S AWS DK 和工具参

哪个用户需要编程式访问权限？	目的	方式
		考指南中的使用长期凭证进行身份验证。 • 有关 AWS API，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。

如果您使用访问密钥，请参阅管理 AWS 访问密钥的最佳实践。

为游戏设置编程式访问权限

大多数游戏使用后端服务通过 AWS 软件开发工具包与 Amazon GameLift 通信。例如，使用后端服务（代表游戏客户端操作）请求游戏会话、让玩家进入游戏以及执行其他任务。这些服务需要编程访问权限和安全凭证来验证对 Amazon GameLift 服务 API 的调用。

对于 Amazon GameLift，您可以通过在 AWS Identity and Access Management (IAM) 中创建玩家用户来管理此访问权限。通过下列选项之一管理玩家用户权限：

- 创建具有玩家用户权限的 IAM 角色，并允许玩家用户在需要时担任该角色。在向 Amazon 发出请求之前，后端服务必须包含代入此角色的代码 GameLift。根据安全最佳实操，角色提供有限的临时访问权限。您可以将角色用于在 AWS 资源上运行的工作负载（IAM 角色）或资源外部 AWS （IAM 角色随处可见）。
- 使用玩家用户权限创建 IAM 用户组，并将您的玩家用户添加到该组中。此选项为您的玩家用户提供长期凭证，后端服务在与 Amazon 通信时必须存储和使用这些证书 GameLift。

有关权限策略语法，请参阅玩家用户权限示例。

有关管理工作负载使用的权限的更多信息，请参阅 IAM 身份：IAM 中的临时凭证。

Amazon GameLift 的 IAM 权限示例

使用这些示例中的语法为需要访问 Amazon GameLift 资源的用户设置 AWS Identity and Access Management (IAM) 权限。有关管理用户权限的更多信息，请参阅管理 Amazon 的用户权限 GameLift。在管理 IAM Identity Center 以外的用户的权限时，最佳实操是始终向 IAM 角色或用户组授予权限，而不是向个人用户授予权限。

如果使用 Amazon GameLift FleetIQ 作为独立解决方案，请参阅为 Amazon GameLift FleetIQ 进行 AWS 账户设置。

管理员权限示例

这些示例为用户提供了管理 Amazon GameLift 游戏托管资源的完全访问权限。

Example Amazon GameLift 资源权限语法

以下示例扩展了对所有 Amazon GameLift 资源的访问权限。

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

Example Amazon GameLift 资源权限语法，支持默认未启用的区域

以下示例扩展了对默认未启用的所有 Amazon GameLift 资源 和 AWS 区域的访问权限。有关默认情况下未启用的区域以及如何启用这些区域的更多信息，请参阅《AWS 一般参考》中的管理 AWS 区域。

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

Example Amazon GameLift 资源和 **`PassRole`** 权限的语法

以下示例扩展了对所有 Amazon GameLift 资源的访问权限，并允许用户将 IAM 服务角色传递给 Amazon GameLift。服务角色使 Amazon GameLift 能够有限地代表您访问其他资源和服务，如为亚马逊设置 IAM 服务角色 GameLift中所述。例如，在响应 `CreateBuild` 请求时，Amazon GameLift 需

要访问您在 Amazon S3 存储桶中的构建文件。有关 PassRole 操作的更多信息，请参阅《IAM 用户指南》中的 IAM：将 IAM 角色传递给特定 AWS 服务。

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

玩家用户权限示例

这些示例允许后端服务或其他实体对 Amazon GameLift API 进行 API 调用。它们涵盖了管理游戏会话、玩家会话和对战的常见场景。有关更多信息，请参阅为游戏设置编程式访问权限。

Example 游戏会话置放权限的语法

以下示例扩展了对 Amazon GameLift API 的访问权限，这些 API 使用游戏会话放置队列来创建游戏会话和管理玩家会话。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "PlayerPermissionsForGameSessionPlacements",
    "Effect": "Allow",
    "Action": [
      "gamelift:StartGameSessionPlacement",
      "gamelift:DescribeGameSessionPlacement",
      "gamelift:StopGameSessionPlacement",
```

```
      "gamelift:CreatePlayerSession",
      "gamelift:CreatePlayerSessions",
      "gamelift:DescribeGameSessions"
    ],
    "Resource": "*"
  }
}
```

Example 对战权限的语法

以下示例扩展了对管理 FlexMatch 对战活动的 Amazon GameLift API 的访问权限。FlexMatch 为玩家匹配新游戏或现有游戏会话，并启动 Amazon GameLift 上托管的游戏的游戏会话放置。有关 FlexMatch 的更多信息，请参阅什么是 Amazon GameLift FlexMatch？

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

Example 手动游戏会话放置权限的语法

以下示例扩展了对 Amazon GameLift API 的访问权限，这些 API 可以在指定实例集上手动创建游戏会话和玩家会话。此场景支持不使用放置队列的游戏，例如允许玩家通过从可用游戏会话列表中进行选择来加入的游戏（“列表和选择”方法）。

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

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

为亚马逊设置 IAM 服务角色 GameLift

Amazon 的某些 GameLift 功能要求您对自己拥有的AWS资源进行有限访问。您可以通过创建 AWS Identity and Access Management (IAM) 角色来做到这一点。IAM 角色是可在账户中创建的一种具有特定权限的 IAM 身份。IAM 角色类似于 IAM 用户，因为它是一个 AWS 身份，具有确定其在 AWS 中可执行和不可执行的操作的权限策略。但是，角色旨在让需要它的任何人代入，而不是唯一地与某个人员关联。此外，角色没有关联的标准长期凭证（如密码或访问密钥）。相反，当您代入角色时，它会为您提供角色会话的临时安全凭证。

本主题介绍如何创建可用于 Amazon GameLift 托管车队的角色。如果您使用亚马逊 GameLift FleetiQ 来优化亚马逊弹性计算云 (Amazon EC2) 实例上的游戏托管，请参阅为亚马逊 FleetiQ 进行设置。AWS 账户 GameLift

在以下步骤中，使用自定义权限策略和允许 Amazon 担任该角色 GameLift 的信任策略创建一个角色。

创建自定义 IAM 角色

步骤 1：创建权限策略。

使用 JSON 策略编辑器创建策略

1. 登录AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧的导航窗格中，选择策略。

   如果这是您首次选择策略，则会显示 Welcome to Managed Policies 页面。选择开始使用。
3. 在页面的顶部，选择创建策略。
4. 在策略编辑器部分，选择 JSON 选项。

5. 输入或粘贴一个 JSON 策略文档。有关 IAM 策略语言的详细信息，请参阅 IAM JSON 策略参考。
6. 解决策略验证过程中生成的任何安全警告、错误或常规警告，然后选择下一步。

> **Note**
>
> 您可以随时在可视化和 JSON 编辑器选项卡之间切换。不过，如果您进行更改或在可视化编辑器中选择下一步，IAM 可能会调整策略结构以针对可视化编辑器进行优化。有关更多信息，请参阅《IAM 用户指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/troubleshoot_policies.html#troubleshoot_viseditor-restructure中的调整策略结构。

7. （可选）在 AWS Management Console 中创建或编辑策略时，您可以生成可在 AWS CloudFormation 模板中使用的 JSON 或 YAML 策略模板。

   为此，请在策略编辑器中选择操作，然后选择生成 CloudFormation模板。如需了解有关 AWS CloudFormation 的更多信息，请参阅《AWS CloudFormation 用户指南》中的 AWS Identity and Access Management 资源类型参考。
8. 向策略添加完权限后，选择下一步。
9. 在查看并创建页面上，为您要创建的策略键入策略名称和描述（可选）。查看此策略中定义的权限以查看策略授予的权限。
10. （可选）通过以密钥值对的形式附加标签来向策略添加元数据。有关在 IAM 中使用标签的更多信息，请参阅《IAM 用户指南》中的标记 IAM 资源。
11. 选择创建策略可保存您的新策略。

步骤 2：创建 Amazon GameLift 可以代入的角色。

1. 在 IAM 控制台的导航窗格中，选择角色，然后选择创建角色。
2. 在选择可信实体页面上，选择自定义信任策略选项。此选项将打开自定义信任策略编辑器。
3. 将默认 JSON 语法替换为以下语法，然后选择下一步继续。

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

```
      "Action": "sts:AssumeRole"
    }
  ]
}
```

4. 在添加权限页面上，找到并选择您在步骤 1 中创建的权限策略。选择下一步以继续。
5. 在名称、查看并创建页面上，为您要创建的角色输入角色名称和描述（可选）。查看信任实体和已添加权限。
6. 选择创建角色以保存您的新角色。

权限策略语法

- Amazon GameLift 担任服务角色的权限

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

- 访问默认情况下未启用的 AWS 区域的权限

```
{
  "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 为开发提供支持 GameLift

Amazon GameLift 提供了一组软件开发工具包，您可以将其与托管游戏托管解决方案配合使用。使用 Amazon GameLift SDK 向需要与亚马逊 GameLift 托管服务交互的多人游戏服务器、游戏客户端和游戏服务添加必要功能。

有关 Amazon GameLift SDK 版本和软件开发工具包兼容性的最新信息，请参阅Amazon GameLift 发行说明。

对于自定义游戏服务器

使用 Amazon 服务器 SDK 创建和部署 64 位自定义游戏 GameLift 服务器。与服务器 SDK 集成并部署用于托管的游戏服务器可以与 Amazon GameLift 服务通信，以启动和管理游戏会话。有关集成服务器软件开发工具包的信息，请参阅为亚马逊准备游戏 GameLift中的主题。

开发操作系统

- Windows
- Linux

支持的编程语言

Amazon GameLift 提供以下语言的服务器软件开发工具包。 下载服务器 SDK。有关特定版本的详细信息，请参阅每个软件包中包含的自述文件。

- C++软件开发工具包
  - 软件开发工具包参考
  - 软件开发工具包集成
- C# 服务器软件开发工具包（版本可能支持.NET 4 和.NET 6）
  - 软件开发工具包参考

  - 软件开发工具包集成
- Go
  - 软件开发工具包参考
  - 软件开发工具包集成

支持的游戏引擎

在任何支持 C++、C# 或 Go 库的引擎中使用特定语言的软件开发工具包。此外，亚马逊还 GameLift 提供以下游戏引擎插件：下载亚马逊 GameLift 插件

- Unity
  - 适用于 Unity 的 C# 服务器软件开发工具包 插件是一个带有预建库的轻量级插件，您可以使用 Unity 包管理器进行安装。将此插件与以下 Unity 版本一起使用：适用于 Windows 和 Mac OS 的 2020.3 LTS、2021.3 LTS 和 2022.3 LTS。它支持 Unity 的 .NET 框架和 .NET 标准配置文件，以及 .NET 标准 2.1 和 .NET 4.x。
    - 将 Amazon GameLift 集成到 Unity 项目中
  - 适用于 Unity 2021.3 LTS 和 2022. 3 LTS 的独立插件是一款功能齐全的插件，包含专为 Unity 构建的 C# SDK 库和用于配置和部署用于托管的亚马逊资源的 GUI 元素。 GameLift
    - 适用于 Unity 的亚马逊 GameLift 插件服务器指南 SDK 5.x
    - 适用于 C# 的 Amazon GameLift 服务器软件开发工具包 参考
- Unreal Engine
  - Unreal Engine 的C++服务器SDK插件是一个由C++虚幻源代码组成的轻量级插件，您可以将其构建到库中，用于 Unreal Engine 版本4、5和5.1。
    - 将 Amazon GameLift 集成到虚幻引擎项目中
    - Amazon GameLift Unreal Engine 服务器软件开发工具包 5.x 参考
  - 虚幻引擎5.0、5.1和5.2的独立插件是一个功能齐全的插件，包含适用于虚幻服务器的C++ SDK库和S AWS DK。该插件安装在虚幻编辑器中，包含用于配置和部署用于托管的Amazon GameLift资源的用户界面元素和支持材料。
    - 将游戏与适用于虚幻引擎的 Amazon GameLift 插件集成
    - Amazon GameLift Unreal Engine 服务器软件开发工具包 5.x 参考

游戏服务器操作系统

使用 Amazon S GameLift erver SDK 构建可在以下平台上运行的游戏服务器：

- Windows Server 2016
- Amazon Linux 2023
- Amazon Linux 2 (AL2)
- Windows Server 2012（参见 Windows 2012 版亚马逊 GameLift 常见问题解答）
- 亚马逊 Linux (AL1)（参见 AL1 的亚马逊 GameLift 常见问题解答）

用于自定义客户端服务

使用 AWS 软件开发工具包和 Amazon GameLift API 创建 64 位自定义客户端服务。此 SDK 使客户服务能够管理游戏会话，让玩家加入亚马逊上托管的游戏 GameLift。要开始使用，请下载 S AWS DK。有关在亚马逊上使用软件开发工具包的更多信息 GameLift，请参阅亚马逊 GameLift API 参考。

对于实时服务器

配置和部署实时服务器来托管您的多人游戏。要允许您的游戏客户端连接到实时服务器，请使用 Amazon GameLift 实时客户端 SDK。游戏客户端使用此开发工具包与 服务器以及连接到该服务器的其他游戏客户端交换消息。要开始使用，请下载 Amazon GameLift 实时客户端 SDK。有关详细的配置信息，请参阅为实时服务器集成游戏客户端。

SDK 支持

客户端开发工具包包含以下语言来源：

- C# (.NET)

开发环境

根据这些支持的开发操作系统和游戏引擎的需求，从源构建开发工具包。

- 操作系统 – Windows、Linux、Android、iOS。
- 游戏引擎 – Unity，支持 C# 库的引擎

游戏服务器操作系统

服务器部署到运行以下平台的托管资源：

- Amazon Linux
- Amazon Linux 2

管理您的游戏托管成本

您的 AWS 账单反映了您的游戏托管成本。您可以在账单控制台上查看当月的预计费用以及前几个月的最终费用，网址为 https://console.aws.amazon.com/billing/。要了解有关可帮助管理 AWS 成本的工具和资源的更多信息，请参阅 AWS Billing 用户指南。该指南有助于审核您的资源开销、决定未来用途，并确定扩展需求。

请特别考虑这些小贴士，以帮助您管理Amazon GameLift 服务的成本。

创建账单提醒以监控使用情况

设置AWS免费套餐使用提醒，以便在您的使用量接近或超过 Amazon GameLift 和其他AWS 服务公司的免费套餐限制时通知您。您可以将警报配置为根据自己的使用水平采取行动。例如，当您的预算达到免费套餐限制时，您可以自动将预算设置为零。

您还可以设置 Amazon CloudWatch 账单提醒，以便在使用量达到自定义阈值时收到通知。

有关更多信息，请参阅《AWS Billing 用户指南》中的以下主题：

- 跟踪您的 AWS 免费套餐使用情况
- 账单提醒偏好

追踪每个 Amazon GameLift 车队的成本

使用AWS成本分配标签，根据 GameLift 亚马逊 Amazon EC2 队列和其他 EC2 资源来组织和跟踪您的游戏托管成本。通过单独或按组标记实例集，您可以创建成本分配报告，根据分配的标签对成本进行分类。您可以使用此类报告来确定实例集如何影响您的托管成本。您还可以使用标签筛选 AWS Cost Explorer 中的视图。

有关更多信息，请参阅以下主题：

- 使用 AWS 成本分配标签，《AWS Billing 用户指南》
- 使用 AWS Cost Explorer 分析成本，《AWS Cost Management 用户指南》

将未使用的实例集容量设置为零

即使不使用实例集托管游戏会话，它们也可能继续产生成本。为避免导致不必要的费用，您可能希望在不使用时缩减实例集。如果您使用自动扩缩，请暂停此活动并手动设置实例集容量。

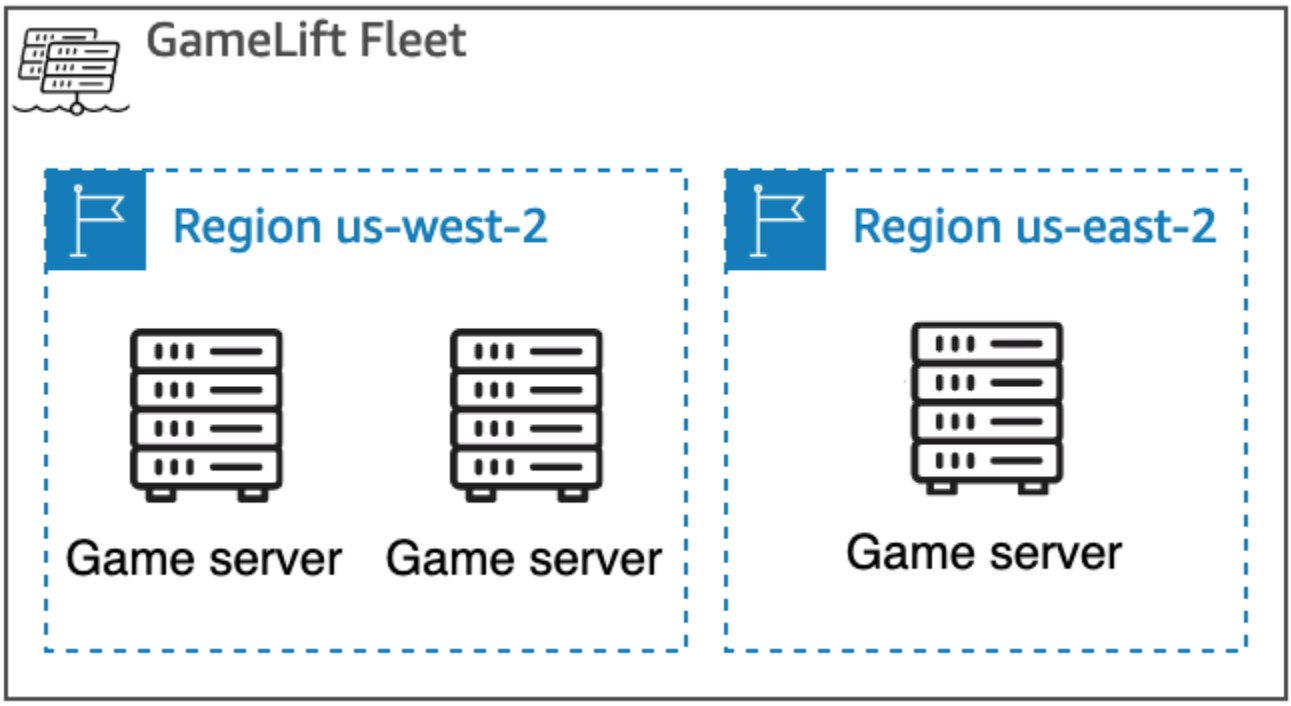
亚马逊 GameLift 托管地点

亚马逊 GameLift 在多个区域AWS 区域和 Local Zones 中可用。有关地点的完整列表，请参阅中的 Amazon GameLift 终端节点和配额AWS 一般参考。

亚马逊 GameLift 托管

当您创建 Amazon GameLift 队列时，Amazon GameLift 会在您的当前队列中创建该队列的资源AWS 区域。Amazon 将该地区 GameLift 称为机队的主产区。要管理实例集，请从其主区域访问该实例集。

多位置实例集将实例部署到实例集主地区以外的其他位置。使用多地点舰队，您可以单独管理每个地点的容量，也可以按地点安排游戏会话。 多地点舰队可以在 Ama GameLift zon 支持的任何区域或本地区域内设有远程位置。下图描绘了多位置实例集，其资源分布在两个区域。在图中，该 us-west-2 区域包括两台游戏服务器，而 us-east-2 区域有一台游戏服务器。



如果您选择在默认情况下未启用的区域中使用多位置实例集，则必须在您的 AWS 账户中启用这些区域。此外，您的 Amazon GameLift 管理员策略必须允许该ec2:DescribeRegions操作。有关默认情况下未启用的区域以及如何启用这些区域的更多信息，请参阅《AWS 一般参考》中的管理 AWS 区域。有关原定设置情况下未启用的区域政策示例，请参阅管理员权限示例。

> **Important**
>
> 要使用默认情况下未启用的区域，请在您的 AWS 账户中将其启用。
>
> - 如果您在 2022 年 2 月 28 日之前创建的未启用区域的实例集不受影响。
> - 要创建新的多位置实例集或更新现有的多位置实例集，请先启用您选择使用的任何区域。

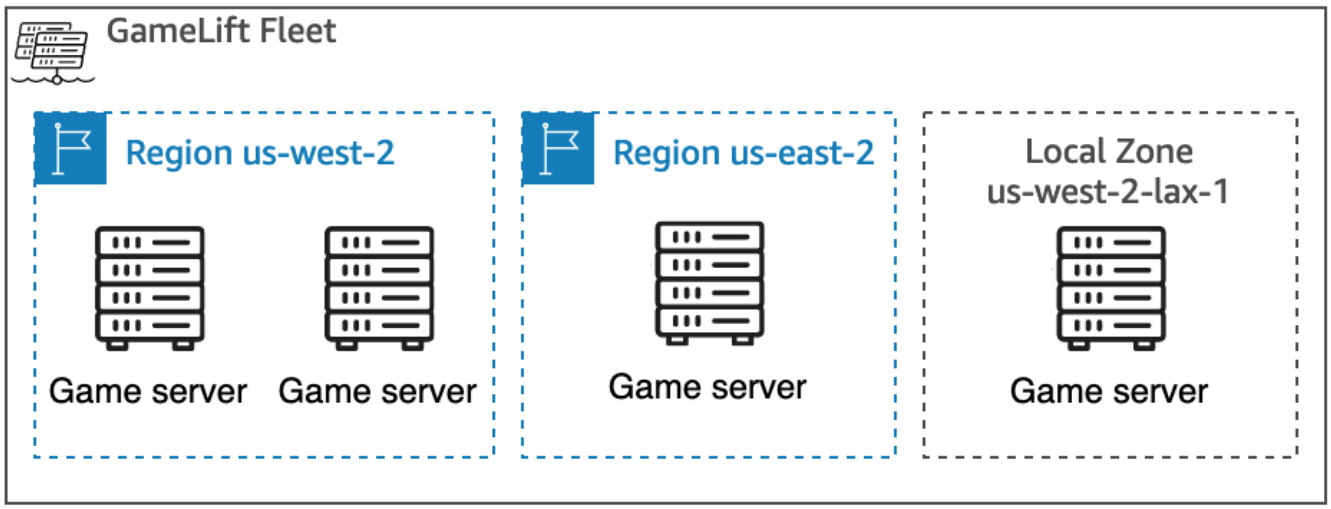
对于游戏会话放置，您可以在 Amazon GameLift 支持的任何位置创建游戏会话队列。Amazon 从您创建队列的位置放 GameLift 置游戏会话。

Local Zones

本地区域是在地理上靠近您的用户的 AWS 区域的扩展。本地区域有自己的互联网连接。本地区域还支持 AWS Direct Connect，以便在本地区域中创建的资源可以通过低延迟通信为本地用户提供服务。有关更多信息，请参阅 AWS Local Zones。

本地区域的代码由其区域代码后跟一个指示其实际位置的标识符组成。例如，`us-west-2-lax-1` 本地区域位于洛杉矶。有关可用的本地区域列表，请参阅可用 Local Zones。

Amazon 将您的游戏 GameLift 托管在您为舰队选择的每个地点。下图描绘了一个实例集，在 `us-west-2` 区域有两台游戏服务器，一台游戏服务器位于 `us-east-2` 区域，一台游戏服务器位于 `us-west-2-lax-1` 本地区域。



可用 Local Zones

下表列出了可用的本地区域及其物理位置。

本地区域	位置（都会区）	
us-east-1-atl-1	亚特兰大	
us-east-1-chi-1	芝加哥	
us-east-1-dfw-1	达拉斯	
us-east-1-iah-1	休斯顿	
us-east-1-mci-1	堪萨斯城	
us-west-2-den-1	丹佛	
us-west-2-lax-1	洛杉矶	
us-west-2-phx-1	Phoenix	

Amazon GameLift Anywhere

您可以使用 Amazon GameLift Anywhere 使用自己的硬件创建舰队，并使用 Ama GameLift zon 管理您的游戏构建、脚本、游戏服务器和客户端。亚马逊 GameLift Anywhere在亚马逊 GameLift 支持的所有地区都可用。有关创建 Anywhere 实例集和测试游戏服务器集成的更多信息，请参阅创建亚马逊 GameLift Anywhere舰队和使用 Amazon GameLift Anywhere 实例集测试您的集成。

通过亚马逊， GameLift Anywhere您可以创建自定义位置，这些位置代表您用来托管亚马逊 GameLift 集成游戏服务器的硬件的物理位置。

Amazon GameLift FlexMatch

对于 FlexMatch，您可以在 Ama GameLift zon 支持的任何位置举办比赛生成的游戏会话。实际的配对活动发生在您选择创建媒人资源AWS 区域的地方. Amazon将匹配请求 GameLift 路由到媒人，并在该位置进行处理。有关亚马逊的更多信息 GameLift FlexMatch，请参阅什么是亚马逊 GameLift FlexMatch？

AWS 区域这些支持 FlexMatch 资源

亚马逊 GameLift 在中国

使用亚马逊 GameLift 获取由光环新网运营的中国（北京）地区或由西云数据运营的中国（宁夏）地区的资源时，您必须拥有单独的AWS（中国）账户。请注意，某些特征在中国区域中不可用。有关 GameLift 在这些地区使用 Amazon 的更多信息，请参阅以下资源：

- 亚马逊云科技中国
- 亚马逊 GameLift（中国亚马逊 Web Services 入门）

Amazon VPC 入门

我们建议您在使用 Amazon GameLift 制作自己的游戏之前，先尝试以下示例。自定义游戏服务器示例为您提供了在 Amazon GameLift 主机中托管游戏的体验。实时服务器示例向您展示了如何使用实时服务器为托管游戏做好准备。

要开始为自己的游戏使用 Amazon GameLift，请参阅Amazon GameLift 托管托管资源路线图。

自定义游戏服务器示例

此示例演示了 Amazon GameLift 上的一款直播定制游戏。该示例将引导您完成以下步骤：

- 创建游戏构建。
- 创建实例集来运行游戏服务器。
- 从示例游戏客户端连接到游戏服务器。
- 查看实例集和游戏会话指标。

您可以启动多个游戏客户端并玩游戏来生成托管数据。一旦您得到一些数据，探索 控制台以查看您的托管资源、跟踪指标并体验扩展托管容量的方法。

要查看现有任务，请登录 Amazon GameLift 控制台。

实时服务器示例游戏

此示例是一款名为 Mega Frog Race 的完整多人游戏，包含源代码。该示例说明如何将您的游戏客户端与实时服务器集成。您也可以使用此示例游戏作为起点，尝试其他 Amazon GameLift 功能，例如 FlexMatch。

要获取实践教程，请参阅 Game 博客文章只用几行 JavaScript 就能创建多人移动游戏服务器。

有关 Mega Frog Race 的源代码，请参阅 GitHub 存储库。

源代码包含以下几个部分：

- 游戏客户端 Unity 创建的 C++ 游戏客户端的源代码。游戏客户端获取游戏会话连接信息，连接到服务器，并与其他玩家交换更新。
- 后端服务 – 管理对 Amazon GameLift 的直接 API 调用的 AWS Lambda 函数的源代码。

- 实时脚本-为游戏配置实时服务器队列的源脚本文件。该脚本包括实时服务器与 Amazon GameLift 通信和托管游戏所需的最低配置。

Amazon GameLift 托管托管资源路线图

本主题可帮助您为基于会话的多人游戏从不同的 Amazon GameLift 托管选项中进行选择。本节的其余主题将引导您了解如何针对托管托管资源使用 Amazon GameLift。

在开始准备将游戏投入生产之前，请填写发布问卷，开始与 Amazon GameLift 团队合作。

主题

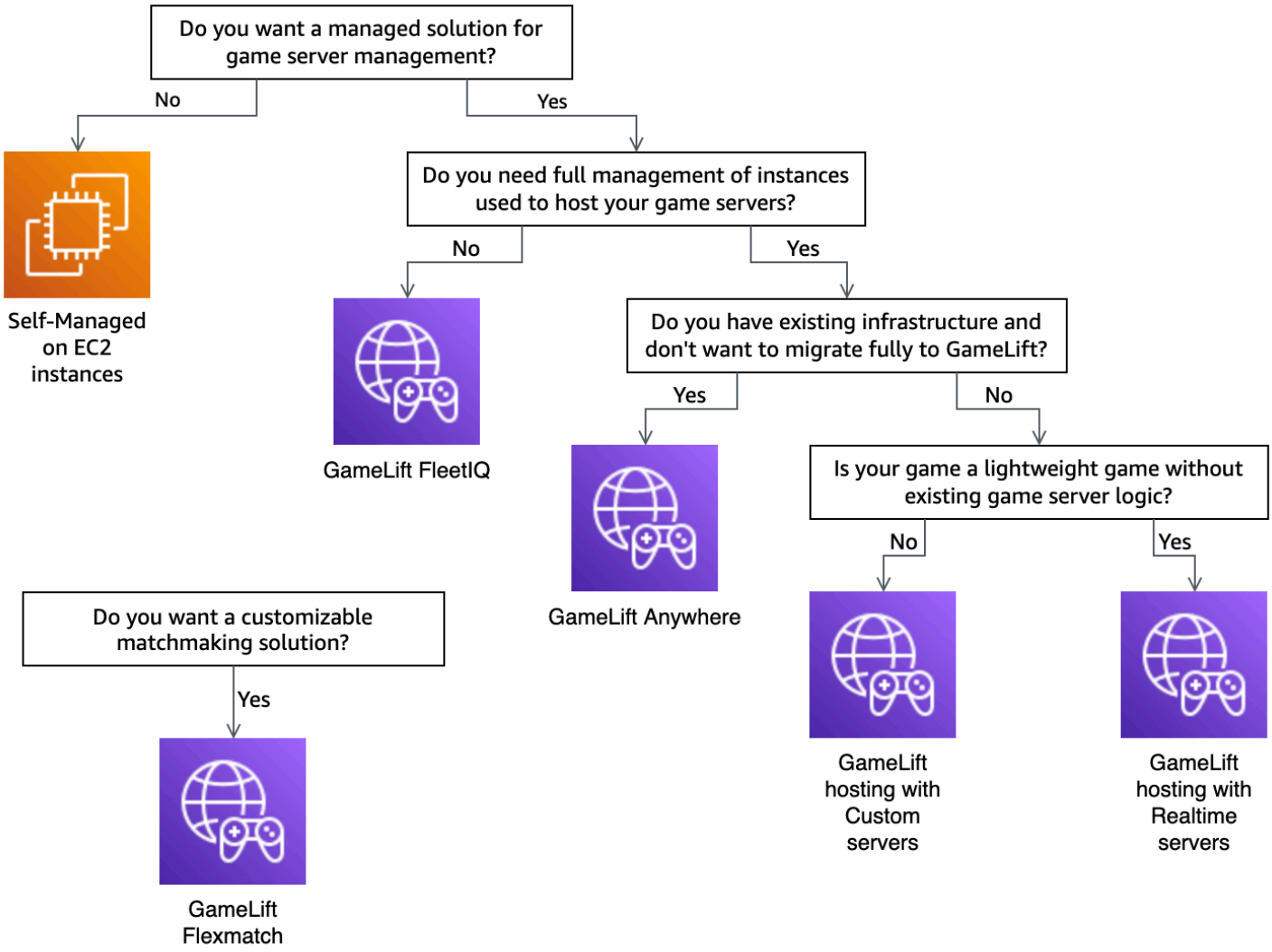
- 选择托管选项
- 让游戏为 Amazon GameLift 做准备
- 测试您与 Amazon GameLift 的集成
- 规划和部署您的 Amazon GameLift 资源
- 设计您的游戏客户端服务
- 为 Amazon GameLift 设置指标和日志记录
- 游戏启动清单

选择托管选项

以下流程图会询问一些问题，引导您找到适合您用例的正确 Amazon GameLift 解决方案。

1. 您想要游戏服务器管理的托管解决方案吗？
   - 是 – 继续执行第二步。
   - 否 – 可以考虑在 Amazon EC2 实例上自行管理游戏服务器。
2. 您是否需要完全控制托管游戏服务器的实例？
   - 是 – 可以考虑 Amazon GameLift FleetIQ。
   - 否 – 继续执行步骤 3。
3. 您是否有想要在 Amazon GameLift 上使用的现有基础设施？
   - 是 – 可以考虑 Amazon GameLift Anywhere。
   - 否 – 继续执行第四步。
4. 如果没有现有的游戏服务器逻辑，您的游戏是轻量级的吗？
   - 是 – 考虑使用实时服务器。

- 否 – 考虑使用自定义服务器。



以下是有关流程图中提到的一些 Amazon GameLift 托管选项的更多信息：

Amazon GameLift Anywhere

借助 Amazon GameLift Anywhere 在自己的硬件上托管游戏，尽享 Amazon GameLift 管理工具的优势。您也可以使用 Anywhere 实例集来迭代测试游戏服务器。有关更多信息，请参阅[创建亚马逊 GameLift Anywhere舰队]。

托管 Amazon GameLift

托管 Amazon GameLift 托管有两种选择：

自定义服务器 – Amazon GameLift 托管您的自定义服务器，该服务器运行您的游戏服务器二进制文件。

实时服务器 – Amazon GameLift 托管您的轻量级游戏服务器。

Amazon GameLift FleetIQ

在流程图中，直接迁移是指当您无法对游戏架构进行更改时的迁移。使用 Amazon GameLift FleetIQ 需要对现有部署进行更少的更改，并提供用于实例集管理的 Amazon GameLift 工具。有关更多信息，请参阅 GameLift 开发人员指南。

如果您决定使用 Amazon GameLift Anywhere 或托管 Amazon GameLift，请继续让游戏为 Amazon GameLift 做准备。

让游戏为 Amazon GameLift 做准备

本主题介绍准备多人游戏以与托管 Amazon GameLift 托管资源集成的步骤。要准备游戏，您必须激活游戏与 Amazon GameLift 之间的通信。

准备自定义游戏服务器

要开启或关闭游戏会话，并执行其他任务，游戏服务器必须能够通知 Amazon GameLift 其状态。要激活与 Amazon GameLift 的通信，请向您的游戏服务器项目添加代码。有关更多信息，请参阅将游戏与自定义游戏服务器集成。

1. 准备自定义游戏服务器以在 Amazon GameLift 上进行托管。
   - 获取 Amazon GameLift 服务器软件开发工具包并针对您的首选编程语言和游戏引擎进行构建。
   - 向您的游戏服务器项目添加代码以激活与 Amazon GameLift 的通信。
2. 让游戏客户端准备好连接到 Amazon GameLift 托管的游戏会话。
   - 将 AWS 软件开发工具包添加到后端服务和游戏客户端项目。有关更多信息，请参阅下载适用于客户服务的 Amazon GameLift 软件开发工具包。
   - 添加在游戏会话中检索信息、放置新游戏会话和在游戏会话中为玩家预留空间的功能。
   - （可选）使用 FlexMatch 进行玩家对战。有关更多信息，请参阅将 FlexMatch 与 Amazon GameLift 托管集成。

准备实时服务器

Amazon GameLift 实时服务器提供轻量级服务器解决方案，可以进行配置来适合您的游戏。实时服务器提供的优势与 Amazon GameLift 为游戏服务器提供的优势相同，但游戏服务器的可定制性会降低。

创建用于在 Amazon GameLift 上托管的实时脚本。

实时脚本包含您的服务器配置和可选的自定义游戏逻辑。实时服务器旨在启动和停止游戏会话、接受玩家连接、管理与 Amazon GameLift 的通信以及游戏中玩家之间的通信。您还可以使用钩子为游戏添加自定义服务器逻辑。实时服务器使用 Node.js 和 JavaScript。有关更多信息，请参阅 创建实时脚本 和 测试您与 Amazon GameLift 的集成：

测试您与 Amazon GameLift 的集成

在测试游戏服务器时，Amazon GameLift 支持快速迭代。本主题将指导您了解可用的测试类型。

自定义游戏服务器

使用 Amazon GameLift 将环境中任何地方的硬件集成到 Amazon GameLift 游戏托管架构中。Amazon GameLift Anywhere 将您的硬件注册到 Anywhere 实例集中的 Amazon GameLift，这样您就可以使用自己的本地开发计算机进行测试。有关使用 Amazon GameLift Anywhere 进行测试的更多信息，请参阅使用 Amazon GameLift Anywhere 实例集测试您的集成。有关使用 Amazon GameLift Anywhere 通过本地解决方案托管游戏的更多信息，请参阅选择 Amazon GameLift 计算资源。

实时服务器

使用实时服务器，您可以随时更新脚本。当您更新实时脚本时，Amazon GameLift 会在几分钟内将新版本分发到您的托管资源中。在 Amazon GameLift 部署新脚本后，所有新的游戏会话都使用新的脚本版本。在 Amazon GameLift 部署新脚本后，您可以立即开始测试。有关此服务器的更多信息，请参阅将游戏与 Amazon GameLift 实时服务器集成。

规划和部署您的 Amazon GameLift 资源

可以使用以下提示帮助规划您的全球 Amazon GameLift 资源部署。有关您可以通过 Amazon GameLift 在哪里托管游戏的信息，请参阅亚马逊 GameLift 托管地点。

要部署您的 Amazon GameLift 资源，请完成以下任务：

- 将您的游戏服务器打包并上传到 Amazon GameLift 或您的硬件。将服务器上传到 Amazon GameLift 时，您只能将其上传到实例集主 AWS 区域。Amazon GameLift 会自动将服务器分发到实例集中的其他位置。有关更多信息，请参阅将构建和脚本上传到 Amazon GameLift。

- 为您的游戏设计和部署 Amazon GameLift 实例集。决定要使用的计算资源类型、要部署到的区域、是否使用队列以及其他选项。有关更多信息，请参阅Amazon GameLift 实例集设计指南。
- 创建队列来管理新的游戏会话放置和竞价型实例策略。有关更多信息，请参阅设计游戏会话队列。
- 启用自动扩缩以管理实例集的托管容量来满足预期的玩家需求。有关更多信息，请参阅扩展 Amazon GameLift 托管容量。
- 在游戏中使用 FlexMatch 对战规则。有关更多信息，请参阅将 FlexMatch 与 Amazon GameLift 托管集成。

自动部署您的 Amazon GameLift 资源

为了简化 Amazon GameLift 资源的全球部署，我们建议您使用基础设施即代码 (IaC) 来定义资源。由于 Amazon GameLift 支持 AWS CloudFormation 模板，因此您可以在模板中为任何特定于部署的配置设置参数。

为了管理 AWS CloudFormation 堆栈的部署，我们还建议使用持续交付 (CI/CD) 工具和服务，例如 AWS CodePipeline。它们可以帮助您在构建游戏服务器二进制文件时自动部署或在获得批准的情况下进行部署。

以下是针对新游戏服务器版本部署 Amazon GameLift 资源的一些常见步骤，您可以使用 CI/CD 工具或服务自动执行该版本：

- 构建和测试您的游戏服务器二进制文件。
- 将二进制文件上传到 Amazon GameLift 或您的硬件。
- 在新构建中部署新的实例集。
- 部署新实例集后，从 Amazon GameLift 队列中移除先前版本的实例集，然后用新的实例集取而代之。
- 在先前版本之后，实例集成功结束所有游戏会话，删除这些实例集的 AWS CloudFormation 堆栈。

也可以使用 AWS Cloud Development Kit (AWS CDK) 定义您的Amazon GameLift 资源。有关 AWS CDK 的更多信息，请参阅 AWS Cloud Development Kit (AWS CDK) 开发人员指南。

设计您的游戏客户端服务

我们建议您实施游戏客户端服务，对您的玩家进行身份验证并与 Amazon GameLift API 进行通信。通过实现自定义游戏客户端服务，您可以：

- 自定义玩家身份验证。
- 控制 Amazon GameLift 如何匹配和启动游戏会话。
- 使用您的玩家数据库获取玩家属性，例如用于对战的技能等级，而不是信任客户端。

使用游戏客户端服务还可以降低游戏客户端直接与您的 Amazon GameLift API 交互所带来的安全风险。

对您的玩家进行身份验证

您可以使用 Amazon Cognito 和玩家会话 ID 对您的游戏客户端进行身份验证。要管理玩家身份的生命周期和属性，请使用 Amazon Cognito 用户群体。

如果您愿意，可以构建自定义身份解决方案并将其托管在 AWS。您还可以使用 Lambda 授权方通过 API Gateway 进行自定义授权逻辑。

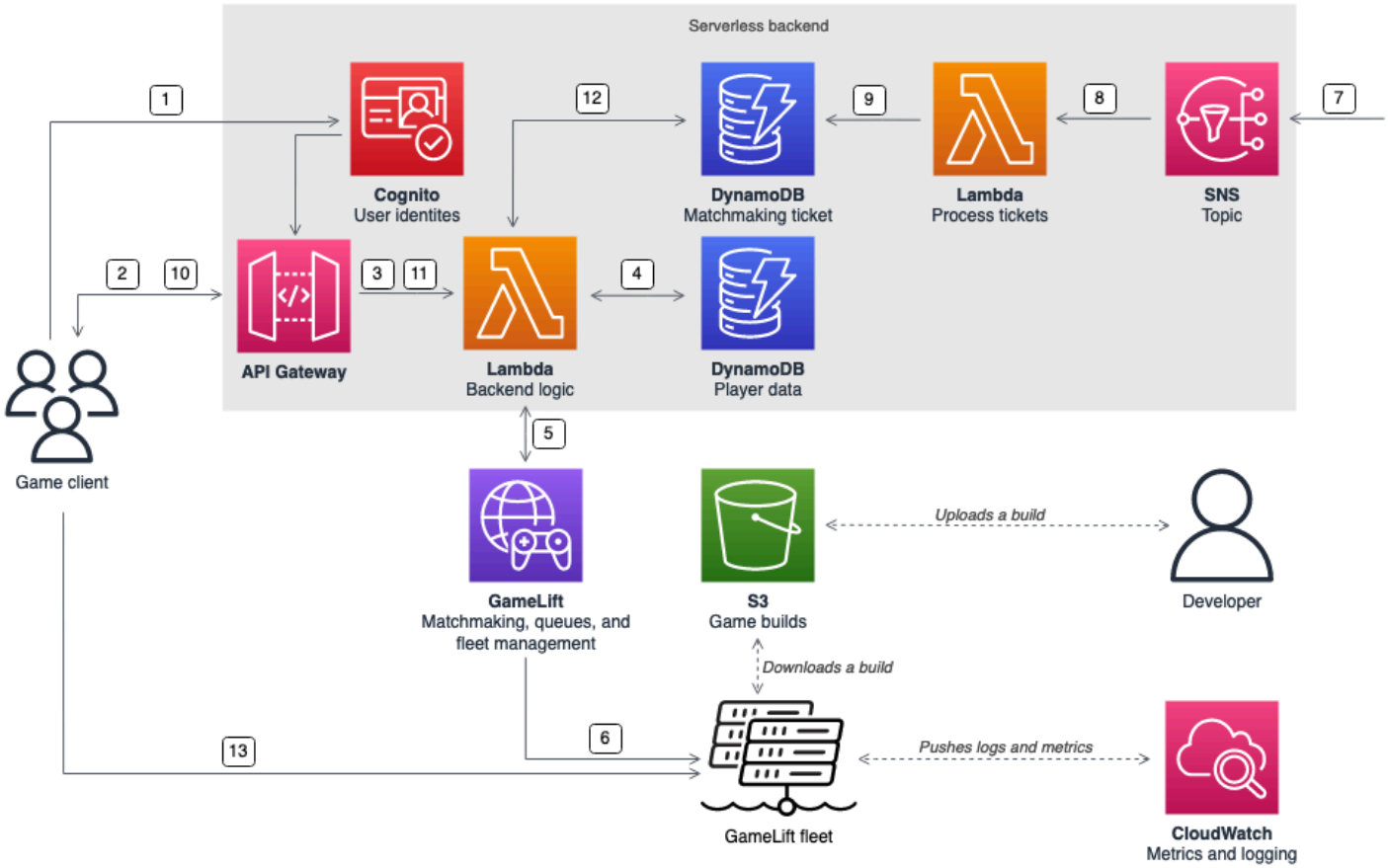
其他资源

- 使用身份池（联合身份）（Amazon Cognito 开发人员指南）
- 用户群体入门（Amazon Cognito 开发人员指南）
- 如何使用 Amazon Cognito 设置玩家身份验证（适用于游戏博客的 AWS）

带有无服务器后端的独立游戏会话服务器

使用无服务器客户端服务架构，后端可以从高度可扩展的数据库中查看对战票证的状态，而不必直接访问Amazon GameLift API。

下图显示了通过 AWS 服务 构建的无服务器后端，该后端可将玩家匹配到在 Amazon GameLift 实例集上运行的游戏。下表提供了对每个带编号的注解的说明。要尝试此示例，请参阅 GitHub 上基于多人会话的 AWS 上的游戏托管。



1. 游戏客户端从 Amazon Cognito 身份池中请求 Amazon Cognito 用户身份。
2. 游戏客户端接收临时访问凭证，并通过 Amazon API Gateway API 请求游戏会话。
3. API Gateway 调用一个 AWS Lambda 函数。
4. Lambda 函数从 Amazon DynamoDB NoSQL 表中请求玩家数据。该函数在请求上下文数据中提供 Amazon Cognito 身份。
5. Lambda 函数通过 Amazon GameLift FlexMatch 对战请求匹配。
6. FlexMatch 匹配一组具有适当延迟的玩家，然后通过 Amazon GameLift 队列请求游戏会话放置。队列中有包含一个或多个 AWS 区域位置的实例集。
7. Amazon GameLift 将会话放到其中一个实例集的位置后，Amazon GameLift 向 Amazon Simple Notification Service (Amazon SNS) 主题发送一个事件通知。
8. Lambda 函数接收并处理 Amazon SNS 事件。
9. 如果对战票证是一个 `MatchmakingSucceeded` 事件，那么 Lambda 函数会将结果以及游戏服务器的端口和 IP 地址写入 DynamoDB 表。
10. 游戏客户端向 API Gateway 发出签名请求，以查看特定时间间隔内的对战票证状态。
11. API Gateway 使用 Lambda 函数来检查对战票证状态。

12Lambda 函数会检查 DynamoDB 表以查看票证是否成功。如果成功，该函数会将游戏服务器的端口和 IP 地址以及玩家会话 ID 发送回客户端。如果票证未成功，则该函数会发送响应，确认匹配尚未准备就绪。

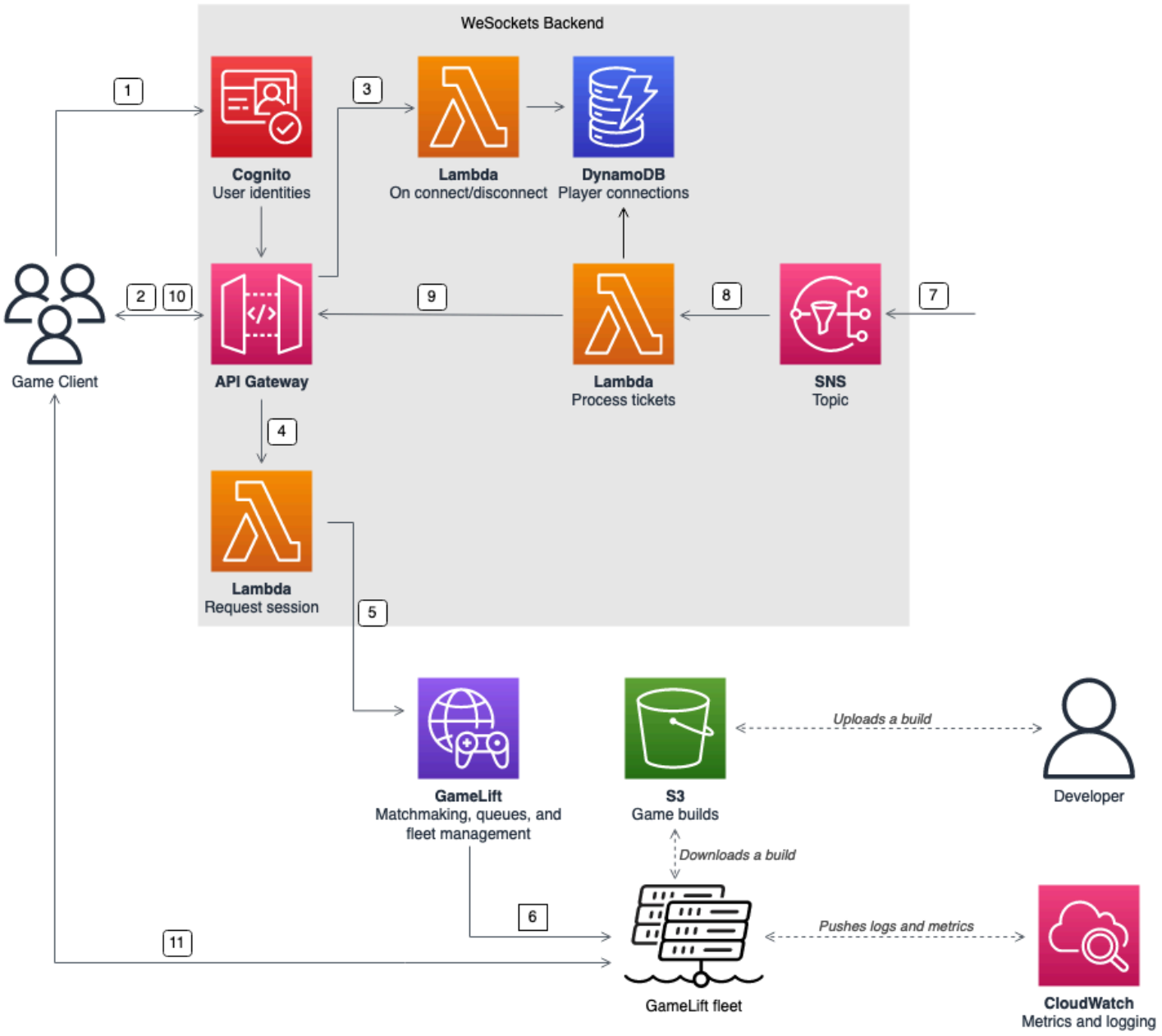
13游戏客户端使用后端服务提供的端口和 IP 地址，使用 TCP 或 UDP 连接到游戏服务器。然后，游戏客户端将玩家会话 ID 发送到游戏服务器，然后游戏服务器使用 Amazon GameLift 服务器软件开发工具包验证该 ID。

带有基于 WebSocket 的后端的独立游戏会话服务器

使用基于 Amazon API Gateway WebSocket 的架构，您可以使用 WebSocket 发出对战请求，并使用服务器启动的消息发送对战完成的推送通知。此架构通过在客户端和服务器之间进行双向通信来提高性能。

有关使用 API Gateway WebSock API 的更多信息，请参阅使用 WebSocket API。

下图显示了基于 WebSocket 的后端架构，该架构使用 API Gateway 和其他 AWS 服务 将玩家匹配到在 Amazon GameLift 实例集上运行的游戏。下表提供了对每个带编号的注解的说明。



1. 游戏客户端从 Amazon Cognito 身份池中请求 Amazon Cognito 用户身份。
2. 游戏客户端使用 Amazon Cognito 凭证签署与 API Gateway API 的 WebSocket 连接。
3. API Gateway 在连接上调用 AWS Lambda 函数。该函数将连接信息存储在 Amazon DynamoDB 表中。
4. 游戏客户端通过 WebSocket 连接经 API Gateway API 向 Lambda 函数发送一条消息，请求会话。
5. Lambda 函数通过 Amazon GameLift FlexMatch 对战接收消息并请求匹配。
6. 在 FlexMatch 匹配一组玩家后，FlexMatch 通过 Amazon GameLift 队列请求游戏会话放置。

7. Amazon GameLift 将会话放到其中一个实例集的位置后，Amazon GameLift 向 Amazon Simple Notification Service (Amazon SNS) 主题发送一个事件通知。
8. Lambda 函数接收并处理 Amazon SNS 事件。
9. 如果对战票证是 `MatchmakingSucceeded` 事件，那么 Lambda 函数会向 DynamoDB 请求正确的玩家连接。该函数随后通过 WebSocket 连接经 API Gateway API 向游戏客户端发送消息。在这种架构中，游戏客户端不会主动轮询对战状态。
10. 游戏客户端通过 WebSocket 连接接收游戏服务器的端口和 IP 地址以及玩家会话 ID。
11. 游戏客户端使用后端服务提供的端口和 IP 地址，使用 TCP 或 UDP 连接到游戏服务器。游戏客户端还将玩家会话 ID 发送到游戏服务器，然后游戏服务器使用 Amazon GameLift 服务器软件开发工具包验证该 ID。

为 Amazon GameLift 设置指标和日志记录

您可以使用从 Amazon GameLift 游戏服务器和资源中收集的数据来帮助识别异常情况。您还可使用指标帮助提高性能。

Amazon GameLift 需要注意的关键领域包括：

- Amazon GameLift 服务指标 – Amazon GameLift 提供有关您的资源的 Amazon CloudWatch 指标，包括游戏服务器、实例集、队列和 FlexMatch。您可以在 Amazon GameLift 控制台和 CloudWatch 控制台中找到这些指标。有关 Amazon CloudWatch 指标的更多信息，请参阅使用 Amazon CloudWatch 监控 Amazon GameLift。
- 游戏服务器指标 – Amazon GameLift 无法访问您的游戏服务器指标。但是，您可以使用 CloudWatch 代理直接从游戏服务器向 CloudWatch 发送自定义指标。您还可以使用队列 AWS Identity and Access Management (IAM) 角色和 AWS 软件开发工具包将指标直接发送到 CloudWatch。有关如何配置指标的示例，请参阅 GitHub 上 AWS 上托管的基于多人游戏会话的游戏。此存储库包含 CloudWatch 代理配置示例，以及 C# StatSD 客户端的代码。
- 游戏服务器日志 – 要在游戏服务器上配置游戏服务器日志文件，请使用 Amazon GameLift 服务器软件开发工具包配置。您还可以将 Amazon CloudWatch Logs 用作实时日志管理解决方案，也可以使用 CloudWatch 代理配置日志。有关更多信息，请参阅在 Amazon GameLift 中记录服务器消息。

游戏启动清单

您可以使用这些清单来验证游戏的部署阶段。在清单中，标有 [关键] 的项目对于您的产品发布至关重要。

主题

- 激活
- 测试
- 启动
- 启动后

激活

使用以下清单来追踪在 Amazon GameLift 托管激活游戏所需的项目。标有 [关键] 的项目对于您的产品发布至关重要。

- [关键] 在 Amazon GameLift 控制台上填写 Amazon GameLift 入门问卷。
- [关键] 设计和实现游戏客户端与游戏服务器交互的后端服务。
- [关键] 创建您提供给 Amazon GameLift 服务器实例以访问其他 AWS 资源的 AWS Identity and Access Management (IAM) 角色。
- [关键] 为 FlexMatch 和队列设计并实施到其他 AWS 区域 的失效转移。
- 考虑到游戏的队列和实例集结构，计划将实例集部署到目标位置。
- 使用基础设施即代码 (IaC) 以及 AWS CloudFormation 和 AWS Cloud Development Kit (AWS CDK) 实现部署自动化。
- 使用 Amazon CloudWatch 和 Amazon Simple Storage Service (Amazon S3) 收集日志和分析。

测试

在使用 Amazon GameLift 托管开发游戏时，使用以下清单来跟踪测试项目。标有 [关键] 的项目对于您的产品发布至关重要。

- [关键] 填写发布问卷，并将填好的问卷提交给 Amazon GameLift 发布团队。可以在 Amazon GameLift 控制台上找到发布问卷。
- [关键] 请求增加 Amazon GameLift 服务限额和其他 AWS 服务 限额，以便您的实时环境可以扩展以满足生产需求。
- [关键] 验证实时实例集上的开放端口是否与您的服务器可以使用的端口范围相匹配。
- [关键] 关闭 RDP 端口 3389 和 SSH 端口 22。

- 为游戏的 DevOps 管理制定计划。如果您使用的是 Amazon CloudWatch Logs 或 Amazon CloudWatch 自定义指标，请为服务器实例集中的严重或关键问题定义警报。模拟故障并测试运行手册。
- 验证在满负荷状态下在实例上运行的服务器数量是否在该服务器实例类型的能力范围内。
- 调整您的扩展策略，使其起初更加保守，并提供比您认为需要的更多的闲置容量。您可以稍后根据成本进行优化。考虑使用基于目标的扩展策略，空闲容量为 20%。
- 使用 FlexMatch 延迟规则来匹配地理位置靠近相同 AWS 区域的玩家。使用来自负载测试客户端的合成延迟数据，测试它在负载下的行为如何。
- 对您的玩家身份验证和游戏会话基础架构进行负载测试，以查看其是否可以有效扩展以满足需求。
- 验证保持运行几天的服务器是否仍然可以接受连接。
- 将您的 AWS Support 计划级别提高到商业级或企业级，以便在出现问题或停机时 AWS 能够对您做出响应。

启动

使用以下清单来追踪在 Amazon GameLift 上托管的游戏的发布项目。标有 [关键] 的项目对于您的产品发布至关重要。

- [关键] 将实例集保护策略设置为对所有实时实例集的全面保护，这样缩减规模就不会停止活动的游戏会话。
- [关键] 将实例集的最大规模设置得足够高，以至少满足预期的高峰需求。我们建议您将最大尺寸增加一倍，以应对突发需求。
- 鼓励您的整个开发团队参与发布活动，并在发布室监控您的游戏发布。
- 监控玩家延迟和玩家体验。

启动后

使用以下清单来跟踪在 Amazon GameLift 上托管的游戏的启动后项目。

- 调整扩展规则以最大限度地减少空闲容量。
- 根据您的延迟要求 修改 FlexMatch 规则或添加其他位置。
- 优化服务器可执行文件，因为其性能效率直接影响实例集成本。要使用相同的基础架构运行更多游戏会话，请增加每个实例的服务器进程数。
- 使用您的分析数据来推动持续开发，改善玩家体验和延长游戏寿命，并优化盈利。

为亚马逊准备游戏 GameLift

要为在亚马逊上托管多人游戏做好准备 GameLift，请设置游戏与亚马逊之间的通信 GameLift。本节中的主题提供了有关将您的游戏与亚马逊 GameLift、自定义游戏服务器和实时服务器集成以及添加配 FlexMatch对的详细帮助。

主题

- 将游戏与自定义游戏服务器集成
- 将游戏与 Amazon GameLift 实时服务器集成
- 将游戏与适用于 Unity 的 Amazon GameLift 插件集成
- 将游戏与适用于虚幻引擎的 Amazon GameLift 插件集成
- 获取 Amazon GameLift 实例的实例集数据
- 添加 FlexMatch 对战

将游戏与自定义游戏服务器集成

提供了一个完整的工具集，用于准备多人游戏并自定义游戏服务器，以在 服务上运行。 Amazon GameLift 软件开发工具包包含游戏客户端和服务器与 Amazon GameLift 通信所需的库。有关软件开发工具包及如何获取它们的详细信息，请参阅 Amazon 为开发提供支持 GameLift。

本部分中的主题包含了详细说明，介绍如何先将所需的 功能添加到游戏客户端和游戏服务器，然后部署到 上。有关使您的游戏在 Amazon GameLift 上启动和正常运行的完整路线图，请参阅 Amazon GameLift 托管托管资源路线图。

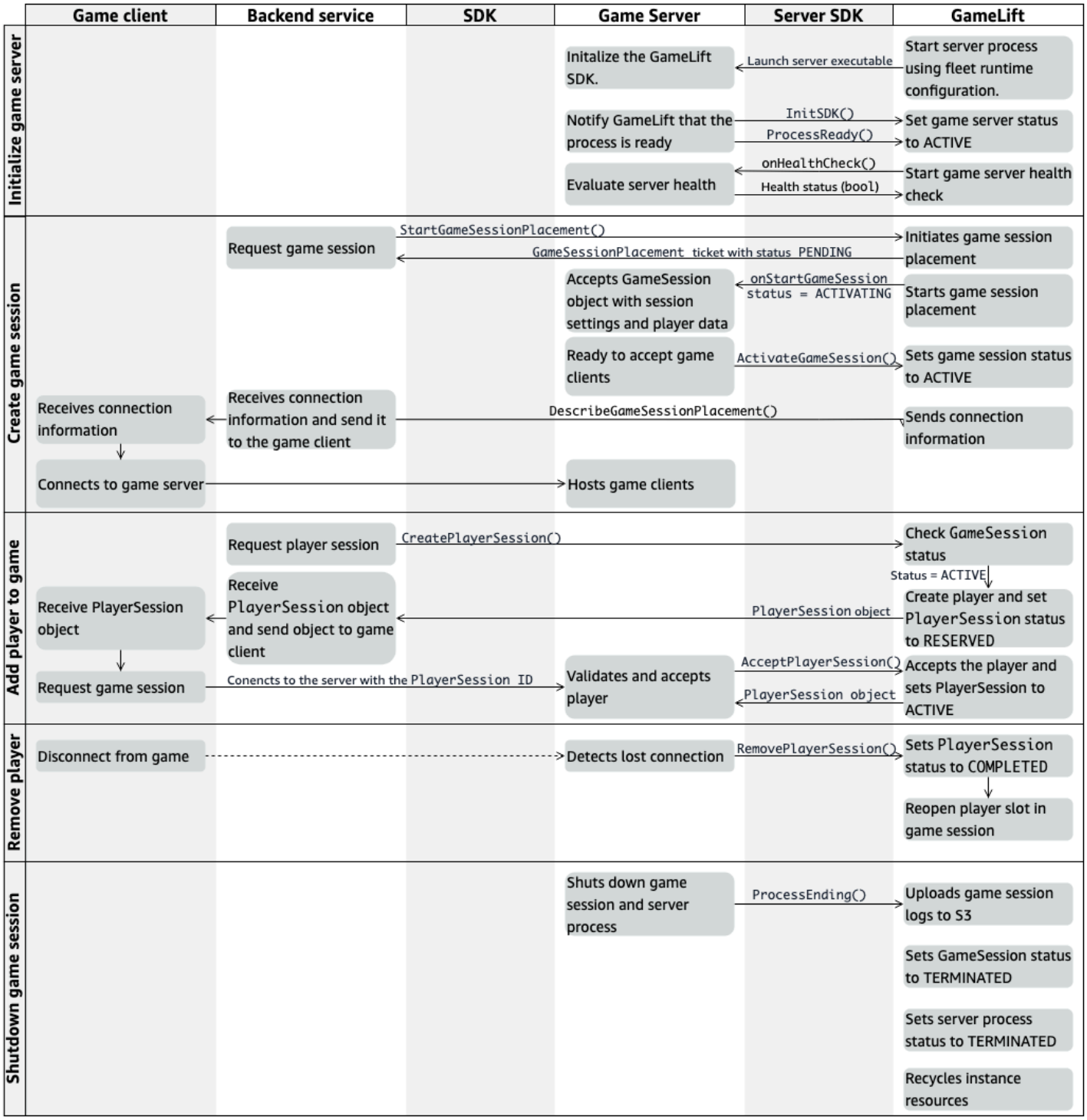
主题

- Amazon GameLift 和游戏客户端服务器的互动
- 将您的游戏服务器与 Amazon GameLift 集成
- 将您的游戏客户端与 Amazon GameLift 集成
- 游戏引擎和 Amazon GameLift
- 使用 Amazon GameLift Anywhere 实例集测试您的集成
- 使用 Amazon GameLift Local 测试您的集成

Amazon GameLift 和游戏客户端服务器的互动

本主题描述了游戏客户端、后端服务、游戏服务器和 Amazon GameLift 之间的交互。

下图说明了游戏客户端、后端服务、Amazon GameLift 软件开发工具包、托管 EC2 游戏服务器、Amazon GameLift 服务器软件开发工具包和 Amazon GameLift 之间的交互。有关所示交互的详细描述，请参阅本页的以下部分。



初始化游戏服务器

以下步骤描述了在准备游戏服务器以托管游戏会话时发生的互动。

1. Amazon GameLift 在 Amazon Elastic Compute Cloud (Amazon EC2) 实例上启动服务器可执行文件。
2. 游戏服务器调用：

   a. `InitSDK()` 初始化服务器软件开发工具包。

   b. `ProcessReady()` 告知游戏会话就绪性、连接信息以及游戏会话日志文件的位置。

   然后，服务器进程会等待 Amazon GameLift 的回调。
3. Amazon GameLift 会将服务器进程的状态更新为 `ACTIVE` 以启用游戏会话放置。
4. Amazon GameLift 开始调用回调 `onHealthCheck`，并在服务器进程处于活动状态时继续定期调用回调。服务器进程可以在一分钟内报告运行状况是否正常。

创建游戏会话

初始化游戏服务器后，当您创建游戏会话来托管玩家时，会发生以下互动。

1. 后端服务调用软件开发工具包操作 `StartGameSessionPlacement()`。
2. Amazon GameLift 会创建一个带有 `PENDING` 状态的 `GameSessionPlacement` 新票证并将其返回给后端服务。
3. 后端服务从队列中获取放置票证状态。有关更多信息，请参阅[请参阅设置游戏会话置放通知。]。
4. Amazon GameLift 通过选择合适的实例集并在包含 `0` 游戏会话的队列中搜索活动服务器进程来开始放置游戏会话。当 Amazon GameLift 找到服务器进程时，Amazon GameLift 会执行以下操作：

   a. 使用游戏会话设置和来自放置请求的玩家数据创建具有 `ACTIVATING` 状态的 `GameSession` 对象。

   b. 调用服务器进程的 `onStartGameSession` 回调。Amazon GameLift `GameSession` 向对象传递信息，表明服务器进程可以设置游戏会话。

   c. 将服务器进程的游戏会话数更改为 `1`。
5. 服务器进程运行 `onStartGameSession` 回调函数。当准备好接受玩家连接时，服务器进程将调用 `ActivateGameSession()` 并等待玩家连接。
6. Amazon GameLift 使用服务器进程的连接信息更新 `GameSession` 对象。（此信息包括报告为 `ProcessReady()` 的端口设置。）Amazon GameLift 也将状态更改为 `ACTIVE`。
7. 后端服务调用 `DescribeGameSessionPlacement()` 以检测更新的票证状态。然后，后端服务使用连接信息将游戏客户端连接到服务器进程并加入游戏会话。

向游戏中添加玩家

此序列描述了向现有游戏会话中添加玩家的过程。玩家会话也可以作为游戏会话放置请求的一部分进行请求。

1. 后端服务使用游戏会话 ID 调用客户端 API `CreatePlayerSession()` 操作。
2. Amazon GameLift 检查游戏会话状态（必须为 `ACTIVE`）并在游戏会话中查找开放的玩家位置。如果有可用位置，Amazon GameLift 将执行以下操作：
   a. 创建新的 `PlayerSession` 对象并将其状态设置为 `RESERVED`。
   b. 使用 `PlayerSession` 对象响应后端服务请求。
3. 后端服务使用玩家会话 ID 直接将游戏客户端连接到服务器进程。
4. 服务器调用服务器 API 操作 `AcceptPlayerSession()` 来验证玩家会话 ID。如果通过验证，则 Amazon GameLift 会将 `PlayerSession` 对象传递到服务器进程。服务器进程接受或拒绝连接。
5. Amazon GameLift 会执行以下操作之一：
   a. 如果连接被接受，则 Amazon GameLift 会将 `PlayerSession` 状态设置为 `ACTIVE`。
   b. 如果后端服务器的原始 `CreatePlayerSession()` 调用在 60 秒内未收到响应，Amazon GameLift 状态 `PlayerSession` 将改为 `TIMEDOUT` 并重新开放游戏会话中的玩家位置。

移除玩家

将玩家从游戏会话中移除以为新玩家的加入创造空间时，会发生以下互动。

1. 玩家断开了与游戏的连接。
2. 服务器检测丢失的连接并调用服务器 API 操作`RemovePlayerSession()`。
3. Amazon GameLift 在游戏会话中将 `PlayerSession` 状态更改为 `COMPLETED` 并重新打开玩家位置。

关闭游戏会话

当服务器进程关闭当前游戏会话时，就会发生这种交互序列。

1. 服务器关闭游戏会话和服务器。
2. 服务器调用 Amazon GameLift 的 `ProcessEnding()`。
3. Amazon GameLift 会执行以下操作：

a. 将游戏会话日志上传到 Amazon Simple Storage Service (Amazon S3)。
b. 将状态 `GameSession` 变为 `TERMINATED`。
c. 更改服务器进程状态为 `TERMINATED`。
d. 回收实例资源。

将您的游戏服务器与 Amazon GameLift 集成

在您的自定义游戏服务器部署并在 Amazon GameLift 实例上运行后，它必须能够与 Amazon GameLift（可能还有其他资源）进行交互。此部分介绍如何游戏服务器软件与 Amazon GameLift 集成

> **Note**
> 这些说明假设您已经创建了一个 AWS 账户并且您有一个现有的游戏服务器项目。

本节主题介绍如何处理以下集成任务：

- 在 Amazon GameLift 和您的游戏服务器之间建立通信。
- 生成并使用 TLS 证书在游戏客户端和游戏服务器之间建立安全连接。
- 为您的游戏服务器软件提供与其他 AWS 资源交互的权限。
- 允许游戏服务器进程获取有关其正在运行的实例集的信息。

主题

- 将 Amazon GameLift 添加到您的游戏服务器
- 与您的实例集中的其他 AWS 资源进行通信

将 Amazon GameLift 添加到您的游戏服务器

您的自定义游戏服务器必须与 Amazon GameLift 通信，因为每个游戏服务器进程都必须能够响应 Amazon GameLift 启动的事件。您的游戏服务器还必须随时向 Amazon GameLift 通报服务器进程状态和玩家连接情况。有关您的游戏服务器、后端服务、游戏客户端和 Amazon GameLift 如何协同管理游戏托管的更多信息，请参阅Amazon GameLift 和游戏客户端服务器的互动。

要将 Amazon GameLift 集成到您的游戏服务器中，请将 Amazon GameLift 软件开发工具包添加到游戏服务器项目并构建此主题介绍的功能。该服务器软件开发工具包支持多种语言。有关 Amazon GameLift 服务器软件开发工具包的更多信息，请参阅Amazon 为开发提供支持 GameLift。

服务器软件开发工具包 API 参考：

- 适用于 C++ 的 Amazon GameLift 服务器软件开发工具包 5.x 参考
- 适用于 C# 和 Unity 的 Amazon GameLift 服务器软件开发工具包 5.x 参考
- Amazon GameLift Unreal Engine 服务器软件开发工具包 5.x 参考

初始化服务器进程

添加代码以与 Amazon GameLift 建立通信并报告服务器进程已准备好托管游戏会话。此代码必须在任何 Amazon GameLift 代码之前运行。

1. 通过调用 `InitSdk()` 初始化 Amazon GameLift API 客户端。要在Amazon GameLift Anywhere 计算资源上初始化服务器进程，请使用以下 `ServerParameters` 调用 `InitSdk()`：

   - 用于连接游戏服务器的 WebSocket 网址。
   - 用于托管游戏服务器的进程的 ID。
   - 托管游戏服务器进程的计算的 ID。
   - 包含您的 Amazon GameLift Anywhere 计算机的 GameLift 实例集的 ID。
   - Amazon GameLift 操作 `GetComputeAuthToken` 生成的授权令牌。

   > **Note**
   >
   > 要在 Amazon GameLift 托管的 Amazon EC2 实例上初始化游戏服务器，请使用默认 `InitSDK()` (C++) (C#) (Unreal) 构造函数（不带参数）构造服务器参数。 Amazon GameLift 为您设置计算环境并自动连接到 Amazon GameLift。

2. 通知 Amazon GameLift，游戏服务器进程已准备好托管游戏会话。使用以下信息调用 `ProcessReady()` (C ++) (C#) (Unreal)。（请注意，每个服务器进程只能调用 `ProcessReady()` 一次）。

   - 服务器进程使用的端口号。后端服务向游戏客户端提供端口号和 IP 地址，用于连接到服务器进程并加入游戏会话。

- 您希望 Amazon GameLift 保留的文件的位置，例如游戏会话日志。服务器进程在游戏会话期间生成这些文件。它们暂时存储在服务器进程正在运行的实例上，当实例关闭时，它们就会丢失。您发布的所有文件都将上传到 Amazon GameLift 中。您可以通过 Amazon GameLift 控制台或调用 Amazon GameLift API 操作 GetGameSessionLogUrl() 访问这些文件。
- Amazon GameLift 可以调用到您的服务器进程的回调函数的名称。游戏服务器代码必须执行这些函数。有关更多信息，请参阅 (C++) (C#) (Unreal) 。
  - （可选）onHealthCheck – Amazon GameLift 定期调用此函数向服务器请求运行状况报告。
  - onStartGameSession – 调用此函数响应客户端请求 CreateGameSession()。
  - onProcessTerminate – Amazon GameLift 强制服务器进程停止，让它正常关闭。
  - （可选）onUpdateGameSession – Amazon GameLift 向游戏服务器交付更新后的游戏会话对象或根据对战回填请求提供状态更新。FlexMatch 回填特征需要此回调。

  您也可以设置游戏服务器来安全地访问您拥有或控制的 AWS 资源。有关更多信息，请参阅与您的实例集中的其他 AWS 资源进行通信。

报告服务器进程运行状况

向游戏服务器添加代码以实现回调函数 onHealthCheck()。Amazon GameLift 会定期调用此回调方法来收集运行状况指标。要实现此回调函数，执行以下操作：

- 评估服务器进程的运行状况。例如，如果任何外部依赖项失败，您可将服务器进程报告为不正常。
- 完成运行状况评估并在 60 秒内响应回调。如果 Amazon GameLift 未在该时间内收到响应，它会自动将服务器进程视为不正常。
- 返回布尔值：true 表示正常，false 表示不正常。

如果您未实现运行状况检查回调，那么 Amazon GameLift 会认为服务器进程正常，除非服务器不响应。

Amazon GameLift 使用服务器进程运行状况来结束不健康的进程并清理资源。如果某个服务器进程在三次连续的运行状况检查中持续报告不正常或不响应， 服务可能会关闭该进程然后启动新进程。Amazon GameLift 收集有关实例集服务器进程运行状况的指标。

（可选）获取 TLS 证书

如果游戏服务器在启用了 TLS 证书生成的实例集上运行，您可以检索 TLS 证书并将其用于与游戏客户端建立安全连接并加密客户端/服务器通信。证书的副本存储在实例上。要获取文件位置，请调用 `GetComputeCertificate()` (C ++) (C#) (Unreal) 。

启动游戏会话

添加代码以实现回调函数 `onStartGameSession`。Amazon GameLift 调用此回调在服务器上启动游戏会话。

`onStartGameSession` 函数采用 GameSession 对象作为输入参数。此对象包含关键的游戏会话信息，例如最大玩家人数。它还可以包括游戏数据和玩家数据。函数实现应完成以下任务：

- 启动操作以基于 `GameSession` 属性创建新的游戏会话。游戏服务器至少必须关联游戏会话 ID，游戏客户端在连接到服务器进程时会引用该会话 ID。
- 根据需要处理游戏数据和玩家数据。这些数据在 `GameSession` 对象中。
- 当新的游戏会话准备好接受玩家时，通知 Amazon GameLift。调用服务器 API 操作 `ActivateGameSession()` (C++) (C#) (Unreal) 。作为对成功调用的响应，`ACTIVE` 服务将游戏会话状态更改为 ACTIVE。

（可选）验证新玩家

如果您正在跟踪玩家会话的状态，请添加代码以在新玩家连接到游戏服务器时对其进行验证。Amazon GameLift 会追踪当前玩家和可用的游戏会话老虎机。

为了进行验证，请求访问游戏会话的游戏客户端必须包含玩家会话 ID。当玩家要求使用 StartGameSessionPlacement() 或 StartMatchmaking() 加入游戏时，Amazon GameLift 会自动生成此 ID。然后，玩家会话将在游戏会话中保留空闲位置。

当游戏服务器进程收到游戏客户端连接请求时，它会使用玩家会话 ID 调用 `AcceptPlayerSession()` (C++) (C#) (Unreal) 。作为回应，Amazon GameLift 会验证玩家会话 ID 是否与游戏会话中预留的空位相对应。在 Amazon GameLift 验证玩家会话 ID 后，服务器进程接受连接。然后，玩家可以加入游戏会话。如果 Amazon GameLift 未验证玩家会话 ID，则服务器进程会拒绝连接。

报告玩家会话结束

如果您正在跟踪玩家会话的状态，请添加代码以便在玩家离开游戏会话时通知 Amazon GameLift。只要服务器进程检测到断开的连接，此代码就应运行。Amazon GameLift 使用此通知来跟踪游戏会话中的当前玩家和可用老虎机。

要处理断开的连接，请在代码中添加对服务器 API 操作 `RemovePlayerSession()` (C++) (C#) (Unreal) 的调用，并附上相应的玩家会话 ID。

结束游戏会话

在服务器进程关闭序列中添加代码，以便在游戏会话结束时通知 Amazon GameLift。为了回收和刷新托管资源，Amazon GameLift 会在游戏会话完成后关闭服务器进程。

在服务器进程关闭代码开始时，调用服务器 API 操作 `ProcessEnding()` (C++) (C#) (Unreal) .。此调用将通知 服务，服务器进程正在关闭。Amazon GameLift 将游戏会话状态和服务器进程状态更改为 `TERMINATED`。调用 `ProcessEnding()` 后，可以安全地关闭进程。

回应服务器进程关闭通知

添加代码以关闭服务器进程，以响应 Amazon GameLift 的通知。当服务器进程持续报告运行状况不佳或服务器进程正在运行的实例被终止时，Amazon GameLift 会发送此通知。Amazon GameLift 可以在容量缩减事件中或响应竞价型实例中断时停止实例。

要处理关机通知，请对游戏服务器代码进行以下更改：

- 实现回调函数 `onProcessTerminate()`。此函数应调用用于关闭游戏服务器的代码。当 Amazon GameLift 调用此操作时，竞价型实例中断会在两分钟内发出通知。收到此调用后，服务器进程有五分钟的时间正常断开玩家连接、保留游戏状态数据和执行其他清除任务。
- 从游戏服务器关闭代码中调用服务器 API 操作 `GetTerminationTime()` (C++) (C#) (Unreal) 。如果 Amazon GameLift 已发出停止服务器进程的调用，则 `GetTerminationTime()` 返回预计的终止时间。
- 在游戏服务器关闭代码开始时，调用服务器 API 操作 `ProcessEnding()` (C++) (C#) (Unreal) 。此调用通知 Amazon GameLift 服务器进程正在关闭，然后 Amazon GameLift 将服务器进程状态更改为 `TERMINATED`。调用 `ProcessEnding()` 后，可以安全地关闭进程。

与您的实例集中的其他 AWS 资源进行通信

当您创建要在 Amazon GameLift 队列上部署的游戏服务器版本时，您可能希望游戏版本中的应用程序能够与您拥有的其他 AWS 资源直接安全地通信。由于 Amazon GameLift 管理您的游戏托管队伍，因此您必须向 Amazon GameLift 提供对这些资源和服务的有限访问权限。

一些示例场景包括：

- 使用 Amazon CloudWatch 代理从托管 EC2 实例集和 Anywhere 实例集中收集指标、日志和跟踪信息
- 将实例日志数据发送到 CloudWatch 日志。
- 将消息存储在 Amazon Simple Storage Service (Amazon S3) 存储桶中。
- 动态读写在 或其他数据存储服务中存储的游戏数据，例如游戏模式或清单。
- 使用 Amazon Simple Queue Service Amazon S将信号直接发送到实例。
- 访问在 Amazon Elastic Compute Cloud (Amazon EC2) 上部署和运行的自定义资源。

Amazon GameLift 支持以下方法来建立访问权限：

- 使用软件开发工具包以 IAM 角色访问 AWS 资源
- 通过 VPC 对等互连访问 AWS 资源

使用软件开发工具包以 IAM 角色访问 AWS 资源

使用 IAM 角色指定谁可以访问您的资源并设置访问限制。受信任方可以“担任”角色并获得临时安全证书，以授权他们与资源进行交互。当各方发出与资源相关的 API 请求时，他们必须提供证书。

要设置由 IAM 角色控制的访问权限，请执行以下任务：

1. 创建 IAM 角色。
2. 修改应用程序以获取凭证
3. 将服务角色与 实例集关联。

创建 IAM 角色。

在此步骤中，您将创建一个 IAM 角色，该角色具有一组控制 AWS 资源访问权限的权限和一个授予 Amazon GameLift 使用该角色权限的信任策略。

有关如何设置 IAM 角色的说明，请参阅为亚马逊设置 IAM 服务角色 GameLift。在创建权限策略时，请选择您的应用程序需要使用的特定服务、资源和操作。作为最佳实操，请尽可能限制权限的范围。

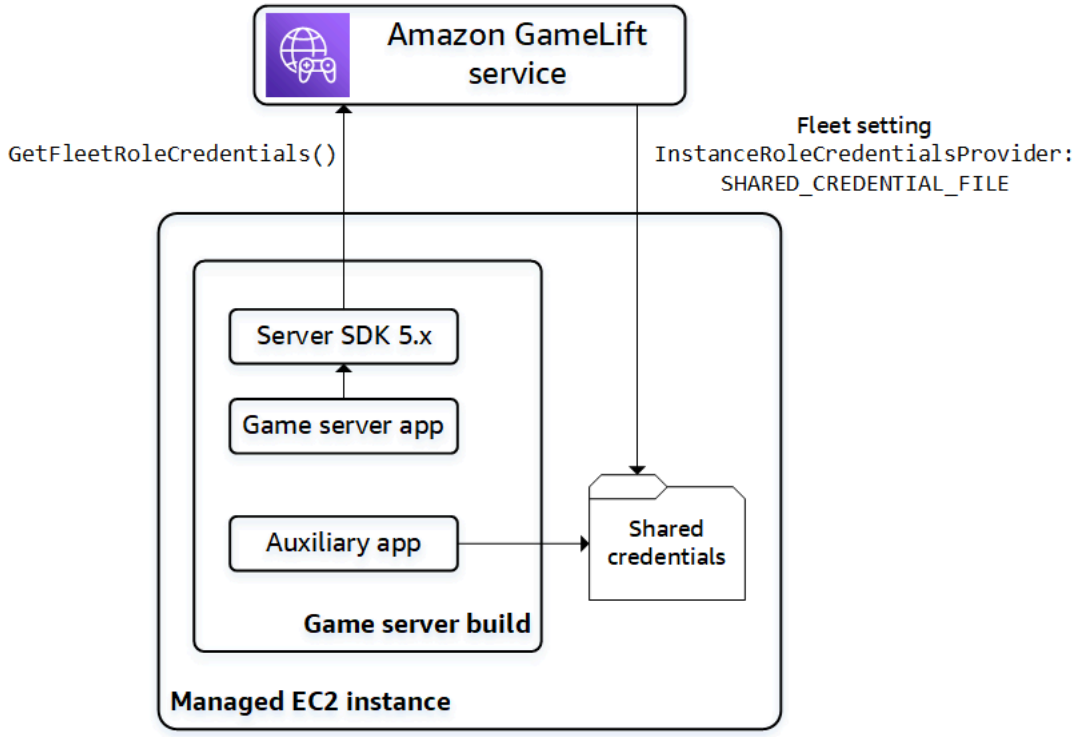
创建角色后，记下其 Amazon 资源名称（ARN）。在创建实例集期间，您需要角色 ARN。

修改应用程序以获取凭证

在此步骤中，您将应用程序配置为获取 IAM 角色的安全凭证，并在与您的 AWS 资源交互时使用这些凭证。参见下表，根据 (1) 应用程序的类型以及 (2) 您的游戏用于与 Amazon GameLift 通信的服务器软件开发工具包 版本来确定如何修改您的应用程序。

	游戏服务器应用程序	其他客户端应用程序
使用服务器软件开发工具包版本 5.x	从您的游戏服务器代码中调用服务器软件开发工具包方法 `GetFleetRoleCredentials()` 。	向应用程序添加代码，以便从队列实例上的共享文件中提取证书。
使用服务器软件开发工具包版本 4 或更早版本	`AssumeRole` 使用角色 ARN 调用 AWS Security Token Service (AWS STS)。	`AssumeRole` 使用角色 ARN 调用 AWS Security Token Service (AWS STS)。

对于与服务器软件开发工具包 5.x 集成的游戏，此图说明了您部署的游戏版本中的应用程序如何获取 IAM 角色的证书。



调用 **GetFleetRoleCredentials()**（服务器软件开发工具包 5.x）

在您的游戏服务器代码（该代码应该已经与 Amazon GameLift 服务器软件开发工具包 5.x 集成）中，调用 `GetFleetRoleCredentials` (C++) (C#) (Unreal) 来检索一组临时凭证。证书过期后，您可以通过再次调用 `GetFleetRoleCredentials` 来刷新凭证。

使用共享凭据（服务器软件开发工具包 5.x）

对于使用服务器软件开发工具包 5.x 与游戏服务器版本一起部署的非服务器应用程序，请添加代码以获取和使用存储在共享文件中的凭据。Amazon GameLift 会为每个队列实例生成一个凭证配置文件。这些证书可供实例上的所有应用程序使用。 Amazon GameLift 会不断刷新临时证书。

您必须将队列配置为在创建队列时生成共享凭证文件。

在每个需要使用共享凭据文件的应用程序中，指定文件位置和配置文件名称，如下所示：

Windows:

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

Linux：

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

示例：设置 CloudWatch 代理以收集 Amazon GameLift 实例集实例的指标

如果您想使用 Amazon CloudWatch 代理从您的 Amazon GameLift 队列收集指标、日志和跟踪，请使用此方法授权代理将数据发送到您的账户。在此场景中，请采取以下步骤：

1. 检索或写入 CloudWatch 代理 `config.json` 文件。
2. 如上所述，更新 `common-config.toml` 代理文件以识别凭证文件名和配置文件名称。
3. 设置游戏服务器版本安装脚本以安装和启动 CloudWatch 代理。

使用 **`AssumeRole()`**（服务器软件开发工具包 4）

向您的应用程序添加代码以担任 IAM 角色并获取用于与您的 AWS 资源交互的凭证。任何在带有服务器软件开发工具包 4 或更早版本的 Amazon GameLift 队列实例上运行的应用程序都可以担任 IAM 角色。

在应用程序代码中，在访问 AWS 资源之前，应用程序必须调用 AWS Security Token Service (AWS STS) `AssumeRole` API 操作并指定角色 ARN。此操作返回一组授权应用程序访问 AWS 资源的临时凭证。有关更多信息，请参阅《IAM 用户指南》中的将临时凭证用于 AWS 资源。

将服务角色与 实例集关联。

在创建 IAM 角色并更新游戏服务器版本中的应用程序以获取和使用访问凭证后，您可以部署队列。配置新队列时，请设置以下参数：

- InstanceoLearn – 将此参数设置为 IAM 角色的 ARN。
- InstanceRoleCredentialsProvider – 要提示 Amazon GameLift 为每个队列实例生成共享凭证文件，请将此参数设置为 `SHARED_CREDENTIAL_FILE`。

创建队列时必须设置这些值。以后，无法对其进行更新。

通过 VPC 对等互连访问 AWS 资源

您可以使用 Amazon Virtual Private Cloud（Amazon VPC）对等在 Amazon GameLift 实例上运行的应用程序与其他 AWS 资源上运行的应用程序之间进行通信。VPC 是您定义的虚拟专用网络，其中包

括通过您 AWS 账户管理的一组资源。每个 Amazon GameQual GameLift 队伍都有自己的 VPC。借助 VPC 对等连接，您可以在实例集的 VPC 与其他 AWS 资源的 VPC 之间建立直接网络连接。

简化了为游戏服务器设置 VPC 对等连接的过程。它会处理对等请求、更新路由表，并根据需要配置连接。有关如何为游戏服务器设置 VPC 对等连接的更多信息，请参阅Amazon GameLift 的 VPC 对等连接。

将您的游戏客户端与 Amazon GameLift 集成

本部分中的主题描述了您可以添加到后端服务的托管 Amazon GameLift 功能。后端服务处理以下任务：

- 从 Amazon GameLift 请求有关活动游戏会话的信息。
- 将玩家加入到现有游戏会话。
- 创建一个新的游戏会话并在其中加入玩家。
- 更改有关现有游戏会话的元数据。

有关游戏客户端如何与 Amazon GameLift 以及在 Amazon GameLift 上运行的游戏服务器进行交互的更多信息，请参阅Amazon GameLift 和游戏客户端服务器的互动。

先决条件

- 一个 AWS 账户。
- 已上传到 Amazon GameLift 中的游戏服务器版本。
- 用于托管游戏的实例集。

主题

- 将 Amazon GameLift 添加到您的游戏客户端
- 生成玩家 ID

将 Amazon GameLift 添加到您的游戏客户端

将 Amazon GameLift 集成到需要游戏会话信息的游戏组件中，创建新的游戏会话以及向游戏中添加玩家。根据游戏架构，此功能可能会放在后端服务中，处理诸如玩家身份验证、对战或游戏会话放置等任务。

> **Note**
>
> 有关如何为亚马逊 GameLift 托管的游戏设置配对的详细信息，请参阅《亚马逊 GameLift FlexMatch开发者指南》。

在后端服务 GameLift 上设置 Amazon

添加用于初始化 Amazon GameLift 客户端和存储密钥设置的代码。此代码必须在任何依赖于 Amazon 的代码之前运行 GameLift。

1. 设置客户端配置。使用默认客户端配置或创建自定义客户端配置对象。有关更多信息，请参阅 AWS::Client::ClientConfiguration(C++) 或 AmazonGameLiftConfig(C#)。

   客户端配置指定了联系亚马逊时要使用的目标区域和终端节点 GameLift。区域确定要使用的已部署资源集（实例集、队列和对战）。默认客户端配置设置位置为美国东部（弗吉尼亚州北部）区域。要使用任何其他区域，请创建自定义配置。
2. 初始化 Amazon GameLift 客户端。将 Aws:GameLift:: GameLiftClient () (C++) 或 AmazonGameLiftClient() (C#) 与默认客户端配置或自定义客户端配置一起使用。
3. 添加为每个玩家生成一个唯一标识符的机制。有关更多信息，请参阅 生成玩家 ID。
4. 收集并存储以下信息：

   - 目标队列 — 许多 Amazon GameLift API 请求都必须指定队列。要指定目标实例集，请使用实例集 ID 或者指向目标实例集的别名 ID。最佳做法是使用实例集别名，这样您就可以在不更新后端服务的情况下将玩家从一个实例集切换到另一个实例集。
   - 目标队列 – 对于使用多实例集队列放置新游戏会话的游戏，请指定要使用的队列名称。
   - AWS 凭证 — 所有向 Amazon 拨打的电话都 GameLift 必须提供托管游戏的凭证。 AWS 账户您可以通过创建玩家用户来获取这些凭证，如为游戏设置编程式访问权限中所述。根据您管理玩家用户访问权限的方式，请执行以下操作：
     - 如果您使用角色来管理玩家用户权限，请在调用 Amazon GameLift API 之前添加代入该角色的代码。承担角色的请求返回一组临时安全凭证。有关更多信息，请参阅 IAM 用户指南中的切换到 IAM 角色 (AWS API)。
     - 如果您拥有长期安全凭证，请配置您的代码以查找和使用存储的凭证。请参阅《AWS 软件开发工具包和工具参考指南》中的使用长期凭证进行身份验证。有关存储凭据的信息，请参阅 (C++) 和 (.NET) 的AWS API 参考。

- 如果您有临时安全凭证，请使用 AWS Security Token Service (AWS STS) 添加代码以定期刷新凭证，如《IAM 用户指南》中的将临时安全凭证与 AWS 软件开发工具包配合使用中所述。该代码必须在旧凭证过期之前请求新的凭证。

获取游戏会话

添加用于发现可用游戏会话和管理游戏会话设置和元数据的代码。

搜索活动的游戏会话。

SearchGameSessions用于获取有关特定游戏会话、所有活跃会话或符合一组搜索条件的会话的信息。此调用会为每个活动游戏会话返回一个与您的搜索请求相匹配的GameSession对象。

使用搜索条件获取经筛选列表，列出可供玩家接入的活动游戏会话。例如，您可以按照以下方式筛选会话：

- 排除已饱和的游戏会话：`CurrentPlayerSessionCount = MaximumPlayerSessionCount`
- 根据会话运行的时长来选择游戏会话：评估 `CreationTime`
- 根据自定义游戏属性查找游戏会话：`gameSessionProperties.gameMode = "brawl"`

管理游戏会话。

使用以下任意一项操作来检索或更新游戏会话信息。

- DescribeGameSessionDetails() — 获取游戏会话的保护状态以及游戏会话信息。
- UpdateGameSession() — 根据需要更改游戏会话的元数据和设置。
- GetGameSessionLogUrl— 访问存储的游戏会话日志。

创建游戏会话

添加用于在已部署的实例集中启动新游戏会话并使其可供玩家接入的代码。创建游戏会话有两个选项，具体取决于你是在多个区域还 AWS 区域 是在单个区域部署游戏。

在多位置队列中创建游戏会话

StartGameSessionPlacement用于在队列中请求新游戏会话。要使用此操作，请创建一个队列。这决定了 Amazon 将新游戏会话 GameLift 放置在哪里。有关队列及其使用方法的更多信息，请参阅《Amazon GameLift 开发人员指南》中的设置游戏会话置放的 GameLift 队列。

创建游戏会话放置时，指定要使用的队列名称、游戏会话名称、最大并发玩家数量以及一组可选的游戏属性。您可以选择提供玩家列表来自动加入游戏会话。如果您包含相关区域的玩家延迟数据，则 Amazon 会 GameLift 使用这些信息将新的游戏会话放置在为玩家提供理想游戏体验的舰队上。

游戏会话放置为异步操作。在放置请求后，您等待其成功或超时。您也可以随时使用取消请求StopGameSessionPlacement。要查看您的安置申请的状态，请致电DescribeGameSessionPlacement。

在特定的实例集中创建游戏会话。

CreateGameSession用于在指定队列上创建新会话。这一同步操作成功与否取决于该实例集是否拥有托管新游戏会话所需的资源。在 Amazon GameLift 创建新游戏会话并返回GameSession对象后，您可以加入玩家的行列。

使用此操作时，请提供实例集 ID 或别名 ID、会话名称和游戏中的最大并发玩家数量。您可以选择包括一组游戏属性。游戏属性是在键值对的数组中定义的。

如果您使用 Amazon GameLift 资源保护功能来限制一个玩家可以创建的游戏会话数量，请提供游戏会话创建者的玩家 ID。

将玩家接入游戏会话

添加在活动的游戏会话中预留玩家位置以及将游戏客户端连接到游戏会话的代码。

1. 在游戏会话中预留玩家位置。

   要预留玩家位置，请在游戏会话中新建一个玩家会话。有关玩家会话的更多信息，请参阅如何将玩家接入游戏。

   您可通过两种方式来创建新的玩家会话：

   - 用于StartGameSessionPlacement在新游戏会话中为一名或多名玩家预留老虎机。
   - 使用或使用游戏会话 ID 为一名CreatePlayerSession或CreatePlayerSessions多名玩家保留玩家位置。

   Amazon GameLift 首先验证游戏会话是否接受新玩家，并且有可用的玩家位置。如果成功，Ama GameLift zon 将为玩家预留一个位置，创建新的玩家会话并返回一个PlayerSession对象。此对象包含游戏客户端连接到游戏会话所需的 DNS 名称、IP 地址和端口。

   玩家会话请求必须包括每个玩家的唯一 ID。有关更多信息，请参阅 生成玩家 ID。

玩家会话可能包括一组自定义玩家数据。这些数据存储在新创建的玩家会话对象中，您可以通过调用 DescribePlayerSessions() 来检索该对象。当玩家直接连接到游戏会话时，Amazon GameLift 还会将此对象传递给游戏服务器。在请求多人游戏会话时，您可以提供每个玩家的玩家数据字符串，在请求中映射到玩家 ID。

2. 连接游戏会话

   将代码添加到游戏客户端来检索包含游戏会话的连接信息的 `PlayerSession` 对象。使用此信息与服务器建立直接连接。

   - 您可以使用指定的端口以及分配给服务器进程的 DNS 名称或 IP 地址进行连接。
   - 如果实例集启用了 TLS 证书生成，则必须使用 DNS 名称和端口进行连接。
   - 如果您的游戏服务器验证了传入的玩家连接，请引用玩家会话 ID。

   建立连接后，游戏客户端和服务器进程直接通信，无需Amazon参与 GameLift。服务器与 Amazon 保持通信， GameLift以报告玩家的连接状态、健康状态等。如果游戏服务器验证了传入的玩家，则它会验证玩家会话 ID 是否与游戏会话中的预留位置相匹配，并接受或拒绝玩家连接。当玩家断开连接时，服务器进程报告断开连接。

使用游戏会话属性

您的游戏客户端可以使用游戏属性将数据传递到游戏会话中。游戏属性是您的游戏服务器可以添加、读取、列出和更改的键值对。您可以在创建新的游戏会话时传入游戏属性，也可以稍后在游戏会话处于活动状态时传入游戏属性。一个游戏会话最多可以包含 16 个游戏属性。您无法删除游戏属性。

例如，您的游戏提供以下难度等级：`NoviceEasy`、`Intermediate`、和`Expert`。玩家选择`Easy`，然后开始游戏。您的游戏客户端使用`StartGameSessionPlacement`或向亚马逊 GameLift 请求新的游戏会话，`CreateGameSession`如前面部分所述。在请求中，客户端传递了这个：`{"Key": "Difficulty", "Value":"Easy"}`。

作为对请求的响应，Amazon GameLift 创建了一个包含指定游戏属性的`GameSession`对象。GameLift 然后，Amazon 会指示可用的游戏服务器启动新的游戏会话并传递`GameSession`对象。游戏服务器以开启游戏会`Difficulty`话`Easy`。

了解更多信息

- GameProperty 数据类型
- SearchGameSessions() 示例

- UpdateGameSession() GameProperties 参数

生成玩家 ID

Amazon GameLift 使用玩家会话表示已连接至游戏会话的玩家。每当玩家使用与 Amazon GameLift 集成的游戏客户端连接到游戏会话时，Amazon GameLift 都会创建一个玩家会话。当玩家离开游戏时，玩家会话将结束。Amazon GameLift 不会重复使用玩家会话。

以下示例代码会随机生成唯一的玩家 ID：

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

有关玩家会话的更多信息，请参阅查看游戏和玩家会话中的数据。

游戏引擎和 Amazon GameLift

您可以在大多数支持 C++ 或 C# 库的游戏引擎中使用托管 服务，包括 Amazon Lumberyard、Unreal Engine 和 Unity。构建游戏所需的版本；有关构建说明和最低要求，请参阅每个版本的自述文件。有关可用 Amazon GameLift 开发工具包以及支持的开发平台和操作系统的更多信息，请参阅适用于游戏服务器的 Amazon 为开发提供支持 GameLift。

除本主题中提供的特定引擎信息外，请参考下列主题，获得有关在您的游戏服务器、客户端和服务中集成 的更多帮助：

- Amazon GameLift 托管托管资源路线图 - 一个六步工作流程，可实现在游戏中成功集成 Amazon GameLift 和设置托管资源。
- 将 Amazon GameLift 添加到您的游戏服务器 - 在游戏服务器中集成 Amazon GameLift 的详细说明。
- 将 Amazon GameLift 添加到您的游戏客户端 - 集成到游戏客户端或服务中的详细说明，包括创建游戏会话和将玩家加入游戏。

O3DE

游戏服务器

使用适用于 C++ 的 Amazon GameLift 服务器软件开发工具包准备用于托管 Amazon GameLift 的游戏服务器。请参阅将 Amazon GameLift 添加到您的游戏服务器，以获取将所需功能集成到游戏服务器的帮助。

游戏客户端和服务

启用游戏客户端和/或游戏服务与 服务进行交互，如查找可用游戏会话或创建新会话，以及将玩家加入游戏。适用于 C++ 的 AWS 软件开发工具包中提供了核心客户端功能。要将 Amazon GameLift 集成到 O3DE 游戏项目中，请参阅将 Amazon GameLift 添加到 O3DE 游戏客户端和服务器和将 Amazon GameLift 添加到您的游戏客户端。

Unreal Engine

游戏服务器

通过将适用于 Unreal Engine 的 Amazon GameLift 服务器软件开发工具包添加到您的项目并实现所需的服务器功能，在 Amazon GameLift 中准备游戏服务器的托管。有关设置 Unreal Engine 插件和添加 Amazon GameLift 代码的帮助，请参阅将 Amazon GameLift 集成到虚幻引擎项目中。

游戏客户端和服务

启用游戏客户端和/或游戏服务与 服务进行交互，如查找可用游戏会话或创建新会话，以及将玩家加入游戏。适用于 C++ 的 AWS 软件开发工具包中提供了核心客户端功能。要将 Amazon GameLift 集成到 Unreal Engine 游戏项目中，请参阅将 Amazon GameLift 添加到您的游戏客户端。

Unity

游戏服务器

通过将适用于 C# 的 Amazon GameLift 服务器软件开发工具包添加到您的项目并实现所需的服务器功能，在 Amazon GameLift 中准备游戏服务器的托管。有关使用 Unity 进行设置和添加 Amazon GameLift 代码的帮助，请参阅将 Amazon GameLift 集成到 Unity 项目中。

游戏客户端和服务

启用游戏客户端和/或游戏服务与 服务进行交互，如查找可用游戏会话或创建新会话，以及将玩家加入游戏。AWS SDK for .NET中提供了核心客户端功能。要将 Amazon GameLift 集成到 Unity 游戏项目中，请参阅将 Amazon GameLift 添加到您的游戏客户端。

其他引擎

有关游戏服务器和客户端可使用的 Amazon GameLift 开发工具包的完整列表，请参阅the section called “ 开发 Linux 支持”。

将 Amazon GameLift 添加到 O3DE 游戏客户端和服务器

您可以使用开源、跨平台、实时 3D 引擎 O3DE 来创建高性能的交互式体验，包括游戏和模拟。O3DE 渲染器和工具封装在模块化框架中，您可以使用首选的开发工具对其进行修改和扩展。

模块化框架使用包含具有标准接口和资产的库的 Gem。选择您自己的 Gem，根据您的要求选择要添加的功能。

Amazon GameLift Gem 提供以下功能：

Amazon GameLift 集成

一个用于扩展 O3DE 网络层并让多人游戏 Gem 与 Amazon GameLift 专用服务器解决方案配合使用的框架。Gem 提供了与 Amazon GameLift 服务器软件开发工具包和 AWS 软件开发工具包客户端（称为 Amazon GameLift 服务本身）的集成。

构建和软件包管理

打包并可选择上传专用服务器版本和 AWS Cloud Development Kit (AWS CDK) (AWS CDK) 应用程序的说明，以设置和更新资源。

Amazon GameLift Gem 设置

按照本节中的步骤在 O3DE 中设置 Amazon GameLift Gem。

先决条件

- 为 Amazon GameLift 设置 AWS 账户。有关更多信息，请参阅设置一个 AWS 账户。
- 为 O3DE 设置 AWS 凭证。有关更多信息，请参阅配置 AWS 凭证。
- 设置 AWS CLI 和 AWS CDK。有关更多信息，请参阅 AWS Command Line Interface 和 AWS Cloud Development Kit (AWS CDK)。

打开 Amazon GameLift Gem 及其依赖项

1. 打开项目管理器。
2. 打开项目下的菜单，然后选择编辑项目设置...。
3. 选择配置 Gem。
4. 打开 Amazon GameLift Gem 和以下依赖宝石：

- AWS Core Gem – 提供在 O3DE 中使用 AWS 服务 的框架。
- 多人游戏 Gem – 通过扩展网络框架提供多人游戏功能。

包括 Amazon GameLift Gem 静态库

1. 为您的项目服务器目标添加 Gem::AWSGameLift.Server.Static 作为 BUILD_DEPENDENCIES。

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

2. 将 AWSGameLiftService 设置为您的项目服务器系统组件必填项。

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

3. （可选）要使用 C++ 提出 Amazon GameLift 服务请求，请在针对您的客户目标的 BUILD_DEPENDENCIES 中包含 Gem::AWSGameLift.Client.Static。

```
ly_add_target(
    NAME YourProject.Client.Static STATIC
    ...
    BUILD_DEPENDENCIES
    PUBLIC
        ...
    PRIVATE
        ...
```

```
        Gem::AWSCore.Static
        Gem::AWSGameLift.Client.Static
}
```

集成您的游戏和专用服务器

使用会话管理集成管理游戏和专用游戏服务器中的游戏会话。要支持 FlexMatch，请参阅 FlexMatch 集成。

将 Amazon GameLift 集成到虚幻引擎项目中

本主题介绍如何为虚幻引擎设置 Amazon GameLift C++ 服务器 SDK 插件并将其集成到您的游戏项目中。

其他资源

- 虚幻下载网站的服务器软件开发工具包 插件
- Amazon GameLift Unreal Engine 服务器软件开发工具包 5.x 参考
- the section called “ 开发 Linux 支持”

先决条件

在继续之前，请确保您满足以下先决条件：

先决条件

- 一台能够运行 Unreal Engine 的计算机。有关 Unreal Engine 要求的更多信息，请参阅 Unreal Engine 的硬件和软件规格文档。
- Microsoft Visual Studio 2019 16.2.4 或更高版本。
- EMR 版本 6.1.0 或更高版本
- Python，版本 3.6 或更高版本。
- PATH 上有一个 Git 客户端。
- 一个 Epic 游戏账号。在 Unreal Engine 官方网站上注册一个账号。
- 与你的虚幻引擎 GitHub 账号关联的账号。如需了解更多信息，请参阅虚幻引擎网站 GitHub上的访问虚幻引擎源代码。

> **Note**
>
> Amazon GameLift 目前支持以下版本的虚幻引擎：
>
> - 4.22
> - 4.23
> - 4.24
> - 4.25
> - 4.26
> - 4.27
> - 5.1.0
> - 5.1.1
> - 5.2
> - 5.3

从源代码构建 Unreal Engine

通过Epic启动器下载的 Unreal Engine 编辑器的标准版本仅允许构建虚幻客户端应用程序。要构建虚幻服务器应用程序，您需要使用 Unreal Engine Github 存储库从源代码下载和构建 Unreal Engine 。如需了解更多信息，请参阅 Unreal Engine 文档网站上的从源代码构建 Unreal Engine 教程。

> **Note**
>
> 如果你还没有这样做，请按照访问虚幻引擎源代码中的说明将你的 GitHub 账户关联 GitHub到你的Epic Games账户。

将 Unreal Engine 源代码克隆到您的开发环境中

1. 在您选择的分支中将 Unreal Engine 源代码克隆到您的开发环境中。

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

2. 查看您用来开发游戏的版本的标签。例如，以下示例查看了 Unreal Engine 版本5.1.1：

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

3. 导航到本地存储库的根文件夹。当您在根文件夹中时，运行以下文件：Setup.bat。
4. 在根文件夹中，还要运行文件：GenerateProjectFiles.bat。
5. 运行前面步骤中的文件后，将创建 Unreal Engine 解决方案文件。UE5.sln打开 Visual Studio，然后在 Visual Studio 编辑器中打开该UE5.sln文件。
6. 在 Visual Studio 中，打开查看菜单，然后选择解决方案资源管理器选项。这将打开虚幻项目节点的快捷菜单。在解决方案资源管理器窗口中，右键单击 UE5.sln 文件（可以将其列为只列出 UE5），然后选择构建，使用开发编辑器 Win64 目标构建 Unreal 项目。

> **Note**
>
> 此教程可在 1 个小时内完成。

构建完成后，您就可以打开虚幻开发编辑器并创建或导入项目了。

为插件配置您的虚幻项目

按照以下步骤为你的游戏 GameLift 服务器项目准备好适用于虚幻引擎的Amazon服务器SDK插件。

为插件配置项目

1. 打开Visual Studio后，导航到解决方案资源管理器窗格并选择UE5文件以打开虚幻项目的快捷菜单。在上下文菜单中，选择设置为启动项目选项。
2. 在 Visual Studio 窗口的顶部，选择开始调试（绿色箭头）。

   这个动作会启动您新的源代码构建的虚幻编辑器实例。有关使用虚幻编辑器的更多信息，请参阅 Unreal Engine 文档网站上的虚幻编辑器界面。
3. 关闭您打开的 Visual Studio 窗口，因为虚幻编辑器会打开另一个包含虚幻项目和您的游戏项目的 Visual Studio窗口。
4. 在 编辑器中，执行以下操作之一：
   - 选择要与Amazon集成的现有虚幻项目 GameLift。
   - 创建新 项目 要尝试使用适用于虚幻引擎的Amazon GameLift 插件，请尝试使用虚幻引擎的第三人称视角模板。有关此模板的更多信息，请参阅 Unreal Engine 文档网站上的第三人称视角模板。

     或者，使用以下设置配置新项目：

     - C++

- 包含入门内容
- Desktop
- Project Name 在本主题的示例中，我们命名了我们的项目GameLiftUnrealApp。

5. 在Visual Studio的解决方案资源管理器中，导航到您的虚幻项目所在的位置。在虚幻Source文件夹中，找到一个名为的文件*Your-application-name*.Target.cs。

   例如：GameLiftUnrealApp.Target.cs。

6. 为文件创建一个副本并将该副本命名为 *Your-application-name*Server.Target.cs。
7. 打开新文件并进行以下更改：

   - 更改class和constructor以匹配文件名。
   - 将 Type 从 TargetType.Game 更改为 TargetType.Server。
   - 最终文件将类似于以下示例：

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

您的项目现已配置为接受 Amazon GameLift 服务器 SDK 插件。

下一个任务是为 Unreal Engine 构建 C++ 服务器软件开发工具包 库，这样您就可以将它们导入到您的项目中。

构建适用于 Unreal 的 C++ 服务器软件开发工具包库。

1. 下载适用于虚幻引擎的 Amazon GameLift C++ 服务器 SDK 插件。

> **Note**
>
> 将软件开发工具包 放在默认下载目录可能会导致构建失败，因为路径超过 260 个字符的限制。例如：`C:\Users\Administrator\Downloads\GameLift-SDK-Release-06_15_2023\GameLift-Cpp-ServerSDK-5.0.4`
> 例如，我们建议您将软件开发工具包移到另一个目录`C:\GameLift-Cpp-ServerSDK-5.0.4`。

2. 下载并安装 Maven。有关下载 OpenSSL 的更多信息，请阅读 Github Op enSSL 构建和安装文档。

   有关更多信息，请阅读适用于 Windows 平台的 OpenSSL 注意事项文档。

   > **Note**
   >
   > 你用来构建亚马逊 GameLift 服务器软件开发工具包的 OpenSSL 版本应与虚幻引擎用来打包游戏服务器的 OpenSSL 版本相匹配。你可以在虚幻安装目录中找到版本信息`...Engine\Source\ThirdParty\OpenSSL`。

3. 下载库后，为 Unreal Engine 构建 C++ 服务器软件开发工具包 库。

   在下载的软件开发工具包 的`GameLift-Cpp-ServerSDK-<version>`目录中，使用`-DBUILD_FOR_UNREAL=1`参数进行编译并构建服务器软件开发工具包。以下示例演示了如何使用 cmake 命令。

   在计算机终端中运行以下命令。

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

   Windows 版本将在该`out\gamelift-server-sdk\Release`文件夹中创建以下二进制文件：

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

将这两个库文件复制到亚马逊 GameLift 虚幻引擎插件包中的ThirdParty\GameLiftServerSDK\Win64文件夹。

使用以下步骤将 Amazon GameLift 插件导入您的示例项目。

导入 Amazon GameLift 插件

1. 找到您在前面的过程中从插件中提取GameLiftServerSDK的文件夹。
2. Plugins在您的游戏项目根文件夹中找到。（如果该文件夹不存在，则在那里创建。）
3. 将该GameLiftServerSDK文件夹复制到Plugins。

   这将允许虚幻项目看到该插件。
4. 将 Amazon GameLift 服务器 SDK 插件添加到游戏.uproject文件中。

   在示例中，应用程序被调用GameLiftUnrealApp，因此文件将被调用GameLiftUnrealApp.uproject。
5. 编辑.uproject文件以将插件添加到您的游戏项目中。

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

6. 确保游戏依赖 ModuleRules 于插件。打开.Build.cs文件并添加 Amazon GameLiftServer SDK 依赖项。此文件位于 *Your-application-name*/Source/*/Your-application-name/*。

   例如，教程的文件路径是../GameLiftUnrealApp/Source/GameLiftUnrealApp/GameLiftUnrealApp.Build.cs。
7. "GameLiftServerSDK"添加到列表的末尾PublicDependencyModuleNames.

```
using UnrealBuildTool;
using System.Collections.Generic;
public class GameLiftUnrealApp : ModuleRules
  {
    public GameLiftUnrealApp(TargetInfo Target)
    {
```

```
        PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject",
"Engine", "InputCore", "GameLiftServerSDK" });
        bEnableExceptions = true;
    }
}
```

该插件现在应该可以用于您的应用程序。继续下一节，将Amazon GameLift 功能集成到您的游戏中。

将 Amazon GameLift 服务器代码添加到你的虚幻项目中

您已经配置并设置了虚幻引擎环境，现在可以将游戏服务器与Amazon集成 GameLift。本主题中介绍的代码对 Amazon GameLift 服务进行必需的调用。它还实现了一组回调函数，用于响应 Amazon GameLift 服务的请求。有关每个函数以及代码作用的更多信息，请参阅初始化服务器进程。有关此代码中使用的软件开发工具包 操作和数据类型的更多信息，请阅读。 Unreal Engine 的 Amazon GameLift 服务器软件开发工具包 参考

要使用 Amazon 初始化游戏服务器 GameLift，请按以下步骤操作。

> **Note**
>
> 下一节中提供的 GameLift Amazon 特定代码取决于`WITH_GAMELIFT`预处理器标志的使用。仅当满足以下两个条件时，此标志才为真：
>
> - `Target.Type == TargetRules.TargetType.Server`
> - 这些插件找到了 Amazon GameLift 服务器 SDK 二进制文件。
>
> 这样可以确保只有虚幻服务器版本才能调用亚马逊 GameLift的后端API。它还允许您编写能够针对您的游戏可能产生的所有不同虚幻目标正确执行的代码。

将游戏服务器与 Amazon 集成 GameLift

1. 在 Visual Studio 中，打开您的应用程序的`.sln`文件。在我们的示例中，该文件`GameLiftUnrealApp.sln`位于根文件夹中。
2. 打开解决方案后，找到应用程序的*`Your-application-name`*`GameMode.h`文件。示例：`GameLiftUnrealAppGameMode.h`。
3. 更改头文件以使其与以下示例代码保持一致。请务必用您自己的应用程序名称替换 GameLiftUnrealApp “”。

```
#pragma once

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

DECLARE_LOG_CATEGORY_EXTERN(GameServerLog, Log, All);

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

public:
    AGameLiftUnrealAppGameMode();

protected:
    virtual void BeginPlay() override;

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

    void InitGameLift();
};
```

4. 打开相关的源文件*Your-application-name*GameMode.cpp文件。在我们的示例:GameLiftUnrealAppGameMode.cpp. 中，更改代码以与以下示例代码保持一致。请务必用您自己的应用程序名称替换 GameLiftUnrealApp “”。

   此示例演示如何添加与亚马逊集成所需的所有元素 GameLift，如将亚马逊 GameLift添加到您的游戏服务器中所述。这包括：

   - 初始化亚马逊 GameLift API 客户端。
   - 实现回调函数以响应 Amazon GameLift 服务的请求OnStartGameSession，包括OnProcessTerminate、和onHealthCheck。
   - 使用指定端口调用 ProcessReady ()，以便在准备举办游戏会话 GameLiftservice 时通知 Amazon。

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

#include "UObject/ConstructorHelpers.h"

DEFINE_LOG_CATEGORY(GameServerLog);

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

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

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

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

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

    //AuthToken returned from the "aws gamelift get-compute-auth-token" API. Note
 this will expire and require a new call to the API after 15 minutes.
    if (FParse::Value(FCommandLine::Get(), TEXT("-authtoken="),
 serverParameters.m_authToken))
    {
```

```
        UE_LOG(GameServerLog, Log, TEXT("AUTH_TOKEN: %s"),
*serverParameters.m_authToken)
    }

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

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

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

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

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

    //Implement callback function onStartGameSession
    //GameLift sends a game session activation request to the game server
    //and passes a game session object with game properties and other settings.
    //Here is where a game server takes action based on the game session object.
    //When the game server is ready to receive incoming player connections,
    //it invokes the server SDK call ActivateGameSession().
```

```
    auto onGameSession = [=](Aws::GameLift::Server::Model::GameSession gameSession)
    {
        FString gameSessionId = FString(gameSession.GetGameSessionId());
        UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"),
*gameSessionId);
        gameLiftSdkModule->ActivateGameSession();
    };

    m_params.OnStartGameSession.BindLambda(onGameSession);

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

    m_params.OnTerminate.BindLambda(onProcessTerminate);

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

    m_params.OnHealthCheck.BindLambda(onHealthCheck);

    //The game server gets ready to report that it is ready to host game sessions
    //and that it will listen on port 7777 for incoming player connections.
    m_params.port = 7777;
```

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

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

5. 为以下两种目标类型构建游戏项目：开发编辑器和开发服务器。

> **Note**
>
> 您不需要重新构建解决方案。相反，只在与您的应用程序名称相匹配的Games文件夹下构建项目。否则，Visual Studio 会重建整个 UE5 项目，这可能需要长达一个小时的时间。

6. 两个构建都完成后，关闭Visual Studio并打开项目`.uproject`文件以在虚幻编辑器中将其打开。
7. 在虚幻编辑器中，打包游戏的服务器版本。要选择目标，请前往 “平台”、“Windows”，然后选择 “***Your-application-name ###***”。
8. 要开始构建服务器应用程序的过程，请转到平台、Windows，然后选择Package Project。构建完成后，您应该有一个可执行文件。在我们的示例中，文件名为`GameLiftUnrealAppServer.exe`。
9. 在虚幻编辑器中构建服务器应用程序会生成两个可执行文件。一个位于游戏构建文件夹的根目录中，用作实际服务器可执行文件的封装器。

   使用您的服务器版本创建 Amazon GameLift 队列时，我们建议您传入实际的服务器可执行文件作为运行时配置启动路径。例如，在您的游戏版本文件夹中，您的根目录可能有一个`GameLiftFPS.exe`文件，另一个位于根目录`\GameLiftFPS\Binaries\Win64\GameLiftFPSServer.exe`。创建队列时，我们建议您使用`C:\GameLiftFPS\Binaries\Win64\GameLiftFPSServer.exe`作为运行时配置的启动路径。
10. 确保在 Amazon GameLift 舰队上打开必要的 UDP 端口，以便游戏服务器可以与游戏客户端通信。默认情况下， Unreal Engine 使用端口7777。有关更多信息，请参阅 UpdateFleetPortSettingsAmazon GameLift 服务 API 参考指南。
11. 为您的游戏版本创建`install.bat`文件。每当游戏版本部署到 Amazon GameLift 舰队时，都会运行此安装脚本。下面给出了一个示例文件：

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

对于某些版本的虚幻引擎，install.bat应该改为

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

> **Note**
>
> 该文件的文件路径是Engine\Extras\Redist\en-us。<>PrereqSetup_x64.exe

12. 现在，您可以将游戏版本打包并上传到 Amazon GameLift。

    您在游戏版本中打包的 OpenSSL 版本需要与游戏引擎在构建游戏服务器时使用的版本相匹配。确保在游戏服务器版本中打包正确的 OpenSSL 版本。对于 Windows 操作系统，OpenSSL 格式为。.dll

    > **Note**
    >
    > Package 将 OpenSSL DLL 打包到你的游戏服务器版本中。请务必打包与您在构建游戏服务器时使用的相同版本的 OpenSSL。
    >
    > - libssl-1_1-x64.dll
    >
    >   libcrypto-1_1-x64.dll
    >
    > Package 将依赖项和游戏服务器可执行文件一起打包到 zip 文件的根目录中。例如，openssl-libdll 应与.exe文件位于同一目录中。

后续步骤

你已经配置并设置了虚幻引擎环境，现在可以开始将Amazon GameLift 集成到你的游戏中了。

有关在游戏中添加 Amazon GameLift 的更多信息，请参阅以下内容：

- 将 Amazon GameLift 添加到您的游戏服务器

- Unreal Engine 的 Amazon GameLift 服务器软件开发工具包 参考

有关测试游戏的说明，请参阅 使用 Amazon GameLift Anywhere 实例集测试您的集成 。

将 Amazon GameLift 集成到 Unity 项目中

本主题介绍如何设置适用于 Unity 的 Amazon GameLift C# 服务器软件开发工具包插件并将其集成到您的游戏项目中。

其他资源

- Amazon GameLift 服务器软件开发工具包下载网址
- 适用于 C# 和 Unity 的 Amazon GameLift 服务器软件开发工具包 5.x 参考
- the section called “ 开发 Linux 支持”

先决条件

要使用 Unity 的游戏 GameLift C# server软件开发工具包 插件，需要以下几个组件：

- 该插件支持的开发环境和 Unity 编辑器版本（请参阅Amazon 为开发提供支持 GameLift）。有关 Unity 版本的信息，请参阅 Unity 文档中的 Unity 系统要求。
- 适用于 Unity 软件包的 Amazon GameLift 服务器软件开发工具包 插件。此软件包包括适用于 C# 的服务器软件开发工具包 5+。您可以从这个网站下载软件包：Amazon GameLift 入门。
- 第三方限定了注册表 UnityNuGet。此工具管理第三方 DLL。有关更多信息，请参阅 UnityNuGet Github 存储库。

设置 UnityNuget

如果您没有为游戏项目设置 UnityNuGet，请按照以下步骤使用 Unity 包管理器安装该工具。或者，您可以使用 NuGet CLI 手动下载 DLL。有关更多信息，请参阅适用于 Unity README 的 Amazon GameLift C# 服务器软件开发工具包。

将 UnityNuGet 集成到游戏项目中

1. 在 Unity 编辑器中打开项目后，进入主菜单并选择编辑、项目设置。从选项中选择包管理器部分，然后打开范围界定的注册表组。
2. 选择 + 按钮，然后为 UnityNuGet 范围内的注册表输入以下值：

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

3. 对于 Unity 2021 版本的用户：

   设置 UnityNuGet 后，请检查 Unity 控制台中是否显示 `Assembly Version Validation` 错误。如果 NuGet 包中强命名程序集的绑定重定向未正确解析到 Unity 项目中的路径，则会发生这些错误。要解决这一问题，请配置 Unity 的程序集版本验证：

   a. 在 Unity 编辑器中，进入主菜单并选择 编辑、项目设置，然后打开玩家部分。

   b. 取消选择程序集版本验证选项。

安装插件

使用以下过程安装适用于 Unity 的 Amazon GameLift C# 服务器软件开发工具包 插件并配置 log4net 日志记录。

安装该插件

1. 在 Unity 编辑器中打开项目后，进入主菜单并选择窗口、包管理器。
2. 选择 + 按钮添加新软件包。选择从 tarball 添加软件包选项。
3. 在选择磁盘上的软件包中，找到适用于 Unity 的 Amazon GameLift C# Server软件开发工具包 插件下载文件，然后选择 Amazon GameLift Server软件开发工具包 文件。选择打开以安装插件。

Amazon GameLift 服务器软件开发工具包 使用 log4net 框架输出日志消息。默认情况下，它配置为向服务器版本的终端输出消息，但是 Unity 需要配置才能添加文件日志记录支持。您可以通过将提供的示例导入 Amazon GameLift Server软件开发工具包 包中来为您的项目添加此支持。按照以下过程添加示例并配置 log4net：

为文件输出配置 log4net

1. 在 Unity 编辑器中打开项目后，进入主菜单并选择窗口、包管理器。
2. 从下拉菜单中选择软件包：在项目中，然后从软件包列表中选择 Amazon GameLift 服务器软件开发工具包。这将打开包裹详情。
3. 在包裹详细信息中，选择样品组选项，然后按导入。

4. log4net.config 文件和随附的 LoggingConfiguration.cs 脚本会自动执行配置，该配置现在已设置在项目的 Assets/Samples 文件夹中。

> **Note**
>
> 如果您需要将 log4net.config 文件移动到项目中的其他文件夹，则还必须使用新路径更新脚本 LoggingConfiguration.cs 中配置文件的文件路径。有关更多信息，请参阅有关配置 log4net 的 log4net 手册。

有关更详细的说明和测试指南，请参阅插件下载中的README。

设置 Amazon GameLift Anywhere 实例集进行测试

您可以将开发工作站设置为 Amazon GameLift Anywhere 托管队列，以迭代测试您的 Amazon GameLift 集成。通过此设置，您可以在工作站上启动游戏服务器进程，向 Amazon GameLift 发送玩家加入或对战请求以开始游戏会话，并将客户端连接到新的游戏会话。将自己的工作站设置为托管服务器后，您可以监控游戏与 Amazon GameLift 集成的各个方面。

有关设置工作站的说明，请参阅使用 Amazon GameLift Anywhere 实例集测试您的集成以完成以下步骤：

1. 为您的工作站创建自定义位置。
2. 使用您的新自定义位置创建 Amazon GameLift Anywhere 实例集。如果成功，此请求将返回实例集 ID。记下 ARN，稍后您将用到它。
3. 将您的工作站注册为新Anywhere队列中的计算设备。为您的工作站提供唯一的计算名称并指定 IP 地址。如果成功，此请求将以 WebSocket 网址的形式返回服务软件开发工具包 端点。记下 ARN，稍后您将用到它。
4. 为您的工作站计算生成身份验证令牌。这种短暂的身份验证包括令牌和到期日期。您的游戏服务器使用它来验证与 Amazon GameLift 服务的通信。将身份验证存储在您的工作站计算机上，以便正在运行的游戏服务器进程可以对其进行访问。

将 Amazon GameLift 服务器代码添加到您的 Unity 项目中

您的游戏服务器与 Amazon GameLift 服务进行通信，以接收指令并报告持续状态。为此，您需要添加使用 Amazon GameLift 服务器软件开发工具包 的游戏服务器代码。

提供的代码示例说明了所需的基本集成元素。它使用MonoBehavior来说明使用 Amazon GameLift 进行简单的游戏服务器初始化。该示例假设游戏服务器在 Amazon GameL Anywhere ift 队列上运行以进行测试。它包括以下代码：

- 初始化 Amazon GameLift API 客户端。该示例使用InitSDK()带有服务器参数的版本进行Anywhere队列和计算。使用上一主题中定义的 WebSocket 网址、队列 ID、计算名称（主机 ID）和身份验证令牌。设置 Amazon GameLift Anywhere 实例集进行测试
- 实现回调函数以响应来自 Amazon GameLift 服务的请求，OnStartGameSession包括OnProcessTerminate、和。onHealthCheck
- 使用指定端口调用 ProcessReady ()，以便在进程准备好托管游戏会话时通知 Amazon GameLift 服务。

本主题中介绍的代码建立了与Amazon GameLift服务的通信，并且. 它还实现了一组回调函数，用于响应来自的请求。有关每个函数以及代码作用的更多信息，请参阅初始化服务器进程。有关此代码中使用的软件开发工具包 操作和数据类型的更多信息，请阅读适用于 C# 的 Amazon GameLift 服务器软件开发工具包 参考。

此示例演示如何添加所有必需元素，如将 Amazon GameLift 添加到游戏服务器中所述。它包括：

有关添加 功能的更多信息，请参阅以下主题：

- 将 Amazon GameLift 添加到您的游戏服务器
- 适用于 C# 的 Amazon GameLift 服务器软件开发工具包 参考

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

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

        //WebSocketUrl from RegisterHost call
```

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

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

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

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

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

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

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

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

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

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

其他资源

使用以下资源来测试您的游戏服务器并扩展其功能：

- 将您的开发计算机设置为 Amazon GameLift Anywhere 实例集，然后将其用于本地测试。请参阅测试您的自定义服务器集成。
- 构建您的游戏服务器并将版本上传到 Amazon GameLift。参阅将自定义服务器构建上传到 Amazon GameLift。
- 将您的游戏服务器版本部署到 Amazon GameLift 托管 EC2 队列中。参见创建新的 Amazon GameLift 实例集。

使用 Amazon GameLift Anywhere 实例集测试您的集成

您可以使用 Amazon GameLift Anywhere 实例集来迭代构建和测试游戏与 Amazon GameLift 的集成。将您自己的硬件设置为与 Amazon GameLift 服务相连的Anywhere实例集，然后在其上安装并运行您的游戏服务器。使用测试应用程序运行场景，例如开始/停止游戏会话、跟踪玩家连接以及处理对战回填等。有了Anywhere实例集，您可以根据需要更新游戏服务器版本，并全面了解托管活动。

您可以使用与 Amazon GameLift 服务器软件开发工具包版本 5 或更高版本集成的游戏的 Amazon GameLift 实例集。

主题

- 初始开发
- 在游戏服务器上进行迭代

初始开发

您已经开发了自己的游戏并正在将其与 Amazon GameLift 服务器软件开发工具包 集成。要测试您的集成，您可以将游戏服务器版本的每个新版本上传到 Amazon GameLift 并创建队列。或者，在开发笔记本电脑上使用Anywhere实例集可以更有效地进行迭代开发和测试。

使用以下过程创建Anywhere队列并使用 Amazon GameLift 主机或 AWS Command Line Interface () 在笔记本电脑上开始游戏会话。AWS CLI

Console

1. 打开 Amazon GameLift 主机。
2. 在导航窗格的托管下，选择位置。
3. 选择创建位置。
4. 在创建位置对话框中，执行以下操作：

a. 输入位置名称。这会标记 Amazon GameLift 用于在队列中运行游戏的计算资源的位置。Anywhere自定义位置名称必须以 c ustom- 开头。

b. 选择创建。

5. 要创建Anywhere，可以执行以下操作：

a. 在导航窗格中，选择托管下的实例集。

b. 在实例集页面上，选择创建实例集。

c. 在选择计算类型步骤中，选择 Anywhere，然后选择下一步。

d. 在定义实例集详细信息步骤中，定义您的新实例集。有关更多信息，请参阅创建新的 Amazon GameLift 实例集。

e. 在选择位置步骤中，选择您创建的自定义位置。

f. 完成剩余的实例集创建步骤以创建您的Anywhere实例集。

6. 在您创建的队列中将您的笔记本电脑注册为计算资源。使用register-compute命令（或RegisterComputeAPI 操作）。包括上一步中fleet-id创建的，然后添加 a compute-name 和您的笔记本电脑ip-address。

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

输出示例：

```
Compute {
    FleetId = fleet-1234,
    ComputeName = DevLaptop,
    Status = ACTIVE,
    IpAddress = 10.1.2.3,
    GameLiftServiceSdkEndpoint = wss://12345678.execute-api.amazonaws.com/,
    Location = custom-location-1
}
```

7. 启动游戏服务器的调试会话。

a. 在您创建的机群中获取笔记本电脑的授权令牌。使用get-compute-auth-token命令（或 GetComputeAuthTokenAPI 操作）。

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

输出示例：

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

b. 运行游戏服务器可执行文件的调试实例。要运行调试实例，您的游戏服务器必须调用InitSDK()。在进程准备好托管游戏会话后，游戏服务器会调用ProcessReady()。

8. 创建游戏会话来测试您与 Amazon Game Anywhere Lift 的首次集成。使用create-game-session命令（或 CreateGameSessionAPI 操作）。指定实例集的自定义位置。

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

输出示例：

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

Amazon GameLift 会向您的注册服务器进程发送 onStartGameSession() 消息。该消息包含上一步中的 GameSession 对象，其中包含游戏属性、游戏会话数据、对战构建器数据以及有关游戏会话的更多信息。

9. 向游戏服务器添加逻辑，以便服务器进程使用 `ActivateGameSession()` 响应 `onStartGameSession()` 消息。该操作会向 Amazon GameLift 发送确认消息，表明您的服务器已收到并接受了创建游戏会话消息。有关更多信息，请参阅Amazon GameLift 服务器软件开发工具包参考。

您的游戏服务器现在正在运行游戏会话，供您测试并用于迭代。要了解如何在游戏服务器上进行迭代，请继续执行下一部分。

AWS CLI

1. 使用`create-location`命令（或 `CreateLocation`API 操作）创建自定义位置。自定义位置会标记 Amazon GameLift 用于在实例集中运行游戏的硬件位置。Anywhere

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

输出示例：

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

2. 使用`create-fleet`命令（或 `CreateFleet`API 操作）使用您的自定义位置创建Anywhere实例集。Amazon GameLift 会在您的家乡和您提供的自定义位置创建实例集。

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

输出示例：

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

```
}
```

3. 在您创建的队列中将您的笔记本电脑注册为计算资源。使用register-compute命令（或RegisterComputeAPI 操作）。包括上一步中fleet-id创建的，然后添加compute-name和您的笔记本电脑的公共场景ip-address。

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

输出示例：

```
Compute {
    FleetId = fleet-1234,
    ComputeName = DevLaptop,
    Status = ACTIVE,
    IpAddress = 10.1.2.3,
    GameLiftServiceSdkEndpoint = wss://12345678.execute-api.amazonaws.com/,
    Location = custom-location-1
}
```

4. 启动游戏服务器的调试会话。

   a. 在您创建的机群中获取笔记本电脑的授权令牌。使用get-compute-auth-token命令（或 GetComputeAuthTokenAPI 操作）。

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

输出示例：

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

b. 运行游戏服务器可执行文件的调试实例。要运行调试实例，您的游戏服务器必须调用InitSDK()。在进程准备好托管游戏会话后，游戏服务器会调用ProcessReady()。

5. 创建游戏会话来测试您与 Amazon Game Anywhere Lift 的首次集成。使用create-game-session命令（或 CreateGameSessionAPI 操作）。

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

输出示例：

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

Amazon GameLift 会向您的注册服务器进程发送 onStartGameSession() 消息。该消息包含上一步中的 GameSession 对象，其中包含游戏属性、游戏会话数据、对战构建器数据以及有关游戏会话的更多信息。

6. 向游戏服务器添加逻辑，以便服务器进程使用 ActivateGameSession() 响应 onStartGameSession() 消息。该操作会向 Amazon GameLift 发送确认消息，表明您的服务器已收到并接受了创建游戏会话消息。有关更多信息，请参阅Amazon GameLift 服务器软件开发工具包参考。

您的游戏服务器现在正在运行游戏会话，供您测试并用于迭代。要了解如何在游戏服务器上进行迭代，请继续执行下一部分。

在游戏服务器上进行迭代

在此用例中，请考虑这样一个场景：您已经设置并测试了游戏服务器并发现了错误。借助 Amazon Anywhere GameLift，您可以对代码进行迭代，避免使用 Amazon EC2 队列的繁重设置。

1. 如果可能GameSession，请清理现有的。如果游戏服务器崩溃或无法调用ProcessEnding()，Amazon GameLift 会在游戏服务器停止发送健康GameSession检查后进行清理。
2. 对游戏服务器进行代码更改，进行编译，为下一次测试做准备。
3. 您之前的Anywhere队列仍处于活动状态，并且您的笔记本电脑仍被注册为队列中的计算资源。要重新开始测试，请创建一个新的调试实例。

   a. 在您创建的队列中检索笔记本电脑的授权令牌。使用get-compute-auth-token命令（或GetComputeAuthTokenAPI 操作）。

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

   输出示例：

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

   b. 运行游戏服务器可执行文件的调试实例。要运行调试实例，您的游戏服务器必须调用InitSDK()。在进程准备好托管游戏会话后，游戏服务器会调用ProcessReady()。

4. 您的实例集现在有了可用的服务器进程。创建您的游戏会话并执行下一个测试。使用create-game-session命令（或CreateGameSessionAPI 操作）。

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

   Amazon GameLift 会向您的注册服务器进程发送 onStartGameSession() 消息。该消息包含上一步中的 GameSession 对象，其中包含游戏属性、游戏会话数据、对战构建器数据以及有关游戏会话的更多信息。

5. 向游戏服务器添加逻辑，以便服务器进程使用 ActivateGameSession() 响应 onStartGameSession() 消息。该操作会向 Amazon GameLift 发送确认消息，表明您的服务器

已收到并接受了创建游戏会话消息。有关更多信息，请参阅Amazon GameLift 服务器软件开发工具包参考。

测试完游戏服务器后，您可以继续使用 Amazon GameLift 来管理您的实例集和游戏服务器。有关更多信息，请参阅创建亚马逊 GameLift Anywhere舰队。

使用 Amazon GameLift Local 测试您的集成

> **Note**
>
> 如果您使用的是版本 4.x 或更早版本的 Amazon GameLift 服务器软件开发工具包，请使用此测试程序。您的服务器软件开发工具包 包中包含兼容版本的 Amazon GameLift Local。如果您使用的是服务器软件开发工具包 版本 5.x，使用 Amazon GameLift Anywhere 实例集测试您的集成请参阅 Amazon GameLift Anywhere 队列进行本地测试。

使用 Local 可以在本地设备上运行托管 服务的有限版本，并在其上测试您的游戏集成。此工具在对您的游戏集成进行迭代开发时非常有用。其替代方法是将每个新生成的游戏上传到 &AGSlong; 并配置实例集来托管游戏，每次都需要 30 分钟或更长时间。

通过 Local，您可以验证以下内容：

- 您的游戏服务器与服务器开发工具包正确集成，并且正确与 服务通信，可启动新游戏会话、接受新玩家和报告运行状况及状态。
- 您的游戏客户端与适用于 AWS 的 AWS 开发工具包正确集成，可以检索现有游戏会话的信息，启动新游戏会话，让玩家加入游戏并连接到游戏会话。

Local 是一个命令行工具，可启动托管 服务的独立版本。 Local 还提供了服务器进程初始化、运行状况检查以及 API 调用和响应的运行事件日志。Amazon GameLift Local 可识别 Amazon GameLift 的软件开发工具包AWS操作的子集。您可以从 AWS CLI 或者从游戏客户端进行调用。在本地执行的所有 API 操作与在 Web 服务上执行时完全一样。

每个服务器进程只能托管一个游戏会话。游戏会话是您用来连接到 Amazon GameLift Local 的可执行文件。游戏会话完成后，您应该调用`GameLiftServerSDK::ProcessEndning`然后退出该进程。使用 Amazon GameLift Local 进行本地测试时，您可以启动多个服务器进程。每个进程都将连接到 Amazon GameLift Local。然后，您可以为每个服务器进程创建一个游戏会话。当您的游戏会话结束时，您的游戏服务器进程应该退出。然后，必须手动启动另一个服务器进程。

Amazon GameLift 本地版支持以下 API：

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

Amazon GameLift Local

Local 作为与服务器开发工具包捆绑的可执行 文件提供。它可以在 Windows 或 Linux 上运行，并可用于任何 支持的语言。

在运行 Local 之前，您还必须已安装以下各项。

- Amazon GameLift Server Server软件开发工具包 版本 3.1.5 至 4.x 的版本。
- Java 8

测试游戏服务器

如果您只要测试游戏服务器，则可以使用 来模拟游戏客户端对 Local 服务的调用。这将验证您的游戏服务器是否按预期执行以下操作：

- 游戏服务器正确启动并初始化 服务器开发工具包。
- 在启动过程中，游戏服务器通知 ，服务器已准备好托管游戏会话。
- 在运行时，游戏服务器每分钟将运行状况发送到 。
- 游戏服务器响应请求，启动新游戏会话。

1. Amazon GameLift Local

   打开命令提示符窗口，导航到包含 *GameLiftLocal.jar* 文件的目录并运行它。默认情况下，Local 在端口 8080 上侦听来自游戏客户端的请求。要指定不同的端口号，请使用 -p 参数，如以下示例所示：

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

Local 启动之后，您可以查看日志，其中指示两个本地服务器已启动，一个列出您的游戏服务器，另一个列出您的游戏客户端或 AWS CLI。日志继续报告两个本地服务器上的活动，包括往返于游戏组件之间的通信。

2. 启动游戏服务器。

   在本地启动您的 集成游戏服务器。您无需更改游戏服务器的终端节点。

   在 Local 命令提示符窗口中，日志消息指示您的游戏服务器已连接至 Local 服务。这意味着游戏服务器已成功初始化 服务器开发工具包（使用 ）。它使用所示的日志路径调用 `ProcessReady()`，如果成功，则已准备好托管游戏会话。在游戏服务器运行期间， 记录来自各游戏服务器的各个运行状况报告。以下日志消息示例显示了成功集成的游戏服务器：

```
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
```

   潜在的错误和警告消息包括以下内容：

   - 错误：“ProcessReady did not find a process with pID: *<process ID>*! 是否已调用 InitSDK()？”
   - 警告：“Process state already exists for process with pID: *<process ID>*! 是否已多次调用 ProcessReady(...)？”

3. 启动 AWS CLI。

在您的游戏服务器成功调用 ProcessReady() 之后，您可以开始进行客户端调用。打开另一个命令提示符窗口并启动 AWS CLI 工具。默认情况下，AWS CLI 使用 Amazon GameLift Web 服务端点。您必须在使用 --endpoint-url 参数的每个请求中，使用 Local 终端节点覆盖此项，如以下示例请求中所示。

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

在 AWS CLI 命令提示符窗口中，AWS gamelift 命令会生成 AWS CLI 命令参考中所示的响应。

4. 创建游戏会话。

使用 AWS CLI 提交 CreateGameSession() 请求。该请求应采用预期的语法。对于 Local，FleetId 参数可以设置为任意有效字符串 (^fleet-\S+)。

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

在 Local 命令提示符窗口中，日志消息指示 Amazon GameLift Local 已向您的游戏服务器发送 onStartGameSession 回调。如果成功创建了游戏会话，您的游戏服务器通过调用 ActivateGameSession 来响应。

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

```
        "arn:aws:gamelift:local::gamesession/
fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d/gsess-abcdef12-3456-7890-abcd-
ef1234567890"
```

在 AWS CLI 窗口中，Amazon GameLift 使用包含游戏会话 ID 的游戏会话对象进行响应。请注意，新游戏会话的状态为“Activating”。游戏会话调用 ActivateGameSession 后，状态将更改为“Active”。如果您希望查看更改后的状态，请使用 AWS CLI 调用 `DescribeGameSessions()`。

```
{
    "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
    }
}
```

测试游戏服务器和客户端

要检查您的完整游戏集成，包括将玩家连接到游戏，您可以在本地运行游戏服务器和客户端。这使您可以测试从游戏客户端对 Local 进行的编程调用。您可以验证以下操作：

- 游戏客户端成功向 Amazon GameLift Local 服务发出 AWS 开发工具包请求，包括创建游戏会话、检索现有游戏会话上的信息以及创建玩家会话。
- 游戏服务器在玩家尝试加入游戏会话时，正确验证玩家。对于通过验证的玩家，游戏服务器可能会检索玩家数据 (如果已实施)。
- 游戏服务器在玩家离开游戏时报告断开连接。
- 游戏服务器报告结束游戏会话。

1. 启动 Amazon GameLift Local。

打开命令提示符窗口，导航到包含 *GameLiftLocal.jar* 文件的目录并运行它。默认情况下，Local 在端口 8080 上侦听来自游戏客户端的请求。要指定不同的端口号，请使用 -p 参数，如以下示例所示。

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

Local 启动之后，您可以查看日志，其中显示两个本地服务器已启动，一个列出您的游戏服务器，另一个列出您的游戏客户端或 AWS CLI。

2. 启动游戏服务器。

   在本地启动您的 Amazon GameLift 集成游戏服务器。有关消息日志的详细信息，请参阅测试游戏服务器。

3. 为 Local 配置您的游戏客户端并启用它。

   要将您的游戏客户端与 Amazon GameLift Local 服务一起使用，您必须对游戏客户端的设置进行以下更改，如在后端服务 GameLift 上设置 Amazon中所述：

   - 更改 `ClientConfiguration` 对象以指向您的 Local 终端节点，例如 `http://localhost:9080`。
   - 设置目标实例集 ID 值。对于 Local，您不需要实际实例集 ID；可以将目标实例集设置为任意有效字符串 (`^fleet-\S+`)，例如 `fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d`。
   - 设置 AWS 凭证。对于 Local，您无需实际的 AWS 凭证，您可以将访问密钥和私有密钥设置为任意字符串。

   在 Local 命令提示符窗口中，在您启动游戏客户端之后，日志消息应指示它已初始化 ，并且已成功与 服务通信。

4. 测试游戏客户端对 服务的调用。

   验证您的游戏客户端已成功进行任意或所有以下 API 调用：

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

在 Local 命令提示符窗口中，只有对 `CreateGameSession()` 的调用才会产生日志消息。日志消息显示 Amazon GameLift Local 何时提示您的游戏服务器启动游戏会话（`onStartGameSession` 回调）并在您的游戏服务器调用它时获取成功的 `ActivateGameSession`。在 AWS CLI 窗口中，记录导致响应或错误消息的所有 API 调用。

5. 确保您的游戏服务器正在验证新玩家连接。

   创建游戏会话和玩家会话之后，建立与游戏会话的直接连接。

   在 Local 命令提示符窗口中，日志消息应显示游戏服务器已发送 `AcceptPlayerSession()` 请求来验证新玩家连接。如果您使用 AWS CLI 调用 `DescribePlayerSessions()`，玩家会话状态应从“Reserved”更改为“Active”。

6. 验证您的游戏服务器正在将游戏和玩家状态报告给 Amazon GameLift 服务。

   要让 Amazon GameLift 管理玩家需求并正确报告指标，您的游戏服务器必须将各种状态报告回 Amazon GameLift。验证 Local 正在记录与以下操作相关的事件。您可能还希望使用 AWS CLI 跟踪状态更改。

   - 玩家从游戏会话断开连接 – Amazon GameLift Local 日志消息应显示游戏服务器调用了 `RemovePlayerSession()`。对 `DescribePlayerSessions()` 的 AWS CLI 调用应体现出状态从 `Active` 更改为 `Completed`。您还可以调用 `DescribeGameSessions()` 来检查游戏会话的当前玩家数减少了一个。
   - 游戏会话结束 – Amazon GameLift Local 日志消息应显示游戏服务器调用了 `TerminateGameSession()`。

     > **Note**
     >
     > 之前的指导是在结束游戏会话时调用 `TerminateGameSession()`。Amazon GameLift 服务器软件开发工具包 v4.0.1 不推荐使用此方法。请参阅结束游戏会话。

   - 服务器进程终止 – Amazon GameLift Local 日志消息应显示游戏服务器调用了 `ProcessEnding()`。对 `DescribeGameSessions()` 的 AWS CLI 调用应体现出状态从 `Active` 更改为 `Terminated`（或 `Terminating`）。

Local 的变化

使用 Amazon GameLift Local 时，请记住以下内容：

- 与 Amazon GameLift Web 服务不同，Local 不跟踪服务器的运行状况和启动 `onProcessTerminate` 回调。Local 仅停止记录游戏服务器的运行状况报告。
- 对于面向 AWS 软件开发工具包的调用，不验证实例集 ID，该 ID 可以是满足参数要求 (`^fleet-\S+`) 的任意字符串值。
- 使用 Local 创建的游戏会话 ID 具有不同结构。它们包括字符串 `local`，如此处所示：

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

将游戏与 Amazon GameLift 实时服务器集成

本主题提供了托管 及 解决方案的概述。概述说明了此解决方案何时适合您的游戏，以及实时服务器如何支持多人游戏。

有关使您的游戏在 上启动和正常运行的完整路线图，请参阅 。

> **Tip**
>
> 要试用 Amazon GameLift 游戏服务器托管，请参阅Amazon VPC 入门。

什么是实时服务器？

是由 提供以供您与多人游戏一起使用的轻量级的即用型游戏服务器。实时服务器取消了自定义游戏服务器的开发、测试和部署过程。此解决方案可以帮助最大限度地减少完成游戏所需的时间和精力。

主要特征

- 充分利用完整的网络堆栈进行游戏客户端/服务器交互。
- 核心游戏服务器功能。
- 可自定义的服务器逻辑。
- 实时更新 配置和服务器逻辑。
- FlexMatch 对战
- 灵活控制托管资源。

通过创建托管资源实例集并提供配置脚本来设置 服务器。在中了解更多有关创建 服务器和如何准备您的游戏客户端的信息。

实时服务器如何管理游戏会话

您可以选择通过将用于游戏会话管理的自定义逻辑内置到 脚本中来添加此逻辑。您可能编写代码来访问服务器特定的对象，使用回调添加事件驱动的逻辑或基于非事件方案（例如计时器或状态检查）添加逻辑。

实时客户端和服务器如何交互

在游戏会话期间，游戏客户端通过后端服务向实时服务器发送消息进行交互。然后，后端服务在游戏客户端之间中继消息，以交换活动、游戏状态和相关的游戏数据。

此外，您可以通过向 脚本添加游戏逻辑来自定义客户端和服务器的交互方式。借助自定义游戏逻辑，可能会实施回调以触发事件驱动的响应。

通信协议

服务器与连接的游戏客户端之间的通信使用两个通道：用于可靠传递的 TCP 连接和用于快速传递的 UDP 通道。创建消息时，游戏客户端选择使用的协议取决于消息的性质。默认情况下，消息传递设置为 UDP。如果 UDP 频道不可用，Amazon GameLift 会使用 TCP 作为后备发送消息。

消息内容

消息内容包含两种元素：必需的操作代码 (opCode) 和可选负载。消息的操作代码标识特定的玩家活动或游戏事件，而负载提供与操作代码相关的附加数据。这两种元素是开发人员定义的。您的游戏客户端会根据它收到的消息中的操作码采取措施。

玩家组

提供管理玩家组的功能。默认情况下，连接到游戏的所有玩家均置于“所有玩家”组中。此外，开发人员可为其游戏设置其他组，而玩家可以同时是多个组的成员。组成员可以向组中的所有玩家发送消息或与组共享游戏数据。组的一种可能用途是建立玩家团队并管理团队沟通。

带有 TLS 证书的实时服务器

在实时服务器中，服务器身份验证和数据包加密内置于服务中。当您打开 TLS 证书生成功能时，这些安全功能即会启用。当游戏客户端尝试与实时服务器连接时，服务器会自动使用客户端验证的 TLS 证书进行响应。加密是通过以下方式处理的：对于 TCP (Websocket) 通信使用 TLS，对于 UDP 流量使用 DTLS。

自定义实时服务器

实时服务器作为无状态中继服务器运行。服务器在连接到游戏的游戏客户端之间中继游戏数据包和消息。但是，实时服务器不评估消息、处理数据或执行任何游戏逻辑。以这种方式使用，每个游戏客户端均保持其自己的游戏状态的视图，并通过中继服务器向其他玩家提供更新。每个游戏客户端均负责合并这些更新并协调自己的游戏状态。

您可以通过添加实时脚本功能来自定义服务器。例如，对于游戏逻辑，您可选择使用游戏状态的服务器授权视图来构建有状态的游戏。

Amazon GameLift 为实时脚本定义了一组服务器端回调。实施这些回调将事件驱动的功能添加到服务器。例如，您可以：

- 当游戏客户端尝试连接到服务器时，对玩家进行身份验证。
- 在请求时验证玩家是否可加入组。
- 评估何时传递特定玩家或目标玩家的消息，或执行其他响应处理。
- 当玩家离开小组或与服务器断开连接时，请采取措施，例如通知所有玩家。
- 评估游戏会话对象或消息对象的内容并使用数据。

部署和更新实时服务器

的一个关键优势是能够随时更新您的脚本。更新脚本时，新版本将在几分钟内传播到所有托管资源。部署新脚本后，在此之后创建的所有新游戏会话都将使用新脚本版本。（现有游戏会话将继续使用原始版本。）

开始将您的游戏与 集成：

- 为实时服务器集成游戏客户端
- 创建实时脚本

为实时服务器集成游戏客户端

本主题介绍如何准备您的游戏客户端以便能够加入并参与 托管的游戏会话。

准备游戏客户端需要两组任务：

- 设置您的游戏客户端以获取有关现有游戏的信息、请求对战、启动新游戏会话以及为玩家预留游戏会话位置。

- 使您的游戏客户端能够加入服务器上托管的游戏会话并交换消息。

查找或创建游戏会话和玩家会话

设置您的游戏客户端以查找或启动游戏会话、请求 对战以及通过创建玩家会话为游戏中的玩家预留空间。作为最佳实操，请创建一个客户端服务，在某个游戏客户端操作触发该服务时使用它直接向 服务发出请求。然后，客户端服务会将相关响应中继回游戏客户端。

1. 将 AWS 软件开发工具包添加到您的客户端服务项目，初始化 AWS 客户端，并将其配置为使用您的实例集和/或队列中的托管资源。该 AWS提供多种语言；请参阅的 开发工具包。
2. 将 GameLift 功能添加到您的客户端服务。有关更多详细说明，请参阅将 Amazon GameLift 添加到您的游戏客户端和添加 FlexMatch 对战。最佳做法是使用游戏会话放置功能来创建新的游戏会话。此方法可让您充分利用 GameLift 快速而智能地放置新游戏会话的功能，并且可以利用玩家延迟数据来最大程度地减少游戏延迟。至少，您的客户端服务必须能够请求新的游戏会话，并处理响应中的游戏会话数据。您还可能需要添加功能，以搜索并获取现有游戏会话的信息和请求玩家会话，这会有效地保留现有游戏会话中的玩家位置。
3. 将连接信息传回游戏客户端。后端游戏服务接收游戏会话和玩家会话对象，以响应对 服务的请求。这些对象包含游戏客户端直接连接到运行在 Realtime Server 上的游戏会话所需的信息，尤其是连接详细信息（IP 地址和端口）以及玩家会话 ID。

在实时服务器上连接游戏

使您的游戏客户端能够与服务器上托管的游戏会话直接连接并与服务器和其他玩家交换消息。

1. 获取客户端软件开发工具包，构建它，并将其添加到您的游戏客户端项目。有关软件开发工具包要求的更多信息以及如何生成客户端库的说明，请参阅自述文件。
2. 使用指定要使用的客户端/服务器连接类型的客户端配置调用 Client()。

> **Note**
>
> 如果要使用 TLS 证书连接到在安全实例集上运行的实时服务器，则必须指定安全的连接类型。

3. 将以下功能添加到您的游戏客户端。有关更多信息，请参阅实时服务器客户端 API (C#) 参考。
   - 连接和断开连接游戏
     - Connect()

    - Disconnect()
- 将消息发送给目标收件人
    - SendMessage()
- 接收和删除消息
    - OnDataReceived()
- 加入组和离开玩家组
    - JoinGroup()
    - RequestGroupMembership()
    - LeaveGroup()

4. 根据需要设置客户端回调的事件处理程序。请参阅实时服务器客户端 API (C#) 参考：异步回调。

与启用了 TLS 证书生成的 Realtime 实例集一起使用时，将使用 TLS 证书自动对服务器进行身份验证。对正在传输的 TCP 和 UDP 通信进行加密，以提供传输层安全性。TCP 流量使用 TLS 1.2 进行加密，而 UDP 流量使用 DTLS 1.2 进行加密。

游戏客户端示例

基本 Realtime 客户端 (C #)

此示例说明与客户端软件开发工具包 (C#) 的基本游戏客户端集成。如下所示，该示例初始化客户端对象，设置事件处理程序并实施客户端回调，连接到服务器，发送消息，然后断开连接。

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

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

```
    {
        public Aws.GameLift.Realtime.Client Client { get; private set; }

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

        /// Initialize a client for GameLift Realtime and connect to a player session.
        /// <param name="endpoint">The DNS name that is assigned to Realtime server</
param>
        /// <param name="remoteTcpPort">A TCP port for the Realtime server</param>
        /// <param name="listeningUdpPort">A local port for listening to UDP traffic</
param>
        /// <param name="connectionType">Type of connection to establish between client
 and the Realtime server</param>
        /// <param name="playerSessionId">The player session ID that is assigned to the
 game client for a game session </param>
        /// <param name="connectionPayload">Developer-defined data to be used during
 client connection, such as for player authentication</param>
       public RealTimeClient(string endpoint, int remoteTcpPort, int listeningUdpPort,
 ConnectionType connectionType,
                     string playerSessionId, byte[] connectionPayload)
        {
            // Create a client configuration to specify a secure or unsecure connection
 type
            // Best practice is to set up a secure connection using the connection type
 RT_OVER_WSS_DTLS_TLS12.
            ClientConfiguration clientConfiguration = new ClientConfiguration()
      {
                // C# notation to set the field ConnectionType in the new instance of
 ClientConfiguration
          ConnectionType = connectionType
      };

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

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

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

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

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

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

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

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

```
        {
        }

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

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

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

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

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

```
        }
    }
}
```

创建实时脚本

要为您的游戏使用 ，您需要提供脚本（以一些 JavaScript 代码的格式）来配置并（可选）自定义 的实例集。本主题涵盖了创建 脚本的关键步骤。脚本准备就绪后，将它上传至 服务并使用它来创建实例集（请参阅）。

要准备用于 的脚本，请将以下功能添加到 脚本。

管理游戏会话生命周期（必需）

至少， 脚本必须包含 函数，该函数准备 服务器来启动游戏会话。同时还强烈建议您提供一种终止游戏会话的方式，以确保实例集上可以继续启动新游戏会话。

回调函数，该函数在调用时传递 会话对象，其中包含 服务器的接口。有关此接口的详细信息，请参阅实时服务器接口。

要顺利结束游戏会话，脚本还必须调用 服务器的 函数。这需要一些机制来确定何时结束会话。此脚本示例代码演示了一个简单的机制，该机制检查玩家连接，并在指定时间长度中没有玩家连接到会话时触发游戏会话中止。

具有最基本配置（服务器进程初始化和终止）的 实际上是充当无状态的中继服务器。 服务器中继连接到游戏的游戏客户端之间的消息和游戏数据，但不会采取独立操作来处理数据或执行逻辑。您可以根据需要选择性地为游戏添加游戏逻辑（由游戏事件或其他机制触发）。

添加服务器端游戏逻辑（可选）

您可以选择将游戏逻辑添加到 脚本。例如，您可以执行以下任一或所有操作。脚本示例代码提供了说明。请参阅Amazon GameLift 实时服务器脚本参考。

- 添加事件驱动的逻辑。实施回调函数来响应客户端-服务器事件。有关终端节点的完整列表，请参阅实时服务器的脚本回调。
- 通过发送消息到服务器来触发逻辑。为从游戏客户端发送到服务器的消息创建一组特殊操作代码，并添加函数来处理接收。使用回调 onMessage，并使用 gameMessage 接口解析消息内容（请参阅 gameMessage.opcode）。
- 启用游戏逻辑以访问您的其他AWS资源。有关详细信息，请参阅 与您的实例集中的其他 AWS 资源进行通信。

- 允许游戏逻辑访问其正在运行的实例的队列信息。有关详细信息，请参阅 获取 Amazon GameLift 实例的实例集数据。

实时服务器脚本示例

此示例说明了部署 以及一些自定义逻辑所需的基本脚本。它包含必需的 Init() 函数，并使用计时器机制，根据没有玩家连接的时间长度来触发游戏会话终止。它还包括一些针对自定义逻辑的挂钩，以及一些回调实施。

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

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

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

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

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

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

// Example groups for user-defined groups
```

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

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

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

    return true;
}

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

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

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

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

```
    return true;
}

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

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

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

// Handle a message to the server
function onMessage(gameMessage) {
    switch (gameMessage.opCode) {
      case OP_CODE_CUSTOM_OP1: {
        // do operation 1 with gameMessage.payload for example sendToGroup
        const outMessage = session.newTextGameMessage(OP_CODE_CUSTOM_OP1_REPLY,
 session.getServerId(), gameMessage.payload);
        session.sendGroupMessage(outMessage, RED_TEAM_GROUP);
```

```
        break;
      }
    }
}

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

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

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

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

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

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

```
        setTimeout(tickLoop, tickTime);
    }
}

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

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

将游戏与适用于 Unity 的 Amazon GameLift 插件集成

本节中的主题介绍适用于 Unity 的 Amazon GameLift 插件，以及如何使用它来准备在亚马逊托管的多人游戏项目 GameLift。使用插件的指导工作流程，完全在 Unity 开发环境中工作，以完成在 Amazon 上托管的基本要求 GameLift。

Amazon GameLift 是一项完全托管的服务，允许游戏开发者管理和扩展用于基于会话的多人游戏的专用游戏服务器。有关 Amazon GameLift 托管的更多信息，请参阅Amazon GameLift 的工作原理。

- 适用于 Unity 的亚马逊 GameLift 插件服务器指南 SDK 5.x，版本 2.0.0，适用于服务器 SDK 5.x 并支持 Amazon。 GameLift Anywhere
- 适用于 Unity 的亚马逊 GameLift 插件服务器指南 SDK 4.x，版本 1.0.0，适用于服务器 SDK 4.x 或更早版本。此版本使用 Amazon L GameLift ocal 进行集成测试。

适用于 Unity 的亚马逊 GameLift 插件服务器指南 SDK 5.x

Amazon GameLift 提供工具，让您的多人游戏服务器做好准备，以便与亚马逊配合使用 GameLift。Unity 版的亚马逊 GameLift 插件可以更轻松地将亚马逊 GameLift 集成到您的 Unity 游戏项目中，测试您与亚马逊的集成 GameLift Anywhere，以及为云托管部署亚马逊 GameLift 资源。

此插件使用AWS CloudFormation模板为常见游戏场景部署托管解决方案。按照提供的方式使用这些解决方案，或者根据游戏的需要对其进行自定义。

主题

- 关于插件
- 插件工作流
- 安装适用于 Unity 的插件
- 设置 AWS 用户配置文件
- 在 Amazon 上设置您的游戏以进行本地测试 GameLift Anywhere
- 使用托管 EC2 实例集将游戏部署到云托管

关于插件

Unity 插件为将您的 Unity 多人游戏与亚马逊集成和托管 Unity 多人游戏提供了简化的入门体验 GameLift。您可以利用插件功能和预先构建的组件来快速启动和运行游戏。

该插件为 Unity 编辑器添加了工具和功能。使用指导式工作流程将 Amazon GameLift 集成到您的游戏项目中，在本地对其进行测试，然后将游戏服务器部署到亚马逊 GameLift 云托管。

使用插件的预构建托管解决方案来部署您的游戏。以您的本地工作站为主机，设置 Amazon GameLift Anywhere 舰队。对于云托管，可以从两种常见的部署方案中进行选择，以不同的方式平衡玩家延迟、游戏会话可用性和成本。一种场景包括简单的 FlexMatch匹配器和规则集. 使用这些场景来制定基本的生产就绪型托管解决方案，然后根据需要进行优化和自定义。

该插件包括以下组件：

- Unity 编辑器的插件模块。安装插件后，新的主菜单项将允许您访问Amazon的 GameLift 功能。
- 适用于具有客户端功能的 Amazon GameLift 服务 API 的 C# 库。
- 适用于亚马逊 GameLift 服务器软件开发工具包的 C# 库（版本 5.x）。
- 示例游戏内容，包括资产和场景，因此 GameLift 即使您没有准备好构建的多人游戏，也可以试用 Amazon。

- 解决方案配置，作为AWS CloudFormation模板提供，供插件在将游戏服务器部署到云端进行托管时使用。

插件工作流

以下步骤描述了使用适用于 Unity 的 Amazon GameLift 插件集成和部署游戏项目的典型方法。您可以通过在 Unity 编辑器和游戏代码中工作来完成这些步骤。

1. 创建与您的AWS账户关联的用户个人资料，并为有权使用 Amazon 的有效账户用户提供访问凭证 GameLift。
2. 将服务器代码添加到您的游戏项目中，以便在正在运行的游戏服务器与 with Amazon GameLift 服务之间建立通信。
3. 将客户端代码添加到您的游戏项目中，允许游戏客户端向 Amazon 发送请求 GameLift 以开始或加入游戏会话，然后连接到游戏服务器。
4. 使用 Anywhere 工作流程将您的本地工作站设置为游戏服务器的 Anywhere 主机。在本地启动游戏服务器和客户端，连接到游戏会话，然后测试您的集成。
5. 使用 EC2 托管工作流程上传您的集成游戏服务器并部署云托管解决方案。游戏服务器准备就绪后，在本地启动游戏客户端，连接到游戏会话并登录，然后玩游戏。

在插件中工作时，您将创建和使用 AWS 资源，这些操作可能会对正在使用的 AWS 账户产生费用。如果您不熟悉AWS，则操作可能包含在AWS免费套餐中。

安装适用于 Unity 的插件

本节介绍如何将插件添加到 Unity 项目。安装插件后，在 Unity 编辑器中打开项目后，插件功能就可用。

开始之前

以下是使用适用于 Unity 的亚马逊 GameLift 插件所需的内容：

- 适用于 Windows 2022 的 Unity LTS 或者适用于 macOS 的 Unity
- 下载适用于 Unity 的亚马逊 GameLift 插件。[下载网站] 下载内容包括两个软件包：
  - 适用于 Unity 的亚马逊 GameLift 独立插件
  - 适用于 Unity 的亚马逊 GameLift C# 服务器 SDK
- Microsoft Visual Studio 2019 或更高版本。

- 一个包含 C# 游戏代码的多人游戏项目。
- 第三方范围内的注册表 UnityNuGet。此工具管理第三方 DLL。有关更多信息，请参阅 UnityNuGetGithub 存储库。

将插件添加到游戏项目。

在 Unity 编辑器和游戏项目文件中完成以下任务。

第 1 步： UnityNuGet 添加到你的游戏项目中

如果您尚未为游戏项目进行 UnityNuGet 设置，请按照以下步骤使用 Unity 包管理器安装该工具。或者，您可以使用 NuGet CLI 手动下载 DLL。有关更多信息，请参阅适用于 Unity 的亚马逊 GameLift C# 服务器 SDK README。

1. 在 Unity 编辑器中打开项目后，进入主菜单并选择编辑、项目设置。从选项中选择包管理器部分，然后打开范围界定的注册表组。
2. 选择 + 按钮，然后为限定 UnityNuGet 范围的注册表输入以下值：

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

对于 Unity 2021 版本的用户：

设置完成后 UnityNuGet，检查 Unity 控制台中是否显示`Assembly Version Validation`错误。如果 NuGet 包中强命名程序集的绑定重定向未正确解析到您的 Unity 项目中的路径，则会发生这些错误。要解决这一问题，请配置 Unity 的程序集版本验证：

1. 在 Unity 编辑器中，进入主菜单并选择 “编辑”、“项目设置”，然后打开 “播放器” 部分。
2. 取消选择程序集版本验证选项。

第 2 步：添加插件和 C# 服务器 SDK 包

1. 解压用于 Unity 下载的 Amazon GameLift 插件，其中包含两个软件包。
2. 在 Unity 编辑器中打开项目后，进入主菜单并选择 “窗口”、“Package Manager”。
3. 选择 + 按钮添加新软件包。选择从 tarball 添加软件包选项。

4. 在 “选择磁盘上的软件包” 中，找到适用于 Unity 的 Amazon GameLift C# Server SDK 插件下载文件，然后选择com.amazonaws.gameliftserver.sdk-<version>.tgz文件。选择打开以安装插件。
5. 在 “选择磁盘上的软件包” 中，找到用于 Unity 下载文件的 Amazon GameLift 独立插件，然后选择文件com.amazonaws.gamelift-<version>.tgz。选择打开以安装插件。
6. 确认独立插件已添加到您的项目中。返回到 Unity 编辑器窗口。在主菜单中查看新的A mazon GameLift 菜单按钮。

第 3 步：导入示例游戏（可选）

Unity 插件附带了一组示例游戏资产，包括场景，您可以将其添加到游戏项目中。通过导入示例游戏，您可以快速使用 Amazon 测试、构建和部署一款简单的多人游戏 GameLift。该示例游戏已与 Amazon GameLift SDK 完全集成，因此您可以跳过集成任务并完成剩余的工作流程任务。

使用示例游戏时，您可以在短短几分钟内设置并加入本地托管的 Amazon GameLift Anywhere 舰队。您可以将游戏部署到 Amazon， GameLift并在不到一小时的时间内加入云端托管的直播游戏。

要导入示例游戏，请执行以下操作：

1. 在 Unity 编辑器中打开游戏项目后，前往亚马逊 GameLift菜单并选择示例游戏、导入示例游戏。
2. 导入文件后，再次进入 Amazon GameLift 菜单，选择示例游戏，初始化设置。此步骤将配置您的项目以构建游戏客户端和服务器。

安装完成后，你会看到两个新场景添加到你的游戏项目中。您还将看到一些额外的项目资产，包括GameLiftClientSettings资产。

有关示例用户界面和游戏玩法的更多详细信息，请参阅示例游戏自述文件。

设置 AWS 用户配置文件

安装插件后，设置配置文件并将其关联到有效的 AWS 账户用户。您可以维护多个配置文件，但一次只能有一个配置文件处于活动状态。每当使用插件时，请选择要使用的配置文件。

维护多个配置文件使您能够在不同的托管方案之间切换。例如，您可以设置具有相同 AWS 凭证但 AWS 区域不同的配置文件。或者，您可以使用不同的 AWS 账户或不同的用户/权限集来设置配置文件。

> **Note**
>
> 如果您已在工作站上安装了 AWS CLI 并且已经配置了配置文件，则 Amazon GameLift 插件可以检测到它并将其列为现有配置文件。该插件会自动选择任何名为 `[default]` 的配置文件。您可以使用现有配置文件或创建新的配置文件。

要设置您的AWS个人资料

1. 在 Unity 编辑器主菜单中，选择 Amazon， GameLift然后选择设置AWS账户资料。此操作将打开插件窗口。打开 “AWS用户配置文件” 页面。
2. 如果插件检测到现有配置文件，则系统不会提示您创建个人资料。选择现有配置文件或选择 “添加其他配置文件” 来创建新的配置文件。
3. 如果插件未检测到现有的配置文件，则会提示您创建一个配置文件。您可以使用新账户或现有 AWS 账户创建新的配置文件。

   > **Note**
   >
   > 您需要使用 AWS 管理控制台创建新 AWS 账户，并使用适当的权限集创建或更新用户。

   在设置配置文件时，您需要提供以下信息：

   - 一个 AWS 账户。如果您需要创建新 AWS 账户，请按照提示创建账户。有关更多详细信息，请参阅创建 AWS 账户。
   - 有权使用 Amazon GameLift 和其他所需AWS服务的AWS账户用户。设置一个 AWS 账户有关设置具有 Amazon GameLift 权限的 AWS Identity and Access Management (IAM) 用户的说明，请参阅。
   - AWS 用户凭证 该用户还需要使用长期凭证进行编程访问。这些凭证包含 AWS 访问密钥和 AWS 私有密钥。有关更多详细信息，请参阅获取访问密钥。
   - AWS 区域。这是您要在其中创建托管 AWS 资源的地理位置。在开发过程中，我们建议使用靠近您的实际位置的区域，以最大限度地减少延迟。查看受支持 AWS 区域列表。
4. 选择或创建配置文件后，请检查配置文件的引导状态并根据需要采取措施。必须启动所有配置文件才能使用 Amazon GameLift 插件功能。

要引导您的配置文件，请执行以下操作：

Bootstrapping 会指定一个 Amazon S3 存储桶用于所选用户个人资料。Amazon S3 是一项用于数据和对象存储的核心AWS服务。用于存储项目配置、构建项目和其他依赖项的存储桶。存储桶不会在其他配置文件之间共享。

> **Note**
>
> Bootstrapping 会创建新的AWS资源，并且可能会产生成本。

1. 在插件窗口 “AWS用户配置文件” 中查看您的个人资料时，请选择要使用的配置文件。如果配置文件尚未被引导，则会显示一条警告消息。
2. 在引导您的配置文件部分，从下拉列表中选择一个配置文件并检查引导状态。如果状态显示不存在存储桶，请选择 Bootstrap 配置文件按钮。您可以将存储桶名称设置为新的存储桶名称，输入您有权访问的现有存储桶，或者保留自动生成的名称。
3. 等待引导状态变为“活动”。这可能需要几分钟的时间。当状态为 “激活” 时，您可以使用配置文件来使用插件功能

在 Amazon 上设置您的游戏以进行本地测试 GameLift Anywhere

在此工作流程中，您可以为 Amazon GameLift 功能添加客户端和服务器游戏代码，然后使用该插件将您的本地工作站指定为测试游戏服务器主机。完成集成任务后，使用该插件来构建您的游戏客户端和服务器组件。

要启动 Amazon GameLift Anywhere 工作流程，请：

- 在 Unity 编辑器主菜单中，选择 Amazon， GameLift然后选择 “随处托管”。此操作将打开插件页面，用于使用 @ Anywhere 舰队设置游戏。该页面提供了集成、构建和启动游戏组件的五步流程。

设置您的个人资料

选择您要在遵循此工作流程时使用的配置文件。您选择的配置文件会影响工作流程中的所有步骤。您创建的所有资源都与配置文件的 AWS 账户相关联，并放置在配置文件的默认 AWS 区域中。配置文件用户的权限决定了您对 AWS 资源和操作的访问权限。

1. 从可用配置文件的下拉列表中选择一个配置文件。如果您还没有个人资料或想要创建新的个人资料，请前往 Amazon GameLift 菜单并选择 “设置AWS账户资料”。
2. 如果引导状态不是“活动”，请选择引导配置文件并等待状态变为“活动”。

将您的游戏与 Amazon 集成 GameLift

> **Note**
>
> 如果您导入了示例游戏，则可以跳过此步骤。示例游戏资产已经准备好了必要的服务器和客户端代码。

在工作流程的这一步中，您需要更新游戏项目中的客户端和服务器代码。

- * 游戏服务器必须能够与 Amazon GameLift 服务通信，才能收到启动游戏会话的提示、提供游戏会话连接信息和报告状态。
- 游戏客户端必须能够获取有关游戏会话的信息、加入或开始游戏会话以及获取连接信息才能加入游戏。

集成您的服务器代码

如果您使用自己的游戏项目和自定义场景，请使用提供的示例代码将所需的服务器代码添加到您的游戏项目中：

1. 在游戏项目文件中，打开文件`Assets/Scripts/Server`夹。如果它不存在，请创建它。
2. 前往 aws/ GitHub 仓库amazon-gamelift-plugin-unity并打开路径。`Samples~/SampleGame/Assets/Scripts/Server`
3. 找到文件 GameLiftServer .cs. 并将其复制到游戏项目的服务器文件夹中。生成服务器可执行文件时，请使用此文件作为生成目标。

示例代码包括以下最低必需元素，它们使用 Amazon GameLift C# 服务器软件开发工具包（版本 5）：

- 初始化亚马逊 GameLift API 客户端。Amazon GameLift Anywhere 队列需要使用服务器参数进行`InitSDK()`调用。这些设置会自动设置为在插件中使用。
- 实现所需的回调函数以响应 Amazon GameLift 服务的请求`OnStartGameSession`，包括`OnProcessTerminate`、和`onHealthCheck`。

- `ProcessReady()`使用指定端口调用，以便在服务器进程准备好托管游戏会话时通知 Amazon GameLift 服务。

如果要自定义示例服务器代码，请参阅以下资源：

- 将 Amazon GameLift 添加到您的游戏服务器
- 适用于 C# 和 Unity 的 Amazon GameLift 服务器软件开发工具包 5.x 参考

集成您的客户端代码

如果您使用自己的游戏项目和自定义场景，则需要将基本功能集成到游戏客户端中。您还需要添加用户界面元素，以便玩家可以登录并加入游戏会话。使用 Amazon GameLift 服务 API（在 AWS SDK 中）获取游戏会话信息、创建新的游戏会话或加入现有游戏会话，

使用 Anywhere 队列构建用于本地测试的客户端时，您可以添加对 Amazon GameLift 服务的直接调用。当你为云托管开发游戏时，或者计划使用 Anywhere 队列进行生产托管时，你需要创建一个客户端后端服务来处理游戏客户端与 Amazon 服务之间的所有通信。 GameLift

要将 Amazon GameLift 集成到您的客户代码中，请使用以下资源作为指导。

- 将客户端与 GitHub repo 中的 GameLiftCoreApi 类集成 aw amazon-gamelift-plugin-unity s/。该类提供玩家身份验证和检索游戏会话信息的控件。
- 查看示例游戏集成，可在 GitHub 存储库 aw amazon-gamelift-plugin-unity s/ 中找到。`Samples~/SampleGame/Assets/Scripts/Client/GameLiftClient.cs`
- 按照将 Amazon 添加 GameLift 到 Unity 游戏客户端中的说明进行操作。

对于连接到 Anywhere 队列的游戏客户端，您的游戏客户端需要以下信息。该插件会自动更新您的游戏项目，以使用您在插件中创建的资源。

- FleetId -您的 Anywhere 车队的唯一标识符。
- FleetLocation -您的 Anywhere 舰队的自定义位置。
- AwsRegion -托管您的 Anywhere 舰队的AWS区域。这是您在用户个人资料中设置的区域。
- ProfileName -本地计算机上的AWS凭据配置文件，允许访问的 AWS SDK GameLift。游戏客户端使用这些凭证对Amazon GameLift 服务的请求进行身份验证。

> **Note**
>
> 凭证配置文件由插件生成并存储在本地计算机上。因此，您必须在本地计算机（或具有相同配置文件的计算机上）上运行客户端。

连接到 Anywhere 舰队

在此步骤中，您将指定要使用的 Anywhere 实例集。Anywhere 实例集定义了一组计算资源，这些资源可以位于任何地方，用于托管游戏服务器。

- 如果您当前使用的AWS账户已有 Anywhere 舰队，请打开舰队名称下拉字段并选择舰队。此下拉列表仅显示当前处于活动状态的用户配置文件所在 AWS 区域的 Anywhere 实例集。
- 如果现有队列不存在，或者您想创建新舰队，请选择创建新的 Anywhere 队列并提供队列名称。

在您为项目选择 Anywhere 队列后，Amazon GameLift 会验证队列状态是否为活跃并显示队列 ID。您可以在 Unity 编辑器的输出日志中跟踪此请求的进度。

注册计算

在此步骤中，您将本地工作站注册为新的 Anywhere 实例集中的计算资源。

1. 输入本地计算机的计算名称。如果您在实例集中添加多个计算，则名称必须是唯一的。
2. 选择 “注册计算”。可以在 Unreal 编辑器的输出日志中跟踪此请求的进度。

该插件使用设置为本地主机 (127.0.0.1) 的 IP 地址注册您的本地工作站。此设置假设你将在同一台计算机上运行游戏客户端和服务器。

作为对这一操作的响应，Amazon GameLift 会验证它是否可以连接到计算并返回有关新注册的计算的信息。

启动游戏

在此步骤中，您将构建游戏组件并启动它们来玩游戏。完成以下任务：

1. 配置您的游戏客户端。在此步骤中，您将提示插件更新游戏项目的`GameLiftClientSettings`资产。该插件使用此资产来存储您的游戏客户端连接到 Amazon GameLift 服务所需的某些信息。

a. 如果您没有导入和初始化示例游戏，请创建一个新GameLiftClientSettings资产。在 Unity 编辑器主菜单中，选择资源、创建 GameLift、客户端设置。如果您在项目 GameLiftClientSettings 中创建了多个副本，插件会自动检测到这一点，并通知您该插件将更新哪个资产。

b. 在 Launch Game 中，选择配置客户端：在任何地方应用设置。此操作会更新您的游戏客户端设置以使用您刚刚设置的 Anywhere 舰队。

2. 构建并运行您的游戏客户端。

a. 使用标准 Unity 编译流程生成客户端可执行文件。在 “文件”、“生成设置” 中，将平台切换到 Windows、Mac、Linux。如果您导入了示例游戏并初始化了设置，则构建列表和构建目标会自动更新。

b. 启动新建游戏客户端可执行文件的一个或多个实例。

3. 在 Anywhere 舰队中启动游戏服务器。选择服务器：在编辑器中启动服务器。此任务将启动一个实时服务器，只要 Unity 编辑器保持打开状态，您的客户端就可以连接到该服务器。

4. 开始或加入游戏会话。在您的游戏客户端实例中，使用用户界面将每个客户端加入游戏会话。如何执行此操作取决于您如何向客户端添加功能。

如果您使用的是示例游戏客户端，则它具有以下特征：

- 玩家登录组件。连接到 Anywhere 舰队上的游戏服务器时，不会进行玩家验证。您可以输入任何值来加入游戏会话。
- 一个简单的加入游戏用户界面。当客户尝试加入游戏时，客户端会自动寻找具有可用玩家插槽的活跃游戏会话。如果没有可用的游戏会话，则客户端会请求新的游戏会话。如果游戏会话可用，则客户端会请求加入可用的游戏会话。使用多个并发客户端测试游戏时，第一个客户端会启动游戏会话，其余客户端会自动加入现有的游戏会话。
- 有四个玩家插槽的游戏会话。您最多可以同时启动四个游戏客户端实例，它们将加入同一个游戏会话。

从服务器可执行文件启动（可选）

您可以构建并启动游戏服务器可执行文件，以便在 Anywhere 队列上进行测试。

1. 使用标准的 Unity 编译流程生成服务器可执行文件。在 “文件”、“编译设置” 中，将平台切换到专用服务器并构建。

2. 使用您的 Anywhere 舰队 ID 和AWS区域调用 AWS CLI 命令 get-compute-auth-token，获取短期身份验证令牌。创建队列时，舰队 ID 会显示在 Connect to a Anywhere 队列中。当您选择有效的个人资料时，该AWS地区将显示在 “设置您的个人资料” 中。

```
aws gamelift get-compute-auth-token --fleet-id [your anywhere fleet ID] --region
 [your AWS region]
```

3. 从命令行启动新建的游戏服务器可执行文件，并传入有效的身份验证令牌。

```
my_project.exe --authToken [token]
```

使用托管 EC2 实例集将游戏部署到云托管

在此工作流程中，您将使用插件为托管在 Amazon 管理的基于云的计算资源上进行游戏准备 GameLift。您可以为亚马逊 GameLift 功能添加客户端和服务器游戏代码，然后将您的服务器版本上传到亚马逊 GameLift 服务进行托管。此工作流程完成后，游戏服务器将在云端运行，并有一个可以与之连接的游戏客户端。

要启动亚马逊 GameLift 托管的 Amazon EC2 工作流程，请执行以下操作：

- 在 Unity 编辑器主菜单中，选择 Amazon， GameLift然后选择使用托管 EC2 的主机。此工作流程提供了一个集成、构建、部署和启动游戏组件的六步流程。

设置您的个人资料

选择您要在遵循此工作流程时使用的配置文件。您选择的配置文件会影响工作流程中的所有步骤。您创建的所有资源都与配置文件的 AWS 账户相关联，并放置在配置文件的默认 AWS 区域中。配置文件用户的权限决定了您对 AWS 资源和操作的访问权限。

1. 从可用配置文件的下拉列表中选择一个配置文件。如果您还没有个人资料或想要创建新的个人资料，请前往 Amazon GameLift 菜单并选择 “设置AWS账户资料”。
2. 如果引导状态不是“活动”，请选择引导配置文件并等待状态变为“活动”。

将您的游戏与 Amazon 集成 GameLift

对于此任务，您需要更新游戏项目中的客户端和服务器代码。

- 游戏服务器必须能够与 Amazon GameLift 服务通信，才能收到启动游戏会话的提示、提供游戏会话连接信息和报告状态。
- 游戏客户端必须能够获取有关游戏会话的信息、加入或开始游戏会话以及获取连接信息才能加入游戏。

> **Note**
>
> 如果您导入了示例游戏，则可以跳过此步骤。示例游戏资产已经准备好了必要的服务器和客户端代码。

集成您的服务器代码

在自定义场景中使用自己的游戏项目时，请使用提供的示例代码将所需的服务器代码添加到游戏项目中。如果您将要测试的游戏项目与 Anywhere 队列集成，则您已经完成了此步骤中的说明。

1. 在游戏项目文件中，打开文件`Assets/Scripts/Server`夹。如果它不存在，请创建它。
2. 前往 aws/ GitHub 仓库amazon-gamelift-plugin-unity并打开路径。`Samples~/SampleGame/Assets/Scripts/Server`
3. 找到该文件`GameLiftServer.cs`并将其复制到游戏项目的`Server`文件夹中。生成服务器可执行文件时，请使用此文件作为生成目标。

示例代码包括以下最低必需元素，它们使用 Amazon GameLift C# 服务器软件开发工具包（版本 5）：

- 初始化亚马逊 GameLift API 客户端。Amazon GameLift Anywhere 队列需要使用带有服务器参数的 initSDK () 调用。这些设置会自动设置为在插件中使用。
- 实现所需的回调函数以响应 Amazon GameLift 服务的请求`OnStartGameSession`，包括`OnProcessTerminate`、和`onHealthCheck`。
- `ProcessReady()`使用指定端口调用，以便在服务器进程准备好托管游戏会话时通知 Amazon GameLift 服务。

如果要自定义示例服务器代码，请参阅以下资源：

- 将 Amazon GameLift 添加到您的游戏服务器
- 适用于 C# 和 Unity 的 Amazon GameLift 服务器软件开发工具包 5.x 参考

集成您的客户端代码

对于连接到基于云的游戏服务器的游戏客户端，最佳做法是使用客户端后端服务来调用 Amazon GameLift 服务，而不是直接从游戏客户端拨打电话。

在托管 EC2 队列上托管的插件工作流程中，每个部署场景都包含一个包含以下组件的预建后端服务：

- 一组 Lambda 函数和 DynamoDB 表，用于请求游戏会话和检索游戏会话信息。这些组件使用 API 网关作为代理。
- 一个 Amazon Cognito 用户池，用于生成唯一的玩家 ID 并对玩家连接进行身份验证。

要使用这些组件，您的游戏客户端需要能够向后端服务发送请求以执行以下操作：

- 在 C AWS ognito 用户池中创建玩家用户并进行身份验证。
- 加入游戏会话并接收连接信息。
- 使用配对加入游戏。

使用以下资源作为指导。

- 将客户端与 GitHub repo 中的 GameLiftCoreApi类集成 aw amazon-gamelift-plugin-unity s/。该类提供玩家身份验证和检索游戏会话信息的控件。
- 要查看示例游戏集成，请访问 GitHub repo aw amazon-gamelift-plugin-unity s/，。`Samples~/SampleGame/Assets/Scripts/Client/GameLiftClient.cs`
- 将亚马逊 GameLift 添加到您的 Unity 游戏客户端。

选择部署方案

在此步骤中，您可以选择此时要部署的游戏托管解决方案。使用任何方案，您都可以对游戏进行多个部署。

- 单区域队列：将您的游戏服务器部署到活动配置文件默认AWS区域中的单个托管资源队列。此方案是测试服务器与 AWS 集成和服务器构建配置的良好起点。它部署了以下资源：
  - 已安装并运行游戏服务器构建的 AWS 实例集（按需型）。
  - Amazon Cognito 用户群体和客户端，使玩家能够进行身份验证和开始游戏。
  - 将用户群体与 API 关联的 API 网关授权器。
  - WebACL，用于限制玩家对 API 网关的过多调用。

  - API 网关 + Lambda 函数，供玩家申请游戏位置。如果两者都不可用，则此函数调用 `CreateGameSession()`。
  - API 网关 + Lambda 函数，供玩家获取游戏请求的连接信息。
- FlexMatch 舰队：将你的游戏服务器部署到一组舰队中，并设置一个带有规则的 FlexMatch 匹配器来创建玩家对战。此场景使用低成本 Spot 托管和多舰队、多地点结构，以实现持久可用性。当您准备开始为托管解决方案设计匹配器组件时，这种方法非常有用。在这种情况下，您将为此解决方案创建基本资源，以后可以根据需要对其进行自定义。它部署了以下资源：
  - FlexMatch 配对配置和配对规则设置为接受玩家请求和表单匹配。
  - 三个 AWS 实例集，安装了游戏服务器构建，并在多个位置运行。包括两个竞价型实例集和一个按需型实例集作为备份。
  - AWS 游戏会话放置队列，通过寻找尽可能好的托管资源（基于可行性、成本、玩家延迟等）并启动游戏会话来满足对提议对战的请求。
  - Amazon Cognito 用户群体和客户端，使玩家能够进行身份验证和开始游戏。
  - 将用户群体与 API 关联的 API 网关授权器。
  - WebACL，用于限制玩家对 API 网关的过多调用。
  - API 网关 + Lambda 函数，供玩家申请游戏位置。此函数调用 `StartMatchmaking()`。
  - API 网关 + Lambda 函数，供玩家获取游戏请求的连接信息。
  - Amazon DynamoDB 表，用于存储玩家的对战票证和游戏会话信息。
  - SNS 主题 + 用于处理事件的 Lambda 函数。 GameSessionQueue

设置游戏参数

在此步骤中，您将描述要上传到 AWS 的游戏。

- 游戏名称：为您的游戏项目提供一个有意义的名称。此名称在插件中使用。
- 队列名称：为您的托管 EC2 队列提供一个有意义的名称。Amazon 在AWS控制台中列出资源时 GameLift 使用此名称（以及舰队 ID）。
- 版本名称：为您的服务器版本提供一个有意义的名称。 AWS使用此名称来指代已上传到 Amazon GameLift 并用于部署的服务器版本的副本。
- 启动参数：输入在托管 EC2 队列实例上启动服务器可执行文件时要运行的可选说明。最大长度为 1024 个字符。
- 游戏服务器文件夹：提供包含您的服务器版本的本地文件夹的路径。
- 游戏服务器文件：指定服务器可执行文件名。

部署场景

在此步骤中，您将根据所选的部署方案将游戏部署到云托管解决方案。在 AWS 验证服务器构建、配置托管资源、安装游戏服务器、启动服务器进程以及让它们做好托管游戏会话的准备时，此过程可能需要长达 40 分钟。

要开始部署，请选择部署 CloudFormation。您可以在此处跟踪您的游戏托管状态。要了解更多详细信息，您可以登录 AWS 管理控制台了解 AWS 和查看事件通知。请务必使用与插件中活跃用户配置文件相同的账户、用户和 AWS 区域登录。

部署完成后，您的游戏服务器将安装在 AWS EC2 实例上。至少有一个服务器进程正在运行并准备开始游戏会话。

启动游戏客户端

成功部署队列后，游戏服务器就会运行并可供托管游戏会话。现在，您可以构建客户端、启动客户端、连接以加入游戏会话。

1. 配置您的游戏客户端。在此步骤中，您将提示插件更新游戏项目的`GameLiftClientSettings`资产。该插件使用此资产来存储您的游戏客户端连接到 Amazon GameLift 服务所需的某些信息。

   a. 如果您没有导入和初始化示例游戏，请创建一个新`GameLiftClientSettings`资产。在 Unity 编辑器主菜单中，选择资源、创建 GameLift、客户端设置。如果您在项目 GameLiftClientSettings 中创建了多个副本，插件会自动检测到这一点，并通知您该插件将更新哪个资产。

   b. 在 Launch Game 中，选择配置客户端：应用托管 EC2 设置。此操作会更新您的游戏客户端设置以使用您刚刚部署的托管 EC2 队列。

2. 构建您的游戏客户端。使用标准 Unity 编译流程生成客户端可执行文件。在 “文件”、“生成设置” 中，将平台切换到 Windows、Mac、Linux。如果您导入了示例游戏并初始化了设置，则构建列表和构建目标会自动更新。
3. 启动新构建的游戏客户端可执行文件。要开始玩游戏，请启动两到四个客户端实例，然后使用每个实例中的用户界面加入游戏会话。

如果您使用的是示例游戏客户端，则它具有以下特征：

- 玩家登录组件。连接到 Anywhere 舰队上的游戏服务器时，不会进行玩家验证。您可以输入任何值来加入游戏会话。

- 一个简单的加入游戏用户界面。当客户尝试加入游戏时，客户端会自动寻找具有可用玩家插槽的活跃游戏会话。如果没有可用的游戏会话，则客户端会请求新的游戏会话。如果游戏会话可用，则客户端会请求加入可用的游戏会话。使用多个并发客户端测试游戏时，第一个客户端会启动游戏会话，其余客户端会自动加入现有的游戏会话。
- 有四个玩家插槽的游戏会话。您最多可以同时启动四个游戏客户端实例，它们将加入同一个游戏会话。

适用于 Unity 的亚马逊 GameLift 插件服务器指南 SDK 4.x

> **Note**
>
> 本主题提供有关适用于 Unity 的 Amazon GameLift 插件的早期版本的信息。版本 1.0.0（2021 年发布）使用亚马逊 GameLift 服务器 SDK 4.x 或更早版本。有关该插件最新版本（该插件使用服务器 SDK 5.x 且支持 Amazon）的文档 GameLift Anywhere，请参阅适用于 Unity 的亚马逊 GameLift 插件服务器指南 SDK 5.x。

亚马逊 GameLift 提供工具，让您的多人游戏服务器做好在亚马逊上运行的准备 GameLift。Unity GameLift 版的亚马逊插件可以更轻松地将亚马逊 GameLift 集成到您的 Unity 游戏项目中，以及为云托管部署亚马逊 GameLift 资源。使用适用于 Unity 的插件访问亚马逊 GameLift API 并为常见游戏场景部署AWS CloudFormation模板。

设置好插件后，你可以试用 Amazon GameLift Unity 示例 GitHub。

主题

- 将亚马逊 GameLift 与 Unity 游戏服务器项目集成
- 将亚马逊 GameLift 与 Unity 游戏客户端项目集成
- 安装并设置插件
- 在本地测试你的游戏
- 部署场景
- GameLift 在 Unity 中将游戏与亚马逊集成
- 导入并运行示例游戏

将亚马逊 GameLift 与 Unity 游戏服务器项目集成

> **Note**
>
> 本主题提供有关适用于 Unity 的 Amazon GameLift 插件的早期版本的信息。版本 1.0.0（2021 年发布）使用亚马逊 GameLift 服务器 SDK 4.x 或更早版本。有关该插件最新版本（该插件使用服务器 SDK 5.x 且支持 Amazon）的文档 GameLift Anywhere，请参阅适用于 Unity 的亚马逊 GameLift 插件服务器指南 SDK 5.x。

本主题可帮助您为在 Amazon 上托管的自定义游戏服务器做好准备 GameLift。游戏服务器必须能够 GameLift 通知Amazon其状态，在出现提示时启动和停止游戏会话以及执行其他任务。有关更多信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

先决条件

在集成游戏服务器之前，请完成以下任务：

- 为亚马逊设置 IAM 服务角色 GameLift
- 安装适用于 Unity 的插件

设置新的服务器进程

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

设置与 Amazon 的通信 GameLift 并报告服务器进程已准备好托管游戏会话。

1. 通过调用 `InitSDK()` 初始化服务器软件开发工具包。
2. 要让服务器做好接受游戏会话的准备，调用 `ProcessReady()` 以及连接端口和游戏会话位置的详细信息。包括 Amazon GameLift 服务调用的回调函数的名称，例如、`OnGameSession()`、`OnGameSessionUpdate()`、`OnProcessTerminate()`。`OnHealthChecl` GameLift 可能需要几分钟才能提供回调。
3. Amazon 将服务器进程的状态 GameLift 更新为`ACTIVE`。

4. 亚马逊onHealthCheck会定期致 GameLift 电。

以下代码示例展示了如何使用 Amazon 设置简单的服务器进程 GameLift。

```
//initSDK
var initSDKOutcome = GameLiftServerAPI.InitSDK();

//processReady
// Set parameters and call ProcessReady
var processParams = new ProcessParameters(
    this.OnGameSession,
    this.OnProcessTerminate,
    this.OnHealthCheck,
    this.OnGameSessionUpdate,
    port,
    // Examples of log and error files written by the game server
    new LogParameters(new List<string>()
        {
            "C:\\game\\logs",
            "C:\\game\\error"
        })
);

var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);

// Implement callback functions
void OnGameSession(GameSession gameSession)
{
    // game-specific tasks when starting a new game session, such as loading map
    // When ready to receive players
    var activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}

void OnProcessTerminate()
{
    // game-specific tasks required to gracefully shut down a game session,
    // such as notifying players, preserving game state data, and other cleanup
    var ProcessEndingOutcome = GameLiftServerAPI.ProcessEnding();
}

bool OnHealthCheck()
{
    bool isHealthy;
```

```
    // complete health evaluation within 60 seconds and set health
    return isHealthy;
}
```

启动游戏会话

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

游戏初始化完成后，您可以开始游戏会话。

1. 实现回调函数 `onStartGameSession`。Amazon GameLift 调用此方法在服务器进程上启动新的游戏会话并接收玩家连接。
2. 要激活游戏会话，请调用 `ActivateGameSession()`。有关软件开发工具包的更多信息，请参阅 Amazon GameLift 服务器软件开发工具包 (C#) 参考：操作。

以下代码示例说明了如何通过 Amazon 启动游戏会话 GameLift。

```
void OnStartGameSession(GameSession gameSession)
{
    // game-specific tasks when starting a new game session, such as loading map
    ...
    // When ready to receive players
    var activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}
```

结束游戏会话

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

游戏会话结束 GameLift 时通知亚马逊。最佳实操是在游戏会话完成后关闭服务器进程，以回收和刷新托管资源。

1. 设置一个名为的函数onProcessTerminate以接收来自 Amazon 的请求 GameLift 并调用ProcessEnding()。
2. 进程状态更改为 TERMINATED。

以下示例描述了如何结束游戏会话的进程。

```
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();

if (processReadyOutcome.Success)
   Environment.Exit(0);

// otherwise, exit with error code
Environment.Exit(errorCode);
```

创建服务器、构建并上传到 Amazon GameLift

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

将游戏服务器与 Amazon 集成后 GameLift，将构建文件上传到队列中，以便亚马逊 GameLift 可以将其部署用于游戏托管。有关如何将服务器上传到 Amazon 的更多信息 GameLift，请参阅将自定义服务器构建上传到 Amazon GameLift。

将亚马逊 GameLift 与 Unity 游戏客户端项目集成

> **Note**
>
> 本主题提供有关适用于 Unity 的 Amazon GameLift 插件的早期版本的信息。版本 1.0.0（2021 年发布）使用亚马逊 GameLift 服务器 SDK 4.x 或更早版本。有关该插件最新版本（该插件使用服务器 SDK 5.x 且支持 Amazon）的文档 GameLift Anywhere，请参阅适用于 Unity 的亚马逊 GameLift 插件服务器指南 SDK 5.x。

本主题可帮助您设置游戏客户端，以便通过后端服务连接到 Amazon GameLift 托管的游戏会话。使用 Amazon GameLift API 启动配对、请求游戏会话放置等。

向后端服务项目添加代码以允许与 Amazon GameLift 服务进行通信。后端服务处理游戏客户端与该 GameLift 服务的所有通信。有关后端服务的更多信息，请参阅设计您的游戏客户端服务。

后端服务器处理以下游戏客户端任务：

- 自定义玩家身份验证。
- 向 Amazon GameLift 服务请求有关活跃游戏会话的信息。
- 创建新的游戏会话。
- 将玩家加入到现有游戏会话。
- 将玩家从现有游戏会话中移除。

主题

- 先决条件
- 初始化游戏客户端
- 按照特定的实例集创建游戏会话
- 向游戏会话中添加玩家
- 从游戏会中移除玩家

先决条件

在设置游戏服务器与 Amazon GameLift 客户端的通信之前，请完成以下任务：

- 设置一个 AWS 账户
- 安装适用于 Unity 的插件
- 将亚马逊 GameLift 与 Unity 游戏服务器项目集成
- 设置 Amazon GameLift 实例集

初始化游戏客户端

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

添加用于初始化游戏客户端的代码。启动时运行此代码，这是其他 Amazon GameLift 功能所必需的。

1. 初始化 AmazonGameLiftClient。使用默认客户端配置或自定义配置调用 AmazonGameLiftClient。有关如何配置客户端的更多信息，请参阅[在后端服务 GameLift 上设置 Amazon]。
2. 为每个玩家生成唯一的玩家 ID 来连接到游戏会话。有关更多信息，请参阅 [生成玩家 ID]。

   以下示例显示了如何设置 Amazon GameLift 客户端。

```
public class GameLiftClient
{
    private GameLift gl;
    //A sample way to generate random player IDs.
    bool includeBrackets = false;
    bool includeDashes = true;
    string playerId = AZ::Uuid::CreateRandom().ToString<string>(includeBrackets,
 includeDashes);


    private Amazon.GameLift.Model.PlayerSession psession = null;
    public AmazonGameLiftClient aglc = null;

    public void CreateGameLiftClient()
    {
        //Access Amazon GameLift service by setting up a configuration.
        //The default configuration specifies a location.
        var config = new AmazonGameLiftConfig();
        config.RegionEndpoint = Amazon.RegionEndpoint.USEast1;

        CredentialProfile profile = null;
        var nscf = new SharedCredentialsFile();
        nscf.TryGetProfile(profileName, out profile);
        AWSCredentials credentials = profile.GetAWSCredentials(null);
        //Initialize GameLift Client with default client configuration.
        aglc = new AmazonGameLiftClient(credentials, config);

    }
}
```

按照特定的实例集创建游戏会话

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

添加用于在已部署的实例集中启动新游戏会话并使其可供玩家接入的代码。在 Amaz GameLift on 创建新游戏会话并返回后GameSession，您可以向其中添加玩家。

- 申请新游戏会话。
  - 如果您的游戏使用实例集，请使用实例集或别名 ID、会话名称和游戏的最大并发玩家数量调用 `CreateGameSession()`。
  - 如果您的游戏使用队列，请调用 `StartGameSessionPlacement()`。

以下示例演示如何创建游戏会话。

```
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;
}
```

向游戏会话中添加玩家

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

在 Amaz GameLift on 创建新游戏会话并返回GameSession对象后，您可以向其中添加玩家。

1. 通过新建玩家会话在游戏会话中预留玩家位置。将 CreatePlayerSession 或 CreatePlayerSessions 与游戏会话 ID 和每个玩家的唯一 ID 一起使用。
2. 连接游戏会话。检索 PlayerSession 对象以获取游戏会话的连接信息。您可以使用此信息与服务器进程建立直接连接：
   a. 使用指定的端口以及分配给服务器进程的 DNS 名称或 IP 地址进行连接。
   b. 使用实例集的 DNS 名称和端口。如果实例集启用了 TLS 证书生成，则需要 DNS 名称和端口。
   c. 引用玩家会话 ID。如果游戏服务器验证传入的玩家连接，则需要玩家会话 ID。

以下示例演示了如何在游戏会话中预留玩家位置。

```
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;
}
```

以下代码说明了如何将玩家连接到游戏会话。

```
public bool ConnectPlayer(int playerIdx, string playerSessionId)
{
    //Call ConnectPlayer with player ID and player session ID.
    return server.ConnectPlayer(playerIdx, playerSessionId);
}
```

从游戏会中移除玩家

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

当玩家离开游戏时，您可以将其从游戏会话中移除。

1. 通知 Amazon GameLift 服务有玩家已断开与服务器进程的连接。使用玩家的会话 ID 调用 `RemovePlayerSession`。
2. 验证 `RemovePlayerSession` 是否返回 `Success`。然后，亚马逊将玩家位置 GameLift 更改为可用，亚马逊 GameLift 可以将其分配给新玩家。

以下示例说明了如何移除玩家会话。

```
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());
    }
}
```

安装并设置插件

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

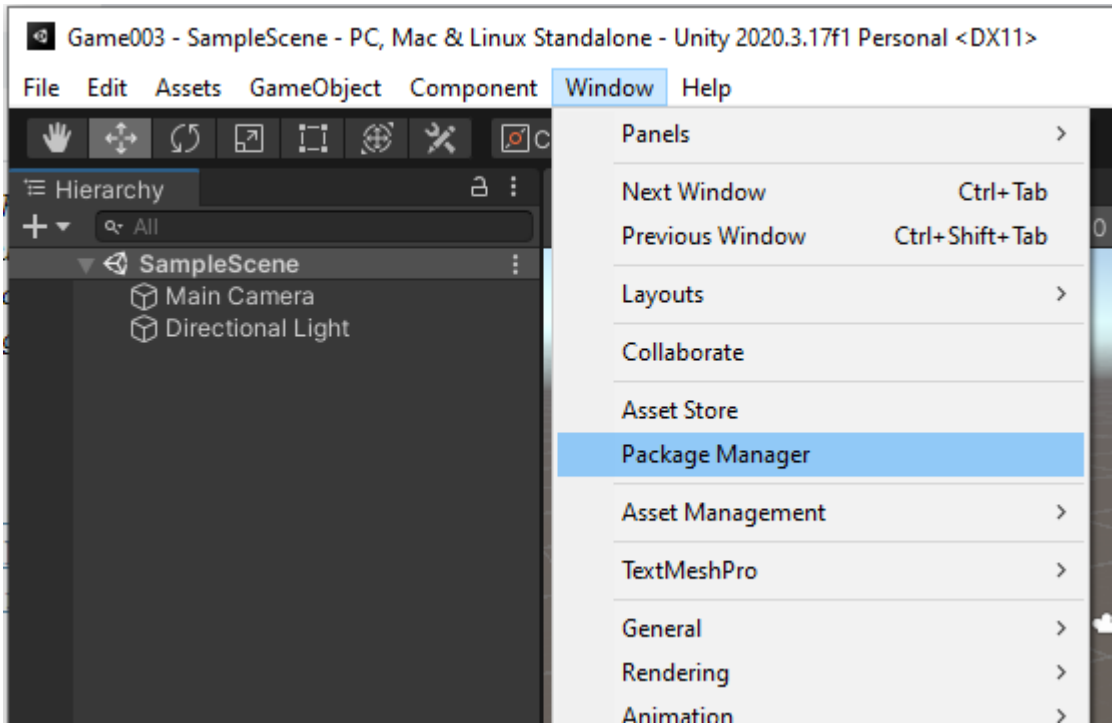
本节介绍如何下载、安装和设置适用于 Unity 的 Amazon GameLift 插件，版本 1.0.0。

先决条件

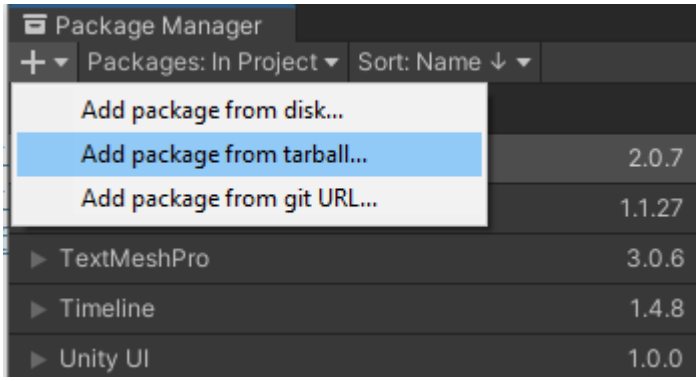
- 适用于 Windows 的 Unity 2019.4 LTS、适用于 Windows 的 2020.3 LTS 或适用于 macOS 的 Unity
- Java 的当前版本
- .NET 4.x 的当前版本

下载并安装适用于 Unity 的插件

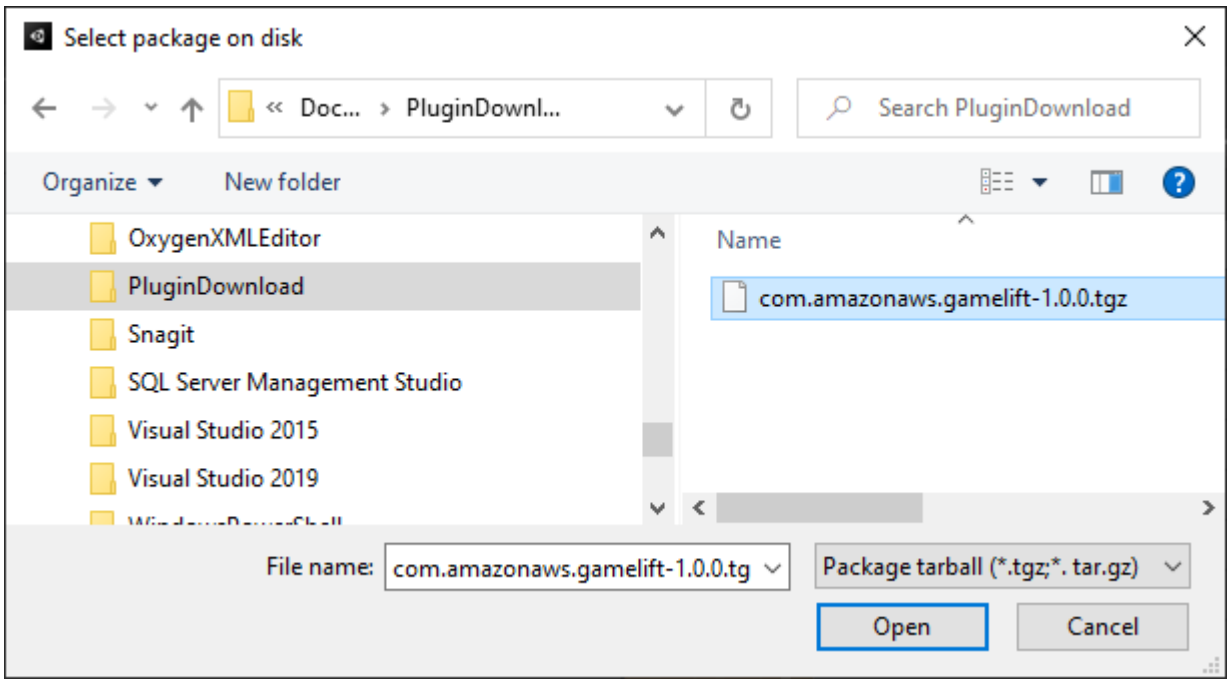
1. 下载适用于 Unity 的亚马逊 GameLift 插件。您可以在适用于 Unity 的 Amazon GameLift 插件存储库页面上找到最新版本。在最新版本下，选择资产，然后下载 `com.amazonaws.gamelift-`*`version`*`.tgz` 文件。
2. 启动 Unity 并选择一个项目。
3. 在顶部导航栏的窗口下，选择 Package Manager：



4. 在 Package Manager 选项卡下，选择 +，然后选择从 tarball 添加软件包...：



5. 在选择磁盘上的软件包窗口中，导航到 `com.amazonaws.gamelift` 文件夹，选择文件 `com.amazonaws.gamelift-`*`version`*`.tgz` ，然后选择打开：



6. Unity 加载插件后，亚马逊 GameLift将作为新项目出现在 Unity 菜单中。安装和重新编译脚本可能需要几分钟时间。“A mazon GameLift 插件设置” 选项卡会自动打开。

7. 在软件开发工具包窗格中，选择使用 .NET 4.x。

   配置后，状态将从未配置更改为已配置。

在本地测试你的游戏

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

使用 Amazon L GameLift ocal GameLift 在您的本地设备上运行亚马逊。无需网络连接，即可使用 Amazon L GameLift ocal 在几秒钟内验证代码更改。

配置本地测试

1. 在 Unity 插件窗口中，选择测试选项卡。
2. 在 “测试” 窗格中，选择 “下载 Amazon L GameLift ocal”。适用于 Unity 的插件会打开浏览器窗口，并将 `GameLift_06_03_2021.zip` 文件下载到您的下载文件夹。

   下载内容包括 C# 服务器软件开发工具包、.NET 源文件和与 Unity 兼容的 .NET 组件。
3. 解压下载的 `GameLift_06_03_2021.zip` 文件。
4. 在 “亚马逊 GameLift 插件设置” 窗口中，选择 “亚马逊 GameLift 本地路径”，导航到解压缩的文件夹，选择文件**`GameLiftLocal.jar`**，然后选择 “打开”。

   配置后，本地测试状态将从未配置更改为已配置。
5. 验证 JRE 的状态。如果状态为未配置，请选择下载 JRE 并安装推荐的 Java 版本。

   安装和配置 Java 环境后，状态将更改为已配置。

运行你的本地游戏

1. 在“适用于 Unity 的插件”选项卡中，选择测试选项卡。
2. 在测试窗格中，选择打开本地测试 UI。
3. 在本地测试窗口中，指定服务器可执行文件路径。选择...选择服务器应用程序的路径和可执行文件名称。
4. 在本地测试窗口中，指定 GL Local 端口。
5. 选择部署并运行以部署和运行服务器。
6. 要停止游戏服务器，请选择停止或关闭游戏服务器窗口。

部署场景

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

场景使用AWS CloudFormation模板来创建为游戏部署云托管解决方案所需的资源。本节介绍了 Amazon GameLift 提供的场景以及如何使用它们。

先决条件

要部署该场景，您需要一个 Amazon GameLift 服务的 IAM 角色。有关如何为 Amazon 创建角色的信息 GameLift，请参阅设置一个 AWS 账户。

每种场景都需要访问以下资源的权限：

- Amazon GameLift
- Amazon S3
- AWS CloudFormation
- API Gateway
- AWS Lambda
- AWS WAFV2
- Amazon Cognito

场景

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

适用于 Unity 的 Amazon GameLift 插件包括以下场景：

仅限身份验证

此场景创建了一个游戏后端服务，该服务在没有游戏服务器功能的情况下执行玩家身份验证。该模板在您的账户中创建以下资源：

- Amazon Cognito 用户群体，用于存储玩家身份验证信息。
- Amazon API Gateway REST 端点支持的 AWS Lambda 处理程序，用于启动游戏并查看游戏连接信息。

单区域实例集

此场景使用单个 Amazon GameLift 队列创建游戏后端服务。其创建了以下资源：

- Amazon Cognito 用户群体，供玩家进行身份验证和开始游戏。
- 一个 AWS Lambda 处理程序，用于搜索实例集上有开放玩家位置的现有游戏会话。如果该处理程序找不到开放位置，就会创建一个新的游戏会话。

带有队列和自定义对战构建器的多区域实例集

此场景通过使用Amazon GameLift 队列和自定义匹配器将等候池中年龄最大的玩家组合在一起来形成匹配项. 其创建了以下资源：

- 亚马逊简单通知服务主题，亚马逊向其 GameLift 发布消息。有关 SNS 主题和通知的更多信息，请参阅 请参阅设置游戏会话置放通知。。
- 一个 Lambda 函数，由传达位置和游戏连接详情的消息调用。
- Amazon DynamoDB 表，用于存储位置和游戏连接详情。`GetGameConnection` 调用从此表读取并将连接信息返回到游戏客户端。

带有队列和自定义对战构建器的竞价型实例集

此场景使用Amazon GameLift 队列和自定义匹配器形成匹配项，并配置三个队列。其创建了以下资源：

- 两个竞价型实例集包含不同的实例类型，可确保竞价型实例不可用性更持久。
- 一种按需型实例集，可作为其他竞价型实例集的备份。有关实例集设计的更多信息，请参阅Amazon GameLift 实例集设计指南。
- Amazon GameLift 队列，用于保持高服务器可用性和低成本。有关队列的更多信息和最佳实操，请参阅设计游戏会话队列。

FlexMatch

此场景使用 FlexMatch托管配对服务将玩家配对在一起。有关更多信息 FlexMatch，请参阅什么是亚马逊 GameLift FlexMatch。此场景创建了以下资源：

- 一个 Lambda 函数，用于在收到 `StartGame` 请求后创建对战票证。
- 一个单独的 Lambda 函数，用于监听 FlexMatch 匹配事件。

为避免您的 AWS 账户产生不必要的费用，请在使用完每个场景创建的资源后将其删除。同时删除相应的 AWS CloudFormation 堆栈。

更新AWS凭证

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

适用于 Unity 的 Amazon GameLift 插件需要安全证书才能部署场景。您可以创建新凭证或使用现有凭证。

有关配置凭证的更多信息，请参阅了解和获取您的 AWS 凭证。

更新 AWS 凭证

1. 在 Unity 的“适用于 Unity 的插件”选项卡中，选择部署选项卡。
2. 在部署窗格中，选择 AWS 凭证。
3. 您可以创建新 AWS 凭证或选择现有凭证。
    - 要创建凭证，请选择创建新的凭证配置文件，然后指定新配置文件名称、AWS 访问密钥 ID、AWS 密钥和 AWS 区域。
    - 要选择现有凭证，请选择选择现有凭证配置文件，然后选择配置文件名称和 AWS 区域。
4. 在更新 AWS 凭证窗口中，选择更新凭证配置文件。

更新账号引导

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

引导位置是部署期间使用的 Amazon S3 存储桶。它用于存储游戏服务器资产和其他依赖项。您为存储桶选择的 AWS 区域必须与用于场景部署的区域相同。

有关 Amazon S3 存储桶的更多信息，请参阅创建、配置和使用 Amazon Simple Storage Service 存储桶。

更新账户引导位置

1. 在 Unity 的“适用于 Unity 的插件”选项卡中，选择部署选项卡。
2. 在部署窗格中，选择更新账户引导。
3. 在账户引导窗口中，您可以选择现有 Amazon S3 存储桶或创建一个新的 Amazon S3 存储桶：
   - 要选择现有存储桶，请选择选择现有 Amazon S3 存储桶，然后选择更新以保存您的选择。
   - 选择创建新的 Amazon S3 存储桶以创建新的 Amazon Simple Storage Service 存储桶，然后选择策略。该策略指定了 Amazon S3 存储桶的过期时间。选择创建以创建存储桶。

部署游戏场景

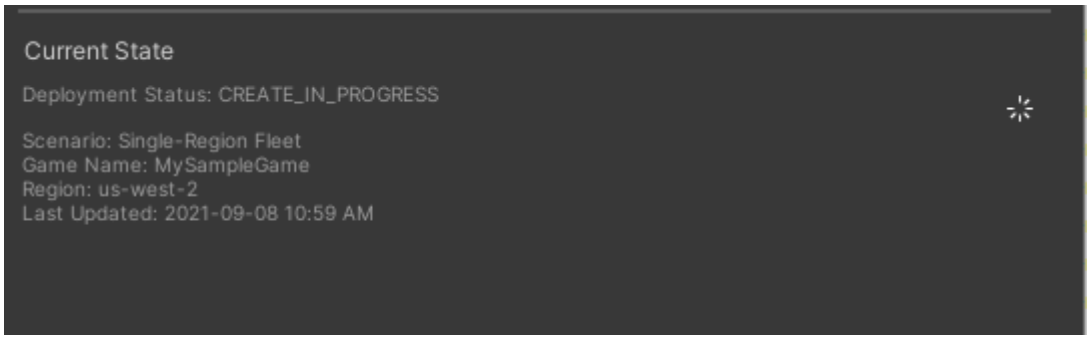
> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

您可以使用场景在 Amazon 上测试您的游戏 GameLift。每个场景都使用 AWS CloudFormation 模板创建一个包含所需资源的堆栈。大多数场景都需要游戏服务器可执行文件和构建路径。部署场景时，作为部署的一部分，Amazon 会将游戏资产 GameLift复制到引导位置。

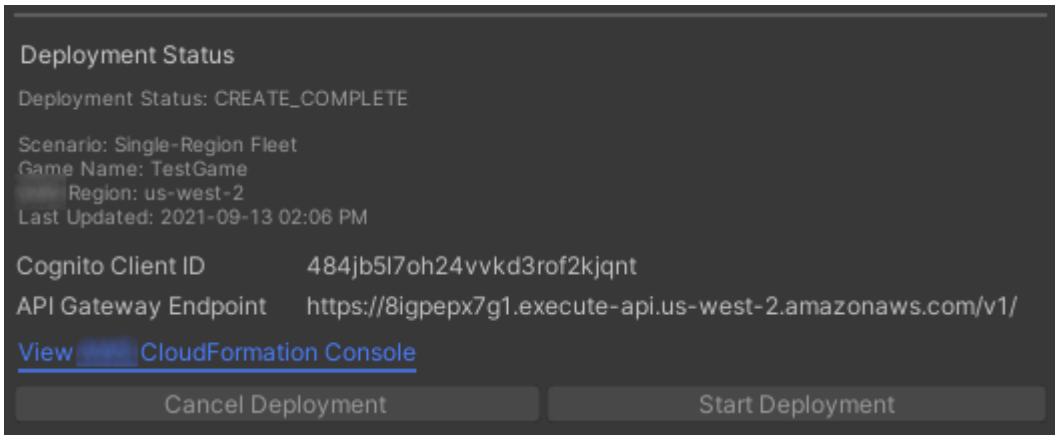
您必须配置 AWS 凭证和 AWS 账户引导才能部署场景。

部署场景

1. 在 Unity 的“适用于 Unity 的插件”选项卡中，选择部署选项卡。
2. 在部署窗格中，选择打开部署 UI。
3. 在部署窗口中，选择一个场景。
4. 输入游戏名称。此名称必须唯一。部署场景时，游戏名称是 AWS CloudFormation 堆栈名称的一部分。
5. 选择游戏服务器构建文件夹路径。构建文件夹路径指向包含服务器可执行文件和依赖项的文件夹。
6. 选择游戏服务器构建 .exe 文件路径。构建可执行文件路径指向游戏服务器可执行文件。
7. 选择开始部署以开始部署场景。您可以在部署窗口的当前状态下关注更新状态。部署场景最多可能需要 30 分钟。



8. 场景完成部署后，当前状态将更新为包括 Cognito 客户端 ID 和 API Gateway 端点，您可以将其复制并粘贴到游戏中。



9. 要更新游戏设置，请在 Unity 菜单上选择转到客户端连接设置。这会在 Unity 屏幕的右侧显示 Inspector 选项卡。
10. 取消选择本地测试模式。
11. 输入 API Gateway 端点和 Coginito 客户端 ID。选择与场景部署相同的 AWS 区域。然后，您可以使用已部署的场景资源重建并运行游戏客户端。

删除场景创建的资源

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

要删除为该场景创建的资源，请删除相应的 AWS CloudFormation 堆栈。

删除由场景创建的资源

1. 在 Unity 部署的 Amazon GameLift 插件窗口中，选择查看AWS CloudFormation控制台以打开AWS CloudFormation控制台。
2. 在 AWS CloudFormation 控制台中，选择堆栈，然后选择包含部署期间指定的游戏名称的堆栈。
3. 要删除该堆栈，请选择删除。删除堆栈可能需要几分钟时间。AWS CloudFormation 删除场景使用的堆栈后，其状态将更改为 `ROLLBACK_COMPLETE`。

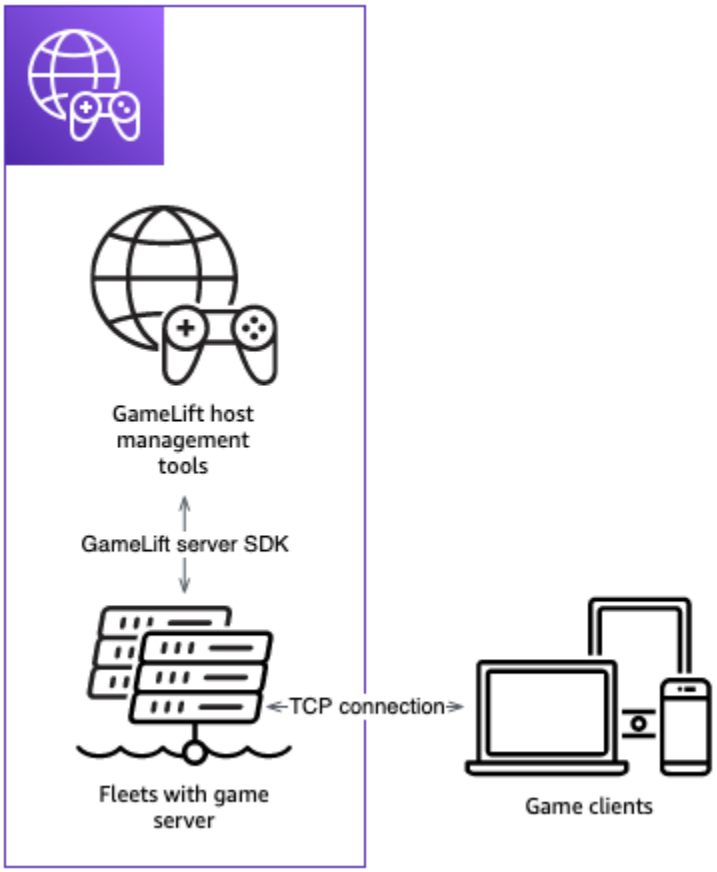
GameLift 在 Unity 中将游戏与亚马逊集成

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

完成以下任务，将您 GameLift 的 Unity 游戏与 Amazon 集成：

- 将亚马逊 GameLift 与 Unity 游戏服务器项目集成
- 将亚马逊 GameLift 与 Unity 游戏客户端项目集成

下图显示了集成游戏的示例流程。在图中，带有游戏服务器的舰队已部署到Amazon GameLift。游戏客户端与游戏服务器通信，游戏服务器与 Amazon 通信。 GameLift



导入并运行示例游戏

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

适用于 Unity 的亚马逊 GameLift 插件包含一个示例游戏，你可以用它来探索将游戏与亚马逊集成的基础知识 GameLift。在本节中，您将构建游戏客户端和游戏服务器，然后使用 Amazon L GameLift ocal 在本地进行测试。

先决条件

- 设置一个 AWS 账户
- 安装并设置插件

构建并运行示例游戏服务器

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

设置示例游戏的游戏服务器文件。

1. 在 Unity 中，在菜单上选择 Amazon GameLift，然后选择导入示例游戏。
2. 在导入示例游戏窗口中，选择导入以导入游戏、其资产和依赖项。
3. 构建游戏服务器。在 Unity 中，在菜单上选择亚马逊 GameLift，然后选择应用 Windows 示例服务器版本设置或应用 macOS 示例服务器版本设置。配置游戏服务器设置后，Unity 会重新编译资产。
4. 在 Unity 中，在菜单上选择文件，然后选择构建。选择服务器构建，选择构建，然后选择专门存放服务器文件的构建文件夹。

   Unity 构建示例游戏服务器，将可执行文件和所需资产放在指定的构建文件夹中。

构建并运行示例游戏客户端

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

设置示例游戏的游戏客户端文件。

1. 在 Unity 中，在菜单上选择亚马逊 GameLift，然后选择应用 Windows 示例客户端版本设置或应用 macOS 示例客户端版本设置。配置游戏客户端设置后，Unity 将重新编译资产。
2. 在 Unity 中，在菜单上选择转到客户端设置。这将在 Unity 屏幕的右侧显示 Inspector 选项卡。在 Amazon GameLift 客户端设置选项卡中，选择本地测试模式。
3. 构建游戏客户端。在 Unity 中，在菜单上文件。确认未选中服务器构建，选择构建，然后选择专门存放客户端文件的构建文件夹。

   Unity 构建示例游戏客户端，将可执行文件和所需资产放在指定的客户端构建文件夹中。

4. 您尚未构建游戏服务器和客户端。在接下来的步骤中，您将运行游戏并查看它如何与Amazon GameLift 互动。

在本地测试示例游戏

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

使用 Amazon L GameLift ocal 运行您导入的示例游戏。

1. 启动游戏服务器。在 Unity 的“适用于 Unity 的插件”选项卡中，选择部署选项卡。
2. 在测试窗格中，选择打开本地测试 UI。
3. 在本地测试窗口中，指定游戏服务器 .exe 文件路径。路径必须包含可执行文件名称。例如，`C:/MyGame/GameServer/MyGameServer.exe`。
4. 选择部署并运行。Unity 插件启动游戏服务器并打开 Amazon GameLift 本地日志窗口。窗口包含日志消息，包括游戏服务器和 Amazon L GameLift ocal 之间发送的消息。
5. 启动游戏客户端。使用示例游戏客户端找到构建位置并选择可执行文件。
6. 在 Amazon GameLift 示例游戏中，提供电子邮件和密码，然后选择 “登录”。电子邮件和密码未经过验证或使用。
7. 在 Amazon GameLift 示例游戏中，选择 “开始”。游戏客户端会寻找游戏会话。如果找不到会话，便会自行创建。然后，游戏客户端开始游戏会话。可以在日志中看到游戏活动。

示例游戏服务器日志

```
...
2021-09-15T19:55:3495 PID:20728 Log :) GAMELIFT AWAKE
2021-09-15T19:55:3512 PID:20728 Log :) I AM SERVER
2021-09-15T19:55:3514 PID:20728 Log :) GAMELIFT StartServer at port 33430.
2021-09-15T19:55:3514 PID:20728 Log :) SDK VERSION: 4.0.2
2021-09-15T19:55:3556 PID:20728 Log :) SERVER IS IN A GAMELIFT FLEET
2021-09-15T19:55:3577 PID:20728 Log :) PROCESSREADY SUCCESS.
2021-09-15T19:55:3577 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
...
2021-09-15T19:55:3634 PID:20728 Log :) GAMELOGIC AWAKE
```

```
2021-09-15T19:55:3635 PID:20728 Log :) GAMELOGIC START
2021-09-15T19:55:3636 PID:20728 Log :) LISTENING ON PORT 33430
2021-09-15T19:55:3636 PID:20728 Log SERVER: Frame: 0 HELLO WORLD!
...
2021-09-15T19:56:2464 PID:20728 Log :) GAMELIFT SESSION REQUESTED
2021-09-15T19:56:2468 PID:20728 Log :) GAME SESSION ACTIVATED
2021-09-15T19:56:3578 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
2021-09-15T19:57:3584 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
2021-09-15T19:58:0334 PID:20728 Log SERVER: Frame: 8695 Connection accepted: playerIdx
 0 joined
2021-09-15T19:58:0335 PID:20728 Log SERVER: Frame: 8696 Connection accepted: playerIdx
 1 joined
2021-09-15T19:58:0338 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from playerIdx 0 Msg:
 CONNECT: server IP localhost
2021-09-15T19:58:0338 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from player 0:CONNECT:
 server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 CONNECT: player index 0
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from playerIdx 1 Msg:
 CONNECT: server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from player 1:CONNECT:
 server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 CONNECT: player index 1
```

Amazon GameLift 本地日志示例

```
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"
```

关闭服务器进程

> **Note**
>
> 本主题涉及适用于 Unity 版本 1.0.0 的亚马逊 GameLift 插件，该插件使用服务器 SDK 4.x 或更早版本。

完成示例游戏后，在 Unity 中关闭服务器。

1. 在游戏客户端中，选择退出或关闭窗口以停止游戏客户端。
2. 在 Unity 中，在本地测试窗口中，选择停止或关闭游戏服务器窗口以停止服务器。

将游戏与适用于虚幻引擎的 Amazon GameLift 插件集成

本节中的主题介绍适用于虚幻引擎 (UE) 的亚马逊 GameLift 插件，以及如何使用它来准备在亚马逊托管的多人游戏项目 GameLift。完全在您的 UE 开发环境中使用插件的指导工作流程，以完成在 Amazon 上托管的基本要求 GameLift。

Amazon GameLift 是一项完全托管的服务，允许游戏开发者管理和扩展用于基于会话的多人游戏的专用游戏服务器。有关 Amazon GameLift 托管的更多信息，请参阅Amazon GameLift 的工作原理。

主题

- 关于插件
- 插件工作流
- 安装适用于 Unreal 的插件
- 设置 AWS 用户配置文件
- 将您的游戏设置为在 Amazon 上进行测试 GameLift Anywhere
- 使用托管 EC2 实例集将游戏部署到云托管

关于插件

该插件将 Amazon GameLift 工具和功能添加到 UE 编辑器中。该插件的指导工作流程用于将 Amazon GameLift 集成到您的游戏项目中，将工作站指定为本地主机进行测试，并将游戏服务器部署到亚马逊 GameLift 云托管。

使用插件的预构建托管解决方案来部署您的游戏。以您的本地工作站为主机，设置 Amazon GameLift Anywhere 舰队。对于云托管，可以从两种常见的部署方案中进行选择，以不同的方式平衡玩家延迟、游戏会话可用性和成本。一种场景包括一个简单的 FlexMatch匹配器和规则集. 使用这些解决方案可以快速开始使用可随时投入生产的托管结构，然后根据需要进行优化和自定义。

该插件包括以下组件：

- UE 编辑器的插件模块。安装插件后，新的主菜单按钮可让您访问Amazon的 GameLift 功能。
- 适用于具有客户端功能的 Amazon GameLift 服务 API 的 C++ 库。
- Amazon GameLift 服务器软件开发工具包的虚幻库（版本 5）。
- 测试内容，包括启动游戏地图和两张测试地图，其中包含用于测试服务器集成的基本蓝图和用户界面元素。
- 可编辑配置，AWS CloudFormation 模板形式，插件在部署游戏服务器进行托管时使用。

插件工作流

以下步骤描述了使用虚幻引擎的Amazon GameLift 插件集成和部署游戏项目的典型方法。您可以通过 UE 编辑器和游戏代码来完成这些步骤。

1. 创建与您的AWS账户关联的用户个人资料，并为有权使用 Amazon 的有效账户用户提供访问凭证 GameLift。
2. 将服务器代码添加到您的游戏项目中，以便在正在运行的游戏服务器与 with Amazon GameLift 服务之间建立通信。
3. 将客户端代码添加到您的游戏项目中，允许游戏客户端向 Amazon 发送请求 GameLift 以启动新的游戏会话，然后连接到这些会话。
4. 使用 Anywhere 工作流程将您的本地工作站设置为游戏服务器的 Anywhere 主机。通过插件在本地启动游戏服务器和客户端，连接到游戏会话，然后测试集成情况。
5. 使用 EC2 托管工作流程上传您的集成游戏服务器并部署云托管解决方案。游戏服务器准备就绪后，通过插件在本地启动游戏客户端、连接到游戏会话并加入游戏。

在插件中工作时，您将创建和使用AWS资源，这些操作可能会对正在使用的AWS账户产生费用。如果您不熟悉AWS，这些操作可能包含在AWS免费套餐中。

安装适用于 Unreal 的插件

本节介绍将插件添加到 Unreal Engine 项目的初始安装任务。当在 Unreal 编辑器中打开项目时，插件功能就可用。

> **Note**
>
> 您可以将 Amazon GameLift 插件与标准版本的 UE 编辑器一起使用，但是在打包游戏服务器版本时，需要使用源代码构建的版本。

开始之前

以下是使用适用于虚幻引擎的Amazon GameLift 插件所需的内容：

- 适用于虚幻引擎的Amazon GameLift 插件发布包。[下载网站]。
- Microsoft Visual Studio 2019 或更高版本。
- Unreal Engine 编辑器的源代码构建版本。需要一个源代码构建版本来打包多人游戏的服务器组件。如需更多详细信息，包括其他先决条件，请参阅 Unreal Engine 文档：
  - 在 You need GitHub 和 Epic Games 账号@@ 上 GitHub 访问虚幻引擎源代码。
  - 基于源代码构建 Unreal Engine 教程。
- 一个包含 C++ 游戏代码的多人游戏项目。如果你正在使用蓝图项目，请参阅虚幻文档，了解如何为项目生成 C++ 源代码。

将插件添加到游戏项目。

完成以下任务，将插件添加到您的游戏项目中。

构建 Amazon GameLift C++ 服务器软件开发工具包

1. 解压适用于虚幻引擎的Amazon GameLift 插件发布包，以提取两个压缩文件：
   - `amazon-gamelift-plugin-unreal-<>-sdk-<>.zip`
   - `GameLift-Cpp-ServerSDK-<>.zip.`

解压缩这些文件。

2. 打开该GameLift-Cpp-ServerSDK-<>文件夹，然后根据您的平台完成以下说明：Linux 或 Microsoft Windows。

Linux

1. 运行以下命令：

```
mkdir out
cd out
cmake -DBUILD_FOR_UNREAL=1 ..
make
```

这些命令会生成/lib/aws-cpp-sdk-gamelift-server.so文件。

2. 复制/lib/aws-cpp-sdk-gamelift-server.so到该amazon-gamelift-plugin-unreal/GameLiftPlugin/Source/GameliftServer/ThirdParty/GameLiftServerSDK/Linux/x86_64-unknown-linux-gnu/目录。

Microsoft Windows

1. 运行以下命令：

```
mkdir out
cd out
cmake -G "Visual Studio 17 2022" -DBUILD_FOR_UNREAL=1 ..
msbuild ALL_BUILD.vcxproj /p:Configuration=Release
```

这些命令生成以下二进制文件。

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

2. 将文件复制到amazon-gamelift-plugin-unreal\GameLiftPlugin\Source\GameliftServer\ThirdParty\GameLiftServerSDK\Win64\目录中。

处理游戏项目文件，完成以下任务。

1. 安装插件文件。

   a. 找到您的游戏项目根文件夹，例如 `... > Unreal Projects/[project-name]/`。如果那里不存在 Plugins 文件夹，请创建它。

   b. 转到解压后的amazon-gamelift-plugin-unreal文件夹。amazon-gamelift-plugin-unreal-<>-sdk-<>.zip将该GameLiftPlugin文件夹从该gamelift-plugin-unreal文件夹复制到游戏项目目录中的Plugins文件夹。

2. 将插件添加到 **.uproject** 文件。

   a. 在您的游戏项目根文件夹中，打开 .uproject 文件。

   b. 更新文件以将 “GameLiftPlugin” 和 “WebBrowserWidget” 添加到该Plugins部分并启用它们。以下代码显示了名为 “MyGame” 的游戏的更新.uproject文件。

```
UnrealProjects > MyGame > MyGame.uproject
{
  ...
  "Plugins": [
    {
      "Name": "ModelingToolsEditorMode",
      "Enabled": true,
      "TargetAllowList": [ "Editor" ]
    },
    {
      "Name": "GameLiftPlugin",
      "Enabled": true
    },
    {
      "Name": "WebBrowserWidget",
      "Enabled": true
    }
  ]
}
```

3. 更改项目的 UE 编辑器版本。

   如果您为一个编辑器版本创建了项目，但现在想要更改为另一个版本（例如源代码构建版本），则需要更新该项目。

   在游戏项目根文件夹中，选择 .uproject 文件并选择切换 Unreal Engine 版本。选择新的编辑器版本。

4. 使用您的更新重新构建项目解决方案。

   a. 在项目根文件夹中，查找解决方案（`*.sln`）文件。如果不存在任何文件，请选择 `.uproject` 文件并选择生成 Visual Studio 项目文件选项。

   b. 打开解决方案文件并构建或重建项目。

5. 确认该插件已在 UE 编辑器中启用。

   > **Note**
   >
   > 如果已经打开了编辑器，则可能需要在编辑器识别新插件之前重新启动编辑器。

   a. 在选择的 UE 编辑器中打开项目。

   b. 在主编辑器工具栏中查看新的Amazon GameLift 菜单按钮 [需要图片]。

   c. 在内容浏览器中查找 Amazon GameLift 插件资产。确保在您的查看选项设置中，已选中显示插件内容选项。

设置 AWS 用户配置文件

安装插件后，设置配置文件并将其关联到有效的 AWS 账户用户。您可以维护多个配置文件，但一次只能有一个配置文件处于活动状态。每当使用插件时，请选择要使用的配置文件。

维护多个配置文件使您能够在不同的托管方案之间切换。例如，您可以设置具有相同 AWS 凭证但 AWS 区域不同的配置文件。或者，您可以使用不同的 AWS 账户或不同的用户/权限集来设置配置文件。

> **Note**
>
> 如果您已在工作站上安装了 AWS CLI 并且已经配置了配置文件，则 Amazon GameLift 插件可以检测到它并将其列为现有配置文件。该插件会自动选择任何名为 `[default]` 的配置文件。您可以使用现有配置文件或创建新的配置文件。

管理您的 AWS 配置文件

1. 在虚幻编辑器主工具栏中，选择Amazon GameLift 菜单，然后选择设置AWS用户配置文件。该操作将打开插件的“项目设置”。展开 AWS 用户配置文件部分。

2. 如果插件未检测到现有的配置文件，它会提示你创建一个配置文件。您可以使用新账户或现有 AWS 账户创建新的配置文件。

   > **Note**
   >
   > 您需要使用 AWS 管理控制台创建新 AWS 账户，并使用适当的权限集创建或更新用户。

   在设置配置文件时，您需要提供以下信息：

   - 一个 AWS 账户。如果您需要创建新 AWS 账户，请按照提示创建账户。有关更多详细信息，请参阅创建 AWS 账户。
   - 有权使用 Amazon GameLift 和其他所需AWS服务的AWS用户。设置一个 AWS 账户有关设置具有 Amazon GameLift 权限和具有长期证书编程访问权限的 AWS Identity and Access Management (IAM) 用户的说明，请参阅。
   - AWS 用户凭证 这些凭证包含 AWS 访问密钥和 AWS 私有密钥。有关更多详细信息，请参阅获取访问密钥。
   - AWS 区域。这是您要在其中创建托管 AWS 资源的地理位置。在开发过程中，我们建议使用靠近您的实际位置的区域。查看受支持 AWS 区域列表。
3. 如果插件检测到现有配置文件，则系统不会提示您创建配置文件。如果您想更新配置文件或创建新的配置文件，请选择管理您的配置文件。

要引导您的配置文件，请执行以下操作：

所有配置文件都必须经过引导才能与 Ama GameLift zon 插件一起使用。引导会创建特定于该配置文件的 Amazon S3 存储桶。它用于存储项目配置、构建工件和其他依赖项。存储桶不会在其他配置文件之间共享。

引导涉及创建新 AWS 资源，并且可能会产生成本。

1. 在虚幻编辑器主工具栏中，选择Amazon GameLift 图标，然后选择设置AWS用户资料。该操作将打开插件的“项目设置”。展开 AWS 用户配置文件部分。
2. 在引导您的配置文件部分，从下拉列表中选择一个配置文件并检查引导状态。如果状态显示不存在存储桶，请选择引导并创建配置文件按钮，为所选配置文件创建 Amazon S3 存储桶。
3. 等待引导状态变为“活动”。这可能需要几分钟的时间。

将您的游戏设置为在 Amazon 上进行测试 GameLift Anywhere

在此工作流程中，您可以为 Amazon GameLift 功能添加客户端和服务器游戏代码，然后使用该插件将您的本地工作站指定为测试游戏服务器主机。完成集成任务后，使用该插件来构建您的游戏客户端和服务器组件。

要启动 Amazon GameLift Anywhere 工作流程，请：

- 在虚幻编辑器主工具栏中，选择 Amazon GameLift 菜单，然后选择 “随处托管”。此操作将打开插件页面部署 Anywhere，其中提供了集成、构建和启动游戏组件的六步流程。

步骤 1：设置配置文件

选择您要在遵循此工作流程时使用的配置文件。您选择的配置文件会影响工作流程中的所有步骤。您创建的所有资源都与个人资料的AWS账户相关联，并放置在个人资料的默认AWS区域中。个人资料用户的权限决定了您对AWS资源和操作的访问权限。

1. 从可用配置文件的下拉列表中选择一个配置文件。如果您还没有个人资料或想要创建新的个人资料，请前往 Amazon GameLift 菜单并选择 “设置AWS用户资料”。
2. 如果引导状态不是 “激活”，请选择 Bootstrap 配置文件并等待状态变为 “活动”。

步骤 2：设置游戏代码

在此步骤中，您将对客户端和服务器代码进行一系列更新，以添加托管功能。如果你还没有设置虚幻编辑器的源代码版本，该插件会提供指向说明和源代码的链接。

使用该插件，可以在集成游戏代码时获享便利。您可以进行最少的集成来设置基本的托管功能。您还可以进行更广泛的自定义集成。本节中的信息描述了最少集成选项。使用插件附带的测试地图，将客户端 Amazon GameLift 功能添加到您的游戏项目中。要进行服务器集成，请使用提供的代码示例更新项目的游戏模式。

集成您的服务器游戏模式

在游戏中添加服务器代码，以便在游戏服务器和 Amazon GameLift 服务之间进行通信。您的游戏服务器必须能够响应来自Amazon的请求 GameLift，例如开始新的游戏会话，并报告游戏服务器健康状况和玩家连接状态。

1. 在代码编辑器中，打开游戏项目的解决方案（`.sln`）文件，该文件通常位于项目根文件夹中。例如：`GameLiftUnrealApp.sln`。

2. 打开解决方案后，找到项目游戏模式头文件：[project-name]GameMode.h 文件。例如：GameLiftUnrealAppGameMode.h。

3. 更改头文件以使其与以下示例代码保持一致。请务必用您自己的项目名称替换 GameLiftServer “”。这些更新特定于游戏服务器；我们建议您备份原始游戏模式文件副本，以供客户端使用。

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

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/GameModeBase.h"
#include "GameLiftServerGameMode.generated.h"

struct FProcessParameters;

DECLARE_LOG_CATEGORY_EXTERN(GameServerLog, Log, All);

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

public:
    AGameLiftServerGameMode();

protected:
    virtual void BeginPlay() override;

private:
    void InitGameLift();

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

4. 打开相关的源文件 [project-name]GameMode.cpp 文件（例如 GameLiftUnrealAppGameMode.cpp）。更改代码以使其与以下示例代码保持一致。请务必用您自己的项目名称替换 GameLiftUnrealApp “”。这些更新特定于游戏服务器；我们建议您备份原始文件副本，以供客户端使用。

   以下示例代码显示如何添加与 Amazon 进行服务器集成所需的最低元素 GameLift：

- 初始化亚马逊 GameLift API 客户端。Amazon GameLift Anywhere 队列需要使用服务器参数进行`InitSDK()`调用。当您连接到 Anywhere 实例集时，插件会将服务器参数存储为控制台参数。示例代码可以在运行时访问这些值。
- 实现所需的回调函数以响应 Amazon GameLift 服务的请求`OnStartGameSession`，包括`OnProcessTerminate`、和`onHealthCheck`。
- 准备好举办游戏会话时，`ProcessReady()`使用指定端口呼叫以通知 Amazon GameLift 服务。

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

#include "GameLiftServerGameMode.h"

#include "UObject/ConstructorHelpers.h"
#include "Kismet/GameplayStatics.h"

#if WITH_GAMELIFT
#include "GameLiftServerSDK.h"
#include "GameLiftServerSDKModels.h"
#endif

#include "GenericPlatform/GenericPlatformOutputDevices.h"

DEFINE_LOG_CATEGORY(GameServerLog);

AGameLiftServerGameMode::AGameLiftServerGameMode() :
    ProcessParameters(nullptr)
{
    // Set default pawn class to our Blueprinted character
    static ConstructorHelpers::FClassFinder<APawn> PlayerPawnBPClass(TEXT("/Game/
ThirdPerson/Blueprints/BP_ThirdPersonCharacter"));

    if (PlayerPawnBPClass.Class != NULL)
    {
        DefaultPawnClass = PlayerPawnBPClass.Class;
    }

    UE_LOG(GameServerLog, Log, TEXT("Initializing AGameLiftServerGameMode..."));
}

void AGameLiftServerGameMode::BeginPlay()
```

```
{
    Super::BeginPlay();

#if WITH_GAMELIFT
    InitGameLift();
#endif
}

void AGameLiftServerGameMode::InitGameLift()
{
#if WITH_GAMELIFT
    UE_LOG(GameServerLog, Log, TEXT("Calling InitGameLift..."));

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

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

    bool bIsAnywhereActive = false;
    if (FParse::Param(FCommandLine::Get(), TEXT("glAnywhere")))
    {
        bIsAnywhereActive = true;
    }

    if (bIsAnywhereActive)
    {
        UE_LOG(GameServerLog, Log, TEXT("Configuring server parameters for
 Anywhere..."));

        // If GameLift Anywhere is enabled, parse command line arguments and pass
 them in the ServerParameters object.
        FString glAnywhereWebSocketUrl = "";
        if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereWebSocketUrl="),
 glAnywhereWebSocketUrl))
        {
            ServerParametersForAnywhere.m_webSocketUrl =
 TCHAR_TO_UTF8(*glAnywhereWebSocketUrl);
        }

        FString glAnywhereFleetId = "";
```

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

        FString glAnywhereProcessId = "";
        if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereProcessId="),
glAnywhereProcessId))
        {
            ServerParametersForAnywhere.m_processId =
TCHAR_TO_UTF8(*glAnywhereProcessId);
        }
        else
        {
            // If no ProcessId is passed as a command line argument, generate a
randomized unique string.
            ServerParametersForAnywhere.m_processId =
                TCHAR_TO_UTF8(
                    *FText::Format(
                        FText::FromString("ProcessId_{0}"),
                        FText::AsNumber(std::time(nullptr))
                    ).ToString()
                );
        }

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

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

        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_YELLOW);
```

```
        UE_LOG(GameServerLog, Log, TEXT(">>>> Web Socket URL: %s"),
 *ServerParametersForAnywhere.m_webSocketUrl);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Fleet ID: %s"),
 *ServerParametersForAnywhere.m_fleetId);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Process ID: %s"),
 *ServerParametersForAnywhere.m_processId);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Host ID (Compute Name): %s"),
 *ServerParametersForAnywhere.m_hostId);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Auth Token: %s"),
 *ServerParametersForAnywhere.m_authToken);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }

    UE_LOG(GameServerLog, Log, TEXT("Initializing the GameLift Server..."));

    //InitSDK will establish a local connection with GameLift's agent to enable
 further communication.
    FGameLiftGenericOutcome InitSdkOutcome = GameLiftSdkModule-
>InitSDK(ServerParametersForAnywhere);
    if (InitSdkOutcome.IsSuccess())
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_GREEN);
        UE_LOG(GameServerLog, Log, TEXT("GameLift InitSDK succeeded!"));
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }
    else
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_RED);
        UE_LOG(GameServerLog, Log, TEXT("ERROR: InitSDK failed : ("));
        FGameLiftError GameLiftError = InitSdkOutcome.GetError();
        UE_LOG(GameServerLog, Log, TEXT("ERROR: %s"),
 *GameLiftError.m_errorMessage);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
        return;
    }

    ProcessParameters = MakeShared<FProcessParameters>();

    //When a game session is created, GameLift sends an activation request to the
 game server and passes along the game session object containing game properties
 and other settings.
    //Here is where a game server should take action based on the game session
 object.
```

```
    //Once the game server is ready to receive incoming player connections, it
 should invoke GameLiftServerAPI.ActivateGameSession()
    ProcessParameters->OnStartGameSession.BindLambda([=]
(Aws::GameLift::Server::Model::GameSession InGameSession)
    {
        FString GameSessionId = FString(InGameSession.GetGameSessionId());
        UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"),
 *GameSessionId);
        GameLiftSdkModule->ActivateGameSession();
    });

    //OnProcessTerminate callback. GameLift will invoke this callback before
 shutting down an instance hosting this game server.
    //It gives this game server a chance to save its state, communicate with
 services, etc., before being shut down.
    //In this case, we simply tell GameLift we are indeed going to shutdown.
    ProcessParameters->OnTerminate.BindLambda([=]()
    {
        UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
        GameLiftSdkModule->ProcessEnding();
    });

    //This is the HealthCheck callback.
    //GameLift will invoke this callback every 60 seconds or so.
    //Here, a game server might want to check the health of dependencies and such.
    //Simply return true if healthy, false otherwise.
    //The game server has 60 seconds to respond with its health status. GameLift
 will default to 'false' if the game server doesn't respond in time.
    //In this case, we're always healthy!
    ProcessParameters->OnHealthCheck.BindLambda([]()
    {
        UE_LOG(GameServerLog, Log, TEXT("Performing Health Check"));
        return true;
    });

    //GameServer.exe -port=7777 LOG=server.mylog
    ProcessParameters->port = FURL::UrlConfig.DefaultPort;
    TArray<FString> CommandLineTokens;
    TArray<FString> CommandLineSwitches;

    FCommandLine::Parse(FCommandLine::Get(), CommandLineTokens,
 CommandLineSwitches);

    for (FString SwitchStr : CommandLineSwitches)
```

```
    {
        FString Key;
        FString Value;

        if (SwitchStr.Split("=", &Key, &Value))
        {
            if (Key.Equals("port"))
            {
                ProcessParameters->port = FCString::Atoi(*Value);
            }
        }
    }

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

    //The game server calls ProcessReady() to tell GameLift it's ready to host game
 sessions.
    UE_LOG(GameServerLog, Log, TEXT("Calling Process Ready..."));
    FGameLiftGenericOutcome ProcessReadyOutcome = GameLiftSdkModule-
>ProcessReady(*ProcessParameters);

    if (ProcessReadyOutcome.IsSuccess())
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_GREEN);
        UE_LOG(GameServerLog, Log, TEXT("Process Ready!"));
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }
    else
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_RED);
        UE_LOG(GameServerLog, Log, TEXT("ERROR: Process Ready Failed!"));
        FGameLiftError ProcessReadyError = ProcessReadyOutcome.GetError();
        UE_LOG(GameServerLog, Log, TEXT("ERROR: %s"),
 *ProcessReadyError.m_errorMessage);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }

    UE_LOG(GameServerLog, Log, TEXT("InitGameLift completed!"));
#endif
```

```
}
```

整合客户端游戏地图

启动游戏地图包含蓝图逻辑和用户界面元素，这些元素已经包含请求游戏会话和使用连接信息连接到游戏会话的基本代码。您可以按原样使用地图，也可以根据需要对其进行修改。将启动游戏地图与其他游戏资产一起使用，例如 Unreal Engine 提供的第三人称模板项目。这些资产可在内容浏览器中找到。您可以使用它们来测试插件的部署工作流程，或者作为为游戏创建自定义后端服务的指南。

该启动地图具有以下特性：

- 它包括 Anywhere 实例集和托管 EC2 实例集的逻辑。在运行客户端时，您可以选择连接到任一实例集。
- 客户端功能包括查找游戏会话 (SearchGameSessions())、创建新的游戏会话 CreateGameSession (()) 以及直接加入游戏会话。
- 它会从项目的 Amazon Cognito 用户群体中获得一个唯一的玩家 ID（这是部署的 Anywhere 解决方案的一部分）。

要使用启动游戏地图

1. 在 UE 编辑器中，打开项目设置、地图和模式页面，然后展开默认地图部分。
2. 对于编辑器启动地图，从下拉列表中选择 StartupMap “”。您可能需要搜索位于 `... > Unreal Projects/[project-name]/Plugins/Amazon GameLift Plugin Content/Maps` 中的文件。
3. 对于游戏默认地图，请从下拉列表中选择相同的 StartupMap “”。
4. 在 “服务器默认地图” 中，选择 “ThirdPersonMap”。这是您的游戏项目中包含的默认地图。这张地图是为游戏中的两个玩家设计的。
5. 打开服务器默认地图的详细信息面板。将 “GameMode 覆盖” 设置为 “无”。
6. 展开默认模式部分，将全局默认服务器游戏模式设置为您为服务器集成而更新的游戏模式。

对项目进行这些更改后，就可以开始构建游戏组件了。

构建您的游戏组件

1. 创建新的服务器和客户端目标文件

a. 在游戏项目文件夹中，转到源文件夹并找到 Target.cs 文件。

b. 将文件 [project-name]Editor.Target.cs 复制到两个名为 [project-name]Client.Target.cs 和 [project-name]Server.Target.cs 的新文件中。

c. 编辑每个新文件以更新类名和目标类型值，如下所示：

```
UnrealProjects > MyGame > Source > MyGameClient.Target.cs
// Copyright Epic Games, Inc. All Rights Reserved.

using UnrealBuildTool;
using System.Collections.Generic;

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

```
UnrealProjects > MyGame > Source > MyGameServer.Target.cs
// Copyright Epic Games, Inc. All Rights Reserved.

using UnrealBuildTool;
using System.Collections.Generic;

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

2. 更新 .Build.cs 文件

a. 打开您的项目的 .Build.cs 文件。此文件位于 UnrealProjects/[project name]/Source/[project name]/[project name].Build.cs 中：

b. 更新 ModuleRules 类，如以下代码示例所示。

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

    if (Target.Type == TargetRules.TargetType.Server)
    {
      PublicDependencyModuleNames.AddRange(new string[]
 { "GameLiftServerSDK" });
      PublicDefinitions.Add("WITH_GAMELIFT=1");
    }
    else
    {
      PublicDefinitions.Add("WITH_GAMELIFT=0");
    }
  }
}
```

3. 重新生成您的游戏项目解决方案。
4. 在 Unreal Engine 编辑器的源代码构建版本中打开游戏项目。
5. 对客户端和服务器执行以下操作：

a. 选择一个目标。前往平台、Windows，然后选择以下选项之一：

- 服务器：[your-application-name]Server
- 客户端：[your-application-name]Client

b. 启动构建。转到平台、Windows、程序包项目。

每个打包过程都会生成一个可执行文件：[your-application-name]Client.exe 或 [your-application-name]Server.exe。

在插件中，在本地工作站上设置客户端和服务器构建可执行文件的路径。

步骤 3：连接到 Anywhere 实例集

在此步骤中，您将指定要使用的 Anywhere 实例集。Anywhere 实例集定义了一组计算资源，这些资源可以位于任何地方，用于托管游戏服务器。

- 如果您当前使用的AWS账户已有 Anywhere 舰队，请打开舰队名称下拉字段并选择舰队。此下拉列表仅显示当前处于活动状态的用户配置文件所在 AWS 区域的 Anywhere 实例集。
- 如果现有实例集不存在，或者您想创建新实例集，请选择创建新的 Anywhere 实例集并提供实例集名称。

在您为项目选择 Anywhere 队列后，Amazon GameLift 会验证队列状态是否为活跃并显示队列 ID。你可以在虚幻编辑器的输出日志中跟踪此请求的进度。

步骤 4：注册工作站

在此步骤中，您将本地工作站注册为新的 Anywhere 实例集中的计算资源。

1. 输入本地计算机的计算名称。如果您在实例集中添加多个计算，则名称必须是唯一的。
2. 为您的本地计算机提供 IP 地址。此字段默认为您计算机的公有 IP 地址。只要你在同一台计算机上运行游戏客户端和服务器，你也可以使用 localhost (127.0.0.1)。
3. 选择注册计算。你可以在虚幻编辑器的输出日志中跟踪此请求的进度。

作为对这一操作的响应，Amazon GameLift 会验证它是否可以连接到计算并返回有关新注册的计算的信息。它还会创建游戏可执行文件在初始化与 Ama GameLift zon 服务通信时所需的控制台参数。

步骤 5：生成身份验证令牌

在 Anywhere 计算机上运行的游戏服务器进程需要身份验证令牌才能调用该 GameLift 服务。每当从插件启动游戏服务器时，该插件都会自动生成并存储 Anywhere 实例集的身份验证令牌。身份验证令牌值存储为命令行参数，您的服务器代码可以在运行时检索该参数。

您无需在此步骤中执行任何操作。

步骤 6：启动游戏

至此，您已经完成了使用 Amazon 在本地工作站上启动和玩多人游戏所需的所有任务 GameLift。

1. 启动游戏服务器。游戏服务器将在准备好举办游戏会话 GameLift 时通知亚马逊。

2. 启动您的游戏客户端，然后使用新功能启动新的游戏会话。此请求 GameLift 通过新的后端服务发送到 Amazon。作为回应 GameLift，Amazon 会调用在本地计算机上运行的游戏服务器以启动新的游戏会话。当游戏会话准备好接受玩家时，Amazon 会 GameLift 提供连接信息供游戏客户端加入游戏会话。

使用托管 EC2 实例集将游戏部署到云托管

在此工作流程中，您可以使用插件修改您的游戏，使其托管在 Amazon 管理的基于云的计算资源上 GameLift。您可以为亚马逊 GameLift 功能添加客户端和服务器游戏代码，然后将您的服务器版本上传到亚马逊 GameLift 服务以部署到基于云的资源。此工作流程完成后，您将拥有一个可以连接到云端游戏服务器的游戏客户端。

要启动亚马逊 GameLift 托管的 Amazon EC2 工作流程，请执行以下操作：

- 在虚幻编辑器主工具栏中，选择Amazon GameLift 菜单，然后选择托管EC2托管。此操作将打开插件页面部署 Amazon EC2 Fleet，其中提供了集成、构建、部署和启动游戏组件的六步流程。

步骤 1：设置配置文件

选择您要在遵循此工作流程时使用的配置文件。您选择的配置文件会影响工作流程中的所有步骤。您创建的所有资源都与个人资料的AWS账户相关联，并放置在个人资料的默认AWS区域中。个人资料用户的权限决定了您对AWS资源和操作的访问权限。

1. 从可用配置文件的下拉列表中选择一个配置文件。如果您还没有个人资料或想要创建新的个人资料，请前往 Amazon GameLift 菜单并选择 “设置AWS用户资料”。
2. 如果引导状态不是 “激活”，请选择 Bootstrap 配置文件并等待状态变为 “活动”。

步骤 2：设置游戏代码

在此步骤中，您将对客户端和服务器代码进行一系列更新，以添加托管功能。如果你还没有设置虚幻编辑器的源代码版本，该插件会提供指向说明和源代码的链接。

如果您已将游戏与 Anywhere 实例集一起使用，则无需对游戏代码进行任何更改。如果您使用的是启动游戏地图，则这也适用于 EC2 部署。

- 设置游戏代码 (Anywhere)
- 构建您的游戏组件

构建游戏服务器后，请完成以下任务，为将其上传到 Amazon 做好准备 GameLift。

打包服务器构建以进行云部署

在 Unreal 编辑器默认打包服务器构建文件的 WindowsServer 文件夹中，添加以下内容

1. 将插件下载中包含的安装脚本复制到 WindowsServer 文件夹的根目录中。查找文件 [project-name]/Plugins/Resources/CloudFormation/extra_server_resources/install.bat。Amazon GameLift 使用此文件在每个 EC2 托管资源上安装服务器版本。
2. 将 Visual Studio 安装中包含的 VC_redist.x64.exe 文件复制到 WindowsServer 文件夹的根目录中。此文件通常位于 C:/Program Files (x86)/Microsoft Visual Studio/2019/Professional/VC/Redist/MSVC/v142。
3. 将游戏服务器构建的 OpenSSL DLL 复制到 WindowsServer/MyGame/Binaries/Win64 文件夹中。确保 DLL 的版本与服务器构建中使用的版本相同。复制以下文件：
   - libssl-3-x64.dll
   - libcrypto-3-x64.dll

步骤 3：选择部署方案

在此步骤中，您可以选择此时要部署的游戏托管解决方案。使用任何方案，您都可以对游戏进行多个部署。

- 单区域队列：将您的游戏服务器部署到活动配置文件默认AWS区域中的单个托管资源队列。此方案是测试服务器与 AWS 集成和服务器构建配置的良好起点。它部署了以下资源：
  - 已安装并运行游戏服务器构建的 AWS 实例集（按需型）。
  - Amazon Cognito 用户群体和客户端，使玩家能够进行身份验证和开始游戏。
  - 将用户群体与 API 关联的 API 网关授权器。
  - WebACL，用于限制玩家对 API 网关的过多调用。
  - API 网关 + Lambda 函数，供玩家申请游戏位置。如果两者都不可用，则此函数调用 CreateGameSession()。
  - API 网关 + Lambda 函数，供玩家获取游戏请求的连接信息。
- FlexMatch 舰队：将你的游戏服务器部署到一组舰队中，并设置一个带有规则的 FlexMatch 匹配器来创建玩家对战。此场景使用低成本 Spot 托管和多舰队、多地点结构，以实现持久可用性。当您准备开始为托管解决方案设计匹配器组件时，这种方法非常有用。在这种情况下，您将为此解决方案创建基本资源，以后可以根据需要对其进行自定义。它部署了以下资源：

- FlexMatch 配对配置和配对规则设置为接受玩家请求和表单匹配。
- 三个 AWS 实例集，安装了游戏服务器构建，并在多个位置运行。包括两个竞价型实例集和一个按需型实例集作为备份。
- AWS 游戏会话放置队列，通过寻找尽可能好的托管资源（基于可行性、成本、玩家延迟等）并启动游戏会话来满足对提议对战的请求。
- Amazon Cognito 用户群体和客户端，使玩家能够进行身份验证和开始游戏。
- 将用户群体与 API 关联的 API 网关授权器。
- WebACL，用于限制玩家对 API 网关的过多调用。
- API 网关 + Lambda 函数，供玩家申请游戏位置。此函数调用 `StartMatchmaking()`。
- API 网关 + Lambda 函数，供玩家获取游戏请求的连接信息。
- Amazon DynamoDB 表，用于存储玩家的对战票证和游戏会话信息。
- SNS 主题 + 用于处理事件的 Lambda 函数。 GameSessionQueue

步骤 4：设置游戏参数

在此步骤中，您将描述要上传到 AWS 的游戏。

- 服务器版本名称：为您的游戏服务器版本提供一个有意义的名称。 AWS使用此名称来指代已上传并用于部署的服务器版本的副本。
- 服务器构建操作系统：输入构建服务器以在其中运行的操作系统。这将告诉 AWS 使用哪种类型的计算资源来托管您的游戏。
- 游戏服务器文件夹：确定本地服务器构建文件夹的路径。
- 游戏服务器构建：确定游戏服务器可执行文件的路径。
- 游戏客户端路径：确定游戏客户端可执行文件的路径。
- 客户端配置输出：此字段需要指向您的客户端构建中包含 AWS 配置的文件夹。在以下位置寻找：`[client-build]/[project-name]/Content/CloudFormation`。

步骤 5：部署方案

在此步骤中，您将根据所选的部署方案将游戏部署到云托管解决方案。在 AWS 验证服务器构建、配置托管资源、安装游戏服务器、启动服务器进程以及让它们做好托管游戏会话的准备时，此过程可能需要长达 40 分钟。

要开始部署，请选择部署 CloudFormation。您可以在此处跟踪您的游戏托管状态。要了解更多详细信息，您可以登录 AWS 管理控制台了解 AWS 和查看事件通知。请务必使用与插件中活跃用户配置文件相同的账户、用户和 AWS 区域登录。

部署完成后，您的游戏服务器将安装在 AWS EC2 实例上。至少有一个服务器进程正在运行并准备开始游戏会话。

步骤 6：启动客户端

至此，您已经完成了启动和玩由Amazon托管的多人游戏所需的所有任务 GameLift。要玩游戏，请启动您的游戏客户端实例。

如果您部署了单个实例集方案，则可以用一个玩家打开一个客户端实例，进入服务器地图并四处移动。打开游戏客户端的其他实例，将第二个玩家添加到同一个服务器游戏地图中。

如果您部署了 FlexMatch 场景，则解决方案会等待至少两个客户端排队等候游戏会话放置，然后玩家才能进入服务器地图。

获取 Amazon GameLift 实例的实例集数据

在某些情况下，您的自定义游戏版本或实时服务器脚本可能需要有关 Amazon GameLift 队列的信息。例如，您的游戏版本或脚本可能包含以下代码：

- 根据实例集数据监控活动。
- 汇总指标以按实例集数据跟踪活动。（许多游戏使用这些数据进行 LiveOps 活动。）
- 为自定义游戏服务提供相关数据，例如用于对战、额外容量扩展或测试。

队列信息以 JSON 文件形式显示在每个实例上，位于以下位置：

- Windows: `C:\GameMetadata\gamelift-metadata.json`
- Linux：`/local/gamemetadata/gamelift-metadata.json`

该 `gamelift-metadata.json` 文件包含 Amazon GameLift 实例集资源的属性。

示例 JSON 筛选条件

```
{
    "buildArn":"arn:aws:gamelift:us-west-2:123456789012:build/
build-1111aaaa-22bb-33cc-44dd-5555eeee66ff",
```

```
    "buildId":"build-1111aaaa-22bb-33cc-44dd-5555eeee66ff",
    "fleetArn":"arn:aws:gamelift:us-west-2:123456789012:fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
    "fleetDescription":"Test fleet for Really Fun Game v0.8",
    "fleetId":"fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
    "fleetName":"ReallyFunGameTestFleet08",
    "fleetType":"ON_DEMAND",
    "instanceRoleArn":"arn:aws:iam::123456789012:role/S3AccessForGameLift",
    "instanceType":"c5.large",
    "serverLaunchPath":"/local/game/reallyfungame.exe"
}
```

添加 FlexMatch 对战

使用 Amazon GameLift FlexMatch 为 Amazon GameLift 托管的游戏添加玩家对战功能。您可以将 FlexMatch 与自定义游戏服务器或实时服务器配合使用。

FlexMatch 可将对战服务与自定义规则引擎搭配使用。您可以根据对您的游戏有意义的玩家属性和游戏模式来设计如何将玩家匹配在一起。FlexMatch 管理着评估正在寻找游戏的玩家、与一支或多支团队对战以及开始游戏会话以托管对战的基本要素。

要使用完整的 FlexMatch 服务，您必须将托管资源设置为队列。Amazon GameLift 使用队列来为跨多个地区和计算类型的游戏找到尽可能好的托管位置。特别是，Amazon GameLift 队列可以使用游戏客户端提供的延迟数据来放置游戏会话，以便玩家在玩游戏时体验到尽可能低的延迟。

有关 FlexMatch 的更多信息，包括将对战集成到游戏中的详细帮助，请参阅以下 Amazon GameLift FlexMatch 开发人员主题：

- Amazon GameLift FlexMatch 的工作原理
- FlexMatch 集成步骤

使用 Amazon GameLift 容器管理托管

本文档适用于公开预览版中的一项功能。本文档随时可能更改。

Amazon GameLift 提供完整的云托管服务，以支持游戏服务器托管的容器化解决方案。借助 Amazon GameLift 集装箱船队，您可以利用集装箱的优势，例如便携性、灵活性和容错性。

主要特征

Amazon GameLift 集装箱船队提供以下功能。

- 开发带有轻量级容器的自定义容器架构，以便在 Amazon GameLift 托管资源上运行您的游戏服务器软件。
- 包括 Amazon A GameLift gent 以管理容器内游戏服务器进程的生命周期。计算机上的代理会执行您的指示，说明何时以及如何启动服务器进程，以及为托管游戏会话需要维护多少进程。
- 自定义 Amazon 提供的资源 GameLift ，以便使用您的游戏服务器应用程序构建容器映像。使用提供的 dockerfile 创建基于 Linux 的容器镜像。将您的集装箱队列的图像存储在亚马逊弹性容器注册表 (Amazon ECR) Container Registry 的私有存储库中。
- 通过将容器舰队资源部署到 Amazon GameLift 支持的任何区域 AWS 区域 或本地区域，提供低延迟的玩家体验。创建多地点集装箱船队，简化船队管理。请参阅 亚马逊 GameLift 托管地点。
- 使用 Ama GameLift Anywhere zon 舰队测试您的容器化游戏托管解决方案。使用 Anywhere 在本地测试您的解决方案开发，包括您的 Amazon GameLift SDK 集成和容器映像配置。
- 使用容器特定的性能指标跟踪游戏托管性能。使用硬件指标监控舰队资源的运行状况。
- 使用 Amazon GameLift 游戏会话放置工具（包括队列和 FlexMatch配对）将玩家与您的容器舰队上托管的最佳游戏会话进行匹配。
- 使用适用于 Amazon 的 AWS CloudFormation 模板管理集装箱船队资源 GameLift。

在公开预览期间使用集装箱舰队

新的集装箱舰队功能目前处于公开预览阶段。在此阶段，支持以下 Amazon GameLift 功能：

- 使用容器队列托管专为 Linux 构建的游戏服务器。容器队列使用`Amazon_Linux_2023`并支持 Linux 容器镜像。不支持 Windows 容器。
- 仅将游戏服务器项目与 Amazon GameLift 服务器 SDK 版本 5+ 集成。不支持以前的版本。

- 使用亚马逊 GameLift 支持的任何 Amazon EC2 按需实例类型。目前不支持 Spot 队列。

容器在 Amazon 中的运作方式 GameLift

本文档适用于公开预览版中的一项功能。本文档随时可能更改。

Amazon GameLift 容器队列旨在让您灵活部署和扩展容器化应用程序。它使用亚马逊弹性容器服务 (Amazon ECS) Service 来管理您的亚马逊 GameLift 舰队的任务部署和执行。本主题描述了在 Amazon GameLift 托管队列上运行容器的基本结构元素，说明了常见架构，并概述了一些核心概念。

集装箱船队组件

实例集

容器队列是运行您的容器化游戏服务器的 Amazon EC2 实例的集合 GameLift，由 Amazon 管理。创建队列时，您可以配置如何将容器架构和游戏服务器软件部署到每个队列实例。您可以将集装箱舰队部署到单个 AWS 区域 或多个地理位置。您可以使用 Amazon GameLift 手动或自动扩展工具来扩展容器舰队的容量，以托管游戏会话和玩家。

实例

Amazon EC2 实例是为游戏托管提供计算容量的虚拟服务器。借助 Amazon GameLift，您可以从各种实例类型中进行选择。每种实例类型都提供不同的 CPU、内存、存储和网络容量组合。

当您创建容器队列时，Amazon 会根据您选择的实例类型和队列配置来 GameLift 部署实例。每个部署的队列实例都是相同的，并且以相同的方式运行您的容器化游戏服务器软件。队列中的实例数量决定了队列的大小和游戏托管容量。

容器组

Amazon GameLift 使用容器组的概念来描述和管理一组容器。容器组类似于容器 “任务” 或 “pod”。在每个容器组中，您可以定义容器如何共享可用 CPU 和内存资源。您还可以在容器之间设置依赖关系并管理容器组的生命周期。

容器组可以在每个队列实例之间进行复制，以优化资源使用。您可以通过设置容器组的调度策略来管理复制，如下所示：

- 副本容器组管理运行游戏服务器应用程序和支持软件的容器。所有容器舰队都必须定义一个副本容器组。一个副本组在每个队列实例上可能有多个副本，具体取决于容器组的要求和所用实例类型的资源。副本组中的所有容器都会自动在实例中一起扩展。

- 守护程序容器组是可选的，可用于运行后台服务或实用程序，例如用于监控。你的游戏服务器软件不直接依赖于守护程序组中的进程。守护程序容器组不会被复制——每个队列实例最多有一个守护程序组副本。这意味着守护程序组中的容器不能与副本组中的容器一起跨队列实例进行扩展。

容器队列必须有一个副本容器组，并且可以选择拥有一个守护程序组。

容器

容器是基于容器的架构中最基本的元素。它由包含软件可执行文件和依赖文件的容器镜像组成。当您定义要用于 Amazon 的容器时 GameLift，您可以配置软件在容器中的运行方式。

集装箱船队中的每个集装箱组必须有一个被指定为 “必备” 的集装箱。基本容器驱动着容器组的生命周期。如果基本容器出现故障，则整个容器组将重新启动。

容器类型包括：

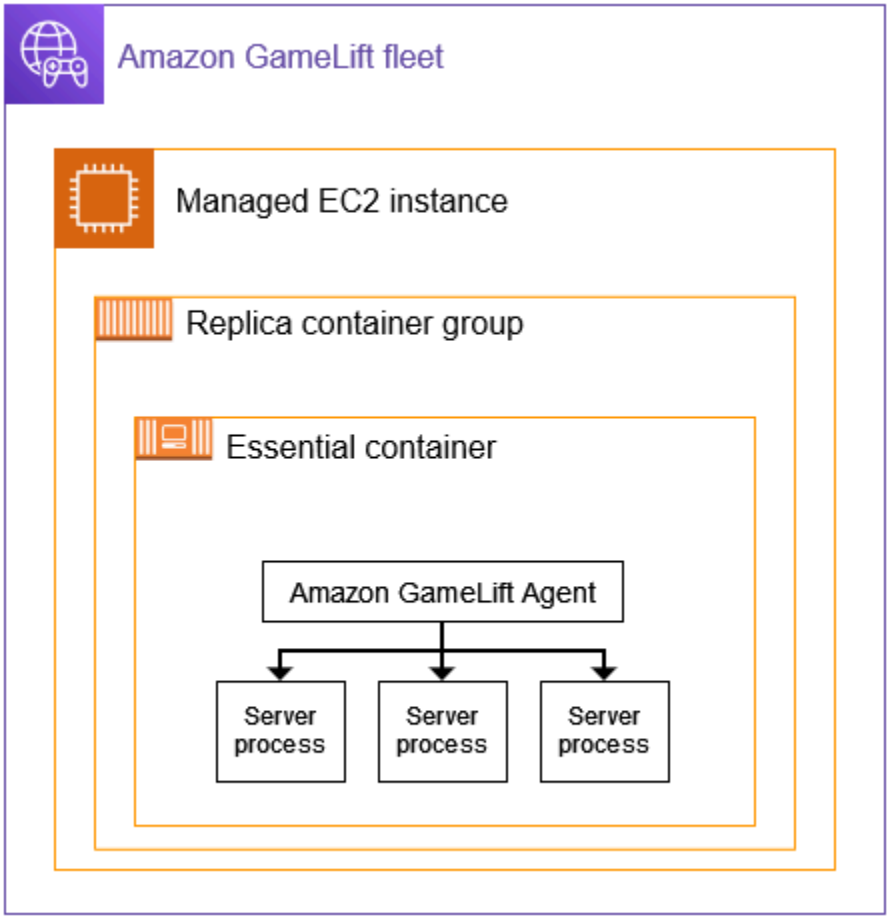
- 基本副本容器包含运行游戏服务器进程和为玩家托管游戏会话所需的一切。它包括您的游戏服务器版本（与 Amazon GameLift 服务器 SDK 集成）和相关软件。它还包括管理游戏服务器进程生命周期的 Amazon A GameLift gent。舰队的仿制集装箱组正好有一个基本的复制集装箱。
- 非必需的副本容器（也称为 “sidecar” 容器）运行软件来支持您的游戏服务器应用程序。使用边车容器可以让你在游戏服务器旁边运行和扩展支持软件，但可以作为单独的容器进行管理。如果这种类型的容器出现故障，则只有容器本身会重新启动；容器组不会受到影响。
- 守护程序容器运行守护程序服务来管理后台进程。守护程序容器的常见用途是运行 A mazon CloudWatch (CloudWatch) 代理来收集容器的指标、日志和跟踪。守护程序容器可以是必需的，也可以是非必需的，具体取决于容器故障何时必须导致容器组重启。

计算

计算是一种队列托管资源，已在 Amazon GameLift 服务中注册并能够与该服务进行通信。在容器队列中，计算是带有管理计算注册过程的进程的容器。在集装箱舰队的基本复制容器中，Amazon A GameLift gent 会自动将此容器注册为计算容器。

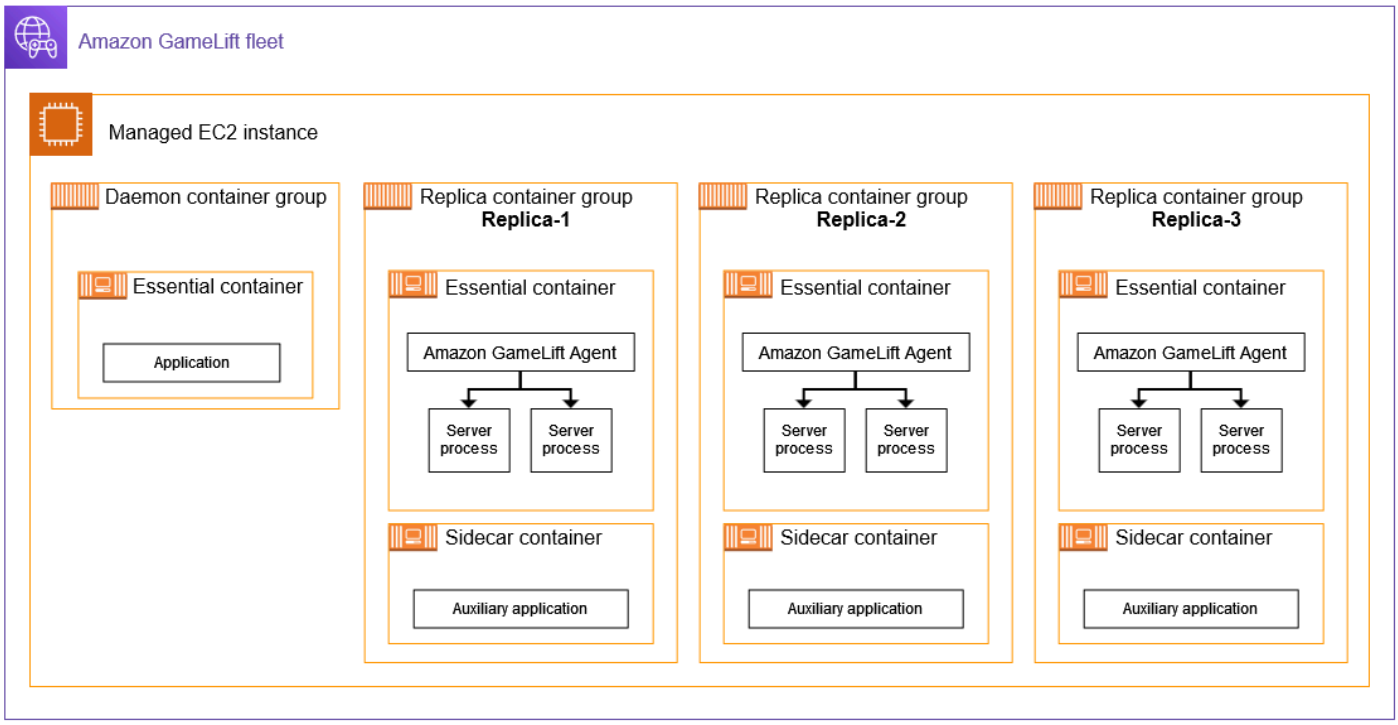
常见架构

下图说明了最简单的集装箱船队结构。在此结构中，队列中的每个实例都维护一个副本容器组的副本。容器组有一个基本容器，用于运行 Amazon A GameLift gent、游戏服务器应用程序以及用于托管游戏会话的所有支持软件。代理实现特定于队列的指令，以同时运行三个服务器进程。由于每个实例有一个副本容器组，因此每个队列实例同时运行三个服务器进程。



第二个示例说明了更复杂的集装箱船队设计。在此示例中，队列有一个包含多个容器的副本容器组和一个包含一个容器的守护程序容器组。队列配置在每个队列实例上放置三个副本容器组副本。守护程序容器组永远不会被复制。

中的每组副本组容器在每个实例上都有三个副本。在每个基本副本容器中，代理被指示同时运行两个服务器进程。因此，每个队列实例同时运行六个服务器进程（三个基本副本容器中各运行两个进程）。



核心概念

本节总结了 Amazon 如何 GameLift 实现一些基本的容器概念。有关如何使用集装箱船队的说明，请参阅本指南中的相关主题。

集装箱组包装

在开发用于部署到集装箱舰队中的容器结构时，一个共同的目标是优化对可用计算能力的使用。要实现此目标，请在不影响游戏服务器性能的情况下找到可以在舰队实例上放置的最大数量的副本容器组。

Amazon GameLift 可以帮助您做到这一点。它根据以下信息计算每个实例的最大副本组数：

- 队列的实例类型以及可用的 CPU 和内存资源。
- 您为副本组中的所有容器设置的 CPU 和内存要求。

  您为守护程序组中的所有容器（如果有）设置的 CPU 和内存要求。
- 为管理每个实例上的容器和其他关键应用程序而预留的资源。

创建集装箱船队时，您可以选择使用计算得出的最大数量，也可以通过指定所需数量来覆盖计算出的数字。作为最佳实践，请对容器化游戏服务器软件进行试验，以确定准确的资源需求。使用这些数据来寻找游戏服务器性能的最佳打包策略。

游戏服务器和 Amazon GameLift 代理

在构建基本副本容器时，您需要将游戏服务器软件和 Amazon A GameLift gent 一起打包到同一个容器镜像中。这个计算代理控制容器中游戏服务器的生命周期。在每个副本容器组中，基本副本容器运行代理和所有游戏服务器进程。

Amazon A GameLift gent 执行集装箱队列运行时配置中的指令。运行时配置标识 (1) 要开始运行的可执行文件，(2) 一组可选的启动参数，以及 (3) 要同时运行的进程数。运行时配置可以包含针对多个不同可执行文件的指令。游戏服务器可执行文件必须至少有一条指令。例如，运行时配置可能指示代理维护游戏服务器可执行文件的 10 个进程以供生产使用，1 个具有特殊启动参数的相同可执行文件用于测试，1 个进程用于日志实用程序。

您可以随时修改队列的运行时配置。Amazon GameLift 代理会定期向该服务请求更新。当有更新的运行时配置可用时，代理会收到该配置并开始执行指令。操作可能包括添加或关闭服务器进程。

Amazon A GameLift gent 是计算代理的开源版本，亚马逊 GameLift 用于托管 EC2 队列。本指南提供了有关如何从源代码构建代理并将其构建到容器映像中的说明。代理处理以下任务：

服务器进程管理：

- 根据运行时配置启动、关闭和替换服务器进程。
- 当服务器进程未及时激活时，将其关闭。
- 服务器进程终止 GameLift 时向 Amazon 报告。
- 为服务器进程发出队列事件。

容器管理：

- 按照 Amazon 的提示关闭服务器进程 GameLift。
- 报告容器运行状况。

日志上传任务：

- 将游戏会话日志上传到指定的 Amazon S3 存储桶。
- 将计算代理日志上传到指定的 Amazon S3 存储桶。

扩展实例集容量

舰队容量衡量舰队在任何时候可以托管的游戏会话数量。您还可以根据舰队可以同时支持的玩家数量来衡量容量。

要增加或减少队列的托管容量，您可以添加或移除队列实例。集装箱舰队的打包策略决定了每个舰队实例上同时运行多少游戏会话。这个数字告诉你在增加或减少舰队容量时增加或减少的游戏会话（和玩家位置）的数量。

对于集装箱船队，您可以使用Amazon GameLift 提供的任何扩展方法。其中包括：

- 通过设置特定的所需队列实例数来手动设置队列容量。
- 通过定位所需的可用实例缓冲区来设置自动扩展（目标跟踪）。此方法会自动维护一组闲置的主机资源，以便新来的玩家可以随时快速进入游戏。随着玩家需求的增加或减少，该缓冲区的大小会不断调整。
- 使用自定义缩放规则设置自动扩展（高级功能）。

游戏客户端/服务器连接

托管 EC2 队列和容器队列以类似的方式处理游戏客户端和云托管游戏服务器之间的连接。当 Amazon GameLift 创建新的游戏会话时，该服务会传达游戏会话的连接信息。游戏客户端使用这些信息直接连接到托管游戏会话的游戏服务器。对于所有类型的舰队，连接信息均由 IP 地址和端口分配组成。

创建集装箱船队时，您需要定义两组端口范围。首先，定义一系列面向外部的连接端口，允许游戏客户端连接到游戏。其次，定义一组仅限内部使用的容器端口，这些端口被分配给在容器中运行的每个游戏服务器进程。Amazon GameLift 动态地将内部容器端口映射到外部连接端口，让玩家可以访问游戏。这种方法可以保护您的游戏服务器免受直接访问容器端口，从而提供额外的安全层。

在为容器队列定义端口范围时，必须为范围提供足够的端口，以容纳在实例上的容器上同时运行的所有服务器进程。

为了获得更多控制，您还可以为队列设置入站权限。入站权限决定哪些连接端口对传入流量开放。您可以随时更改舰队的入站权限。有了入站权限，您可以根据需要快速关闭所有连接端口、打开某些端口或全部打开。

Amazon GameLift 容器的开发路线图

本文档适用于公开预览版中的一项功能。本文档随时可能更改。

以下工作流程总结了让游戏服务器在 Amazon GameLift 容器队列上运行的步骤。

第 1 步：将您的游戏与 Amazon 集成 GameLift

为您的游戏服务器添加功能，使其在部署到集装箱舰队时可以与 Amazon GameLift 服务进行通信。如果您使用 FlexMatch 配对，请将此功能添加到您的游戏服务器和客户端。有关详细信息，请参阅将您的游戏与 Amazon 集成 GameLift。

- 获取 Amazon GameLift 服务器 SDK（版本 5+），并使用您的游戏项目进行设置。服务器 SDK 有 C++、C# 和 Go 版本。
- 修改您的游戏服务器代码以添加所需的服务器 SDK 功能。
- Package 为你的 Linux 游戏服务器版本打包。如果你在 Windows 上进行开发，则此步骤可能需要额外的工作才能设置 Linux 环境。
- （可选）使用 Amazon GameLift Anywhere 舰队测试您的游戏服务器集成。在准备容器映像之前进行测试，以隔离集成工作中的问题。要测试游戏客户端/服务器连接，请同时集成您的游戏客户端。

> **Note**
>
> 如果你在 Windows 上进行开发，请设置一个单独的 Linux 工作区，或者使用诸如适用于 Linux 的 Windows 子系统 (WSL) 之类的工具。你需要一个 Linux 环境来测试你的游戏服务器版本，以及构建和测试你的容器镜像。

第 2 步：准备游戏服务器容器镜像

创建运行游戏服务器进程的容器镜像，并将其存储在亚马逊弹性容器注册表 (Amazon ECR) Container Registry 存储库中以供亚马逊使用。 GameLift有关详细说明，请参阅 使用游戏服务器软件准备容器镜像。

- 为容器映像设置工作目录，包括您的 Linux 游戏构建、安装脚本以及所有支持的软件和依赖项。
- 获取 Amazon A GameLift gent 源代码，对其进行编译，然后将该`jar`文件添加到您的工作目录中。
- 获取默认 Dockerfile 并对其进行修改，以便使用游戏服务器软件配置容器镜像。
- 构建您的容器镜像。在 Linux 环境中执行此步骤。
- 创建 Amazon ECR 私有存储库并将您的容器映像推送到该存储库。在您计划部署容器队列 AWS 账户 的相同 AWS 区域 位置创建存储库。
- （可选）使用Anywhere队列测试您的容器映像。您可以设置运行时配置，将指令传递给 Amazon GameLift 代理。

第 3 步：创建您的容器和容器组

为在 Amazon 上托管游戏设计容器架构 GameLift。请参阅设计一支亚马逊 GameLift 集装箱船队和为 Amazon 集装箱队列创建 GameLift 容器组定义。

- 定义您的容器配置。对于每个容器，您将定义诸如运行时进程、内存分配、运行状况检查、网络端口等问题。
- 使用 Amazon GameLift 控制台或 AWS CLI 使用您的容器配置创建容器组定义。当您创建容器组定义时，Amazon GameLift 会拍摄当时每个容器镜像的快照。

第 4 步：将您的容器化游戏服务器部署到容器舰队

使用在上一步中创建的容器组定义来创建容器队列并部署您的容器化游戏服务器软件。请参阅 创建 Amazon GameLift 集装箱船队。

- 使用 Amazon GameLift 控制台或 AWS CLI 创建集装箱舰队。
- 在部署和激活队列实例时跟踪队列状态。检查队列创建事件，以验证队列是否已成功部署到所有地点。
- 验证游戏客户端是否可以请求和加入游戏会话并玩游戏。如果您设置了配对，请测试这些场景。

第 5 步：管理您的车队

在为生产级使用做准备时，请构建游戏托管解决方案并管理托管生命周期。

- 在其他地方创建多地点舰队和舰队 AWS 区域 来支持你的玩家群。
- 使用队列或 FlexMatch 配对配置游戏托管位置。请参阅以下资源：
  - 《Amazon GameLift 开发人员指南》中的设置游戏会话置放的 GameLift 队列
  - FlexMatch 开发人员指南
- 设置自动扩展，根据玩家对游戏会话的需求管理舰队容量。
- 为您的集装箱船队设置监控。使用 Amazon GameLift 指标，检索游戏会话日志和容器日志，设置对单个容器的远程访问。
- 建立集装箱船队的长期管理。使用船队别名来简化集装箱船队的更新流程。创建 AWS CloudFormation 模板来管理车队生命周期。请参阅以下资源：
  - 向 Amazon GameLift 舰队添加别名
  - 使用 AWS CloudFormation 管理资源

将您的游戏与 Amazon 集成 GameLift

本文档适用于公开预览版中的一项功能。本文档随时可能更改。

在使用游戏服务器软件创建容器镜像并将其部署到亚马逊 GameLift 进行云托管之前，请先将您的游戏项目与 Amazon GameLift 服务器 SDK 集成，然后构建一个在 Linux 上运行的游戏服务器。本主题介绍了 Amazon GameLift 提供的各种集成工具。

托管的游戏服务器必须能够与 Amazon GameLift 服务通信。通过将 Amazon S GameLift erver SDK（版本 5+）添加到您的游戏项目并修改游戏的服务器代码来设置通信。Amazon GameLift 提供服务器 SDK 资源和文档，以支持多种语言和游戏引擎。

容器化游戏服务器的集成过程与将游戏服务器集成到托管的 EC2 或 Ama GameLift Anywhere zon 队列上托管的过程几乎相同。

集成工具

Amazon GameLift 为集成提供了以下工具和语言支持：

适用于虚幻引擎开发者

使用虚幻引擎的轻量级插件。此插件包括具有必需的 Amazon GameLift 功能的 C++ 服务器 SDK 库。使用文档为插件配置虚幻游戏项目，并使用提供的代码块更新游戏代码，为服务器和客户端版本添加所需的功能。

- SDK 插件下载
- 指南：将你的虚幻项目与 Amazon 集成 GameLift
- 参考指南：适用于虚幻引擎的 C++ 服务器 SDK 5

注意：适用于虚幻引擎的Amazon GameLift 独立插件不支持使用容器舰队。

适用于 Unity 开发

使用适用于 Unity 的轻量级插件。此插件包括具有必需的 Amazon GameLift 功能的 C# 服务器 SDK 库。使用文档为插件配置虚幻游戏项目，并使用提供的代码块更新游戏代码，为服务器和客户端版本添加所需的功能。

- SDK 插件下载
- 指南：将您的 Unity 项目与亚马逊集成 GameLift
- 参考指南：适用于 Unity 的 C# 服务器 SDK 5

注意：适用于 Unity GameLift 的 Amazon 独立插件不支持使用集装箱舰队。

适用于使用其他游戏引擎的开发者

请遵循以下服务器和客户端集成通用指南：

- 集成游戏服务器
- 集成游戏客户端

Amazon GameLift 提供适用于以下语言的服务器 SDK 5 库：

- 适用于 C++ 的服务器 SDK 5 [SDK 下载] [参考指南]
- 适用于 C# 的服务器 SDK 5 [SDK 下载] [参考指南]
- 适用于 Go 的服务器 SDK 5 [SDK 下载] [参考指南]

为 Linux 构建你的游戏服务器

Amazon GameLift 容器舰队支持在 Linux 平台上运行的游戏服务器。以下是一些针对 Linux 目标构建游戏服务器的技巧：

- 如果您使用 Unity 游戏引擎开发游戏，则游戏编辑器提供内置支持，无需特殊要求即可针对 Linux 进行构建。
- 如果您使用 C++ 开发游戏，则在构建适用于 C++ 的亚马逊 GameLift 服务器 SDK 和构建游戏服务器时，必须包含适用于 Linux 的 OpenSSL 库。还要在游戏服务器容器镜像中包含相同的库。
- 如果你在Windows上使用虚幻引擎开发游戏，可以考虑以下选项：
  - 使用虚幻引擎建立交叉编译工具链。
  - 设置单独的 Linux 工作区，或者使用诸如适用于 Linux 的 Windows 子系统 (WSL) 之类的工具。你可以使用这个环境在 Linux 上运行虚幻编辑器来构建你的游戏服务器。

在本地测试您的集成

您可以使用 Amazon GameLift Anywhere 舰队在本地测试您的游戏集成。这种方法是帮助隔离与集成直接相关问题的最佳实践。Anywhere舰队是运行测试应用程序和游戏场景的有用工具，例如启动/停止游戏会话和跟踪玩家连接。使用Anywhere队列，您可以更快地进行迭代构建和测试，从而提高对托管活动的可见性。

使用 Amazon GameLift Anywhere 实例集测试您的集成有关使用 Amazon GameLift Anywhere 队列进行集成测试的帮助，请参阅。设置测试环境的工作流程如下所示：

1. 设置一台运行 Linux 的本地设备。

2. 组建Anywhere舰队。为本地设备创建自定义位置，创建Anywhere队列，然后将本地设备注册为队列中的计算设备。
3. 获取游戏服务器的身份验证令牌。您的集成服务器进程需要一个令牌才能通过 Amazon GameLift 服务进行身份验证。您可以对同时运行的多个服务器进程重复使用相同的令牌。只有在使用Anywhere队列进行集成测试时才需要执行此步骤。

   > **Note**
   >
   > 身份验证令牌是临时的，必须定期刷新。考虑在服务器构建包中添加脚本以请求新令牌。

4. 更新您的游戏服务器代码Anywhere。在 Anywhere 队列上运行时，游戏服务器需要使用以下服务器参数调用服务器 SDK 操作 `InitSdk()` (C ++) (C#) (Unreal)。只有在使用Anywhere队列进行集成测试时才需要执行此步骤。将 Amazon A GameLift gent 添加到容器映像后，它会自动处理这些参数。

   作为最佳实践，请将服务器代码设置为从环境变量或启动时指定的控制台参数中提取这些值。

   - `webSocketUrl`— 使用的值`GameLiftServiceSdkEndpoint`，该值在调用时返回`register-compute`。
   - `processId`— 为服务器进程分配唯一标识符。
   - `fleetId`— Anywhere 队列标识符，在调用时返回该标识符`create-fleet`。
   - `authToken`— 有效的身份验证令牌，在调用时返回`get-compute-auth-token`。

5. 在本地计算机上，设置游戏服务器编译软件并启动服务器进程。

   如果您的服务器集成成功，则服务器进程会调用服务器 SDK 操作`InitSDK()`来建立与 Amazon GameLift 服务的连接，然后调用`ProcessReady()`以通知该服务已准备好托管游戏会话。

6. 开始游戏会话。如果您已集成游戏客户端来请求游戏会话，则可以使用它来请求新的游戏会话。如果不是，请使用 C AWS LI 命令 create-game-session。Amazon GameLift 创建一个`GameSession`对象并启动启动新游戏会话的进程。

   如果您的集成正在运行，Amazon 会在您的本地工作站上 GameLift 调用服务器进程来启动新的游戏会话（使用`onStartGameSession()`回调）。当游戏会话准备好可供玩家使用时，服务器进程就会调用`ActivateGameSession()`。作为响应，Amazon 会 GameLift 更新`GameSession`状态和连接信息，以便游戏客户端可以连接到游戏会话并玩游戏。

使用游戏服务器软件准备容器镜像

本文档适用于公开预览版中的一项功能。本文档随时可能更改。

集装箱是 Amazon 集装 GameLift 箱船队中最基本的组成部分。您的容器包括您的游戏服务器及其依赖项，例如 SDK、软件、目录和文件。

要在容器队列中运行，您的游戏服务器必须在 Linux 上运行，并与服务器 SDK 5.x 集成。

主题

- 设置你的工作目录
- 构建您的容器镜像
- 将您的容器镜像推送到 Amazon ECR

设置你的工作目录

您的工作目录是您存放构建容器映像和定义 Amazon 如何 GameLift 运行容器映像所需的所有文件的地方。

设置您的容器工作目录

1. 创建您要在其中使用 Amazon GameLift 容器映像的目录。

   Example

   例如：

   ```
   [~/]$ mkdir -p work/glc/gamebuild && cd work && find .
   .
   ./glc
   ./glc/gamebuild
   ```

2. 克隆亚马逊 GameLift 代理。

   Example

   例如：

   ```
   [~/work]$ git clone https://github.com/aws/amazon-gamelift-agent.git
   ```

```
Cloning into 'amazon-gamelift-agent'...
```

3. GameLiftAgent 使用 Maven 进行构建。

   Example

   例如：

   ```
   [~/work]$ cd amazon-gamelift-agent
   ```

   Example

   ```
   [~/work/amazon-gamelift-agent]$ mvn clean compile assembly:single && \
   mv target ../glc && cd .. && find glc
   ```

4. 添加已与服务器 SDK 5.x 集成、构建并打包成.ZIP文件的游戏服务器。
5. 将您的.ZIP文件复制到~/work/glc/gamebuild/。

如果您没有 SDK 5.x 游戏服务器，则可以下载并使用我们的示例SimpleServer游戏来尝试使用容器舰队。

Example

```
[~/work]$ curl -o glc/gamebuild/SimpleServer.zip \
'https://ws-assets-prod-iad-r-iad-ed304a55c2ca1aee.s3.us-
east-1.amazonaws.com/086bb355-4fdc-4e63-8ca7-af7cfc45d4f2/
AmazonGameLiftSampleServerBinary.zip' &&
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
Dload  Upload   Total   Spent    Left  Speed
100 5140k  100 5140k    0     0  12.3M      0 --:--:-- --:--:-- --:--:-- 12.3M
glc
glc/target
glc/target/GameLiftAgent-1.0.jar
glc/gamebuild
glc/gamebuild/SimpleServer.zip
```

构建您的容器镜像

您的 Dockerfile 指定了构建容器的环境、软件和说明。

创建你的 Dockerfile

1. 移至glc子目录。

   Example

   ```
   [~/work]$ cd glc && find

   .
   ./target
   ./target/GameLiftAgent-1.0.jar
   ./gamebuild
   ```

2. 创建并打开一个新的 Dockerfile。

   Example

   例如：

   ```
   [~/work/glc]$ nano Dockerfile
   ```

3. 从以下模板之一复制，然后将内容粘贴到您的 Dockerfile 中。

适用于你的游戏服务器的 Dockerfile 模板

此模板包含容器在 Amazon GameLift 车队中使用所需的最低限度说明。根据需要修改游戏服务器的内容。

```
# Base image
# ----------
  # Add the base image that you want to use over here,
  # Make sure to use an image with the same architecture as the
  # Instance type you are planning to use on your fleets.
  # We require JDK to be installed in the base image, so that
  # it can be used to run the &AGS; Agent
FROM public.ecr.aws/amazoncorretto/amazoncorretto:17-amd64
  #
# Game build directory
# --------------------
  # Add your game build to gamebuild directory and add the zip file name in the
 'GAME_BUILD_ZIP' env variable below.
  # The game build provided over here needs to be integrated with gamelift server sdk.
```

```
  # This template assumes that the game build is in a zip format.
ENV GAME_BUILD_ZIP="<ADD_GAME_BUILD_ZIP_FILE_NAME>" \
  #
# Default directory
# -----------------
  # Default directory, the value provided here should be where the game executable
 exists.
  # Provide this same value as your launch path in RuntimeConfiguration when creating a
 fleet.
  # Ref: https://docs.aws.amazon.com/gamelift/latest/apireference/
API_ServerProcess.html
GAME_EXECUTABLE="<ADD NAME OF EXECUTABLE WITHIN THE GAME BUILD>" \
HOME_DIR="/local/game" \
  #
  # Registered compute in anywhere fleet (not used in container fleets)
  # ---------------------------------------------------------------------
  # Add the name for the registered compute in an anywhere fleet.
  # This environment variable is required only for anywhere fleets, but not for
 container fleets.
  # If it is set for container fleets, it will be overridden by Gamelift.
  GAMELIFT_COMPUTE_NAME="<ADD_COMPUTE_NAME>" \
  #
# Default Gamelift Agent jar
# --------------------------
GAMELIFT_AGENT_EXEC="GameLiftAgent-1.0.jar" \
  #
  # This env variable defines the name of the S3 bucket that stores the GameLift Agent
 logs.
  # This S3 bucket should exist in the customer AWS account.
  # In order to allow GameLift agent to upload logs to this s3 bucket, customers would
 need to
  # include s3:PutObject permission in the IAM role provided as instanceRoleArn during
 CreateFleet operation.
GAMELIFT_AGENT_LOGS_BUCKET_NAME="<ADD NAME OF GAMELIFT AGENT LOGS S3 BUCKET>" \
  #
# -----------
  # This env variable defines the name of the S3 bucket that stores the game session
 logs.
  # This S3 bucket should exist in the customer AWS account.
  # In order to allow GameLift agent to upload logs to this s3 bucket, customers would
 need to
  # include s3:PutObject permission in the IAM role provided as instanceRoleArn during
 CreateFleet operation.
# -----------
```

```
GAME_SESSION_LOGS_BUCKET_NAME="<ADD NAME OF GAME SESSION LOGS S3 BUCKET>" \
#
# -----------
GAMELIFT_AGENT_LOGS_PATH="/local/game/agentlogs/" \
  #
# NOT USED in container fleets - USED in Anywhere fleets
# ------------------------------------------------------------
  # Specifiy the type of compute resource used to host the game servers.
  # This env variable is required only for anywhere fleets, but not for container
 fleets.
  # If it is set for container fleets, it will be overridden by Gamelift.
#
# -----------
COMPUTE_TYPE="ANYWHERE" \
  #
  # Specify the credential to be used for creating the client.
  # This env variable is required only for anywhere fleets, but not for container
 fleets.
  # If it is set for container fleets, it will be overridden by Gamelift.
#
# -----------
CREDENTIAL_PROVIDER="environment-variable"

USER root

    # intall dependencies as necessary
    RUN yum install -y sudo \
                        unzip \
                        git \
                        shadow-utils \
                        iputils \
                        tar \
                        gcc \
                        make \
                        openssl-devel \
                        zlib-devel \
                        vim \
                        net-tools \
                        nc \
                        procps

    # Set up the ground for 'gamescale' user
    RUN groupadd -r gamescale -g 500 && \
```

```
      useradd -u 500 -r -g gamescale -m -s /sbin/nologin -c "Gamescale user" gamescale
&& \
      echo "gamescale ALL=(ALL) NOPASSWD: ALL" | (EDITOR="tee -a" visudo) && \
      mkdir -p $HOME_DIR && \
      mkdir $HOME_DIR/mono && \
      chown -R gamescale:gamescale $HOME_DIR

   WORKDIR $HOME_DIR

   # extract game build as necessary
   COPY ./gamebuild/$GAME_BUILD_ZIP .
   RUN unzip ./$GAME_BUILD_ZIP -d ./

   # copy Gamelift Agent jar
   COPY ./gameliftAgent/$GAMELIFT_AGENT_EXEC ./

   # Add permissions to game build and gamelift agent jar
   RUN chmod +x ./$GAME_EXECUTABLE
   RUN chmod +x ./$GAMELIFT_AGENT_EXEC

   # Check if java is installed on the image, if not then the Agent will not be able
to run
   RUN java --version

   USER gamescale

   ENV PATH="$PATH:$HOME_DIR/bin:$JAVA_HOME"

   # Change directory to bin
   WORKDIR $HOME_DIR

   # check path before starting the container
   RUN echo $PATH

   # Create logs directory for GameLift Agent & server processes
   RUN mkdir logs
   RUN mkdir agentlogs

   # Start the GameLift Agent
   ENTRYPOINT sleep 90 && java -jar $GAMELIFT_AGENT_EXEC -ip "192.168.1.1" -gslb
"$GAME_SESSION_LOGS_BUCKET_NAME" -galb "$GAMELIFT_AGENT_LOGS_BUCKET_NAME" -galp
"$GAMELIFT_AGENT_LOGS_PATH" -glc environment-variable
```

示例的 Dockerfile **SimpleServer**

```
# Base image
# ----------
  # Add the base image that you want to use over here,
  # Make sure to use an image with the same architecture as the
  # Instance type you are planning to use on your fleets.
  # We require JDK to be installed in the base image, so that
  # it can be used to run the &AGS; Agent
FROM public.ecr.aws/amazoncorretto/amazoncorretto:17-amd64
  #
# Game build directory
# --------------------
  # Add your game build to gamebuild directory and add the zip file name in the
 'GAME_BUILD_ZIP' env variable below.
  # The game build provided over here needs to be integrated with gamelift server sdk.
  # This template assumes that the game build is in a zip format.
ENV GAME_BUILD_ZIP="SimpleServer.zip" \
  #
# Default directory
# -----------------
  # Default directory, the value provided here should be where the game executable
 exists.
  # Provide this same value as your launch path in RuntimeConfiguration when creating a
 fleet.
  # Ref: https://docs.aws.amazon.com/gamelift/latest/apireference/
API_ServerProcess.html
GAME_EXECUTABLE="GameLiftSampleServer" \
HOME_DIR="/local/game" \
  #
  # Registered compute in anywhere fleet (not used in container fleets)
  # ------------------------------------------------------------------
  # Add the name for the registered compute in an anywhere fleet.
  # This environment variable is required only for anywhere fleets, but not for
 container fleets.
  # If it is set for container fleets, it will be overridden by Gamelift.
  GAMELIFT_COMPUTE_NAME="<ADD_COMPUTE_NAME>" \
  #
# Default Gamelift Agent jar
# --------------------------
GAMELIFT_AGENT_EXEC="GameLiftAgent-1.0.jar" \
  #
  # This env variable defines the name of the S3 bucket that stores the GameLift Agent
 logs.
```

```
  # This S3 bucket should exist in the customer AWS account.
  # In order to allow GameLift agent to upload logs to this s3 bucket, customers would
 need to
  # include s3:PutObject permission in the IAM role provided as instanceRoleArn during
 CreateFleet operation.
GAMELIFT_AGENT_LOGS_BUCKET_NAME="<ADD NAME OF GAMELIFT AGENT LOGS S3 BUCKET>" \
  #
# -----------
  # This env variable defines the name of the S3 bucket that stores the game session
 logs.
  # This S3 bucket should exist in the customer AWS account.
  # In order to allow GameLift agent to upload logs to this s3 bucket, customers would
 need to
  # include s3:PutObject permission in the IAM role provided as instanceRoleArn during
 CreateFleet operation.
# -----------
GAME_SESSION_LOGS_BUCKET_NAME="<ADD NAME OF GAME SESSION LOGS S3 BUCKET>" \
#
# -----------
GAMELIFT_AGENT_LOGS_PATH="/local/game/agentlogs/" \
  #
# NOT USED in container fleets - USED in Anywhere fleets
# ------------------------------------------------------------
  # Specifiy the type of compute resource used to host the game servers.
  # This env variable is required only for anywhere fleets, but not for container
 fleets.
  # If it is set for container fleets, it will be overridden by Gamelift.
#
# -----------
COMPUTE_TYPE="ANYWHERE" \
  #
  # Specify the credential to be used for creating the client.
  # This env variable is required only for anywhere fleets, but not for container
 fleets.
  # If it is set for container fleets, it will be overridden by Gamelift.
#
# -----------
CREDENTIAL_PROVIDER="environment-variable"

USER root

    # intall dependencies as necessary
    RUN yum install -y sudo \
                        unzip \
```

```
                        git \
                        shadow-utils \
                        iputils \
                        tar \
                        gcc \
                        make \
                        openssl-devel \
                        zlib-devel \
                        vim \
                        net-tools \
                        nc \
                        procps

   # Set up the ground for 'gamescale' user
   RUN groupadd -r gamescale -g 500 && \
     useradd -u 500 -r -g gamescale -m -s /sbin/nologin -c "Gamescale user" gamescale
&& \
     echo "gamescale ALL=(ALL) NOPASSWD: ALL" | (EDITOR="tee -a" visudo) && \
     mkdir -p $HOME_DIR && \
     mkdir $HOME_DIR/mono && \
     chown -R gamescale:gamescale $HOME_DIR

   WORKDIR $HOME_DIR

   # extract game build as necessary
   COPY ./gamebuild/$GAME_BUILD_ZIP .
   RUN unzip ./$GAME_BUILD_ZIP -d ./

   # copy Gamelift Agent jar
   COPY ./target/$GAMELIFT_AGENT_EXEC ./

   # Add permissions to game build and gamelift agent jar
   RUN chmod +x ./$GAME_EXECUTABLE
   RUN chmod +x ./$GAMELIFT_AGENT_EXEC

   # Check if java is installed on the image, if not then the Agent will not be able
to run
   RUN java --version

   USER gamescale

   ENV PATH "$PATH:$HOME_DIR/bin:$JAVA_HOME"

   # Change directory to bin
```

```
    WORKDIR $HOME_DIR

    # check path before starting the container
    RUN echo $PATH

    # Create logs directory for GameLift Agent & server processes
    RUN mkdir logs
    RUN mkdir agentlogs

    # Start the GameLift Agent
    ENTRYPOINT sleep 90 && java -jar $GAMELIFT_AGENT_EXEC -ip "192.168.1.1" -gslb
"$GAME_SESSION_LOGS_BUCKET_NAME" -galb "$GAMELIFT_AGENT_LOGS_BUCKET_NAME" -galp
"$GAMELIFT_AGENT_LOGS_PATH" -glc environment-variable
```

> **Note**
>
> 注意：Dockerfile 中的某些环境变量可以被覆盖。ContainerDefinition

构建您的容器镜像

1. 构建您的容器镜像。

   如果你使用的是自己的 SDK 5.x 服务器

   你可以指定任何你想要的本地存储库名称。

   Example

   ```
   [~/work/glc]$ docker build -t <local repository name>:<optional tag> .
   ```

   如果你正在使用我们的**SimpleServer**样本

   Example

   ```
   [~/work/glc]$ docker build -t simple-server:version-1 .
   Successfully built 0123456789012
   Successfully tagged simple-server:version-1
   ```

> **Note**
>
> 在以下示例中，我们使用 s *impler-server* 作为初始REPOSITORY值和version-1值。TAG

2. 查看图像列表并记下REPOSITORY和的IMAGE ID值。你需要在下面的步骤中使用它们。

   Example

```
[~/work/glc]$ docker images
REPOSITORY       TAG          IMAGE ID        CREATED           SIZE
simple-server    version-1    0123456789012    14 minutes ago    1.24GB
```

将您的容器镜像推送到 Amazon ECR

将您的容器映像上传到 Amazon ECR 中的私有存储库。创建容器组定义时，您需要引用此存储库位置，以便 Amazon GameLift 可以拍摄您的容器映像的快照并在部署容器队列时使用它。

> **Note**
>
> 如果您还没有 Amazon ECR 私有存储库，请创建一个。

获取您的 Amazon ECR 凭证

- 在将容器映像推送到 Amazon ECR 之前，您必须以临时形式获取 AWS 凭证并将其提供给 Docker。获取您的亚马逊 ECR 凭证，这样 Docker 就可以登录了。

  Example

```
[~/work/glc]$ aws ecr get-login-password --region us-west-2 | docker login --
username AWS --password-stdin aws_account_id.dkr.ecr.us-west-2.amazonaws.com
WARNING! Your password will be stored unencrypted in
/home/user-name/.docker/config.json.
Configure a credential helper to remove this warning.
See https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

将您的容器镜像推送到 Amazon ECR

1. 复制您要使用的 Amazon ECR 私有存储库的 URI。
2. 将 Amazon ECR 标签应用于您的容器映像。

   Example

   ```
   [~/work/glc]$ docker tag <IMAGE ID from above> <Amazon ECR private repository
    URI>:<optional tag>
   ```

3. 将您的容器镜像推送到 Amazon ECR

   Example

   ```
   [~/work/glc]$ docker image push  <Amazon ECR private repository URI>
   ```

设计一支亚马逊 GameLift 集装箱船队

本文档适用于公开预览版中的一项功能。本文档随时可能更改。

这些主题介绍了您在设置 Amazon GameLift 集装箱船队时要做出的关键决策。您的决策会影响您为容器、容器组和舰队配置设置的方式。

主题

- 设计您的舰队集装箱结构
- 设置资源限制
- 指定基本容器
- 配置网络连接
- 为容器设置运行状况检查
- 设置容器依赖关系
- 配置集装箱舰队

设计您的舰队集装箱结构

首先，确定托管游戏服务器所需的软件和资源，包括以下内容：

- 您的游戏服务器应用程序。该应用程序必须与 Amazon 托管 GameLift功能集成，包括服务器 SDK 版本 5+。请参阅 将您的游戏与 Amazon 集成 GameLift。
- 亚马逊 GameLift 代理。此计算机代理与 Amazon GameLift 服务保持通信，并管理所有游戏服务器进程的生命周期。有关更多详细信息，请参阅游戏服务器和 Amazon GameLift 代理。
- 根据需要提供其他软件和资源。这可能包括运行游戏服务器应用程序所需的软件。常用的支持软件用于记录和监控、安全、内容交付和数据同步。

接下来，决定如何为 Amazon GameLift 集装箱船队构建软件和资源。Amazon GameLift 使用容器组来组织容器。一个队列始终有一个副本容器组，并且可以选择拥有一个守护程序容器队列。有关更多详细信息，请参阅集装箱船队组件。

- 首先设计您的副本容器组。请考虑以下准则：
  - 将您的游戏服务器应用程序和 Amazon A GameLift gent 捆绑到同一个容器中。将此容器设置为副本组的唯一必需容器。
  - 将游戏服务器的所有其他软件整理到容器中。您可以选择将所有内容放入副本组中的单个容器中。或者你可以选择创建一个或多个 sidecar 容器。使用边车的一些原因包括：
    - 为单个软件设置启动/关闭顺序。您可以通过将软件放在单独的容器中并在它们之间设置依赖关系来实现这一点。
    - 设置容器特定的内存和 CPU 使用限制。
    - 为每个容器指定不同的容器配置设置，例如启动命令、入口点、工作目录、环境变量或运行状况检查。
- 决定是否需要为队列设置守护程序容器组。请考虑以下事项：
  - 守护程序容器通常用于运行后台或监视进程。
  - 守护程序组中的容器不会在队列实例上复制。这意味着守护程序组中的容器无法与副本容器组一起扩展。
  - 一个守护程序组可以有多个容器。您可以将守护程序组中的任何容器指定为必需容器。

设置资源限制

对于每个容器组，确定该组运行其软件需要多少内存和 CPU。Amazon GameLift 依靠这些信息来管理容器组的资源。它还使用这些信息来计算舰队映像可以容纳多少个副本容器组。您也可以为单个容器设置限制。

为容器设置可选限制

通过设置容器特定的资源限制，您可以更好地控制各个容器如何使用群组的资源。如果您未设置特定于容器的限制，则群组中的所有容器将共享群组资源。共享为在需要的地方使用资源提供了更大的灵活性。它还增加了进程相互竞争并导致容器故障的可能性。

为任何容器设置以下任意ContainerDefinition属性。

- SoftLimit（内存）— 保留最少的内存量供容器专用。集装箱始终有可用的预留金额。如果有额外资源可用，则随时可能超过此最低限额。
- HardLimit（内存）-为容器设置最大内存限制。如果容器超过此限制，则会导致重启。
- Cpulimit — 保留最低数量的 CPU 资源供容器专用。集装箱始终有可用的预留金额。如果有额外资源可用，则随时可能超过此最低限额。（1024 个 CPU 单位相当于 1 个 vCPU。）

为容器组设置总资源限制

告诉 Amazon 每个容器组需要 GameLift 多少内存和 CPU 资源。目标是分配足够的资源来优化游戏服务器的性能。Amazon GameLift 使用这些限制来计算如何在队列实例上打包副本容器组。您还将在为集装箱队列选择实例类型时使用它们。

计算组中每个容器中所有进程所需的总内存和 CPU。请考虑以下事项：

- 哪些进程在容器组中的所有容器中运行？ 将这些过程所需的资源相加。
- 您计划在每个容器组中运行多少个并发游戏服务器进程？ 您可以将此值设置为队列运行时配置的一部分，但需要在此处为它们计划足够的内存（请参阅优化运行时配置）。

根据您对容器组需求的估计，设置以下ContainerGroupDefinition属性：

- TotalMemoryLimit— 为容器组设置最大内存限制。组中的所有容器共享分配的内存。如果您设置单个容器限制，则总内存限制必须为：
  - 等于或大于所有容器软内存限制的总和
  - 等于或大于组中容器的最大硬内存限制
- TotalCpuLimit — 为容器组设置最大 CPU 限制。组中的所有容器共享分配的 CPU 资源。如果您设置单个容器限制，则总 CPU 限制必须为：
  - 等于或大于所有容器 CPU 限制的总和。作为最佳实践，可以考虑将此值设置为容器 CPU 限制之和的两倍。

示例方案

假设我们正在定义一个包含以下三个容器的副本容器组：

- 容器 A 是我们必不可少的复制容器。它运行游戏服务器进程和 Amazon GameLift 代理。我们估计一台游戏服务器的资源需求为 512 MiB 和 1024 CPU。我们计划让容器运行 10 个服务器进

程。由于此容器运行我们最关键的软件，因此我们将软内存储备设置为 6144 MiB，并且没有硬盘内存限制或 CPU 预留限制。

- 容器 B 运行支持软件，资源需求估计为 1024 MiB 和 1536 CPU。我们将软内存储备限制设置为 1024 MiB，硬内存限制为 2048 MiB，CPU 预留限制为 1024 CPU。
- 容器 C 运行非关键日志记录和其他监控工具。我们将硬盘内存限制设置为 512 MiB，将 CPU 预留限制设置为 512 CPU。

使用这些信息，我们为容器组设置了以下总限制：

- 总内存限制：7680 MiB。该值超过 (1) 软内存限制 (6144+1024 MiB) 和 (2) 最高硬内存限制 (1024 MiB) 的总和。
- CPU 总限制：13312 CPU。此值超过了 CPU 限制的总和（1024+512 CPU）。

指定基本容器

对于每个容器，将容器指定为必需或非必需容器。所有容器组必须至少有一个基本容器。基本容器负责容器组的关键工作，例如托管游戏服务器。必需容器始终处于运行状态。如果失败，则整个容器组将重新启动。

- 您的舰队的复制集装箱组只能有一个基本集装箱。此容器运行 Amazon A GameLift gent，并由其管理的游戏服务器进程。
- 如果您的队列具有守护程序容器组，则可以指定多个基本容器。如果您希望容器故障提示容器组重启，请将守护程序容器设置为必备项。

将每个容器的`ContainerDefinition`属性`Essential`设置为 true 或 false。

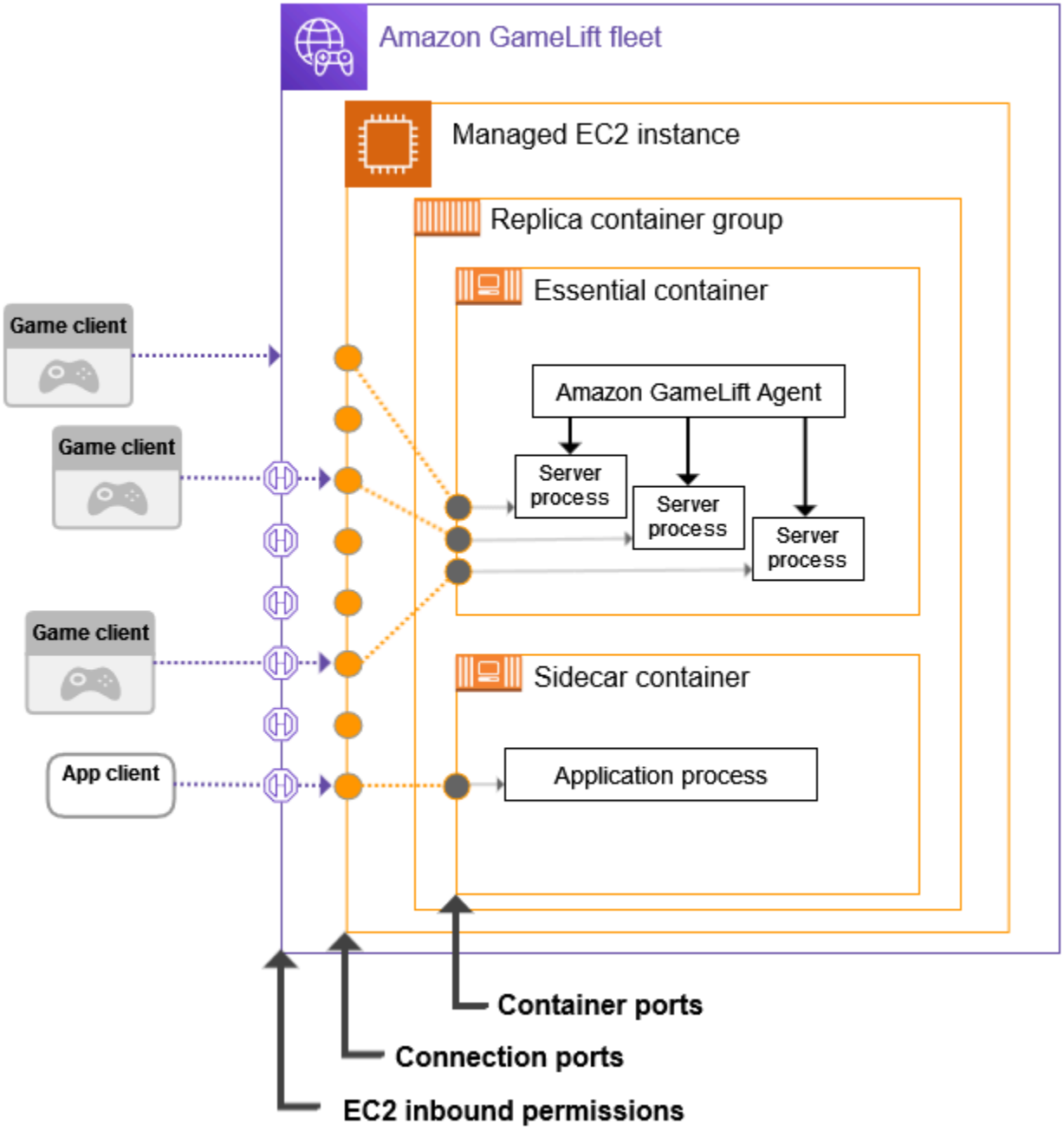
配置网络连接

您可以建立网络访问权限，让外部流量连接到集装箱队列中的任何容器。例如，您必须与运行游戏服务器进程的容器建立网络连接，以便游戏客户端可以加入并玩您的游戏。游戏客户端使用端口和 IP 地址连接到游戏服务器。

在容器舰队中，客户端和服务器之间的连接不是直接的。在内部，容器中的进程监听容器端口。在外部，传入流量使用连接端口连接到队列实例。Amazon GameLift 维护内部容器端口和面向外部的连接端口之间的映射，以便将传入流量路由到实例上的正确进程。

Amazon 为您的网络连接 GameLift 提供了额外的控制层。每个容器舰队都有入站权限设置，允许您控制对每个面向外部的连接端口的访问权限。您无法更改现有舰队的端口配置，但您可以根据需要通过调

整入站权限来允许或限制访问。例如，您可以移除所有连接端口的权限，以关闭对舰队容器的所有访问权限。



设置容器端口范围

为容器定义配置足够的容器端口，以供任何需要外部访问的进程使用。有些集装箱不需要任何端口。其他端口必须有足够的端口才能为每个需要端口的进程分配一个端口。

运行游戏服务器的基本副本容器组需要一个端口，用于每个并发运行的游戏服务器进程（如队列中的配置RuntimeConfiguration）。游戏服务器进程监听分配的端口，并将其报告给 Amazon GameLift。

创建容器组定义时，请为每个需要网络访问的容器定义一个容器端口范围（请参阅 ContainerDefinitionInput:PortConfiguration）。确保范围足够大，可以为每个需要端口的进程分配一个端口。必须在容器的端口配置中为进程分配端口号。

设置连接端口范围

使用一组连接端口配置您的集装箱舰队。连接端口提供对运行您的容器的队列实例的外部访问权限。 GameLift Amazon 会根据需要分配连接端口并将其映射到容器端口。

创建容器队列时，请定义连接端口范围（请参阅 ContainerGroupsConfiguration:ConnectionPortRange）。确保该范围有足够的端口来映射到舰队实例中的每个集装箱端口。要计算所需的最小连接端口，请使用以下公式：

```
[Total number of container ports defined for containers in the replica container group] * [Number of replica container groups per instance] + [Total number of container ports defined for containers in the daemon container group]
```

最佳做法是将连接端口的最小数量增加一倍。

> **Note**
>
> 连接端口的数量可能会限制每个实例的副本容器组的数量。如果队列的连接端口仅足以容纳每个实例的一个副本容器组，则 Amazon GameLift 将仅部署一个副本容器组，即使这些实例具有足够的计算能力来容纳多个副本容器组。

设置入站权限

入站权限通过指定要为传入流量打开哪些连接端口来控制对集装箱队列的外部访问。您可以使用此设置根据需要开启和关闭舰队的网络访问权限。

创建容器队列时，请定义一组入站权限（请参阅:EC2 CreateFleet）。InboundPermissions将入站权限端口属性设置为包含队列连接端口设置中的部分或全部值。要更改现有集装箱舰队的入站权限，请致电 UpdateFleetPortSettings。

示例方案

此示例说明如何设置所有三个网络连接属性。

- 我们舰队的复制容器组有 1 个容器，用于运行游戏服务器进程。运行时确认会告诉容器运行 10 个并发的游戏服务器进程。

  在副本容器组定义中，我们按如下方式设置此容器的PortConfiguration参数：

```
"PortConfiguration": {
  "ContainerPortRanges": [ { "FromPort": 10, "ToPort": 20, "Protocol": "TCP"} ]
  }
```

- 我们的舰队还有一个包含 1 个容器的守护程序容器组。它有 1 个需要网络访问的进程。在守护程序容器组定义中，我们按如下方式为该容器设置PortConfiguration参数：

```
"PortConfiguration": {
  "ContainerPortRanges": [ { "FromPort": 25, "ToPort": 25, "Protocol": "TCP"} ]  }
```

- 我们的舰队为每个舰队实例配置了 3 个副本容器组。有了这些信息，我们可以使用以下公式来计算我们需要的连接端口数：
  - 最少：31 个端口 [10 个副本容器端口 * 每个实例 3 个副本容器组 + 1 个守护程序容器端口]
  - 最佳实践：62 个端口 [最少端口 * 2]

  在创建集装箱舰队时，我们按ContainerGroupsConfiguration如下方式设置ConnectionPortRange参数：

```
"ConnectionPortRange": { "FromPort": 1010, "ToPort": 1071 }
```

- 我们希望允许访问所有可用的连接端口。在创建集装箱舰队时，我们按如下方式设置EC2InboundPermissions参数：

```
"EC2InboundPermissions": [
  {"FromPort": 1010, "ToPort": 1071, "IpRange": "10.24.34.0/23", "Protocol":
 "TCP"} ]
```

为容器设置运行状况检查

如果容器遇到终端故障并停止运行，它会自动重启。如果容器必不可少，则整个容器组将重新启动。

您可以定义其他自定义标准来衡量容器运行状况，并使用运行状况检查来测试该标准。要设置容器运行状况检查，您可以在 docker 容器镜像或容器定义中对其进行定义。如果您在容器定义中设置了运行状况检查，它将覆盖容器镜像中的所有设置。

根据容器类型设置可选的运行状况检查，如下所示：

- 对于基本副本容器，请不要配置运行状况检查。Amazon A GameLift gent 会自动处理此集装箱的运行状况报告。
- 对于非必要的副本容器和任何守护程序容器，您可以选择设置运行状况检查参数。

为容器运行状况检查设置以下ContainerDefinition属性：

- Command— 提供一个命令来检查容器运行状况的某些方面。您可以决定使用什么标准来衡量健康状况。该命令的退出值必须为 1（不正常）或 0（正常）。
- StartPeriod— 指定运行状况检查失败开始计数之前的初始延迟。这种延迟使容器有时间引导其进程。
- Interval— 决定运行运行状况检查命令的频率。您希望以多快的速度检测和解决容器故障？
- Timeout— 决定在重试运行状况检查命令之前等待成功或失败的时间。运行状况检查命令需要多长时间才能完成？
- Retries— 在记录失败之前，应重试运行状况检查命令多少次？

设置容器依赖关系

在每个容器组中，您可以根据容器状态设置容器之间的依赖关系。依赖关系会根据另一个容器的状态影响依赖容器的启动或关闭时间。

依赖关系的一个关键用例是为容器组创建启动和关闭序列。

例如，您可能希望容器 A 先启动并成功完成，然后再启动容器 B 和 C。要实现这一点，首先要在容器 A 上为容器 B 创建一个依赖关系，条件是容器 A 必须成功完成。然后在容器 A 上为容器 C 创建具有相同条件的依赖关系。启动顺序与关机顺序相反。

配置集装箱舰队

创建集装箱船队时，请考虑以下决策点。这些要点大多取决于您的容器架构和配置。

决定要将舰队部署到哪里

通常，您希望将舰队部署在靠近玩家的地理位置，以最大限度地减少延迟。您可以将您的集装箱船队部署到 Amazon GameLift 支持 AWS 区域 的任何容器舰队。如果您想将同一台游戏服务器部署到其他地理位置，则可以将远程位置添加到队列中，包括 AWS 区域 和 Local Zones。对于多地点车队，您可以独立调整每个车队位置的容量。有关支持的舰队位置的更多信息，请参阅亚马逊 GameLift 托管地点。

为您的队列选择实例类型和大小

Amazon GameLift 支持各种 Amazon EC2 实例类型，所有这些类型都可用于集装箱队列。实例类型、可用性和价格因地点而异。您可以在 Amazon GameLift 控制台中查看按位置筛选的支持实例类型列表（在 “资源”、“实例” 和 “服务配额” 下）。

选择实例类型时，首先要考虑实例系列。实例系列提供 CPU、内存、存储和网络功能的各种组合。获取有关 EC2 实例系列的更多信息。在每个系列中，您都有一系列实例大小可供选择。选择实例大小时，请考虑以下问题：

- 可以支持您的工作负载的最小实例大小是多少？ 使用此信息可以消除任何过小的实例类型。
- 哪种实例类型大小最适合您的容器架构？ 理想情况下，您希望选择一种能够容纳副本容器组多个副本的大小，同时最大限度地减少浪费的空间。
- 哪种缩放粒度对你的游戏有意义？ 扩展队列容量涉及添加或删除实例，每个实例都代表托管特定数量的游戏会话的能力。考虑您要为每个实例添加或移除多少容量。如果玩家的需求每分钟相差数千个，那么使用可以托管成百上千个游戏会话的超大型实例可能是有意义的。相比之下，您可能更喜欢使用较小的实例类型进行更精细的扩展控制。
- 是否可以根据尺寸节省成本？ 您可能会发现，某些实例类型的成本因可用性而因地点而异。

优化您的运行时配置

舰队的运行时配置是一组关于如何运行服务器进程以托管游戏会话的指令。这些指令由 Amazon A GameLift gent 在队列中的每个副本容器组中实施。

队列的运行时配置决定了每个副本容器组中有多少服务器进程同时运行。此设置会影响您计算容器组资源限制的方式以及为队列选择实例类型的方式。在设计车队时，您需要平衡这三个要素。

有关如何使用运行时配置的更多信息，请参阅管理 Amazon GameLift 如何启动游戏服务器。

设置其他可选舰队设置

在配置容器队列时，您可以使用以下可选功能：

- 设置游戏服务器以访问其他 AWS 资源。请参阅 与您的实例集中的其他 AWS 资源进行通信。
- 保护活跃玩家的游戏会话在缩小规模活动期间过早终止。

- 限制一个人可以在有限的时间段内在舰队上创建的游戏会话数量。

为 Amazon 集装箱队列创建 GameLift 容器组定义

本文档适用于公开预览版中的一项功能。本文档随时可能更改。

容器组定义描述了如何将容器化游戏服务器应用程序部署到容器队列。它是一个蓝图，用于确定要在舰队上运行的容器集以及如何运行它们。创建容器队列时，您需要指定要部署到队列的容器组定义。有关容器组的更多信息，请参阅集装箱船队组件。

开始之前

完成以下任务：

- 设计用于托管游戏服务器的容器架构。请参阅 设计一支亚马逊 GameLift 集装箱船队。
- 规划要包含在容器组中的容器定义。如果您使用的是 AWS CLI，请在 JSON 文件中创建容器定义。
- 将最终的容器镜像推送到亚马逊弹性容器注册表 (Amazon ECR) Container Registry，该注册表与您计划创建容器组的位置 AWS 区域 相同。Amazon 会在您创建容器组定义时 GameLift 存储每个图像的快照，并在部署到容器队列时使用该副本。请参阅 使用游戏服务器软件准备容器镜像。
- 验证您的 AWS 用户是否具有 IAM 权限来访问 Amazon ECR 存储库。请参阅 管理 Amazon 的用户权限 GameLift。您至少需要权限才能执行以下操作：
  - `ecr:DescribeImages`
  - `ecr:BatchGetImage`
  - `ecr:GetDownloadUrlForLayer`

克隆容器组定义

您可以使用 Amazon GameLift 控制台克隆现有的容器组定义。

克隆容器组

1. 在 Amazon GameLift 控制台中，前往左侧导航窗格并选择容器组。
2. 在容器组列表页面上，选择要克隆的现有容器组。选择容器组后，“克隆” 按钮处于活动状态。
3. 选择克隆。此操作将打开带有预填设置的容器组创建向导。
4. 为克隆的容器组输入新名称。同一区域中的容器组必须具有唯一的名称。
5. 浏览容器组和容器定义页面，查看并创建新的容器组。

创建副本容器组定义

副本容器组管理您的游戏服务器软件。副本容器组至少有一个运行 Amazon A GameLift gent 和您的游戏服务器进程的容器。该小组可能还有其他 “sidecar” 容器来运行支持软件。

本主题介绍如何使用 Amazon GameLift 控制台或 AWS CLI 工具创建容器组定义。有关设置容器组配置的更多详细信息，请参阅设计一支亚马逊 GameLift 集装箱船队。

Console

在 Amazon GameLift 控制台中，选择要创建容器组 AWS 区域 的位置。

打开控制台的左侧导航栏并选择容器组。在容器组页面上，选择创建容器组。

步骤 1：定义群组详细信息。

1. 输入容器组定义名称。此名称必须是 AWS 账户 和区域所独有的。在控制台中，群组定义按名称列出，因此分配有意义的标签会很有帮助。
2. 选择副本调度策略。
3. 在总内存限制中，输入容器组可用的最大内存。有关计算此值的帮助，请参阅设置资源限制。
4. 在 CPU 总限制中，输入容器组可用的最大计算能力。有关计算此值的帮助，请参阅设置资源限制。

步骤 2：添加容器定义。

使用您的游戏服务器应用程序和 Amazon A GameLift gent 定义容器。这是您的必备复制容器。

1. 提供容器定义名称。为该组定义的每个容器都必须具有唯一的名称值。
2. 识别容器镜像的 Amazon ECR 镜像 URI。输入以下任意格式：
   - 仅限图片 URI：`[AWS ##].dkr.ecr.[AWS ##].amazonaws.com/[repository ID]`
   - 图片 URI + 摘要：`[AWS ##].dkr.ecr.[AWS ##].amazonaws.com/[repository ID]@[digest]`
   - 图片 URI + 标签：`[AWS ##].dkr.ecr.[AWS ##].amazonaws.com/[repository ID]:[tag]`
3. 对于基本容器，第一个容器定义自动选择 “是”。如果您添加其他容器定义，则可以为每个定义开启或关闭此设置。有关更多详细信息，请参阅指定基本容器。

4. 设置一个或多个内部容器端口范围。此容器托管您的游戏服务器，因此请定义一个具有足够端口的范围，以便每个服务器进程在容器组中运行。有关更多详细信息，请参阅配置网络连接。
5. 可选设置 Overrides 和 E nvironment 变量允许您指定在启动时传递给容器的值。您在此处设置的值会覆盖容器镜像中已有的所有设置。
6. 设置可选的容器限制以管理此容器的资源分配。有关更多详细信息，请参阅设置资源限制。
7. 根据需要定义其他非必要容器：
   - 提供容器定义名称和 ECR 镜像 URI。非必需容器不得运行 Amazon GameLift 代理。
   - 仅当容器具有需要网络访问的进程时，才设置内部容器端口范围。
   - （可选）为容器设置 He alth 检查。当非必要容器未通过运行状况检查时，它只会提示重启失败的容器。
   - （可选）根据需要设置覆盖、环境变量和资源分配限制。

步骤 3：配置依赖关系。

如果您的容器组定义中有多个容器，则可以定义它们之间的依赖关系。使用依赖关系根据容器条件设置启动和关闭顺序。有关更多详细信息，请参阅设置容器依赖关系。

1. 确定要为其添加依赖项的容器名称。在满足依赖条件之前，此容器不会启动。
2. 确定依赖项容器名称和条件。在依赖容器启动之前，此容器必须满足条件。
3. 根据需要设置其他依赖关系。您可以为任何容器创建多个依赖关系。避免创建循环依赖关系。

步骤 4：查看并创建。

1. 查看所有容器组定义设置。创建容器组定义后，您无法更改其配置。使用 “编辑” 对任何部分进行更改，包括群组的每个容器定义。
2. 审阅完毕后，选择 “创建”。

   如果您的请求成功，控制台将显示新容器组定义资源的详细信息页面。最初的状态是COPYING，当 Amazon GameLift 开始为该群组拍摄所有容器映像的快照时。此阶段完成后，容器组定义状态将更改为READY。容器组定义必须处于READY状态，然后才能使用它创建容器队列。

AWS CLI

使用 AWS CLI 创建容器组定义时，请将容器定义配置保存在单独JSON的文件中。您可以在 CLI 命令中引用该文件。创建容器定义JSON文件有关架构示例，请参阅。

创建容器组定义

要创建新的容器组定义，请使用 `create-container-group-definition` CLI 命令。有关此命令的更多信息，请参见《AWS CLI 命令参考》create-container-group-definition中的。

Example：副本容器组

此示例说明了对副本容器组定义的请求。用于创建副本和守护程序组定义的命令结构基本相同。每种类型的群组的具体细节在各个容器定义中进行了描述。

此示例假设您已使用该组的容器定义创建了一个 JSON 文件。

```
aws gamelift create-container-group-definition \
    --name MyAdventureGameContainerGroup \
    --operating-system AMAZON_LINUX_2023 \
    --scheduling-strategy REPLICA \
    --total-memory-limit 4096 \
    --total-cpu-limit 1024 \
    --container-definitions file://SimpleServer.json
```

创建容器定义JSON文件

创建容器组定义时，还要为该组定义容器。容器定义指定存储容器映像的 Amazon ECR 存储库，以及网络端口的可选配置、CPU 和内存使用限制以及其他设置。我们建议创建一个包含容器组中所有容器的配置的单个JSON文件。维护文件对于存储、共享、版本跟踪这些关键配置非常有用。如果您使用 AWS CLI 创建容器组定义，则可以在命令中引用该文件。

创建容器定义

1. 创建并打开一个新.JSON文件。例如：

   ```
   [~/work/glc]$ vim SimpleServer.json
   ```

2. 为该组的每个容器创建单独的容器定义。复制以下示例内容，并根据需要对其进行修改，以适应您的容器。有关容器定义语法的详细信息，请参阅 Amazon GameLift API 参考ContainerDefinitionInput中的。

3. 将文件保存在本地，以便您可以在 AWS CLI 命令中引用该文件。

示例：基本副本容器定义

Example

此示例描述了您的副本容器组的基本容器。必不可少的副本容器包括您的游戏服务器应用程序、Amazon A GameLift gent，还可能包括用于托管游戏的其他支持软件。定义必须包括名称、图像 URI 和端口配置。此示例还设置了一些特定于容器的资源限制。

```
[
  {
    "ContainerName": "SimpleServer",
    "ImageUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/gl-containers:complex-
server",
    "Essential": true,
    "Cpu": 256,
    "MemoryLimits": {
      "HardLimit": 128
  },
     "PortConfiguration": {
            "ContainerPortRanges": [
               {
                  "FromPort": 2000,
                  "Protocol": "TCP",
                  "ToPort": 2100
               }
            ]
      }
    }
]
```

创建 Amazon GameLift 集装箱船队

本文档适用于公开预览版中的一项功能。本文档随时可能更改。

创建容器组定义后，使用 Amazon GameLift 控制台或 AWS Command Line Interface (AWS CLI) 创建集装箱队列。

创建新队列后，当 Amazon 将您的容器组 GameLift部署到每个队列实例上并启动游戏服务器时，队列的状态会经历几个阶段。当舰队达到状态时ACTIVE，它就可以举办游戏会话了。有关实例集创建问题的帮助，请参阅调试 Amazon GameLift 实例集问题。

Console

在 Amazon GameLift 控制台中，选择要创建队列 AWS 区域 的位置。容器组定义必须位于您要创建队列的同一区域。

打开主机的左侧导航栏并选择 Fleets。在舰队页面上，选择创建舰队。

步骤 1：选择计算类型

- 选择容器计算类型。

第 2 步：定义舰队详细信息

1. 在舰队详细信息部分，输入舰队名称和描述。
2. 在容器组详细信息部分中，确定要部署到队列的容器组。您必须添加副本容器组。您可以选择添加守护程序容器组。每个群组都必须处于状态READY。
3. 设置舰队的连接端口范围。有关更多详细信息，请参阅配置网络连接。
4. （可选）为每个要部署的实例指定所需的副本。您可以指定所需的数字，也可以让 Amazon GameLift 计算可能的最大数字。如果您指定的所需数字大于计算的最大值，则队列创建将失败。创建队列后，您无法更改此设置。有关副本容器组装的更多详细信息，请参阅核心概念。
5. （可选）在“其他详细信息”下：

    a. 对于实例角色，请指定一个 IAM 角色，该角色授权游戏版本中的应用程序访问您账户中的其他 AWS 资源。有关更多信息，请参阅 与您的实例集中的其他 AWS 资源进行通信。要创建具有实例角色的实例集，您的账户必须拥有 IAM PassRole 权限。有关更多信息，请参阅 Amazon GameLift 的 IAM 权限示例。

    b. 对于指标组，输入新的或现有实例集指标组的名称。可以通过将多个实例集添加到同一个指标组来聚合指标。

步骤 3：定义实例详细信息

1. 在实例部署中，选择一个或多个要将实例部署到的远程位置。系统会自动选择主区域（这是您创建舰队的区域）。如果您选择其他位置，则实例集实例也将部署在这些位置。

> **Important**
>
> 要使用默认情况下未启用的区域，请在您的中将其启用 AWS 账户。
>
> - 如果您在 2022 年 2 月 28 日之前创建的未启用区域的实例集不受影响。
> - 要创建新的多位置实例集或更新现有的多位置实例集，请先启用您选择使用的任意区域。
>
> 有关默认情况下未启用的区域以及如何启用这些区域的更多信息，请参阅《AWS 一般参考》中的管理 AWS 区域。

2. 为队列选择实例配置。控制台会自动计算所需的最低 vCPU 和内存（基于您为每个容器组设置的总限制）。它会根据资源需求和您输入的位置筛选可用实例类型的完整列表。您可以根据需要添加其他过滤器。

   有关选择实例类型的更多信息，请参阅配置集装箱舰队。您选择的实例类型的大小将影响副本容器组打包到每个队列实例上的方式。根据您的选择，可以考虑查看每个实例所需的副本的设置。

步骤 4：配置运行时

运行时配置决定了游戏服务器进程的启动和运行方式。这些指令将传递给 Amazon A GameLift gent，Amazon Agent 在每个副本容器组中以相同的方式实现这些指令。您可以通过调用来更新队列的运行时配置UpdateRuntimeConfiguration。

1. 在启动路径中，输入游戏可执行文件的路径。
2. （可选）对于启动参数，输入要作为一组命令行参数传递给游戏可执行文件的信息。
3. 指定要在每个副本容器组中保持运行的并发进程数。查看每个实例的服务器进程数量的 Amazon GameLift 配额。对每个实例的并发服务器进程数的限制应用到所有配置的并发进程总数上。如果配置实例集超出限制，实例集无法激活。
4. 设置并发游戏会话激活的可选限制。这些设置允许您限制在开始新游戏会话时消耗的资源量。激活游戏会话可能会对现有游戏会话的性能产生影响。
5. 设置 EC2 端口设置以允许外部流量访问队列上运行的进程。指定为队列定义的部分或全部连接端口号。创建队列时不必设置这些端口，但是如果没有这些端口，流量就无法连接到游戏服务器。要稍后更新舰队的端口设置，请致电 UpdateFleetPortSettings
6. 在游戏会话资源设置下，配置以下可选功能：

a. 开启或关闭游戏缩放保护策略。启用保护后，如果实例正在托管活跃的游戏会话，Amazon 就 GameLift 不会在缩小规模活动期间关闭这些实例。

b. 设置资源创建的最大限制，以限制一个玩家在指定时间段内可以创建的游戏会话数量。

步骤 5：配置标签

- （可选）通过输入键和值对向构建添加标签。选择下一步继续查看实例集创建。

步骤 6：查看并创建。

- 查看您的舰队配置设置。

  您可以随时更新实例集的元数据和配置，无论实例集状态如何。有关更多信息，请参阅 管理 Amazon GameLift 实例集。您能在实例集进入 ACTIVE 状态之后更新实例集容量。有关更多信息，请参阅 扩展 Amazon GameLift 托管容量。您还可以添加或删除远程位置。

  审阅完毕后，选择 “创建”。

  如果您的请求成功，控制台将显示新舰队资源的详细信息页面。最初的状态是NEW，当 Amazon GameLift 开始队列创建过程时。您可以在实例集页面上跟踪新实例集的状态。舰队在达到状态时已准备好举办游戏会话ACTIVE。

AWS CLI

要使用创建集装箱舰队 AWS CLI，请打开命令行窗口并使用create-fleet命令。有关此命令的更多信息，请参阅《AWS CLI 命令参考》create-fleet中的。

下面显示的示例create-fleet请求创建了一个具有以下特征的新集装箱舰队：

- ContainerGroupsConfiguration 指定了单个副本容器组的定义：MegaFrogRaceServer.NA.v2。副本组的三个副本将部署到每个队列实例。每个实例有 30 个连接端口可供访问该实例上的进程。
- 队列使用 c5.large 按需实例。
- 它将容器组部署到以下位置：
  - us-west-2（主区域）
  - ca-central-1（远程位置）

- 一个实例上的每个副本容器组将同时运行 5 个游戏服务器进程，从而使每个实例一次最多可以托管 15 个游戏会话。
- 在每个副本容器组中，Amazon GameLift 允许同时激活两个新的游戏会话。它还会终止任何未准备好在 300 秒内托管玩家的正在激活的游戏会话。
- 在该实例集中的实例上托管的所有游戏会话将会开启游戏会话保护。
- 单个玩家可以在 15 分钟内创建三个新的游戏会话。

```
aws gamelift create-fleet \
    --name SampleFleet123 \
    --description "The sample test fleet" \
    --compute-type "CONTAINER" \
    --container-groups-configuration
 "ContainerGroupDefinitionNames=['MegaFrogRaceServer.NA.v2'],
 DesiredReplicaContainerGroupPerInstance=3,
 ConnectionPortRange={FromPort=1010,ToPort=1040}" \
    --ec2-instance-type c5.large \
    --region us-west-2 \
    --locations "Location=ca-central-1" \
    --fleet-type ON_DEMAND \
    --runtime-configuration "GameSessionActivationTimeoutSeconds=300,
 MaxConcurrentGameSessionActivations=2, ServerProcesses=[{LaunchPath=/local/game/
MegaFrogRace/server.exe,ConcurrentExecutions=5}]" \
    --new-game-session-protection-policy "FullProtection" \
    --resource-creation-limit-policy "NewGameSessionsPerCreator=3,
 PolicyPeriodInMinutes=15" \
    --ec2-inbound-permissions
 "FromPort=1010,ToPort=1040,IpRange=0.0.0.0/0,Protocol=UDP" \
```

如果创建队列请求成功，Amazon 将 GameLift 返回一组队列属性，其中包括您请求的配置设置和新的队列 ID。 GameLift 然后，Amazon 将舰队状态和位置状态设置为 “新建”，并启动舰队激活流程。您可以使用以下 CLI 命令跟踪实例集的状态并查看其他实例集信息：

- 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

您可以使用以下命令，根据需要更改实例集的容量和其他配置设置：

- update-fleet-attributes
- update-fleet-capacity
- update-fleet-port-settings
- update-runtime-configuration
- create-fleet-locations
- delete-fleet-locations

管理您的亚马逊 GameLift 集装箱船队

本文档适用于公开预览版中的一项功能。本文档随时可能更改。

当您想要获取有关集装箱船队的信息或进行更改时，可以使用以下操作来管理您的集装箱船队。

查看 资源

您可以通过以下几种方式获取有关集装箱船队中资源的信息。

- DescribeCompute-返回有关注册为计算的容器的详细信息。
- DescribeContainerGroupDefinition-返回有关容器组定义的详细信息。此资源描述了该组及其容器的配置方式。
- DescribeFleetAttributes-获取舰队属性，包括连接端口范围和其他属性。
- DescribeFleetCapacity-获取舰队中副本容器组的数量及其状态。
- DescribeRuntimeConfiguration-描述在每个副本容器组中运行的服务器进程。
- GetComputeAccess-提供对托管容器组的实例的远程访问。
- GetComputeAuthToken-向 Amazon 请求身份验证令牌， GameLift 以获取容器队列中的计算资源。
- ListCompute-列出注册为计算的容器组。
- ListContainerGroupDefinitions-列出容器组定义。

- ListFleets-列出使用特定集装箱组的舰队。

更新资源

以下是您可以创建和修改容器队列资源的一些方法。

- CreateContainerGroupDefinition-创建容器组定义。
- CreateFleet-设置为时ComputeType创建集装箱舰队CONTAINER。
- RegisterCompute-使用集装箱舰队注册计算。
- UpdateFleetAttributes-更新队列的可变属性，例如 Anywhere 舰队配置选项。
- UpdateFleetCapacity-更新托管 EC2 队列或集装箱队列的容量设置。
- UpdateRuntimeConfiguration-更新运行时配置，该配置描述了在注册为 compute 的每个副本容器组中要运行的服务器进程。

删除资源

以下是移除容器队列资源的一些方法。

- DeleteContainerGroupDefinition-删除容器组定义。
- DeleteFleet-删除舰队。
- DeregisterCompute-从容器队列中移除计算资源。

扩展 Amazon GameLift 集装箱船队

本文档适用于公开预览版中的一项功能。本文档随时可能更改。

游戏托管最具挑战性的任务之一是扩展容量以满足玩家的需求，同时又不会将成本浪费在不需要的资源上。在集装箱舰队中，您可以通过添加或移除舰队实例来扩展舰队容量。

当您创建新队列时，Amazon 会将队列的所需容量 GameLift 设置为一个实例，并在队列所在区域部署一个实例。对于多地点队列，Amazon 将一个实例 GameLift 部署到主区域和每个远程位置。队列状态达到后ACTIVE，您可以提高所需的容量以扩大规模，也可以降低所需的容量以缩小规模。

您可以使用 Amazon GameLift 扩展功能手动更改容量，也可以根据玩家需求设置自动扩展：

- 使用目标跟踪设置自动缩放。请参阅 基于目标的自动扩缩。

- 手动更改舰队的容量。请参阅 手动设置 Amazon GameLift 实例集的容量。

在扩展容器队列时，请考虑添加或删除实例会如何影响队列托管游戏会话和玩家的能力。

- 每个实例的游戏会话数
  - 实例上运行的每个游戏服务器进程都代表托管一个游戏会话的容量。
  - 使用以下公式计算在容器队列实例上同时运行的游戏会话数：

```
[Game sessions per instance] = [# of processes per replica container group] * [# of
 replica container groups per instance]
```

    - 对于每个副本容器组的进程，请调用 DescribeRuntimeConfiguration并计算游戏服务器进程的并发执行次数。
    - 对于每个实例的副本容器组，请调用DescribeFleetAttributes以获取`DesiredReplicaContainerGroupPerInstance`值。如果未设置此值，请使用该`MaxReplicaContainerGroupsPerInstance`值。
- 每个实例的玩家数
  - 您可以决定每个游戏会话中允许的玩家插槽数量。根据您的托管解决方案处理游戏会话放置的方式，您可以在配对配置中或在开始游戏会话放置的通话中定义每个游戏会话的玩家。
  - 使用以下公式计算可以在容器舰队实例上同时玩游戏的玩家数量：

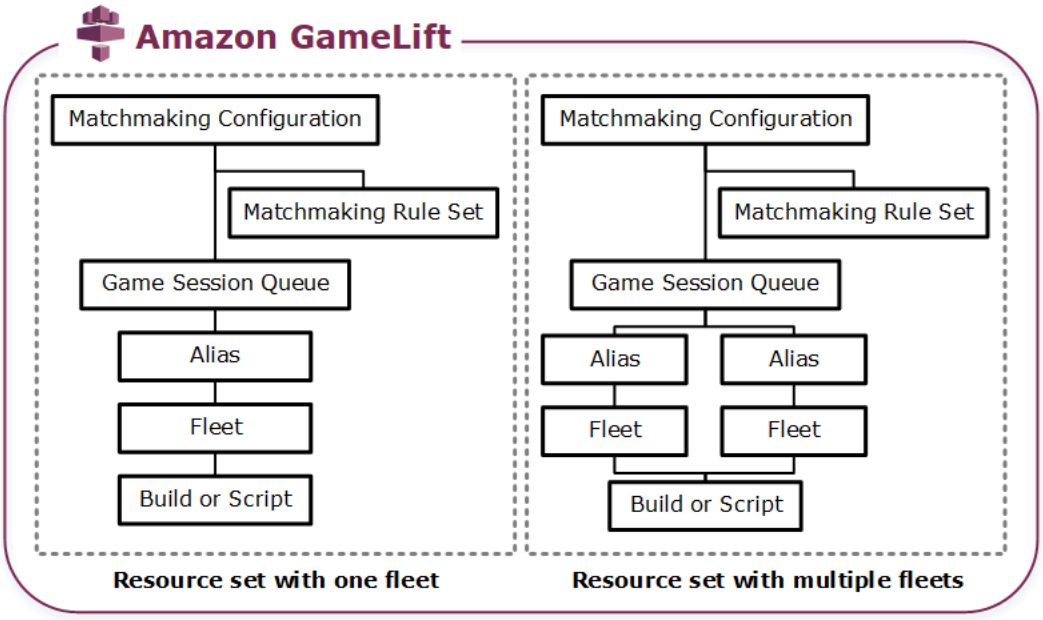
```
[Players per instance] = [# of game sessions per instance] * [# of player slots per
 game session]
```

要获取集装箱船队当前的总容量，请致电DescribeFleetCapacity或 C DescribeFleetLocation apac ity 以获取舰队中副本集装箱组的数量。活跃群组是指当前正在举办游戏会话的群组。闲置群组已准备好举办新的游戏会话。将这些值乘以每个副本容器组的服务器进程数。

管理 Amazon GameLift 托管资源

本节提供有关设置 Amazon GameLift 托管资源以运行游戏服务器和为玩家托管游戏会话的详细信息。您必须配置和部署资源，扩展容量以满足玩家需求，并找到可用资源来托管游戏会话。

下图说明了 Amazon GameLift 资源对象如何相互关联。使用构建或脚本创建实例集，为实例集指定别名，并使用别名将其添加到游戏会话队列中。对于使用 FlexMatch 对战的游戏，使用游戏会话队列和对战规则集来创建对战配置。



游戏服务器代码

- 构建 – 在 Amazon GameLift 上运行并为玩家托管游戏会话的自定义生成游戏服务器软件。游戏构建表示在特定操作系统上运行游戏服务器的一组文件，并且必须与 Amazon GameLift 集成。将游戏构建文件上传到计划设置实例集的 AWS 区域 中的 Amazon GameLift。有关更多信息，请参阅将自定义服务器构建上传到 Amazon GameLift。
- 脚本 – 用于实时服务器的配置和自定义游戏逻辑。您可以使用 JavaScript 创建脚本，为游戏客户端配置实时服务器，并添加自定义游戏逻辑来托管玩家的游戏会话。有关更多信息，请参阅将实时服务器脚本上传到 Amazon GameLift。

实例集

运行游戏服务器并托管玩家的游戏会话的计算资源集合。有关可以在何处部署实例集的信息，请参阅 亚马逊 GameLift 托管地点。有关创建实例集的信息，请参阅设置 Amazon GameLift 实例集。

别名

实例集的抽象标识符，可用于随时更改玩家连接的实例集。有关更多信息，请参阅向 Amazon GameLift 舰队添加别名。

游戏会话队列

一种游戏会话放置机制，用于接收新游戏会话的请求并搜索可用的游戏服务器来托管新会话。有关游戏会话队列的更多信息，请参阅《Amazon GameLift 开发人员指南》中的设置游戏会话置放的 GameLift 队列。

将构建和脚本上传到 Amazon GameLift

在部署多人游戏服务器以通过 Amazon GameLift 进行托管之前，您需要上传游戏服务器文件。本节中的主题提供有关准备和上传自定义游戏服务器构建文件或实时服务器脚本文件的指导。

主题

- 将自定义服务器构建上传到 Amazon GameLift
- 将实时服务器脚本上传到 Amazon GameLift

将自定义服务器构建上传到 Amazon GameLift

将游戏服务器与 Amazon GameLift 集成后，将构建文件上传到 Amazon GameLift。本主题介绍如何使用 AWS Command Line Interface (AWS CLI) 或 AWS 软件开发工具包打包游戏构建文件，创建可选的构建安装脚本，然后上传文件。

主题

- 打包游戏构建文件
- 创建 Amazon GameLift 构建
- 更新您的构建文件
- 添加构建安装脚本

打包游戏构建文件

在将您配置的游戏服务器上传到 Amazon GameLift 之前，请将游戏构建文件打包到构建目录中。此目录必须包含运行您的游戏服务器和托管游戏会话所需的所有组件，如下所示：

- 游戏服务器二进制文件 – 运行游戏服务器所需的二进制文件。构建可以包括多个为相同平台构建的游戏服务器的二进制文件。有关受支持平台的列表，请参阅Amazon 为开发提供支持 GameLift。
- 依赖项 – 运行游戏服务器可执行文件所需的任何相关文件。示例包括资产、配置文件和相关库。

> **Note**
>
> 对于使用适用于 C++ 的 Amazon GameLift 服务器软件开发工具包创建的游戏构建（包括使用 Unreal 插件创建的构建），请包含与您构建服务器软件开发工具包时使用的相同 OpenSSL 版本的 OpenSSL DLL。有关更多详细信息，请参阅服务器软件开发工具包自述文件。

- 安装脚本（可选）– 用于处理在 Amazon GameLift 托管服务器上安装游戏构建的任务的脚本文件。将此文件放置到构建目录的根目录中。Amazon GameLift 在创建实例集时运行安装脚本。

您可以在构建中设置任何应用程序，包括您的安装脚本，以确保安全访问您在其他 AWS 服务上的资源。有关如何执行此操作的信息，请参阅与您的实例集中的其他 AWS 资源进行通信。

在您打包构建文件后，请确保您的游戏服务器运行在目标操作系统的干净安装上。这有助于验证您在程序包内包含了所有必需的依赖项，并且您的安装脚本准确。

创建 Amazon GameLift 构建

创建构建和上传文件时，有几个选项可供选择：

- 从文件目录创建构建。这是最简单且最常用的方法。
- 使用 Amazon Simple Storage Service (Amazon S3) 中的文件创建构建。使用此选项，您可以在 Amazon S3 中管理您的构建版本。

通过这两种方法，Amazon GameLift 使用唯一的构建 ID 和其他元数据创建新的构建资源。生成以已初始化状态开始。Amazon GameLift 获取游戏服务器文件后，构建将变为就绪状态。

构建准备就绪后，您可以将其部署到新的 Amazon GameLift 实例集中。有关更多信息，请参阅创建 Amazon GameLift 托管实例集。当 Amazon GameLift 设置新实例集时，它会将构建文件下载到每个实例集实例并安装构建文件。

从文件目录创建构建

要创建存储在任意位置（包括本地目录）的游戏构建，请使用 `upload-build` AWS CLI 命令。此命令在 Amazon GameLift 中创建一个新的构建记录并从您指定的位置上传文件。

发送上传请求。在命令行窗口中，键入以下 upload-build 命令和参数。

```
aws gamelift upload-build \
    --name user-defined name of build \
    --operating-system supported OS \
    --server-sdk-version Amazon GameLift server SDK version \
    --build-root build path \
    --build-version user-defined build number \
    --region region name
```

- operating-system – 游戏服务器构建的运行时环境。您必须指定操作系统值。您稍后无法更新。
- server-sdk-version – 您的游戏服务器与之集成的 Amazon GameLift 服务器软件开发工具包版本。如果您没有提供值，则 Amazon GameLift 会使用默认值 `4.0.2`。如果您指定的服务器软件开发工具包版本不正确，则在调用 `InitSdk` 建立与 Amazon GameLift 服务的连接时，游戏服务器构建可能会失败。
- build-root – 您的构建文件的目录路径。
- name – 新构建的描述性名称。
- build-version – 构建文件的版本详细信息。
- region – 要创建构建的 AWS 区域。在要部署实例集的区域中创建构建。如果您在多个区域中部署游戏，则需在每个区域中创建一个构建。

> **Note**
>
> 使用 aws configure get region 查看您当前的默认区域。要更改默认区域，请使用 aws configure set region ***region name*** 命令。

示例

```
aws gamelift upload-build \
    --operating-system AMAZON_LINUX_2023 \

    --server-sdk-version "5.0.0" \
    --build-root "~/mygame" \
    --name "My Game Nightly Build" \
    --build-version "build 255" \
    --region us-west-2
```

```
aws gamelift upload-build \
    --operating-system WINDOWS_2016 \
    --server-sdk-version "5.0.0" \
    --build-root "C:\mygame" \
    --name "My Game Nightly Build" \
    --build-version "build 255" \
    --region us-west-2
```

为了回应您的上传请求，Amazon GameLift 会提供上传进度。成功上传后，Amazon GameLift 会返回新的构建记录 ID。上传时间取决于游戏文件的大小和连接速度。

使用 Amazon S3 中的文件创建构建

您可以将构建文件存储在 Amazon S3 中，然后从那里将其上传到 Amazon GameLift。在创建构建时，指定 S3 存储桶的位置，然后 Amazon GameLift 会从 Amazon S3 直接检索构建文件。

创建构建资源

1. 将构建文件存储在 Amazon S3 中。创建一个 .zip 文件，其中包含打包的构建文件，并将其上传到您 AWS 账户中的 S3 存储桶。记下存储桶标签和文件名；您在创建 Amazon GameLift 构建时需要这些内容。
2. 向 Amazon GameLift 提供对您构建文件的访问权限。按照在 Amazon S3 中访问游戏构建文件中的说明创建 IAM 角色。创建角色后，记录新角色的 Amazon 资源名称 (ARN)，您在创建构建时需要该名称。
3. 创建构建 使用 Amazon GameLift 控制台或 AWS CLI 创建新的构建记录。您必须拥有 `PassRole` 权限，如Amazon GameLift 的 IAM 权限示例中所述。

Console

1. 在 Amazon GameLift 控制台的导航窗格中，选择托管、构建。
2. 在构建页面上，选择创建构建。
3. 在创建构建页面的构建设置下，执行以下操作：

    a. 对于名称，输入脚本名称。

    b. 对于版本，输入版本。由于构建内容可以更新，版本数据则有助于跟踪更新。

    c. 对于操作系统 (OS)，选择您的游戏服务器构建的操作系统。您稍后无法更新此值。

    d. 对于游戏服务器构建，输入您上传到 Amazon S3 的构建目标的 S3 URI，然后选择目标版本。如果您忘记了 Amazon S3 URI 和对象版本，选择浏览 S3，然后搜索构建对象。

e. 对于 IAM 角色，选择您创建的允许 Amazon GameLift 访问您的 S3 存储桶和构建对象的角色。

4. （可选）在标签下，通过输入键和值对向构建添加标签。

5. 选择创建。

Amazon GameLift 为新构建分配一个 ID 并上传指定的.zip 文件。您可以在构建页面上查看新构建，包括状态。

AWS CLI

使用 `create-build` 命令定义新构建并上传您的服务器构建文件。

1. 打开命令行窗口，然后切换到您可以使用 AWS CLI 的目录。

2. 输入以下 create-build 命令：

```
aws gamelift create-build \
    --name user-defined name of build \
    --server-sdk-version Amazon GameLift server SDK version \
    --operating-system supported OS \
    --build-version user-defined build number \
    --storage-location "Bucket"=S3 bucket label,"Key"=Build .zip file
 name,"RoleArn"=Access role ARN} \
    --region region name
```

- name – 新构建的描述性名称。
- server-sdk-version – 您用于集成游戏服务器和 Amazon GameLift 的 Amazon GameLift 服务器软件开发工具包版本。如果您没有提供值，则 Amazon GameLift 会使用默认值 `4.0.2`。
- operating-system – 游戏服务器构建的运行时环境。您必须指定操作系统值。您稍后无法更新。
- build-version – 构建文件的版本详细信息。这些信息可能很有用，因为游戏服务器的每个新版本都需要新的构建资源。
- storage-location
  - Bucket – 包含您的构建的 S3 存储桶的名称。示例：“my_build_files”。
  - Key – 包含您的构建文件的 .zip 文件的名称。示例：“my_game_build_7.0.1, 7.0.2”。
  - RoleARN – 分配给您创建的 IAM 角色的 ARN。示例：“arn:aws:iam::111122223333:role/GameLiftAccess”。有关策略示例，请参阅在 Amazon S3 中访问游戏构建文件。

- region – 您必须在要部署实例集的 AWS 区域中创建构建。如果您在多个区域中部署游戏，则需在每个区域中创建一个构建。

> **Note**
>
> 我们建议使用 `configure get` 命令命令检查当前的默认区域。要更改默认区域，请使用 `configure set` 命令。

示例

```
aws gamelift create-build \
    --operating-system WINDOWS_2016 \
    --storage-location
 "Bucket"="my_game_build_files","Key"="mygame_build_101.zip","RoleArn"="arn:aws:iam::111
gamelift" \
    --name "My Game Nightly Build" \
    --build-version "build 101" \
    --region us-west-2
```

3. 要查看新构建，请使用 `describe-build` 命令。

更新您的构建文件

您可以使用 Amazon GameLift 控制台或 `update-build` AWS CLI 命令更新构建资源的元数据。

创建 Amazon GameLift 构建后，您将无法更新与之关联的构建文件。对于每组新文件，都要创建一个新的 Amazon GameLift 构建。使用 `upload-build` 命令，Amazon GameLift 会自动为每个请求创建新的构建记录。如果您使用 `create-build` 命令提供构建文件，则使用其他名称上传新构建 .zip 文件到 Amazon S3 并通过引用新的文件名创建构建。

对于部署更新的构建，请尝试这些提示：

- 根据需要使用队列和换出实例集。在通过 Amazon GameLift 设置游戏客户端时，指定队列而不是实例集。借助队列，您可以向队列添加新实例集和新构建，然后删除旧实例集。有关更多信息，请参阅《Amazon GameLift 开发人员指南》中的设置游戏会话置放的 GameLift 队列。
- 使用别名将玩家传输到新游戏构建。当您将游戏客户端集成到 Amazon GameLift 时，请指定实例集别名而不是实例集 ID。有关更多信息，请参阅向 Amazon GameLift 舰队添加别名。

- 设置自动构建更新。有关将 Amazon GameLift 部署整合到构建系统的示例脚本和信息，请参阅 AWS 游戏技术博客上的自动部署到 Amazon GameLift。

添加构建安装脚本

创建适用于您的游戏构建操作系统 (OS) 的安装脚本：

- Windows：创建名为 `install.bat` 的批处理文件。
- Linux：创建名为 `install.sh` 的 Shell 脚本文件。

在创建安装脚本时，请注意以下事项：

- 该脚本不接受任何用户输入。
- Amazon GameLift 在托管服务器上的以下位置安装构建并重新创建构建包中的文件目录：
  - Windows 实例集：`C:\game`
  - Linux 实例集：`/local/game`
- 在 Linux 实例集安装过程中，run-as 用户具有对实例文件结构的有限访问权限。此用户对安装您的构建文件的目录拥有全部权限。如果您的安装脚本执行的操作需要管理员权限，请使用 sudo 指定管理员访问权限。默认情况下，Windows 实例集的 run-as 用户具有管理员权限。与安装脚本相关的权限失败会生成一条事件消息，此消息指示脚本出现问题。
- 在 Linux 上，Amazon GameLift 支持常用的 Shell 解释器语言，例如 bash。在安装脚本的顶部添加 shebang (例如 `#!/bin/bash`)。要验证对您的首选 Shell 命令的支持，可以远程访问活动的 Linux 实例并打开 Shell 提示符。有关更多信息，请参阅远程连接到 Amazon GameLift 舰队实例。
- 安装脚本不能依赖于 VPC 对等连接。直到 Amazon GameLift 在实例集实例上安装构建后，VPC 对等连接才可用。

Example Windows 安装 bash 文件

此示例 `install.bat` 文件安装游戏服务器所需的 Visual C++ 运行时组件，然后将结果写入日志文件。脚本包含根目录下构建包中的组件文件。

```
vcredist_x64.exe /install /quiet /norestart /log c:\game\vcredist_2013_x64.log
```

Example Linux 安装 Shell 脚本

此示例 `install.sh` 文件在安装脚本中使用 bash 并将结果写入日志文件。

```
#!/bin/bash
echo 'Hello World' > install.log
```

此示例 `install.sh` 文件展示了如何使用 Amazon CloudWatch 代理收集系统级和自定义指标以及处理日志轮换。由于 Amazon GameLift 在服务 VPC 中运行，因此您必须向 Amazon GameLift 授予代表您担任 AWS Identity and Access Management (IAM) 角色的权限。要允许 Amazon GameLift 担任角色，请创建一个包含 AWS 托管策略 `CloudWatchAgentAdminPolicy` 的角色，并在创建实例集时使用该角色。

```
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
```

将实时服务器脚本上传到 Amazon GameLift

当准备好为游戏部署实时服务器时，请将完成的实时服务器脚本文件上传到 Amazon GameLift。为此，请创建 Amazon GameLift 脚本资源并指定脚本文件的位置。您还可以通过上传现有脚本资源的新文件来更新已部署的服务器脚本文件。

当您创建新的脚本资源时，Amazon GameLift 会为其分配一个唯一的脚本 ID（例如，`script-1111aaaa-22bb-33cc-44dd-5555eeee66ff`），并上传脚本文件的副本。上传时间取决于脚本文件的大小和连接速度。

创建脚本资源后，Amazon GameLift 将使用新的实时服务器实例集部署该脚本。Amazon GameLift 将您的服务器脚本安装到实例集中的每个实例上，并将脚本文件放入 `/local/game`。

要排查与服务器脚本相关的实例集激活问题，请参阅调试 Amazon GameLift 实例集问题。

打包脚本文件

您的服务器脚本可以包含一个或多个文件，合并成单个 .zip 文件以供上传。.zip 文件必须包含脚本运行所需的所有文件。

您可以将压缩后的脚本文件存储在本地文件目录，或存储在 Amazon Simple Storage Service (Amazon S3) 存储桶。

从本地目录上传脚本文件

如果您的脚本文件存储在本地，则可以从该位置将其上传到 Amazon GameLift。要创建脚本资源，请使用 Amazon GameLift 控制台或 AWS Command Line Interface (AWS CLI)。

Amazon GameLift console

创建脚本资源

1. 打开 Amazon GameLift 控制台。
2. 在导航窗格中，选择托管，脚本。
3. 在脚本页面上，选择创建脚本。
4. 在创建脚本页面的脚本设置下，执行以下操作：
   a. 对于名称，输入脚本名称。
   b. （可选）对于版本，输入版本信息。由于脚本内容可以更新，版本数据则有助于跟踪更新。
   c. 对于脚本源，选择上传 .zip 文件。
   d. 对于脚本文件，选择选择文件，浏览包含脚本的 .zip 文件，然后选择该文件。
5. （可选）在标签下，通过输入键和值对向脚本添加标签。
6. 选择创建。

   Amazon GameLift 为新脚本分配一个 ID 并上传指定的 .zip 文件。您可以在脚本页面查看新脚本（包括其状态）。

AWS CLI

使用 `create-script` AWS CLI 命令定义新脚本并上传您的服务器脚本文件。

创建脚本资源

1. 将 .zip 文件放入可以使用 AWS CLI 的目录中。
2. 打开命令行窗口，然后切换到放置 .zip 文件的目录。
3. 输入以下 create-script 命令和参数。对于 --zip-file 参数，请务必在 .zip 文件的名称前加上字符串 `fileb://`。它将文件标识为二进制文件，以便 Amazon GameLift 处理压缩后的内容。

```
aws gamelift create-script \
    --name user-defined name of script \
    --script-version user-defined version info \
    --zip-file fileb://name of zip file \
    --region region name
```

示例

```
aws gamelift create-script \
    --name "My_Realtime_Server_Script_1" \
    --script-version "1.0.0" \
    --zip-file fileb://myrealtime_script_1.0.0.zip \
    --region us-west-2
```

作为对您的请求的响应，Amazon GameLift 服务会返回新的脚本对象。

4. 要查看新脚本，请致电 describe-script。

从 Amazon S3 上传脚本文件

您可以选择在 Amazon S3 存储桶中存储脚本文件，并从这里将它们上传到 Amazon GameLift。在创建脚本时，您可以指定 S3 存储桶的位置，然后 Amazon GameLift 会从 Amazon S3 检索您的脚本文件。

创建脚本资源

1. 在 S3 存储桶中存储脚本文件。创建包含您的服务器脚本文件的 .zip 文件，并将其上传到您控制的 AWS 账户中的 S3 存储桶。记下对象 URI – 创建 Amazon GameLift 脚本时需要它。

   > **Note**
   >
   > Amazon GameLift 不支持从名称包含句点 (.) 的 S3 存储桶进行上传。

2. 允许 Amazon GameLift 访问脚本文件。要创建允许 Amazon GameLift 访问包含服务器脚本的 S3 存储桶的 AWS Identity and Access Management (IAM) 角色，请按照为亚马逊设置 IAM 服务角色 GameLift中的说明进行操作。创建新角色后，请记下其名称，创建脚本时需要使用该名称。
3. 创建脚本。使用 Amazon GameLift 控制台或 AWS CLI 创建新的脚本记录。要提出此请求，您必须拥有 IAM `PassRole` 权限，如Amazon GameLift 的 IAM 权限示例中所述。

Amazon GameLift console

1. 在 Amazon GameLift 控制台的导航窗格中，选择托管、脚本。
2. 在脚本页面上，选择创建脚本。
3. 在创建脚本页面的脚本设置下，执行以下操作：

a. 对于名称，输入脚本名称。

b. （可选）对于版本，输入版本信息。由于脚本内容可以更新，版本数据则有助于跟踪更新。

c. 对于脚本源，选择 Amazon S3 URI。

d. 输入您上传到 Amazon S3 的脚本对象的 S3 URI，然后选择对象版本。如果您忘记了 Amazon S3 URI 和对象版本，请选择浏览 S3，然后搜索脚本对象。

4. （可选）在标签下，通过输入键和值对向脚本添加标签。

5. 选择创建。

Amazon GameLift 为新脚本分配一个 ID 并上传指定的 .zip 文件。您可以在脚本页面查看新脚本（包括其状态）。

AWS CLI

使用 `create-script` AWS CLI 命令定义新脚本并上传您的服务器脚本文件。

1. 打开命令行窗口，然后切换到您可以使用 AWS CLI 的目录。

2. 输入以下 create-script 命令和参数。--storage-location 参数指定脚本文件的 Amazon S3 存储桶位置。

```
aws gamelift create-script \
    --name [user-defined name of script] \
    --script-version [user-defined version info] \
    --storage-location "Bucket"=S3 bucket name,"Key"=name of zip file in S3
 bucket,"RoleArn"=Access role ARN \
    --region region name
```

示例

```
aws gamelift create-script \
    --name "My_Realtime_Server_Script_1" \
    --script-version "1.0.0" \
    --storage-location "Bucket"="gamelift-
script","Key"="myrealtime_script_1.0.0.zip","RoleArn"="arn:aws:iam::123456789012:role/
S3Access" \
    --region us-west-2
```

作为对您的请求的响应，Amazon GameLift 服务会返回新的脚本对象。

3. 要查看新脚本，请致电 `describe-script`。

更新脚本文件

您可以使用 Amazon GameLift 控制台或 `update-script`AWS CLI 命令更新脚本资源的元数据。

您也可以更新脚本资源的脚本内容。Amazon GameLift 会将脚本内容部署到使用更新后的脚本资源的所有实例集实例。部署更新的脚本后，实例会在启动新的游戏会话时使用该脚本。更新时已经在运行的游戏会话不使用更新后的脚本。

更新脚本文件

- 对于存储在本地的脚本文件，要上传更新后的脚本 .zip 文件，请使用 Amazon GameLift 控制台或 update-script 命令。
- 对于存储在 Amazon S3 存储桶中的脚本文件，请将更新后的脚本文件上传到 S3 存储桶。Amazon GameLift 会定期检查是否有更新的脚本文件并直接从 S3 存储桶中检索它们。

设置 Amazon GameLift 实例集

本节提供有关设计、构建和维护实例集以便与 Amazon GameLift 结合使用的详细信息。您可以使用 Amazon GameLift 实例集来部署自定义游戏服务器和实时服务器。

实例集将托管资源表示为 Amazon Elastic Compute Cloud (Amazon EC2) 实例或物理硬件的集合。实例集的位置决定了实例或硬件的部署位置，以便为玩家托管游戏会话。实例集的大小，以及它可以支持的游戏会话和玩家数量，取决于实例的数量或您为其提供的硬件数量。您可以手动调整虚拟实例，也可以使用自动扩展来调整虚拟实例。

生产环境中的大多数游戏需要多个实例集。例如，您可以使用多个实例集来同时运行多个版本的游戏服务器、为竞价型实例集提供备份容量或内置冗余。

要了解如何创建专为游戏需求而设计的实例集，请从Amazon GameLift 实例集设计指南开始。实例集运行后，请参阅扩展 Amazon GameLift 托管容量、向 Amazon GameLift 舰队添加别名和《Amazon GameLift 开发人员指南》中的设置游戏会话置放的 GameLift 队列。

主题

- Amazon GameLift 实例集设计指南
- 创建新的 Amazon GameLift 实例集

- 管理 Amazon GameLift 实例集
- 向 Amazon GameLift 舰队添加别名
- 调试 Amazon GameLift 实例集问题
- 远程连接到 Amazon GameLift 舰队实例

Amazon GameLift 实例集设计指南

本设计指南涵盖了创建用于 Amazon GameLift 的托管资源实例集的最佳实操。选择托管资源组合，并了解如何配置它们以最适合您的游戏。

主题

- 选择 Amazon GameLift 计算资源
- 管理 Amazon GameLift 如何启动游戏服务器
- 在 Amazon 上使用竞价型实例 GameLift

选择 Amazon GameLift 计算资源

为了为您的玩家部署游戏服务器和托管游戏会话，亚马逊 GameLift 使用名为实例的亚马逊弹性计算云 (Amazon EC2) 资源或您的物理硬件。设置新实例集时，决定所需的实例类型以及如何对其运行游戏服务器进程。当托管 EC2 队列处于活动状态并准备好托管游戏会话时，您可以根据需要添加或删除实例以满足玩家的需求。

您可以将您的 Amazon GameLift 游戏服务器部署在两种计算类型的组合上：

- 托管 EC2 – 托管 EC2 实例集使用 Amazon EC2 实例来托管您的游戏服务器。Amazon 可以 GameLift 管理实例，减轻托管游戏的硬件和软件管理负担。
- 亚马逊 GameLift Anywhere — 亚马逊 GameLift Anywhere舰队使用您现有的基础设施托管游戏服务器，而亚马逊则 GameLift 管理您的配对和队列。

在为实例集选择计算资源时，请考虑以下因素：

- 可用硬件
- 实例集位置
- 按需型实例和竞价型实例
- 操作系统
- 实例类型

- 服务限额

可用硬件

在实施中考虑现有的基础架构。当您将游戏迁移到 Amazon 时 GameLift，您可以继续使用您的基础架构。借助 Amazon GameLift Anywhere，您可以将自己的基础设施与 Amazon GameLift 托管 EC2 实例一起使用。您还可以使用现有基础架构，在 Amazon 支持的地点允许的范围之外在距离玩家更近 GameLift的地方托管游戏。有关设置 Amazon GameLift Anywhere 车队的更多信息，请参阅创建亚马逊 GameLift Anywhere舰队。

实例集位置

考虑一下您计划部署游戏服务器的地理位置。实例类型的可用性因地区 AWS 区域 和本地区域而异。

对于多位置队列，实例可用性和限额取决于实例集所在主区域和选定远程位置的组合。有关实例集位置的更多信息，请参阅亚马逊 GameLift 托管地点。

对于 Amazon GameLift Anywhere 车队，您可以确定物理硬件的位置。有关自定义位置的更多信息，请参阅Amazon GameLift Anywhere。

按需型实例和竞价型实例

Amazon EC2 按需型实例和竞价型实例提供相同的硬件和性能，但它们在可用性和成本上有所不同。

按需型实例

您始终可以在需要按需型实例时获取它并将它保存任意长的时间。按需型实例具有固定成本，意味着您将为使用这些实例的时间量付费，并且没有任何长期承诺。

竞价型实例

通过利用未使用的 AWS 计算容量，竞价型实例可以为按需实例提供经济实惠的替代方案。竞价型实例的价格根据每个地点每种实例类型的供需情况而波动。 AWS 可以在竞价型实例需要恢复容量时将其中断。Amazon GameLift 使用队列和 FleetiQ 算法来确定 AWS 这将中断竞价型实例，从而使该实例处于回收状态。然后，当实例上没有活跃的游戏会话时，Amazon 会 GameLift 尝试替换它。

有关如何使用竞价型实例的更多信息，请参阅在 Amazon 上使用竞价型实例 GameLift。

操作系统

亚马逊 GameLift 实例支持在微软 Windows 或亚马逊 Linux 上运行的游戏服务器版本。将游戏版本上传到 Amazon 时 GameLift，请指定游戏的操作系统。当您创建 Amazon EC2 队列来部署游戏版本

时，Amazon GameLift 会自动使用该版本的操作系统设置实例。有关受支持的游戏服务器操作系统的更多信息，请参阅Amazon 为开发提供支持 GameLift。

使用 Amazon GameLift Anywhere 队列时，您可以使用您的硬件支持的任何操作系统。Amazon GameLift Anywhere 队列要求您将游戏版本部署到硬件，同时使用 Amazon GameLift 在一个地方管理您的资源。

实例类型

Amazon EC2 实例集的实例类型决定了实例使用的硬件类型。不同实例类型提供了计算能力、内存、存储和网络功能的不同组合。

在为您的游戏选择可用实例类型时，请考虑：

- 游戏服务器的计算架构：x64 或 Arm（AWS Graviton）。

  > **Note**
  >
  > Graviton Arm 实例需要在 Linux 操作系统上构建亚马逊 GameLift 服务器。C++ 和 C# 需要服务器软件开发工具包 5.1.1 或更高版本。Go 需要服务器软件开发工具包 5.0 或更高版本。这些实例不 out-of-the-box 支持在亚马逊 Linux 2023 (AL2023) 或亚马逊 Linux 2 (AL2) 上安装 Mono。

- 您的游戏服务器构建的计算、内存和存储要求。
- 您计划在每个实例上运行的服务器进程数。

通过使用更大的实例类型，您可能能够在每个实例上运行多个服务器进程。这可以减少满足玩家需求所需的实例数量。

有关更多信息：

- 有关实例类型，请参阅 Amazon EC2 实例类型。
- 关于每个实例运行多个进程，请参阅管理 Amazon GameLift 如何启动游戏服务器。

服务限额

要查看 Amazon GameLift 的默认服务配额以及您的当前配额 AWS 账户，请执行以下操作：

- 有关亚马逊的一般服务配额信息 GameLift，请参阅中的亚马逊 GameLift终端节点和配额AWS 一般参考。

- 要查看您的账户每个位置的可用实例类型列表，请打开 Amazon GameLift 控制台的服务配额页面。该页面还会显示您的账户在每个位置的每种实例类型的当前使用情况。
- 要查看您的账户当前每个区域的实例类型配额列表，请运行 AWS Command Line Interface (AWS CLI) 命令`describe-ec2-instance-limits`。此命令返回您在默认区域（或您指定的其他区域）中拥有的活动实例数量。

准备发布游戏时，请在 Amazon GameLift 主机中填写发布问卷。Amazon GameLift 团队使用发布问卷来确定您的游戏的正确配额和限制。

管理 Amazon GameLift 如何启动游戏服务器

您可以设置托管 EC2 实例集的运行时配置，以便对每个实例运行多个游戏服务器进程。这样可以更有效地使用您的托管资源。

实例集如何管理多个进程

Amazon GameLift 使用实例集的运行时配置来确定要在每个实例上运行的进程的类型和数量。运行时配置至少包含一个代表一个游戏服务器可执行文件的服务器进程配置。您可以定义其他服务器进程配置，以运行与您的游戏相关的其他类型的进程。每个服务器进程配置都包含以下信息：

- 游戏构建中可执行文件的文件名和路径。
- （可选）要在启动时传递给进程的参数。
- 要同时运行的进程的数量。

当实例集中的实例激活时，它会启动在运行时配置中定义的服务器进程。对于多个进程，Amazon GameLift 会错开每个进程的启动。服务器进程具有有限的生命周期。当它们结束时，Amazon GameLift 会启动新的进程，以保持运行时配置中定义的服务器进程的数量和类型。

您可以随时通过添加、更改或删除服务器进程配置来更改运行时配置。每个实例都会定期检查对实例集运行时配置的更新，以实施更改。此处介绍 Amazon GameLift 如何采纳运行时配置更改：

1. 该实例向 Amazon GameLift 发送请求以获取最新版本的运行时配置。
2. 该实例将其活动进程与最新的运行时配置进行比较，然后执行以下操作：
   - 如果更新的运行时配置删除了某个服务器进程类型，此类型的活动服务器进程将继续运行直到结束。该实例不会取代这些服务器进程。
   - 如果更新的运行时配置减少了某个服务器进程类型的并发进程数，此类型的多余服务器进程将继续运行直到结束。该实例不会取代这些多余服务器进程。

- 如果更新的运行时配置添加了新的服务器进程类型或增加了现有类型的并发进程，该实例将启动新的服务器进程，直至达到 Amazon GameLift 的上限。在这种情况下，该实例在现有进程结束时启动新的服务器进程。

针对多个进程优化实例集

要在实例集上使用多个进程，请执行以下操作：

- 创建构建，其中包含您要部署到实例集并将构建上传到 Amazon GameLift 的游戏服务器可执行文件。构建中的所有游戏服务器都必须在同一个平台上运行并使用 Amazon GameLift 服务器软件开发工具包。
- 使用一个或多个服务器进程配置和多个并发进程创建运行时配置。
- 将游戏客户端与 AWS 软件开发工具包版本 2016-08-04 或更高版本集成。

要优化实例集性能，建议您执行以下操作：

- 处理服务器进程关闭方案，以便 Amazon GameLift 可以高效率地回收进程。例如：
  - 向调用服务器 API `ProcessEnding()` 的游戏服务器代码添加关闭程序。
  - 在您的游戏服务器代码中实现回调函数 `OnProcessTerminate()` 以处理来自 Amazon GameLift 的终止请求。
- 确保 Amazon GameLift 已关闭并重新启动运行状况不正常的服务器进程。通过在游戏服务器代码中实现 `OnHealthCheck()` 回调函数，将运行状况状态报告给 Amazon GameLift。Amazon GameLift 自动关闭连续三次报告为不正常的服务器进程。如果不实施 `OnHealthCheck()`，Amazon GameLift 假定服务器进程正常运行，除非该进程无法响应通信。

选择每个实例的进程数

在决定要在实例上运行的并发进程数时，需要记住以下内容：

- Amazon GameLift 限制每个实例的最大并发进程数。实例集服务器进程配置的所有并发进程的总和不能超过此限额。
- 为维持可接受的性能水平，Amazon EC2 实例类型可能会限制可并发运行的进程数。测试游戏的不同配置以找出您首选实例类型的合适进程数。
- Amazon GameLift 运行的并发进程数不会超过配置的总数。这意味着从以前的运行时配置到新配置的过渡可能会逐渐发生。

在 Amazon 上使用竞价型实例 GameLift

在设置 Amazon GameLift 托管 EC2 队列时，您可以使用竞价型实例、按需实例或两者的组合。要详细了解 Amazon 如何 GameLift 使用竞价型实例，请参阅按需型实例和竞价型实例。要使用竞价型实例集，您的游戏集成需要进行此页面上列出的调整。

你在用配 FlexMatch 对吗？ 您可以将竞价型实例集添加到现有的游戏会话实例集中，以便进行对战放置。

1. 为竞价型实例设计游戏会话队列。

   使用队列管理游戏会话放置是一个最佳实操，它在使用竞价型实例时是必需的。要设计队列，请考虑以下事项：

   - 位置 – 要获得最佳的玩家体验，请选择地理位置靠近玩家的位置。
   - 实例类型 – 考虑您的游戏服务器硬件要求以及您选择的位置的实例可用性。

   要尝试使用可优化竞价型实例可用性和弹性的队列，请参阅教程：为竞价型实例设置游戏会话队列。
2. 为针对竞价型实例进行了优化的队列创建实例集。

   根据您的队列设计，创建实例集以将游戏服务器部署到所需的位置和实例类型。请参阅创建 Amazon GameLift 托管实例集以帮助您创建和配置新实例集。
3. 创建游戏会话队列。

   添加实例集目的地，配置游戏会话放置流程，并定义放置优先级。请参阅创建游戏会话队列以帮助您创建和配置新队列。
4. 更新您的游戏客户端服务以使用队列。

   当您的游戏客户端使用队列请求资源时，队列会避开中断可能性很高的资源，并选择与您定义的优先级相匹配的位置。要获得在游戏客户端中实施游戏会话放置的帮助，请参阅创建游戏会话。
5. 更新游戏服务器以处理竞价型实例中断。

   AWS 当竞价型实例需要恢复容量时，可以在 2 分钟内发出通知来中断竞价型实例。设置游戏服务器以处理中断，以最大限度地减少对玩家的影响。

   在 AWS 回收竞价型实例之前，它会发送终止通知。Amazon GameLift 通过调用 Amazon Server SDK 回调函数`onProcessTerminate()`将通知传递给所有受影响的 GameLift服务器进程。实现

此回调以结束游戏会话或将游戏会话和玩家移至新实例。请参阅回应服务器进程关闭通知以帮助您实施 onProcessTerminate()。

> **Note**
>
> AWS 会尽一切努力在回收实例之前提供通知，但有可能在警告到来之前 AWS 收回竞价型实例。将游戏服务器准备就绪，以应做好应对意外中断的准备。

6. 评估 竞价型实例集和队列的性能。

   在亚马逊 GameLift 控制台中或通过亚马逊查看亚马逊 GameLift 指标 CloudWatch ，以查看绩效。有关 Amazon GameLift 指标的更多信息，请参阅使用 Amazon CloudWatch 监控 Amazon GameLift。关键指标包括：

   - 中断率 – 使用 InstanceInterruptions 和 GameSessionInterruptions 指标跟踪实例和游戏会话的竞价型实例相关中断的数量和频率。通过 AWS 回收的游戏会话的状态为TERMINATED，状态原因为INTERRUPTED。
   - 队列有效性 – 跟踪放置成功率、平均等待时间和队列深度，以确认竞价型实例集不会影响队列性能。
   - 实例集使用情况 – 监控有关实例、游戏会话和玩家会话的数据。按需型实例集的使用情况可以表明队列为了避免中断而避免放置到竞价型实例集中。

创建新的 Amazon GameLift 实例集

创建新的实例集并部署您的自定义游戏服务器构建或用于托管的实时服务器。您可以部署上传到 Amazon GameLift 的任何游戏构建或脚本资源。

主题

- Amazon GameLift 实例集创建的工作原理
- 创建 Amazon GameLift 托管实例集
- 创建亚马逊 GameLift Anywhere舰队

Amazon GameLift 实例集创建的工作原理

在创建新实例集时，Amazon GameLift 会启动一个工作流程，在每个实例集位置都创建带有一个 Amazon Elastic Compute Cloud (Amazon EC2) 实例的实例集。当 Amazon GameLift 完成工作流程的每个步骤时，实例集会发出事件，Amazon GameLift 会更新实例集的状态。可以使用 Amazon

GameLift 控制台或调用 Amazon GameLift API 操作 DescribeFleetEvents 来跟踪所有事件。您还可以使用 DescribeFleetLocationAttributes 跟踪各个位置的状态。

EC2 实例集 创建工作流程：

- Amazon GameLift 在实例集所在区域和实例集中定义的每个远程位置创建实例集资源。
- Amazon GameLift 将所需容量设置为一个实例。
- Amazon GameLift 将实例集和位置状态设置为新。
- Amazon GameLift 开始将事件写入实例集事件日志。
- Amazon GameLift 在每个实例集位置为一个新实例分配请求的计算资源。
- Amazon GameLift 会将游戏服务器文件下载到每个实例，并将实例集状态设置为正在下载。
- Amazon GameLift 会验证每个实例上下载的游戏服务器文件，以验证下载过程中是否出现错误。Amazon GameLift 将实例集状态设置为正在验证。
- Amazon GameLift 在每个实例上构建游戏服务器，并将实例集状态设置为正在构建。
- 按照实例集的运行时配置中的指令，Amazon GameLift 开始在每个实例上启动服务器进程。如果您将实例集配置为每个实例运行多个并发服务器进程，那么 Amazon GameLift 会将启动过程错开几秒钟。当每个流程上线时，它会向 Amazon GameLift 报告就绪情况。Amazon GameLift 将实例集状态设置为正在激活。
- 当服务器进程报告准备就绪时，Amazon GameLift 会将实例集状态和位置状态设置为活动。

Amazon GameLift Anywhere fleet creation

- Amazon GameLift 创建了实例集资源。对于实例集的主区域和实例集中定义的每个自定义位置，Amazon GameLift 会将实例集和位置状态设置为新。
- Amazon GameLift 开始将事件写入实例集事件日志。
- 队列中的一个服务器进程通知 Amazon GameLift 已准备就绪后，Amazon GameLift 会将实例集状态和位置状态设置为活动。当其他实例集位置的服务器进程报告准备就绪时，Amazon GameLift 会将每个实例集位置的状态设置为活动。

有关实例集创建问题的排查，请参阅调试 Amazon GameLift 实例集问题。

创建 Amazon GameLift 托管实例集

使用 Amazon GameLift 控制台或 AWS Command Line Interface (AWS CLI) 创建托管实例集。

在您创建新的托管 EC2 实例集之后，在 Amazon GameLift 部署实例集并安装和启动游戏服务器的过程中，实例集的状态需要经过多个阶段。当实例集进入 ACTIVE 状态时，即表示已准备好托管游戏会话。有关实例集创建问题的帮助，请参阅调试 Amazon GameLift 实例集问题。

Console

创建托管 EC2 实例集

1. 在 Amazon GameLift 控制台的导航窗格中，选择实例集。
2. 在实例集页面上，选择创建实例集。
3. 选择托管 EC2。
4. 在实例集详细信息页面上，执行以下操作：

   a. 对于名称，输入新名称。我们建议将实例集类型（竞价型或按需型）包含在实例集名称中。这使得在查看实例集列表时可以更轻松地识别实例集类型。

   b. 在描述中，提供对实例集的简短描述。

   c. 对于二进制类型，选择构建或脚本来定义 Amazon GameLift 部署到此实例集的游戏服务器类型。

   d. 从已上传的脚本或构建的下拉列表中选择脚本或构建。

5. （可选）在其他详细信息下显示以下内容：

   a. 对于实例角色，请指定一个 IAM 角色，该角色授权游戏构建中的应用程序访问您账户中的其他 AWS 资源。有关更多信息，请参阅与您的实例集中的其他 AWS 资源进行通信。要创建具有实例角色的实例集，您的账户必须拥有 IAM PassRole 权限。有关更多信息，请参阅Amazon GameLift 的 IAM 权限示例。

   如果您要授权不是服务器可执行文件的应用程序，例如 CloudWatch 代理，请启用共享凭证选项。

   无法在创建实例集后更新这些设置。

   b. 要生成认证，请选择让 Amazon GameLift 为实例集生成 TLS 证书。您可以使用实例集 TLS 证书在连接时让游戏客户端对游戏服务器进行身份验证，并加密所有客户端/服务器通信。对于启用 TLS 的实例集中的每个实例，Amazon GameLift 还使用证书创建一个新的 DNS 条目。使用这些资源为您的游戏设置身份验证和加密。

   c. 对于指标组，输入新的或现有实例集指标组的名称。可以通过将多个实例集添加到同一个指标组来聚合指标。

实例集创建后，您无法更新指标组。

6. 选择下一步。
7. 在选择位置页面上，选择一个或多个其他远程位置来部署实例。系统会根据您访问控制台的区域自动选择主区域。如果您选择其他位置，则实例集实例也将部署在这些位置。

> **Important**
>
> 要使用默认情况下未启用的区域，请在您的 AWS 账户中将其启用。
>
> - 您在 2022 年 2 月 28 日之前创建的具有未启用区域的实例集不受影响。
> - 要创建新的多位置实例集或更新现有的多位置实例集，请先启用您选择使用的任意区域。
>
> 有关默认情况下未启用的区域以及如何启用这些区域的更多信息，请参阅《AWS 一般参考》中的管理 AWS 区域。

8. 选择下一步。
9. 在定义实例详细信息页面上，选择

   a. 此实例集的按需型或竞价型实例。有关实例集类型的更多信息，请参阅按需型实例和竞价型实例。

   b. 从筛选架构菜单中选择 x64 或 Arm。

   > **Note**
   >
   > Graviton Arm 实例需要基于 Linux 操作系统的 Amazon GameLift 服务器构建。C++ 和 C# 需要服务器软件开发工具包 5.1.1 或更高版本。Go 需要服务器软件开发工具包 5.0 或更高版本。这些实例不为 Amazon Linux 2023 (AL2023) 或 Amazon Linux 2 (AL2) 上安装 Mono 提供开箱即用的支持。

   有关 Amazon EC2 Arm 架构的信息，请参阅 AWS Graviton 处理器和 Amazon EC2 实例类型。

有关 Amazon GameLift 支持的实例类型的信息，请参阅 CreateFleet() 请求参数下的 `EC2InstanceType` 值。

10. 从列表中选择 Amazon EC2 实例类型 有关选择实例类型的更多信息，请参阅实例类型。创建实例集后无法更改实例类型。
11. 选择下一步。
12. 在配置运行时系统页面的运行时配置下，执行以下操作：

    a. 对于启动路径，键入您的构建或脚本中游戏可执行文件的路径。在 Windows 实例上，游戏服务器构建到路径 `C:\game`。在 Linux 实例上，游戏服务器的构建到 `/local/game`。示例：**`C:\game\MyGame\server.exe`**、**`/local/game/MyGame/server.exe`** 或 **`MyRealtimeLaunchScript.js`**。

    b. （可选）对于启动参数，输入要作为一组命令行参数传递给游戏可执行文件的信息。示例：**`+sv_port 33435 +start_lobby`**。

    c. 对于并发进程，请选择要在实例集中的每个实例上同时运行的服务器进程数。查看 Amazon GameLift 对并发服务器进程数量的限制。

    对每个实例的并发服务器进程数的限制应用到所有配置的并发进程总数上。如果配置实例集超出限制，实例集无法激活。
13. 在游戏会话激活下，提供在此实例集中的实例上激活新游戏会话的限制：

    a. 对于最大并发游戏会话激活，输入实例中同时激活的游戏会话的数量。当启动多个新的游戏会话可能会对在实例上运行的其他游戏会话造成性能影响时，此限制非常有用。

    b. 对于新激活超时，输入等待会话激活的时长。如果游戏会话在超时之前没有变为 `ACTIVE` 状态，Amazon GameLift 将终止游戏会话激活。
14. （可选）在EC2 端口设置下，执行以下操作：

    a. 选择添加端口设置以定义连接到部署在此实例集上的服务器进程的入站流量的访问权限。

    b. 对于类型，选择自定义 TCP 或自定义 UDP。

    c. 对于端口范围，输入允许入站连接的端口号范围。端口范围必须使用格式 `nnnnn[-nnnnn]`，值介于 1025 和 60000 之间。示例：**`1500`** 或 **`1500-20000`**。

    d. 对于 IP 地址范围，输入 IP 地址的范围。使用 CIDR 表示法。示例：**`0.0.0.0/0`**（此示例允许尝试连接的任何人的访问。）
15. （可选）在游戏会话资源设置下，执行以下操作：

    a. 对于游戏扩展保护策略，请打开或关闭扩展保护。即使具有保护的实例托管了活动游戏会话，在缩减事件中 Amazon GameLift 将不会终止该实例。

    b. 对于资源创建限制，输入策略期限内玩家可以创建的最大游戏会话数。

16. 选择下一步。
17. （可选）通过输入键和值对向构建添加标签。选择下一步继续查看实例集创建。
18. 选择创建。Amazon GameLift 会分配一个 ID 至新实例集并开始实例集激活过程。您可以在实例集页面上跟踪新实例集的状态。

您可以随时更新实例集的元数据和配置，无论实例集状态如何。有关更多信息，请参阅管理 Amazon GameLift 实例集。您能在实例集进入 ACTIVE 状态之后更新实例集容量。有关更多信息，请参阅扩展 Amazon GameLift 托管容量。您还可以添加或删除远程位置。

AWS CLI

要使用 AWS CLI 创建实例集，请打开命令行窗口，然后使用 `create-fleet` 命令。有关 `create-fleet` 命令的更多信息，请参阅《AWS CLI 命令参考》中的 `create-fleet`。

下面显示的示例 `create-fleet` 请求将创建一个具有以下特征的新实例集：

- 该实例集将 c5.large 按需型实例与适合所选游戏构建的操作系统一起使用。
- 它部署指定的游戏服务器构建，以下位置必须处于准备就绪状态：
  - us-west-2（主区域）
  - sa-east-1（远程位置）
- 已启用 TLS 证书生成。
- 该实例集中的每个实例将并发运行十个相同的游戏服务器进程，从而使每个实例能够同时托管多达 10 个游戏会话。
- 在每个实例上，Amazon GameLift 允许同时激活两个新的游戏会话。它还会终止任何未准备好在 300 秒内托管玩家的正在激活的游戏会话。
- 在该实例集中的实例上托管的所有游戏会话将会开启游戏会话保护。
- 单个玩家可以在 15 分钟内创建三个新的游戏会话。
- 在该实例集上托管的每个游戏会话都有一个位于指定的 IP 地址和端口范围内的连接点。
- Amazon GameLift 将该实例集的指标添加到 `EMEAfleets` 指标组中，在此示例中，该指标组合并了 EMEA 区域中的所有实例集的指标。

```
aws gamelift create-fleet \
    --name SampleFleet123 \
    --description "The sample test fleet" \
    --ec2-instance-type c5.large \
    --region us-west-2 \
```

```
    --locations "Location=sa-east-1" \
    --fleet-type ON_DEMAND \
    --build-id build-92f061ed-27c9-4a02-b1f4-6f85b2385620 \
    --certificate-configuration "CertificateType=GENERATED" \
    --runtime-configuration "GameSessionActivationTimeoutSeconds=300,
 MaxConcurrentGameSessionActivations=2, ServerProcesses=[{LaunchPath=C:\game
\Bin64.dedicated\MultiplayerSampleProjectLauncher_Server.exe, Parameters=+sv_port
 33435 +start_lobby, ConcurrentExecutions=10}]" \
    --new-game-session-protection-policy "FullProtection" \
    --resource-creation-limit-policy "NewGameSessionsPerCreator=3,
 PolicyPeriodInMinutes=15" \
    --ec2-inbound-permissions
 "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"
 "FromPort=33235,ToPort=33235,IpRange=0.0.0.0/0,Protocol=UDP" \
    --metric-groups  "EMEAfleets"
```

如果创建实例集请求成功，Amazon GameLift 将返回一组实例集属性，其中包含您请求的配置设置和新的实例集 ID。然后，Amazon GameLift 启动实例集激活流程，并将实例集状态和位置状态设置为新。您可以使用以下 CLI 命令跟踪实例集的状态并查看其他实例集信息：

- 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

您可以使用以下命令，根据需要更改实例集的容量和其他配置设置：

- update-fleet-attributes
- update-fleet-capacity
- update-fleet-port-settings
- update-runtime-configuration
- create-fleet-locations

- delete-fleet-locations

创建亚马逊 GameLift Anywhere舰队

使用亚马逊 GameLift 将您环境中的硬件集成到您的亚马逊 GameLift 游戏托管中。亚马逊将您的硬件 GameLift Anywhere注册到亚马逊机Anywhere队 GameLift 中。您可以在对战构建器和游戏会话队列中集成 Anywhere 和托管 EC2 实例集，以管理对战和游戏放置。

有关使用 Amazon 测试游戏服务器的更多信息 GameLift Anywhere，请参阅使用 Amazon GameLift Anywhere 实例集测试您的集成。

首先，请使用Amazon 为开发提供支持 GameLift版本 5 或更高版本，并查看以下使用 Amazon GameLift Anywhere 队列的概念。

自定义位置

Amazon GameLift Anywhere 队列使用自定义位置来表示您的基础设施的物理位置。

设备注册

要让 Amazon GameLift Anywhere 队列与您的计算资源通信，请先注册您的设备。您可以使用`RegisterCompute`操作从 Amazon GameLift AWS SDK 完成设备注册。此操作使用设备的 IP 地址将其与车队位置关联并与 Amazon 通信 GameLift。

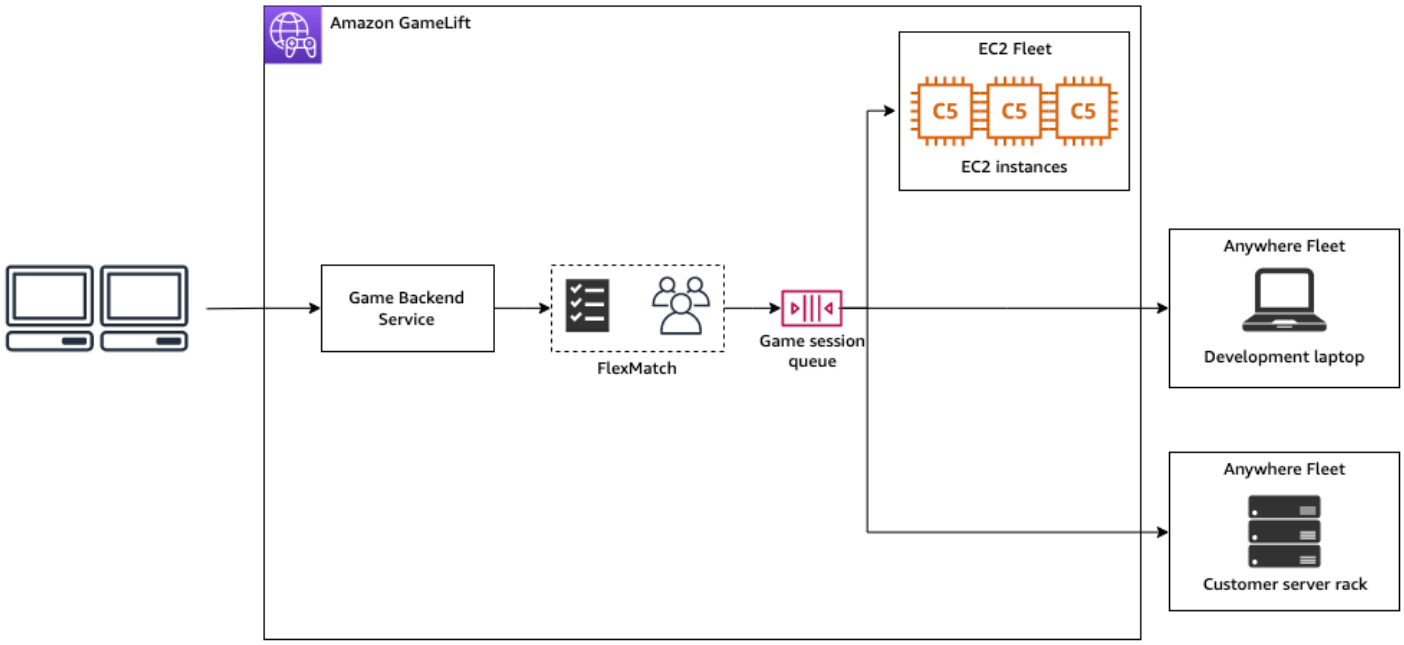
身份验证令牌

当您在计算机上初始化游戏服务器时，Amazon S GameLift erver SDK 会使用身份验证令牌向亚马逊 GameLift验证您的游戏服务器。在身份验证令牌到期时间之前，您可以在同一台计算机上为所有游戏服务器重复使用相同的身份验证令牌。要检索身份验证令牌，请调用 `get-compute-auth-token` AWS Command Line Interface (AWS CLI) 命令。根据需要将令牌传递给每个游戏服务器。

游戏会话

计算机上的每个游戏会话都使用在将计算机注册到实例集位置时创建的相同身份验证令牌。

下图显示了使用 FlexMatch 配对和多个舰队的游戏会话队列。这些队列包括带有 C5 实例的 EC2 实例集、带有开发笔记本电脑的 Anywhere 实例集和带有客户托管服务器机架的 Anywhere 实例集。



主题

- 创建自定义位置
- 创建实例集
- 注册您的计算
- 运行服务器进程
- 创建游戏会话
- 迁移到托管 EC2

创建自定义位置

要开始使用您的计算资源托管游戏，请创建一个描述您的计算所在位置的自定义位置。

Console

创建自定义位置

1. 打开亚马逊 GameLift 控制台。
2. 在导航窗格的托管下，选择位置。
3. 在位置页面上，选择创建位置。
4. 在创建位置对话框中，执行以下操作：

a. 输入位置名称。这会标记 Amazon GameLift 用于在Anywhere舰队中运行游戏的硬件的位置。Ama GameLift zon 会在您的自定义位置名称后面加上 cu stom-。

b. （可选）以键值对的形式向自定义位置添加标签。为每个要添加的标签选择添加新标签。

c. 选择创建。

AWS CLI

使用 `create-location` 命令创建自定义位置。这些`location-name`标签标明了亚马逊GameLift用于在Anywhere舰队中运行游戏的硬件的位置。创建自定义位置时，位置名称必须以`custom-` 开头。

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

输出

```
{
    "Location": {
        "LocationName": "custom-location-1",
        "LocationArn": "arn:aws:gamelift:us-east-1:111122223333:location/custom-
location-1"
    }
}
```

创建实例集

使用 Amazon GameLift 控制台或创建Anywhere队列。 AWS CLI

创建新 Anywhere 实例集后，实例集的状态将从 `NEW` 变为 `ACTIVE`。当实例集进入 `ACTIVE` 状态时，即表示已准备好托管游戏会话。有关实例集创建问题的帮助，请参阅调试 Amazon GameLift 实例集问题。

Console

创建 Anywhere 实例集

1. 打开亚马逊 GameLift 控制台。
2. 在导航窗格中，选择托管下的实例集。

3. 在实例集页面上，选择创建实例集。
4. 在计算类型步骤中，选择 Anywhere，然后选择下一步。
5. 在实例集详细信息步骤中，定义详细信息，然后选择下一步。
6. 在自定义位置步骤中，选择您创建的自定义位置，然后选择下一步。Amazon GameLift 会自动选择房屋 AWS 区域 作为您创建车队的区域。您可以使用主区域来访问和使用您的资源。
7. 完成剩余的实例集创建步骤，然后选择提交以创建您的 Anywhere 实例集。

AWS CLI

使用 `create-fleet` 命令创建 Anywhere 实例集。在 `locations` 中加入您的自定义位置。Amazon 在您的家乡地区和您提供的自定义地点 GameLift 创建车队。在以下示例中，将 *`FleetName`* 和 *`custom-location-1`* 替换为您自己的信息。变量 `custom-location-1` 是在 创建自定义位置 步骤中创建的位置的名称。

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

示例输出

```
{
    "FleetAttributes": {
        "FleetId": "fleet-cebb4da2-52a8-4c27-9b85-587f945c6445",
        "FleetArn": "arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445",
        "Name": "HardwareAnywhere",
        "CreationTime": "2023-02-23T17:57:42.293000+00:00",
        "Status": "ACTIVE",
        "MetricGroups": [
            "default"
        ],
        "CertificateConfiguration": {
            "CertificateType": "DISABLED"
        },
        "ComputeType": "ANYWHERE"
    }
}
```

注册您的计算

要在您创建的实例集中注册您的计算资源，请使用 register-compute 命令。将 *fleet-id* 替换为上一步中 fleet-id 返回的或控制台中实例集详情页面中找到的实例集 ARN。用计算资源的 IP 地址替换 *compute-name* 和 *ip-address*。

> **Note**
>
> 我们建议从register-compute与游戏服务器分开的脚本或进程管理器中同时调用和get-compute-auth-token命令。

```
aws gamelift register-compute \
    --compute-name HardwareAnywhere \
    --fleet-id arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445 \
    --ip-address 10.1.2.3 \
    --location custom-location-1
```

示例输出

```
{
    "Compute": {
        "FleetId": "fleet-cebb4da2-52a8-4c27-9b85-587f945c6445",
        "FleetArn": "arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445",
        "ComputeName": "HardwareAnywhere",
        "ComputeArn": "arn:aws:gamelift:us-east-1:111122223333:compute/
HardwareAnywhere",
        "IpAddress": "10.1.2.3",
        "ComputeStatus": "Active",
        "Location": "custom-location-1",
        "CreationTime": "2023-02-23T18:09:26.727000+00:00",
        "GameLiftServiceSdkEndpoint": "wss://us-east-1.api.amazongamelift.com"
    }
}
```

运行服务器进程

1. 从您创建的实例集中获取计算资源的身份验证令牌。

您的游戏服务器使用身份验证令牌向 Amazon 进行身份验证 GameLift。每个身份验证令牌都有过期时间。要继续使用计算资源托管游戏服务器，请在到期前检索新的身份验证令牌。

> **Note**
>
> Amazon GameLift 建议从`register-compute`与游戏服务器分开的脚本或进程管理器中同时调用和`get-compute-auth-token`命令。

在以下示例中，将 `fleet-id` 替换为在前面步骤中创建的实例集的 ARN 或实例集 ID。将 `compute-name` 替换为您在上一步中使用 `register-compute` 命令创建的计算的名称。

```
aws gamelift get-compute-auth-token \
    --fleet-id arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445 \
    --compute-name HardwareAnywhere
```

输出示例：

```
{
    "FleetId": "fleet-cebb4da2-52a8-4c27-9b85-587f945c6445",
    "FleetArn": "arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445",
    "ComputeName": "HardwareAnywhere",
    "ComputeArn": "arn:aws:gamelift:us-east-1:111122223333:compute/
HardwareAnywhere",
    "AuthToken": "0c728041-3e84-4aaa-b927-a0fb202684c0",
    "ExpirationTimestamp": "2023-02-23T18:47:54+00:00"
}
```

2. 运行游戏服务器可执行文件的实例。

要运行游戏服务器，请通过调用 `InitSDK()` 并传递服务器参数来初始化游戏服务器。有关更多信息，请参阅 服务器参数。

服务器软件开发工具包输入：

```
//Define the server parameters
ServerParameters serverParameters = new ServerParameters(
    webSocketUrl=wss://us-east-1.api.amazongamelift.com,
```

```
    processId=PID1234,
    hostId=HardwareAnywhere,
    fleetId=arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445,
    authToken=0c728041-3e84-4aaa-b927-a0fb202684c0);

//InitSDK establishes a connection with GameLift's websocket server for
 communication.
var initSDKOutcome = GameLiftServerAPI.InitSDK(serverParameters);
```

3. 服务器进程准备好托管游戏会话后，请ProcessReady()从您的游戏服务器致电 Amazon GameLift。有关这些参数的更多信息，请参阅ProcessParameters

```
// Set parameters and call ProcessReady
var processParams = new ProcessParameters(
   this.OnStartGameSession,
   this.OnProcessTerminate,
   this.OnHealthCheck,
   this.OnUpdateGameSession,
   port=1024,
   new LogParameters(new List<string>()          // Examples of log and error files
 written by the game server
   {
      "C:\\game\\logs",
      "C:\\game\\error"
   })
);

var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);
```

创建游戏会话

1. 向游戏服务器添加逻辑，以便服务器进程使用 ActivateGameSession() 响应 onStartGameSession() 消息。此操作没有参数，但它会向 Amazon 发送确认消息 GameLift ，表明您的服务器已收到并接受创建游戏会话消息。

```
void OnStartGameSession(GameSession gameSession)
{
    // game-specific tasks when starting a new game session, such as loading map

    // When ready to receive players
```

```
    var activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}
```

2. 在游戏客户端后端服务中，使用 start-matchmaking、start-game-session-placement 或 create-game-session 命令启动游戏会话。

```
aws gamelift create-game-session \
    --fleet-id arn:aws:gamelift:us-east-1:682428703967:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445 \
    --name GameSession1 \
    --maximum-player-session-count 2 \
    --location custom-location-1
```

输出示例：

```
GameSession {
  FleetId = arn:aws:gamelift:us-east-1:682428703967:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445,
  GameSessionId = 4444-4444,
  Name = GameSession1,
  Location = custom-location-1,
  IpAddress = 10.2.3.4,
  Port = 1024,
  ...
}
```

Amazon GameLift 会向您注册的服务器进程发送onStartGameSession()一条消息。该消息包含上一步中的 GameSession 对象，其中包含游戏属性、游戏会话数据、对战构建器数据以及有关游戏会话的更多信息。

3. 游戏会话完成后，结束游戏服务器进程。

服务器软件开发工具包输入：

```
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();
if (processReadyOutcome.Success)
   Environment.Exit(0);
// otherwise, exit with error code
Environment.Exit(errorCode);
```

4. 通过调用 ProcessReady(processParams) 启动另一个游戏服务器进程。

迁移到托管 EC2

在您开发完游戏服务器并准备好准备制作之后，您可以让 Amazon GameLift 管理您的硬件。要迁移到托管 EC2 队列，请将您的版本上传到 Amazon GameLift 并创建托管 EC2 队列。有关上传构建和设置实例集的更多信息，请参阅将自定义服务器构建上传到 Amazon GameLift和创建 Amazon GameLift 托管实例集。

管理 Amazon GameLift 实例集

使用 Amazon GameLift 控制台或 AWS CLI 更新您的实例集设置、更改远程位置或删除实例集。

更新实例集配置

您可以使用 Amazon GameLift 控制台或 AWS CLI 更新可变实例集属性、端口设置和运行时配置。要更改扩展限制，请参阅使用 Amazon GameLift 自动扩缩实例集容量。

Amazon GameLift console

1. 在 Amazon GameLift 控制台的导航窗格中，选择实例集。
2. 选择要更新的实例集。实例集必须处于 ACTIVE 状态才能进行编辑。
3. 在实例集详情页面的以下任何部分中，选择编辑。
   - 实例集设置
     - 更改实例集属性，如名称和说明。
     - 添加或删除指标组，Amazon CloudWatch 使用这些指标组来跟踪多个实例集的 Amazon GameLift 汇总指标。
     - 更新资源创建限制设置。
     - 打开或关闭游戏会话保护。
   - 运行时配置 – 您可以更改运行时配置的以下任何设置，也可以添加或删除运行时配置。
     - 更改游戏服务器的启动路径。
     - 添加、移除或更改可选的启动参数。
     - 更改游戏服务器运行的并发进程数。
   - 游戏会话激活 – 通过更新最大并发游戏会话激活次数和新激活超时，更改服务器进程的运行方式和托管游戏会话的方式。
   - EC2 端口设置 – 更新允许入站访问实例集的 IP 地址和端口范围。
4. 选择确认以保存所做的更改。

AWS CLI

可使用以下 AWS CLI 命令来更新实例集：

- update-fleet-attributes
- update-fleet-port-settings
- update-runtime-configuration

更新实例集位置

您可以使用 Amazon GameLift 控制台或 AWS CLI 添加或移除实例集的远程位置。您无法更改实例集的主区域。

Amazon GameLift console

1. 在 Amazon GameLift 控制台的导航窗格中，选择实例集。
2. 选择要更新的实例集。实例集必须处于 ACTIVE 状态才能进行编辑。
3. 在实例集详情页面上，选择位置选项卡以查看实例集的位置。
4. 要添加新的远程位置，请选择添加并选择要将实例部署到的位置。此列表不包括实例集实例类型不可用的实例。
5. 选择新位置后，选择添加。Amazon GameLift 将新位置添加到列表中，状态设置为 NEW。然后，Amazon GameLift 开始在每个添加的位置配置一个实例，并使其准备就绪来托管游戏会话。
6. 要从实例集中移除现有的远程位置，请使用复选框选择一个或多个列出的位置。
7. 选择一个或多个实例集后，选择移除。已移除的位置仍保留在列表中，状态设置为 DELETING。然后，Amazon GameLift 开始终止已移除位置的活动。如果存在托管游戏会话的活动实例，Amazon GameLift 会使用游戏服务器终止流程正常结束游戏会话、终止游戏服务器和关闭实例。

AWS CLI

可使用以下 AWS CLI 命令来更新实例集位置：

- create-fleet-locations
- delete-fleet-locations

删除实例集

当您不再需要某个实例集时，可以删除它。删除实例集将永久删除所有与游戏会话和玩家会话相关的数据以及所有收集的指标数据。作为替代方案，您可以保留实例集，禁用自动扩缩，并将实例集手动缩减为 0 个实例。

> **Note**
>
> 如果要删除的实例集具有 VPC 对等连接，您首先需要通过调用 CreateVpcPeeringAuthorization 来请求授权。在实例集删除期间，Amazon GameLift 会删除 VPC 对等连接。

您可以使用 Amazon GameLift 控制台或 AWS CLI 工具来删除实例集。

Amazon GameLift console

1. 在 Amazon GameLift 控制台的导航窗格中，选择实例集。
2. 选择要删除的实例集。您只能删除处于 `ACTIVE` 或 `ERROR` 状态的实例集。
3. 选择删除。
4. 在删除实例集对话框中，输入**`delete`**确认删除。
5. 选择删除。

AWS CLI

使用以下 AWS 命令删除实例集：

- delete-fleet

向 Amazon GameLift 舰队添加别名

Amazon GameLift 别名用于抽象车队名称。舰队名称告诉亚马逊 GameLift 在为玩家创建新游戏会话时在哪里搜索可用资源。通过使用别名而不是特定实例集 ID，通过更改别名的目标位置来无缝地将玩家流量从一个实例集切换到另一个实例集。

别名有两种类型的路由策略：

- 简单 – 将玩家流量路由到指定的实例集 ID。您可以随时更新别名的实例集 ID。
- 终端 – 将消息传回客户端。例如，你可以将正在使用 out-of-date 客户端的玩家引导到他们可以升级的地方。

实例集的生命周期是有限的，在游戏的生命周期中切换实例集有几个原因。您无法更新实例集的游戏服务器构建或更改现有实例集上的某些计算资源属性。相反，使用这些更改创建新实例集，然后将玩家切换到新实例集。利用别名，切换实例集对游戏的影响最小，并且对玩家不可见。

别名对于不使用队列的游戏非常实用。在队列中切换实例集是一个简单的事情，即创建一个新的实例集，将其添加到队列，然后删除旧实例集，这些操作全部对玩家不可见。相比之下，不使用队列的游戏客户端在与 Amazon GameLift 服务通信时必须指定要使用哪个队列。在不使用别名的情况下，实例集切换将需要更新您的游戏代码，并且可能需要将更新后的游戏客户端分配给玩家。

更新别名指向的实例集 ID 时，会有长达 2 分钟的过渡期，在此期间，别名上的游戏会话可能会在旧实例集上结束。

创建新别名

您可以使用亚马逊 GameLift 控制台（如此处所述）或使用 AWS CLI 命令 c reate-alias 创建别名。

1. 在 Amazon GameLift 控制台的导航窗格中，选择别名。
2. 在别名页面，选择创建别名。我们建议在您的别名中包含实例集类型。这使得在查看别名列表时可以更轻松地识别实例集类型。
3. 在创建别名页面上的别名详细信息下，执行以下操作：
   a. 在名称中，输入别名。
   b. 对于描述，输入简短的描述以进行识别。
   c. 选择简单或终端路由类型。
4. （可选）在标签下，通过输入键和值对向别名添加标签。
5. 选择创建。

编辑别名

您可以使用亚马逊 GameLift 控制台或 AWS CLI 命令 update-alia s 编辑别名。

1. 在 Amazon GameLift 控制台的导航窗格中，选择别名。
2. 在别名页面上，选择要编辑的别名。

3. 在别名页面上，选择编辑。
4. 在 Edit alias 页面上，可以编辑以下内容：
   - 别名 – 别名的易记名称。
   - 描述 – 别名的简短描述。
   - 类型 – 玩家流量的路由策略。选择简单更改关联的实例集，或选择终端编辑终端消息。
5. 选择保存更改。

调试 Amazon GameLift 实例集问题

本主题提供有关 Amazon GameLift 托管解决方案的实例集配置问题的指导。对于其他故障排除，您可以在实例集处于活动状态后远程访问实例集实例。请参阅远程连接到 Amazon GameLift 舰队实例。

实例集创建问题

创建实例集后，Amazon GameLift 服务会启动一个工作流程，在实例集的每个位置部署一个新实例，并为运行游戏服务器做好准备。有关详细说明，请参阅Amazon GameLift 实例集创建的工作原理。实例集在进入活动状态之前无法托管游戏会话和玩家。本节讨论阻碍实例集进入活动状态的最常见问题。

下载和验证

在此阶段，如果提取构建文件存在问题，安装脚本将不会运行，或者如果运行时配置中指定的可执行文件未包含在构建文件中，实例集创建可能失败。Amazon GameLift 提供与每个问题相关的日志。

如果日志未显示问题，则可能问题是由内部服务错误造成的。在这种情况下，请再次尝试创建实例集。如果问题仍然存在，请考虑重新上传游戏构建（如果文件已损坏）。您还可以联系 Amazon GameLift 支持或在论坛上发布问题。

构建

导致生成阶段故障的问题几乎肯定是由于游戏构建文件和/或安装脚本的问题。验证上传到 Amazon GameLift 时，游戏构建文件是否可以安装在运行适当操作系统的计算机上。确保使用干净的操作系统安装，而不是现有的开发环境。

激活

激活阶段出现的最常见实例集创建问题。此阶段测试大量元素，包括游戏服务器的可行性、运行时配置设置以及游戏服务器使用服务器软件开发工具包与 Amazon GameLift 服务进行交互的能力。实例集激活过程中遇到的常见问题包括：

服务器进程无法启动。

首先检查您是否在实例集的运行时配置中正确设置了启动路径和可选启动参数。您可以使用实例集详细信息页面、详细信息部分或者调用 AWS CLI 命令 describe-runtime-configuration 来查看实例集的当前运行时配置。如果运行时配置正确，请检查游戏构建文件和/或安装脚本是否存在问题。

服务器进程启动但实例集无法激活。

如果服务器进程启动并成功运行，但实例集未变为活动状态，可能的原因是服务器进程无法通知 Amazon GameLift 已准备好托管游戏会话。检查您的游戏服务器是否正确调用了服务器 API 操作 `ProcessReady()` (请参阅初始化服务器进程)。

VPC 对等连接请求失败。

对于使用 VPC 对等连接创建的实例集（请参阅使用新实例集设置 VPC 对等连接），VPC 对等连接在此激活阶段中完成。如果 VPC 对等连接由于任何原因失败，新实例集将无法转入激活状态。通过调用 describe-vpc-peering-connections，您可以跟踪对等请求的成功或失败。请务必检查存在有效的 VPC 对等连接授权 (describe-vpc-peering-authorizations，因为授权仅在 24 小时内有效。

服务器进程问题

服务器进程启动但快速失败或者报告运行状况不佳。

不同于游戏构建中的问题，同时在实例上尝试运行了过多服务器进程时可能会发生这种情况。并发进程的最佳数目取决于实例类型和您的游戏服务器的资源要求。请尝试减少并发进程数量，该值在实例集的运行时配置中设置，以查看性能是否有所改进。您可以使用 Amazon GameLift 控制台（编辑实例集的容量分配设置）或者调用 AWS CLI 命令 update-runtime-configuration 来更改实例集的运行时配置。

实例集删除问题

由于最大实例计数而无法终止实例集。

错误消息指示正在删除的实例集仍有活动的实例，这种情况是不允许的。您必须首先将实例集缩减到零个活动实例。要执行此操作，可以手动将实例集所需的实例计数设置为“0”，然后等待缩减生效。请务必关闭自动扩展，否则会抵消手动设置。

VPC 操作未获授权。

此问题仅适用于您专门为其创建 VPC 对等连接的实例集（请参阅Amazon GameLift 的 VPC 对等连接）。之所以出现这种情况，是因为删除实例集的过程还包括删除实例集的 VPC 和所有 VPC 对

等连接。必须首先通过调用 Amazon GameLift 服务 API CreateVpcPeeringAuthorization() 或使用 AWSCLI 命令`create-vpc-peering-authorization`来获得授权。获得授权之后，您就可以删除该实例集。

实时服务器实例集问题

僵尸游戏会话：这些会话启动和运行游戏，但永不结束。

您可能会观察到此问题在以下任意场景中出现：

- 实例集的实时服务器未选取脚本更新。
- 实例集快速达到最大容量，但在玩家活动（例如新游戏会话请求）减少时不缩减。

这几乎可以肯定是无法在您的实时脚本中成功调用 `processEnding` 的结果。虽然实例集进入活动状态并且启动了游戏会话，但没有方法可以停止它们。因此，运行游戏会话的实时服务器永远不会释放资源来启动新会话，而新游戏会话只能在新实时服务器启动时启动。此外，对实时脚本的更新不影响已经运行的游戏会话，仅影响新的会话。

为了防止出现这种情况，脚本需要提供一种机制来触发 `processEnding` 调用。如 实时服务器脚本示例中所示，一种方法是编写空闲会话超时，如果在特定时间长度内没有玩家连接，则脚本将结束当前游戏会话。

但是，如果您出现了这种情况，还有多种解决方法可以让实时服务器摆脱卡顿的状况。诀窍是触发实时服务器进程（或底层实例集实例）重启。在这种情况下，GameLift 会自动关闭您的游戏会话。一旦释放实时服务器，它们就可以使用实时脚本的最新版本来启动新游戏会话。

根据问题的普遍性，有几种方法可以做到这一点：

- 缩减整个实例集。此方法执行起来最简单，但具有扩散效应。将实例集缩减为零个实例，等待实例集完全缩减，然后将其扩展回。这将清除所有现有游戏会话，并让您使用最近更新的实时脚本来从头开始。
- 远程访问该实例并重新启动该过程。如果您只有几个进程需要修复，这是一个很好的选项。如果您已登录到该实例，例如用于跟踪日志或调试，则这可能是最快的方法。请参阅远程连接到 Amazon GameLift 舰队实例。

如果您选择不在实时脚本中包含调用 `processEnding` 的方法，则可能会出现一些棘手的情况，即使实例集进入活动状态并且游戏会话已启动。首先，正在运行的游戏会话不结束。因此，游戏会话永远不会释放正在运行的服务器进程来启动新游戏会话。其次，实时服务器不会选取任何脚本更新。

远程连接到 Amazon GameLift 舰队实例

您可以连接到处于活动状态的 Amazon GameLift 托管 EC2 队列中的任何实例。访问实例的常见原因包括：

- 对游戏服务器集成问题进行故障排除
- 微调您的运行时配置和其他特定于舰队的设置
- 获取实时游戏服务器活动，例如日志跟踪。
- 使用实际玩家流量运行基准测试工具。
- 调查游戏会话或服务器进程的具体问题。

连接实例时，请考虑以下潜在问题：

- 您可以连接到活跃队列中的实例。非活动舰队，即处于激活状态或处于错误状态的舰队，可能在短时间内可以访问。有关实例集激活问题的帮助，请参阅调试 Amazon GameLift 实例集问题。
- 连接到活动实例不会影响该实例的托管活动。该实例继续根据运行时配置启动和停止服务器进程。它激活并主持游戏会话。它可能会因缩小规模事件或其他事件而关闭。
- 您对实例上的文件或设置所做的任何更改都可能影响该实例的活动游戏会话和连接的玩家。

以下说明描述了如何使用AWS命令行界面 (CLI) 远程连接到实例。您还可以使用AWS软件开发工具包进行编程调用，如亚马逊 GameLift 服务 API 参考中所述。

收集实例数据

收集以下信息：

- 您要连接的实例的 ID。您可以使用实例 ID 或 ARN。
- 实例上使用的 Amazon GameLift 服务器 SDK 版本。服务器 SDK 与在实例上运行的游戏版本集成。

检索实例数据

以下步骤假设您拥有要连接的实例的托管 EC2 队列 ID。

1. 获取计算名称。

调用托管 EC2 队列的 list-comp ute 以获取队列中所有活跃计算的列表。对于单一地点的舰队，请指定舰队 ID 或 ARN。对于多地点舰队，请指定舰队 ID 或 ARN 以及地点。对于托管 EC2 队列，计算是 EC2 实例，返回的属性ComputeName是实例 ID。例如：

请求

```
aws gamelift list-compute \
  --fleet-id  "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa" \
  --location ""sa-east-1"
```

响应

```
{
  "ComputeList": [
    {
      "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
      "FleetArn": "arn:aws:gamelift:us-west-2::fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
      "ComputeName": "i-0abc12d3e45fa6b78",
      "IpAddress": "00.00.000.00",
      "DnsName":
 "b08444ki909kvqu6zpw3is24x5pyz4b6m05i3jbxvpk9craztu0lqrbbrbnbkks.uwp57060n1k6dnlnw49b78hg1
west-2.amazongamelift.com",
      "ComputeStatus": "Active",
      "Location": "sa-east-1",
      "CreationTime": "2023-07-09T22:51:45.931000-07:00",
      "OperatingSystem": "AMAZON_LINUX",
      "Type": "c4.large"
    }
  ]
}
```

2. 查找服务器 SDK 版本。

   服务器 SDK 版本是编译资源的一个属性。

   a. describe-fleet-attributes使用舰队 ID 致电以获取舰队的构建 ID 和 ARN。

   b. 使用构建 ID 或 ARN 调用 describe-b uild 以获取版本的服务器 SDK 版本。

      例如：

请求

```
aws gamelift describe-fleet-attributes /
  --fleet-ids  "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
```

响应

```
{
  "FleetAttributes": [
    {
      "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
      "ComputeType": "EC2",
      "BuildId": "build-3333cccc-44dd-55ee-66ff-00001111aa22",
      . . .
    }
  ]
}
```

请求

```
aws gamelift describe-build /
  --build-id "build-3333cccc-44dd-55ee-66ff-00001111aa22"
```

响应

```
"Build": {
  "BuildId": "build-1111aaaa-22bb-33cc-44dd-5555eeee66ff",
  "Name": "My_Game_Server_Build_One",
  "OperatingSystem": "AMAZON_LINUX_2",
  "ServerSdkVersion": "5.1.1",
  . . .
}
```

连接到实例（服务器 SDK 5）

如果您要连接的实例正在使用服务器 SDK 版本 5.x 运行游戏版本，请按照以下说明使用 Amazon EC2 Systems Manager (SSM) 连接到该实例。您可以访问在 Windows 或 Linux 上运行的远程实例。

1. 请求实例的访问凭证。如果您有要连接的实例的计算名称和队列 ID，请调用get-compute-access。如果成功，Amazon 将 GameLift 返回一组用于访问该实例的临时证书。例如：

   请求

   ```
   aws gamelift get-compute-access \
   --compute-name i-11111111a222b333c \
   --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa
   --region us-west-2
   ```

   响应

   ```
   {
     "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. 导出访问凭证。您可以选择将凭证导出到环境变量，并使用它们为默认用户配置 AWS CLI。有关更多详细信息，请参阅《AWS Command Line Interface用户指南》中的用于配置 AWS CLI 的环境变量。

   ```
   export AWS_ACCESS_KEY_ID=ASIAIOSFODNN7EXAMPLE
   export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   export AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of session token>
   ```

3. Connect 连接到舰队实例。使用您要连接的实例启动 SSM 会话。包括实例的AWS地区或位置。有关更多信息，请参阅 Amazon EC2 Systems Manager 用户指南中的启动会话 (AWSCLI)。使用您在步骤 1 中获得的凭证。例如：

   ```
   aws ssm start-session \
   --target i-11111111a222b333c \
   --region us-west-2
   ```

连接到实例（服务器 SDK 4.x 或更早版本）

如果您要连接的实例正在使用服务器 SDK 版本 4 或更早版本运行游戏版本，请按照以下说明进行操作。您可以连接到运行 Windows 或 Linux 的实例。使用远程桌面协议 (RDP) 客户端连接到 Windows 实例。使用 SSH 客户端连接到 Linux 实例。

1. 请求实例的访问凭证。如果您有实例 ID，请使用命令请求get-instance-access访问证书。如果成功，Amazon 将 GameLift 返回实例的操作系统、IP 地址和一组证书（用户名和密钥）。凭证格式取决于实例的操作系统。使用以下说明来检索 RDP 或 SSH 的凭证。

   - 对于 Windows 实例 – 要连接到 Windows 实例，RDP 需要用户名和密码。`get-instance-access` 请求会以简单字符串的格式返回这些值，因此您可以原样使用返回的值。凭证示例：

     ```
     "Credentials": {
         "Secret": "aA1bBB2cCCd3EEE",
         "UserName": "gl-user-remote"
     }
     ```

   - 对于 Linux 实例 – 要连接到 Linux 实例，SSH 需要用户名和私有密钥。Amazon GameLift 发放 RSA 私钥并将其作为单个字符串返回，换行符 (\n) 表示换行符。要使私钥可用，请执行以下步骤：(1) 将字符串转换为`.pem`文件，以及 (2) 为新文件设置权限。返回凭证示例：

     ```
     "Credentials": {
         "Secret": "-----BEGIN RSA PRIVATE KEY-----
     nEXAMPLEKEYKCAQEAy7WZhaDsrA1W3mRlQtvhwyORRX8gnxgDAfRt/gx42kWXsT4rXE/b5CpSgie/
     \nvBoU7jLxx92pNHoFnByP+Dc21eyyz6CvjTmWA0JwfWiW5/akH7iO5dSrvC7dQkW2duV5QuUdE0QW
     \nZ/aNxMniGQE6XAgfwlnXVBwrerrQo+ZWQeqiUwwMkuEbLeJFLhMCvYURpUMSC1oehm449ilx9X1F
     \nG50TCFeOzfl8dqqCP6GzbPaIjiU19xX/azOR9V+tpUOzEL+wmXnZt3/nHPQ5xvD2OJH67km6SuPW
     \noPzev/D8V+x4+bHthfSjR9Y7DvQFjfBVwHXigBdtZcU2/wei8D/HYwIDAQABAoIBAGZ1kaEvnrqu
     \n/uler7vgIn5m7lN5LKw4hJLAIW6tUT/fzvtcHK0SkbQCQXuriHmQ2MQyJX/0kn2NfjLV/
     ufGxbL1\nmb5qwMGUnEpJaZD6QSSs3kICLwWUYUiGfc0uiSbmJoap/
     GTLU0W5Mfcv36PaBUNy5p53V6G7hXb2\nbahyWyJNfjLe4M86yd2YK3V2CmK+X/
     BOsShnJ36+hjrXPPWmV3N9zEmCdJjA+K15DYmhm/
     tJWSD9\n81oGk9TopEp7CkIfatEATyyZiVqoRq6k64iuM9JkA3OzdXzMQexXVJ1TLZVEH0E7bhlY9d8O1ozR
     \noQs/FiZNAx2iijCWyv0lpjE73+kCgYEA9mZtyhkHkFDpwrSM1APaL8oNAbbjwEy7Z5Mqfql
     +lIp1\nYkriL0DbLXlvRAH+yHPRit2hHOjtUNZh4Axv+cpg09qbUI3+43eEy24B7G/Uh
     +GTfbjsXsOxQx/x\np9otyVwc7hsQ5TA5PZb
     +mvkJ5OBEKzet9XcKwONBYELGhnEPe7cCgYEA06Vgov6YHleHui9kHuws
     \nayav0elc5zkxjF9nfHFJRry21R1trw2Vdpn+9g481URrpzWVOEihvm+xTtmaZlSp//lkq75XDwnU
     \nWA8gkn6O3QE3fq2yN98BURsAKdJfJ5RL1HvGQvTe10HLYYXpJnEkHv+Unl2ajLivWUt5pbBrKbUC
     \ngYBjbO+OZk0sCcpZ29sbzjYjpIddErySIyRX5gV2uNQwAjLdp9PfN295yQ+BxMBXiIycWVQiw0bH
     \noMo7yykABY7Ozd5wQewBQ4AdSlWSX4nGDtsiFxWiI5sKuAAeOCbTosy1s8w8fxoJ5Tz1sdoxNeGs
     ```

```
\nArq6Wv/G16zQuAE9zK9vvwKBgF+09VI/1wJBirsDGz9whVWfFPrTkJNvJZzYt69qezxlsjgFKshy
\nWBhd4xHZtmCqpBPlAymEjr/TOlbxyARmXMnIOWIAnNXMGB4KGSyl1mzSVAoQ+fqR+cJ3d0dyPl1j
\njjb0Ed/NY8frlNDxAVHE8BSkdsx2f6ELEyBKJSRr9snRAoGAMrTwYneXzvTskF/S5Fyu0iOegLDa
\nNWUH38v/nDCgEpIXD5Hn3qAEcju1IjmbwlvtW+nY2jVhv7UGd8MjwUTNGItdb6nsYqM2asrnF3qS
\nVRkAKKKYeGjkpUfVTrW0YFjXkfcrR/V+QFL5OndHAKJXjW7a4ejJLncTzmZSpYzwApc=\n-----END
 RSA PRIVATE KEY-----",
    "UserName": "gl-user-remote"
}
```

使用 AWS CLI 时，您可以通过在请求中包含 --query和--out put 参数来自动生成.pem文件。get-instance-access

要在 .pem 文件上设置权限，请运行以下命令：

```
$ chmod 400 MyPrivateKey.pem
```

2. 为远程连接打开端口。您可以通过 GameLift 队列配置中授权的任何端口访问 Amazon 队列中的实例。您可以使用命令 describe-fleet-port-settings 查看实例集的端口设置。

   作为最佳实操，我们建议您仅在需要时为远程访问打开这些端口，并在完成后关闭它们。在创建队列之后但在队列处于活动状态之前，您无法更新端口设置。如果您遇到问题，请在打开端口设置的情况下重新创建舰队。

   使用命令 update-fleet-port-settings 为远程连接添加端口设置 (例如 SSH 为 22，RDP 为 3389)。对于 IP 范围值，指定您计划用于连接的设备的 IP 地址 (转换为 CIDR 格式)。例如：

   ```
   $ 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"
   ```

   以下示例在 Windows 实例集上打开端口 3389

   ```
   $ 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. 打开远程连接客户端。为 Windows 实例使用远程桌面，为 Linux 实例使用 SSH。使用 IP 地址连接到实例、端口设置和访问凭证。

SSH 示例：

```
ssh -i MyPrivateKey.pem gl-user-remote@192.0.2.0
```

查看远程实例上的文件

当远程连接到实例时，您具有完整的用户和管理访问权限。这意味着，您也具备导致游戏托管错误和故障的能力。如果该实例托管有活跃玩家的游戏，则可能会导致游戏会话崩溃并丢掉玩家，或者中断游戏关闭过程并导致保存的游戏数据和日志出现错误。

在托管实例上寻找以下资源：

- 游戏构建文件。这些文件是您上传到 Amazon 的游戏版本 GameLift。它们包括一个或多个游戏服务器可执行文件、资产和依赖项。游戏编译文件位于名为game：的根目录中
  - 在 Windows 上: `c:\game`
  - 在 Linux 上: `/local/game`
- 游戏日志文件。无论您指定的任何目录路径，都可以在game根目录中找到游戏服务器生成的日志文件。
- 亚马逊 GameLift 托管资源。根目录Whitewater包含 Amazon GameLift 服务用来管理游戏托管活动的文件。请勿出于任何原因修改这些文件。
- 运行时配置。不要访问单个实例的运行时配置。要更改运行时配置属性，请更新队列的运行时配置（请参阅 AWS SDK 操作UpdateRuntimeConfiguration或 AWS CLI update-runtime-configuration）。
- 实例集数据。JSON 文件包含有关实例所属队列的信息，供实例上运行的服务器进程使用。JSON 文件位于以下位置：
  - 在 Windows 上: `C:\GameMetadata\gamelift-metadata.json`
  - 在 Linux 上: `/local/gamemetadata/gamelift-metadata.json`
- TLS 证书。如果实例位于启用了 TLS 证书生成的队列上，请在以下位置查找证书文件，包括证书、证书链、私钥和根证书：
  - 在 Windows 上: `c:\\GameMetadata\Certificates`
  - 在 Linux 上: `/local/gamemetadata/certificates/`

扩展 Amazon GameLift 托管容量

以实例为单位的托管容量表示 Amazon GameLift 可以同时托管的游戏会话数量以及这些游戏会话可以容纳的并发玩家数量。游戏托管最具挑战性的任务之一是扩展容量以满足玩家的需求，同时又不会将成本浪费在不需要的资源上。有关更多信息，请参阅扩展实例集容量。

容量在实例集位置级别进行调整。所有实例集都至少有一个位置：实例集的主 AWS 区域。查看或扩展容量时，会按位置列出信息，包括实例集主区域和任何其他远程位置。

您可以手动设置要维护的实例数量，也可以设置自动扩缩，以便随着玩家需求的变化动态调整容量。我们建议您刚开始时应启用基于目标的自动扩缩选项。基于目标的自动扩缩旨在保持足够的托管资源来容纳当前玩家，外加一点额外的资源来应对玩家需求的意外激增。对于大多数游戏来说，基于目标的自动扩缩提供了一种非常有效的缩放解决方案。

本节中的主题提供有关下列任务的详细帮助信息：

- 设置容量扩展的最低和最高限制
- 手动设置容量级别
- 使用基于目标的自动扩缩
- 管理基于规则的自动扩缩(高级功能)
- 临时禁用自动扩缩

可以使用 Amazon GameLift 控制台执行大多数实例集扩展活动。也可以将 AWS 软件开发工具包或 AWS Command Line Interface (AWS CLI) 与 Amazon GameLift 服务 API 配合使用。

在控制台中管理实例集容量

1. 打开 Amazon GameLift 控制台。
2. 在导航窗格中，选择托管，实例集。
3. 在实例集页面上，选择活跃实例集的名称以打开该实例集的详情页面。
4. 选择扩展选项卡。在此选项卡上，您可以：
    - 查看整个实例集的历史扩展指标。
    - 查看和更新每个实例集位置的容量设置，包括扩展限制和当前容量设置。
    - 更新基于目标的自动扩缩，查看应用于整个实例集的基于规则的自动扩缩策略，并暂停每个位置的自动扩缩活动。

主题

- 设置 Amazon GameLift 的容量限制
- 手动设置 Amazon GameLift 实例集的容量
- 使用 Amazon GameLift 自动扩缩实例集容量

设置 Amazon GameLift 的容量限制

手动或通过自动扩缩扩展 Amazon GameLift 实例集位置的托管容量时，请考虑该位置的扩展限制。所有实例集位置都有最小和最大限制，用于定义该位置容量的允许范围。默认情况下，实例集位置的限制设置为最少 0 个实例，最多 1 个实例。在缩放实例集位置之前，请先调整限制。

如果您使用的是自动扩缩，则最大限制允许 Amazon GameLift 纵向扩展实例集位置以满足玩家需求，但可以防止托管成本失控，例如在DDOS攻击期间。将 Amazon CloudWatch 警报设置为在容量接近最大限制时发出警报，这样，您就可以评估情况并根据需要手动调整。（您也可以创建账单警报来监控 AWS 成本。）即使玩家需求很低，最低限额也有助于保持托管的可用性。

您可以在 Amazon GameLift 控制台中或使用 AWS Command Line Interface (AWS CLI) 为实例集的位置设置容量限制。

设置容量限制

Console

1. 打开Amazon GameLift 控制台。
2. 在导航窗格中，选择托管，实例集。
3. 在实例集页面上，选择活跃实例集的名称以打开该实例集的详情页面。
4. 在扩展选项卡上的扩展容量下，选择实例集位置，然后选择编辑。
5. 在编辑扩展容量对话框中，为最小大小、所需实例和最大大小设置实例计数。
6. 选择确认。

AWS CLI

1. 检查当前容量设置。在命令行窗口中，使用 describe-fleet-location-capacity 命令以及要更改容量的实例集 ID 和位置。此命令将返回 FleetCapacity 对象，其中包括该位置的当前容量设置。确定新的实例限制是否能适应当前所需的实例设置。

```
aws gamelift describe-fleet-location-capacity \
    --fleet-id <fleet identifier> \
    --location <location name>
```

2. 更新限制设置。在命令行窗口中，使用带有以下参数的 update-fleet-capacity 命令。您可以使用此同一个命令同时调整实例限量和所需的实例计数。

```
--fleet-id <fleet identifier>
--location <location name>
--max-size <maximum capacity for scaling>
--min-size <minimum capacity for scaling>
--desired-instances <fleet capacity goal>
```

示例：

```
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
```

如果请求成功，Amazon GameLift 将返回实例集 ID。如果新的 `max-size` 或 `min-size` 值与当前 `desired-instances` 设置冲突，Amazon GameLift 将返回错误。

手动设置 Amazon GameLift 实例集的容量

当您创建新队列时，Amazon GameLift 会自动将所需实例设置为每个实例集位置的一个实例。然后，Amazon GameLift 在每个位置部署一个新实例。要更改实例集容量，您可以添加基于目标的自动扩缩策略，也可以手动设置某个位置所需的实例数量。有关更多信息，请参阅扩展实例集容量。

当您不需要自动扩缩或需要将容量保持在指定级别时，手动设置实例集的容量会很有用。只有在不使用基于目标的自动扩缩策略时，手动设置容量才有效。如果有基于目标的自动扩缩策略，它将根据自己的扩展规则立即重置所需的容量。

您可以在 Amazon GameLift 控制台中或使用 AWS Command Line Interface (AWS CLI) 手动设置容量。实例集的状态必须为活动状态。

暂停自动扩缩

您可以暂停每个实例集位置的所有自动扩缩活动。暂停自动扩缩后，除非手动更改，否则实例集位置中所需的实例数量将保持不变。当您暂停某个位置的自动扩缩时，它会影响实例集的当前策略以及您将来可能定义的任何策略。

手动设置实例集容量

Console

1. 打开 Amazon GameLift 控制台。
2. 在导航窗格中，选择托管，实例集。
3. 在实例集页面上，选择活跃实例集的名称以打开该实例集的详情页面。
4. 在扩展选项卡上的暂停自动扩缩位置下，选择要暂停自动扩缩的每个位置，然后选择暂停。
5. 在扩展容量下，选择要手动设置的位置，然后选择编辑。
6. 在编辑扩展容量对话框中，设置所需实例的首选值，然后选择确认。此值会告知 Amazon GameLift 保持活动状态且随时可托管游戏会话的实例数量。

   Amazon GameLift 会通过部署其他实例或关闭不需要的实例来应对此类变化。当 Amazon GameLift 完成此过程时，该位置的活动实例数量会发生变化，以匹配更新的所需实例值。此过程可能需要一点时间。

AWS CLI

1. 检查当前容量设置。在命令行窗口中，使用 describe-fleet-location-capacity 命令以及要更改容量的实例集 ID 和位置。此命令将返回 FleetCapacity 对象，其中包括该位置的当前容量设置。确定实例限制是否将适应新的所需的实例设置。

   ```
   aws gamelift describe-fleet-location-capacity \
       --fleet-id <fleet identifier> \
       --location <location name>
   ```

2. 更新所需容量。使用 update-fleet-capacity 命令与所需实例的实例集 ID 和一个新的值。如果此值不在当前限制范围内，则可以在相同的命令中调整限制值。

   ```
   --fleet-id <fleet identifier>
   --location <location name>
   --desired-instances <fleet capacity as an integer>
   --max-size <maximum capacity>    [Optional]
   ```

```
--min-size <minimum capacity>    [Optional]
```

示例：

```
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
```

如果请求成功，Amazon GameLift 将返回实例集 ID。如果新的所需实例设置超出了最小/最大限制，Amazon GameLift 会返回错误。

使用 Amazon GameLift 自动扩缩实例集容量

使用 Amazon GameLift 中的自动扩缩可以动态扩展实例集容量，以响应游戏服务器活动。当玩家到达并开始游戏会话时，自动扩缩可以添加更多实例；当玩家需求消减时，自动扩缩可以终止不需要的实例。自动扩缩是一种可最大程度地降低托管资源和成本的有效方式，同时仍提供流畅、快速的玩家体验。

要使用自动扩缩，您需要创建扩展策略，告诉 Amazon GameLift 何时扩大或缩小规模。有两种类型的扩展策略：基于目标和基于规则。基于目标的方法（目标跟踪）是一个完整的解决方案。我们建议将其作为最简单、最有效的选择。基于规则的扩展策略，它要求您定义自动扩缩决策过程的每个环节，可用于解决特定的问题。它最适合作为对基于目标的自动扩缩的补充。

您可以使用 Amazon GameLift 控制台、AWS Command Line Interface (AWS CLI)或 AWS 软件开发工具包管理基于目标的自动扩缩。仅可使用 AWS CLI 或 AWS 软件开发工具包管理基于规则的自动扩缩，但您可以在控制台中查看基于规则的扩展策略。

主题

- 基于目标的自动扩缩
- 使用基于规则的策略自动扩缩

基于目标的自动扩缩

Amazon GameLift 的基于目标的自动扩缩功能会根据实例集指标 `PercentAvailableGameSessions` 调整容量级别。该指标表示实例集的可用缓冲区应对玩家需求激增的情况。

维护容量缓冲区的主要原因是玩家等待时间。当游戏会话槽准备就绪并等待时，新玩家进入游戏会话需要数秒钟。如果没有资源可用，玩家必须等待现有游戏会话结束或新资源变为可用。启动新实例和服务器进程可能需要数分钟时间。

在设置基于目标的自动扩缩时，需指定您希望实例集维护的缓冲区的大小。由于 `PercentAvailableGameSessions` 衡量的是可用资源的百分比，因此实际缓冲区大小是实例集总容量的百分比。Amazon GameLift 添加或删除实例以保持目标缓冲区大小。如果缓冲区较大，则会最大程度地减少等待时间，但您也要为可能未使用的额外资源付费。如果您的玩家更能容忍等待时间，则可通过设置较小的缓冲区来降低成本。

设置基于目标的自动扩缩

Console

1. 打开 Amazon GameLift 控制台。
2. 在导航窗格中，选择托管，实例集。
3. 在实例集页面上，选择活跃实例集的名称以打开该实例集的详情页面。
4. 选择扩展选项卡。此选项卡显示实例集的历史扩展指标，并包含用于调整当前扩展设置的控件。
5. 在扩展容量下，检查最小大小和最大大小限制是否适合实例集。启用自动扩缩后，容量可能会在这两个限制之间调整。
6. 在基于目标的自动扩缩策略中，选择编辑。
7. 在编辑基于目标的自动扩缩策略对话框中，在可用游戏会话百分比中，设置要保持的百分比，然后选择确认。在您确认设置后，Amazon GameLift 会在基于目标的自动扩缩策略下添加新的基于目标的策略。

AWS CLI

1. 设置容量限制。使用 update-fleet-capacity 命令设置限制值。有关更多信息，请参阅设置 Amazon GameLift 的容量限制。

2. 创建新策略。打开命令行窗口，并使用 put-scaling-policy 命令来设定您的策略的参数设置。要更新现有策略，请指定策略的名称并提供完整版本的更新策略。

```
--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>
```

示例：

```
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"
```

使用基于规则的策略自动扩缩

当自动扩缩实例集的容量以应对玩家活动时，Amazon GameLift 中基于规则的扩展策略提供精细的控制措施。对于每个策略，您可以将扩展与若干可用的实例集指标之一关联、确定触发器点并自定义对应的扩展或缩减事件。基于规则的策略尤其适用于对基于目标的扩展进行补充以应对特殊情况。

基于规则的策略类似于以下语句：“如果实例集指标达到或超过阈值持续特定时间长度，则按指定的数量更改实例集容量。” 本主题介绍用于构建策略语句的语法，并提供帮助以创建和管理基于规则的策略。

管理基于规则的策略

将 AWS 软件开发工具包或 AWS Command Line Interface (AWS CLI) 与 Amazon GameLift 服务 API 结合使用，以创建、更新或删除基于规则的策略。您可以在 Amazon GameLift 控制台中查看所有活动的策略。

要临时禁用实例集的所有扩展策略，请使用 AWS CLI 命令 stop-fleet-actions。

创建或更新基于规则的扩展策略 (AWS CLI)：

1. 设置容量限制。使用 update-fleet-capacity 命令设置其中一个限制值，或同时设置这两个限制值。有关更多信息，请参阅设置 Amazon GameLift 的容量限制。

2. 创建新策略。打开命令行窗口，并使用 put-scaling-policy 命令来设定您的策略的参数设置。要更新现有策略，请指定策略的名称并提供完整版本的更新策略。

```
--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>
```

示例：

```
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
```

使用 AWS CLI 删除基于规则的扩展策略：

- 打开命令行窗口，并使用 delete-scaling-policy 命令以及实例集 ID 和策略名称。

  示例：

```
aws gamelift delete-scaling-policy \
    --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
    --name "Scale up when AGS<50"
```

自动扩缩规则的语法

要构建基于规则的扩展策略语句，需指定六个变量：

如果 <####> 保持为 <#####> <##> 长达 <###>，使用 <#### > 将实例集容量更改为 <###>，或者按后者幅度更改。

例如，只要实例集的额外容量低于处理 50 个新游戏会话所需的容量，此策略语句就会触发扩展事件。

如果 AvailableGameSessions 保持为 less than 50 长达 10 minutes，则使用 ChangeInCapacity 按 1 instances 的幅度更改实例集容量。

指标名称

要触发扩展事件，请将自动扩缩策略与以下实例集特定的指标之一关联。有关更完整的指标说明，请参阅Amazon GameLift 实例集指标。

- 激活游戏会话
- 有效游戏会话
- 可用的游戏会话
- 可用游戏会话所占百分比
- 活动实例
- 可用的玩家会话
- 当前玩家会话
- 空闲实例
- 空闲实例所占百分比

如果实例集包含在游戏会话队列中，则可能使用以下指标：

- 队列深度 – 最适合将此实例集作为可用托管位置的待处理游戏会话请求数。
- 等待时间 – 实例集特定的等待时间。最早的待处理游戏会话请求已等待执行的时间长度。实例集的等待时间等于队列中当前最早请求的时间。

比较运算符

告知 Amazon GameLift 如何将指标数据与阈值进行比较。有效比较运算符包括大于 (>)、小于 (<)、大于或等于 (>=) 以及小于或等于 (<=)。

阈值

当指定的指标值达到或超过阈值时，它将启动扩展事件。此值始终为正整数。

评估期

该指标必须在评估期的完整长度内达到或超过阈值，之后才能启动扩展事件。评估期长度是连续的；如果指标回退到阈值之下，则评估期重新开始。

调整类型和值

这组变量一起使用，用于指定在启动扩展事件时，Amazon GameLift 应该如何调整实例集的容量。从三种可能的调整类型中选择：

- 容量更改 – 按指定的实例数增加或减少当前容量。将调整值设置为要在实例集中增加或删除的实例数。正值表示添加实例，而负值表示删除实例。例如，值为“-10”则将缩减实例集的 10 个实例，而不考虑实例集的总大小。
- 容量更改百分比 – 按指定百分比增加或减少当前容量。将调整值设置为您要增加或减少实例集容量的百分比。正值表示添加实例，而负值表示删除实例。例如，对于一个具有 50 个实例的实例集，“20”的百分比更改向实例集添加 10 个实例。
- 精确容量 – 将当前容量增加或减少到特定值。将调整值设置为您希望在实例集中维护的确切的实例数。

基于规则的自动扩缩的提示

以下建议可以帮助您通过基于规则的策略充分利用自动扩缩。

使用多个策略

您可以同时让多个自动扩缩策略在实例集上生效。最常见的场景是：使用一个基于目标的策略管理大多数扩展需求；而使用基于规则的策略来处理边缘情况。对于使用多个策略没有限制。

在使用多个策略时，将独立运行每个策略。无法控制扩展事件的顺序。例如，如果您有多个策略驱动纵向扩展，那么玩家活动可能会同时触发多个扩展事件。避免使用相互启动的政策。例如，如果您创建了将容量设置为超过各自阈值的扩展和缩减策略，就会创建无限循环。

设置最大和最小容量

每个实例集都具有最大和最小容量限制。此功能在使用自动扩缩时尤为重要。自动扩缩从不将容量设置为此范围之外的值。默认情况下，新创建的实例集具有最小值 0 和最大值 1。要想让您的自动扩缩策略按预期影响容量，需增加最大值。

实例集容量也受到实例集实例类型以及您 AWS 账户上服务限额的约束。您不能设置超出这些限制和账户限额的最小值和最大值。

在容量更改后跟踪指标

在更改了容量以响应自动扩缩策略之后，Amazon GameLift 将等待十分钟，然后再响应该相同策略的触发器。这种等待允许 Amazon GameLift 有时间添加新实例，启动游戏服务器，连接玩家，以及开始

从新实例收集数据。在这段时间内，Amazon GameLift 根据指标评估策略并跟踪策略的评估期，该评估期在扩展事件出现之后重新开始。这意味着，扩展策略可以在等待时间结束后立即启动另一个扩展事件。

不同自动扩缩策略启动的扩展事件之间没有等待时间。

《Amazon GameLift 开发人员指南》中的设置游戏会话置放的 GameLift 队列

游戏会话队列是处理新的游戏会话请求并找到可用的游戏服务器来托管这些请求的主要机制。队列为游戏开发人员和玩家提供了显著的好处。其中包括：

- 队列可提供最佳位置。在处理游戏会话放置请求时，队列使用 Amazon GameLift 算法根据一组已定义的首选项来确定队列位置的优先级。
- 在价格较低的竞价型实例集上托管游戏。使用队列来优化 AWS Spot 队列的使用，从而显著降低托管成本。默认情况下，队列总是尝试在 Spot 队列中放置新的游戏会话。
- 在需求旺盛时更快地发布新游戏。队列使用多个可能的位置进行放置。这意味着，如果首选放置位置不可用，则始终有备用容量。
- 使游戏可用性更具弹性。可能会发生中断。在使用多区域队列时，速度减慢或停机不一定会影响到玩家访问您的游戏。
- 更高效地使用额外的实例集容量。为了处理意外的玩家需求高峰，合理的做法是提供对额外托管容量的快速访问。队列中的实例集位置为彼此提供备用容量。位置可根据玩家需求向上或向下扩展。
- 获取有关游戏会话放置和队列性能的指标。 会发出队列特定的指标，包括放置成功和失败次数、队列中的请求数以及请求在队列中花费的平均时间这样的统计信息。您还可以在 CloudWatch 控制台中查看这些指标。

若要开始使用队列，请参阅设计游戏会话队列。

设计游戏会话队列

本主题介绍如何设计一个队列，该队列既能以最小的延迟提供玩家体验，又能有效地使用托管资源。有关游戏会话队列及其工作原理的更多信息，请参阅《Amazon GameLift 开发人员指南》中的设置游戏会话置放的 GameLift 队列。

这些 Amazon GameLift 功能需要队列：

- 使用 FlexMatch 配对

- 在 Amazon 上使用竞价型实例 GameLift

定义队列的范围

您的游戏的玩家群体中可能有一群不应该一起玩的玩家。例如，如果您以两种语言发布游戏，则每种语言都应有自己的游戏服务器。

要为您的玩家群体设置游戏会话位置，请为每个玩家区段创建一个单独的队列。确定每个队列的范围，将玩家放到正确的游戏服务器中。确定队列范围的一些常见方法包括：

- 按地理位置划分。在多个地理区域部署游戏服务器时，您可以为每个位置的玩家建立队列以减少玩家延迟。
- 按版本或脚本变体排列。如果您的游戏服务器有多个变体，则可能支持无法在同一游戏会话中玩游戏的玩家组。例如，游戏服务器版本或脚本可能支持不同的语言或设备类型。
- 按事件类型划分。您可以创建一个特殊队列来管理锦标赛或其他特殊活动的参与者的游戏。

创建玩家延迟政策

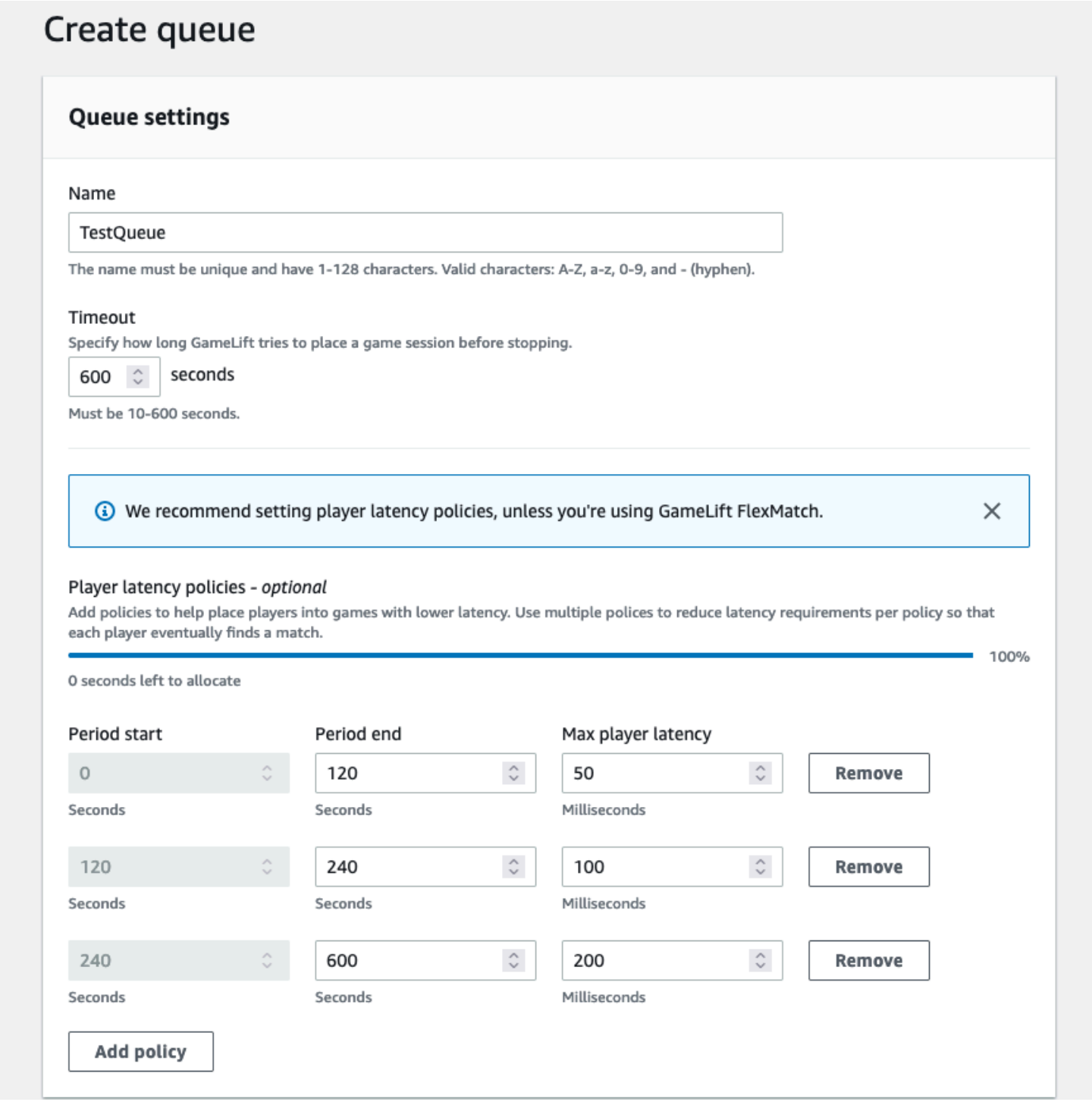
如果您的展示位置请求包含玩家延迟数据，则该算法会查找所有玩家平均延迟时间最低的位置的游戏会话。根据玩家的平均延迟放置游戏会话可防止 Amazon GameLift 将大多数玩家置于延迟较高的游戏中。但是，Amazon GameLift仍然会给玩家带来极大的延迟。为了容纳这些玩家，请制定玩家延迟策略。

玩家延迟策略可防止 Amazon GameLift 将请求的游戏会话放置在请求中的玩家将延迟超过最大值的任何地方。玩家延迟策略还可以阻止 Amazon GameLift 将游戏会话请求与延迟较高的玩家进行匹配。

> **Tip**
>
> 要管理特定于延迟的规则，例如要求小组中所有玩家的延迟时间相似，您可以使用 Amazon GameLift FlexMatch 来创建基于延迟的对战规则。

例如，假设这个队列的超时时间为 5 分钟，并且有以下玩家延迟策略：

1. 花费 120 秒钟搜索所有玩家延迟均低于 50 毫秒的目标，然后...
2. 花费 120 秒钟搜索所有玩家延迟均低于 100 毫秒的目标，然后...
3. 花费剩余的队列超时时间搜索所有玩家延迟均低于 200 毫秒的目标。



建立多位置队列

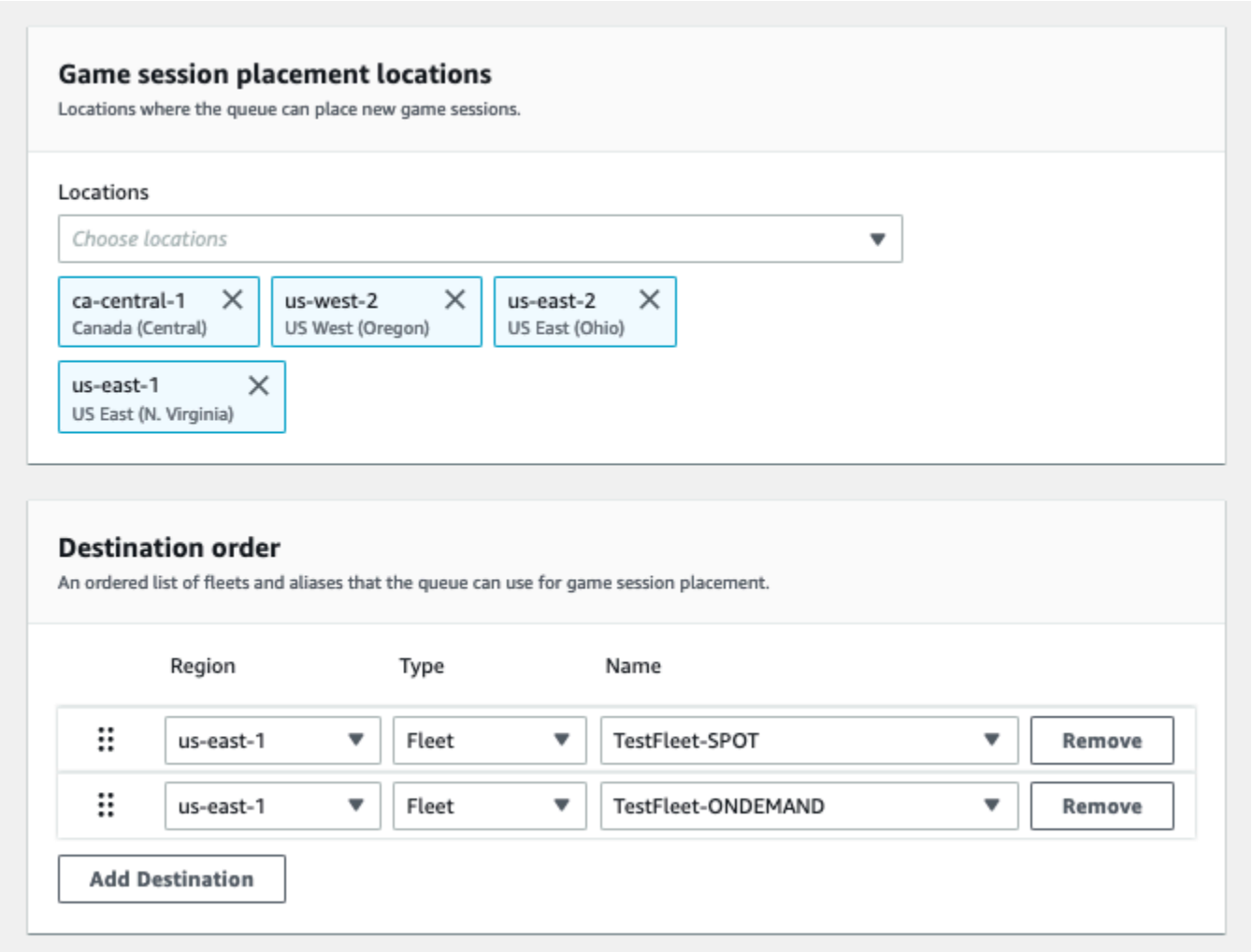
我们建议所有队列都采用多位置设计。这种设计可以提高放置速度和托管弹性。需要采用多位置设计，才能使用玩家延迟数据让玩家以最小的延迟进入游戏会话。如果您要构建使用竞价型实例队列的多位置队列，请按照中的说明进行操作。[教程：为竞价型实例设置游戏会话队列]

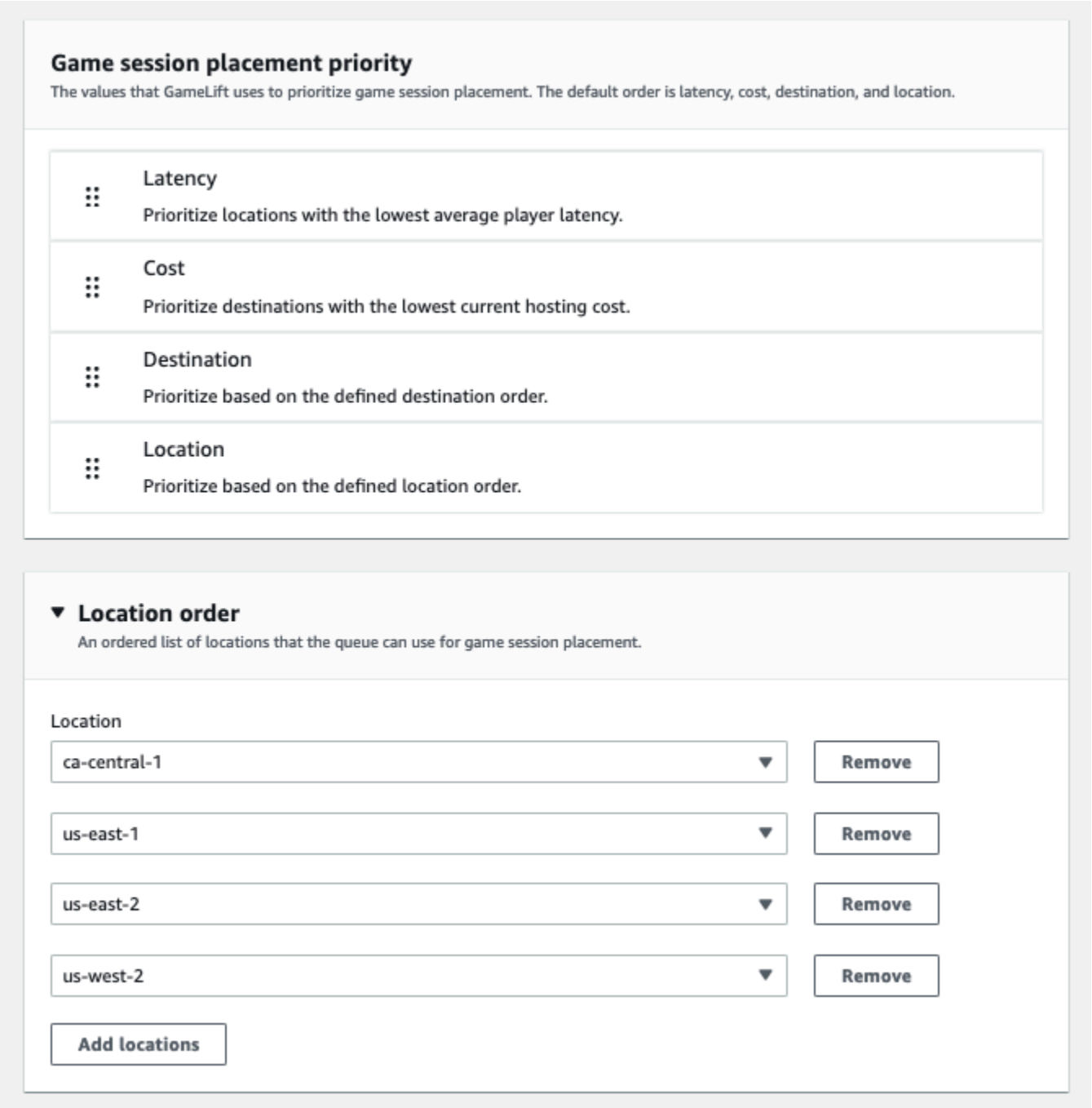
创建多位置队列的一种方法是将多位置队列添加到队列中。这样，队列就可以在实例集的任何位置放置游戏会话。您还可以添加其他具有不同配置或总部位置的实例集以实现冗余。如果您使用的是多位置竞价型实例队列，请遵循最佳实操，并包括具有相同位置的按需型实例队列。

以下示例概述了设计基本多位置队列的过程。在此示例中，我们使用两个队列：一个竞价型实例队列和一个按需型实例队列。每个实例集都有以下AWS 区域放置位置：us-east-1us-east-2、ca-central-1、和us-west-2。

使用多位置实例集创建基本的多位置队列

1. 选择要在其中创建队列的位置。您可以将队列放在部署客户端服务位置附近的位置，从而最大限度地减少请求延迟。在此示例中，我们在中的队列在中的位置us-east-1。
2. 创建新队列并将我们的实例集添加为队列目标。目标顺序决定了 Amazon GameLift 如何放置游戏会话。在此示例中，我们首先列出了竞价型实例队列，然后列出了按需型实例队列。
3. 定义队列的游戏会话放置优先顺序。此顺序决定队列首先在哪里搜索可用的游戏服务器。在此示例中，我们使用默认的优先顺序。
4. 定义位置顺序。如果您未定义位置顺序，Amazon GameLift 会按字母顺序使用这些位置。


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



优先考虑游戏会话放置

Amazon GameLift 使用 FleetiQ 算法根据一组有序的标准来确定在哪里放置新游戏会话。您可以使用默认的优先顺序，也可以自定义顺序。

默认优先顺序

对于包含玩家延迟数据的放置请求，FleetiQ会按以下默认顺序优先考虑游戏会话放置标准：

1. 延迟-请求中所有玩家的最低平均延迟。
2. 成本-如果多个位置的延迟时间相等，则托管成本最低。托管费用主要取决于实例类型和位置的组合。
3. 目的地-目的地订单，如果多个位置的延迟和成本相等。FleetIQ 会根据实例集配置中所列的目标顺序确定优先级。
4. 位置-位置顺序，如果多个位置的延迟、成本和目的地相等。FleetIQ 会根据实例集配置中所列的目标顺序确定优先级。

自定义优先顺序

要在 Amazon GameLift 主机中自定义队列的优先顺序，请将优先级值拖到您想要的位置。要使用 AWS Command Line Interface (AWS CLI) 自定义队列的优先级顺序，请使用带--priority-configuration选项的create-game-session-queue命令。您可以使用此命令创建新队列或更新现有队列。

FleetiQ 算法会根据默认顺序将任何未明确提及的标准附加到列表末尾。如果在优先级配置中包含位置标准，则还必须提供有序的位置列表。

根据需要设计多个队列

根据您的游戏和玩家，您可能需要创建多个游戏会话队列。当游戏客户端或客户端服务请求新游戏会话时，它指定用于放置的队列。为了帮助您确定是否使用多个队列，请考虑：

- 游戏服务器的变体。您可以为游戏服务器的每个变体创建单独的队列。队列中的所有实例集都必须部署兼容的游戏服务器。这是因为使用队列加入游戏的玩家必须能够在队列的任何游戏服务器上玩游戏。
- 不同的玩家群体。您可以根据玩家群体自定义 Amazon GameLift 如何放置游戏会话。例如，您可能需要为某些需要特殊实例类型或运行时配置的游戏模式自定义队列。或者，您可能需要一个特殊的队列来管理锦标赛或其他赛事的排名。
- 游戏会话队列指标。您可以根据想要如何收集游戏会话展示位置指标来设置队列。有关更多信息，请参阅Amazon GameLift 队列指标。

评估队列指标

使用指标来评估您的队列的执行情况。您可以在 Amazon GameLift 控制台或 Amazon CloudWatch 中查看与队列相关的指标。有关队列指标的列表和说明，请参阅Amazon GameLift 队列指标。

队列指标可以提供有关以下内容的见解：

- 队列总体性能-队列指标表明队列响应放置请求的成功程度。这些指标还可以帮助您确定展示位置失败的时间和原因。对于具有手动缩放队列的队列，AverageWaitTime和QueueDepth指标可以指示何时应调整队列的容量。
- FleetIQ 算法性能 – 对于使用 FleetIQ 算法的放置请求，指标显示该算法找到理想游戏会话位置的频率。展示位置可以优先使用玩家延迟最低的资源或成本最低的资源。还有一些错误指标可以确定 Amazon GameLift 找不到理想位置的常见原因。有关指标的更多信息，请参阅 使用 Amazon CloudWatch 监控 Amazon GameLift。
- 特定位置的展示位置-对于多位置队列，指标会按位置显示成功展示位置。通过使用 的队列，此数据提供对玩家活动出现位置的有用见解。

当评估 性能的指标，请考虑以下提示：

- 要跟踪队列寻找理想位置的速度，请将该PlacementsSucceeded指标与FleetiQ指标结合使用，以实现最低延迟和最低价格。
- 要提高队列寻找理想位置的速度，请查看以下错误指标：
  - 如果FirstChoiceOutOfCapacity为高，请调整队列队列的容量扩展。
  - 如果FirstChoiceNotViable错误指标很高，请查看您的竞价型实例队列。当特定实例类型的中断率太高时，竞价型实例被认为“不可行”。要解决此问题，请更改队列以使用具有不同实例类型的竞价型实例集。我们建议您在每个位置加入具有不同实例类型的竞价型实例队列。

Amazon GameLift 游戏会话队列最佳实操

以下是一些最佳实操，可以帮助您为游戏会话放置建立有效的游戏会话队列。

适用于任何实例集类型的队列最佳实操

队列包含可放置新游戏会话的实例集目标的列表。每个实例集都可以在多个地理位置部署实例。选择位置时，队列会选择实例集和实例集位置的组合。您可以为队列提供一组优先级，以便在选择位置时使用。

请考虑以下指南和最佳实操：

- 在可以掩护玩家的位置添加实例集。您可以在任意可用位置添加实例集和别名。如果您根据报告的玩家延迟进行放置，那么位置很重要。
- 为所有实例集使用别名。为队列中的每个实例集分配一个别名，并在队列中设置目标时使用别名。

- 为所有实例集使用相同或相似的游戏构建或脚本。队列可能会让玩家进入队列中任何实例集的游戏会话。玩家必须能够在任何实例集上的任意游戏会话中玩游戏。
- 在至少两个位置创建实例集。通过将游戏服务器托管在至少一个其他位置，可以减轻区域中断对玩家的影响。您可以缩减备份实例集的规模，并在使用量增加时使用自动扩缩来增加容量。
- 优先设置游戏会话放置 队列根据多个元素（包括目标列表顺序）来确定放置选择的优先级。
- 在与客户端服务相同的位置创建队列。通过将队列放在客户端服务附近的位置，可以最大限度地减少通信延迟。
- 使用具有多个位置的实例集。使用队列过滤器配置来防止队列将游戏会话放置在指定位置。在区域停机期间，可以使用至少两个具有不同主位置的多位置实例集，以减轻游戏放置的影响。
- 对所有实例集使用相同的 TLS 证书设置。连接到实例集中游戏会话的游戏客户端必须具有兼容的通信协议。

带有竞价型实例集的队列的最佳实操

如果您的队列包含竞价型实例集，请设置弹性队列。这利用了竞价型实例集节省成本的优势，同时最大限度地减少了游戏会话中断的影响。有关正确构建实例集和游戏会话队列以用于竞价型实例集的帮助，请参阅教程：为竞价型实例设置游戏会话队列。有关竞价型实例的更多信息，请参阅在 Amazon 上使用竞价型实例 GameLift。

除了上一节中的一般最佳实操外，还可以考虑以下特定于竞价型实例的最佳实操：

- 在每个位置至少创建一个按需型实例集。按需型实例集为您的玩家提供备用游戏服务器。您可以缩减备份实例集的规模，直到需要它们为止，并使用自动扩缩在竞价型实例集不可用时增加按需容量。
- 在一个位置的多个竞价型实例集中选择不同的实例类型。如果一种竞价型实例类型暂时不可用，则中断仅影响该位置的一个竞价型实例集。最佳实操是选择广泛可用的实例类型，并使用同一系列中的实例类型（例如 m5.large、m5.xlarge、m5.2xlarge、m5.2xlarge）。使用 Amazon GameLift 控制台查看实例类型的历史定价数据。

创建游戏会话队列

队列用于借助多个实例集和区域中的最佳托管资源来放置新游戏会话。要了解有关为您的游戏构建队列的更多信息，请参阅设计游戏会话队列。

在游戏客户端中，新的游戏会话在启动时，通过使用放置请求加入到队列中。在创建游戏会话中了解有关游戏会话放置的更多信息

更新队列中的队列目标时，有一个很短的过渡期（最多 30 秒），在此期间，放置在队列目标上的游戏会话可能仍会出现在旧队列上。

Console

1. 在 Amazon GameLift 控制台的导航页面中，选择队列。
2. 在 Queues (队列) 页面中，选择 Create queue (创建新队列)。
3. 在创建队列页面上，在队列设置下执行以下操作：

   a. 对于 Name（名称），输入新名称。

   b. 在超时中，输入您希望 Amazon GameLift 在停止之前尝试放置游戏会话的时长。 将继续在任意实例集中搜索可用资源，直到请求超时。

   c. （可选）对于玩家延迟策略，请输入 Amazon GameLift 应在定义的最大延迟内寻找资源的时间。添加其他策略以逐步放宽最大延迟。要添加其他策略，请选择添加策略。
4. 在游戏会话放置位置下，选择要包含在队列中的位置。默认情况下，包括所有位置。队列中的所有实例集必须具有相同的证书配置（GENERATED 或 DISABLED）。队列中的所有机群必须运行与使用该队列的游戏客户端相兼容的游戏生成文件。
5. 在 Destinations 下，向队列添加一个或多个目标。

   a. 选择 Add destination。

   b. 选择目的地所在的位置。

   c. 选择目的地的类型。

   d. 在实例集或别名的结果列表中，选择要添加的实例集或别名。

   e. 如果您有多个目的地，请通过将六点图标拖到目的地左侧来设置默认顺序。 会按照此顺序搜索可用资源的目标以放置新的游戏会话。
6. 对于游戏会话放置优先级，请添加并拖动延迟、成本、目的地和位置值，以定义 Amazon GameLift 如何对队列中的队列进行优先排序。有关确定实例集优先级的更多信息，请参阅优先考虑游戏会话放置。
7. 将位置添加到您的位置顺序中，然后将其拖动到队列应使用的优先级。如果位置是游戏会话放置的最后优先级，那么 Amazon GameLift 会将其用作决胜局。
8. （可选）在事件通知设置下，执行以下操作：

   a. 选择或创建 SNS 主题以接收与展示位置相关的事件通知。有关事件通知的更多信息，请参阅请参阅设置游戏会话置放通知。。

   b. 添加自定义事件数据以附加到该队列创建的事件。

9. [可选] 添加标签。有关标记的更多信息，请参阅标记 资源（ ）。
10. 选择创建。

AWS CLI

Example 创建队列

以下示例创建一个具有以下配置的游戏会话队列：

- 超时五分钟
- 两个实例集目的地
- 筛选仅允许在us-east-1、中的位置us-east-2。 us-west-2，以及 ca-central-1
- 根据成本确定目的地的优先级，然后按定义的顺序排列位置的优先级。

```
aws gamelift create-game-session-queue \
    --name "sample-test-queue" \
    --timeout-in-seconds 300 \
    --destinations DestinationArn="arn:aws:gamelift:us-east-1:111122223333:fleet/
fleet-772266ba-8c82-4a6e-b620-a74a62a93ff8" DestinationArn="arn:aws:gamelift:us-
east-1:111122223333:fleet/fleet-33f28fb6-aa8b-4867-85b4-ceb217bf5994" \
    --filter-configuration "AllowedLocations=us-east-1, ca-central-1, us-east-2, us-
west-2" \
    --priority-configuration
 PriorityOrder="LOCATION","DESTINATION",LocationOrder="us-east-1","us-east-2","ca-
central-1","us-west-2" \
    --notification-target "arn:aws:sns:us-east-1:111122223333:gamelift-test.fifo"
```

> **Note**
>
> 您可以通过使用实例集或别名 ID 调用 describe-fleet-attributes 或 describe-alias 来获取实例集和别名 ARN 的值。

如果 请求成功，则 将返回包含新队列配置的 GameSessionQueue 对象。您现在可以使用 StartGameSessionPlacement 向队列提交请求。

Example 使用玩家延迟策略创建队列

以下示例创建一个具有以下配置的游戏会话队列：

- 超时十分钟
- 三个实例集目的地
- 您可以设置玩家延迟策略以：

```
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" \
```

如果 请求成功，则 将返回包含新队列配置的 GameSessionQueue 对象。

请参阅设置游戏会话置放通知。

您可以使用Events 来监控各个放置请求的状态。我们建议为所有有大量投放活动的游戏设置事件通知。

有两个选项可用于设置事件通知。

- 使用队列将事件通知发布到 Amazon Simple Notification Service (Amazon SNS) 主题。
- 使用自动发布的 Amazon EventBridge 赛事及其工具套件来管理事件。

有关 Amazon GameLift 发出的游戏会话放置事件的列表，请参阅游戏会话放置事件。

设置 SNS 主题。

要让 Amazon GameLift 将游戏会话队列生成的所有事件发布到某个主题，请将通知目标字段设置为主题。

为 事件通知设置 SNS 主题

1. 访问 https://console.aws.amazon.com/sns/v3/home，登录 AWS Management Console 并打开 Amazon SNS 控制台。
2. 在 SNS 控制面板中，选择 Create topic (创建主题)，然后按照说明创建主题。
3. 在 Create Policy（创建策略）下，执行以下操作。

   a. 选择高级方法。

   b. 将以粗体显示的 JSON 数据块添加到现有策略。

```
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__default_statement_ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SNS:GetTopicAttributes",
        "SNS:SetTopicAttributes",
        "SNS:AddPermission",
        "SNS:RemovePermission",
        "SNS:DeleteTopic",
        "SNS:Subscribe",
        "SNS:ListSubscriptionsByTopic",
        "SNS:Publish"
      ],
      "Resource": "arn:aws:sns:your_region:your_account:your_topic_name",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "your_account"
        }
      }
    },
    {
      "Sid": "__console_pub_0",
      "Effect": "Allow",
      "Principal": {
        "Service": "gamelift.amazonaws.com"
```

```
      },
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:your_region:your_account:your_topic_name",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn":
 "arn:aws:gamelift:your_region:your_account:gamesessionqueue/your_queue_name"
        }
      }
    }
  ]
}
```

c. （可选）通过向资源策略添加条件，为主题添加其他访问控制。

4. 选择 Create topic（创建主题）。
5. 创建 SNS 主题后，在创建队列时将其添加到队列中，或者编辑现有队列以将其添加。

使用服务器端加密设置 Amazon SNS 主题

借助服务器端加密（SSE），您可以采用加密主题的方式存储敏感数据。SSE 使用 AWS Key Management Service（AWS KMS）中托管的密钥保护 Amazon SNS 主题中消息的内容。有关 Amazon S3 如何执行加密的更多信息，请参阅 Amazon Simple Storage Service 开发人员指南中的使用服务器端加密保护数据。

要使用服务器端加密设置 SNS 主题，请查看下面的主题：

- 《AWS Key Management Service 开发人员指南》中的创建密钥。
- 将 S imple Notification Service 开发人员指南中的主题启用 SSE

创建 KMS 密钥时，请使用以下 KMS 密钥策略：

```
{
    "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"
        }
    }
}
```

设置EventBridge

Amazon GameLift 会自动将所有游戏会话放置事件发布到 EventBridge。使用 EventBridge，您可以设置规则将事件路由到目标进行处理。例如，您可以设置一条规则，将事件路由到PlacementFulfilled一个处理连接到游戏会话之前的任务的AWS Lambda函数。有关 EventBridge 的更多信息，请参阅 Amazon EventBridge 用户指南中的 Amazon EventBridge 入门。

以下是用于 Amazon GameLift 队列的赛事桥规则的一些示例：

匹配来自所有 Amazon GameLift 队列的事件

```
{
    "source": [
        "aws.gamelift"
    ],
    "detail-type": [
        "GameLift Queue Placement Event"
    ]
}
```

匹配特定队列中的事件

```
{
    "source": [
        "aws.gamelift"
    ],
    "detail-type": [
        "GameLift Queue Placement Event"
```

```
    ],
    "resources": [
        "arn:aws:gamelift:your_region:your_account:gamesessionqueue/your_queue_name"
    ]
}
```

教程：为竞价型实例设置游戏会话队列

简介

本教程介绍如何为部署在低成本竞价型实例集上的游戏设置游戏会话位置。竞价型实例集需要额外的步骤来保持游戏服务器对玩家的持续可用性。

目标受众

本教程适用于想要使用竞价型实例集托管自定义游戏服务器或实时服务器的游戏开发人员。

您将了解

- 定义您的游戏会话队列所服务的玩家组。
- 构建实例集基础设施以支持游戏会话队列的范围。
- 为每个实例集分配一个别名以抽象实例集 ID。
- 创建队列，添加实例集，并对 Amazon GameLift 放置游戏会话的位置进行优先排序。
- 添加玩家延迟策略以帮助最大限度地减少延迟问题。

先决条件

在创建实例集和队列以放置游戏会话之前，请完成以下任务：

- 审核Amazon GameLift 的工作原理。
- 将游戏服务器与 Amazon GameLift 集成。
- 上传游戏服务器构建或实时脚本至 Amazon GameLift。
- 规划实例集配置。

步骤 1：定义队列范围

在本教程中，我们为具有一个游戏服务器构建变体的游戏设计队列。在启动时，我们将在两个位置发布该游戏：亚太地区（首尔）和亚太地区（新加坡）。由于这些位置彼此靠近，因此延迟对我们的玩家来说不是问题。

在这个例子中，有一个玩家区段，这意味着我们创建一个队列。将来，当我们在北美发布游戏时，我们可以为北美玩家创建第二个队列。

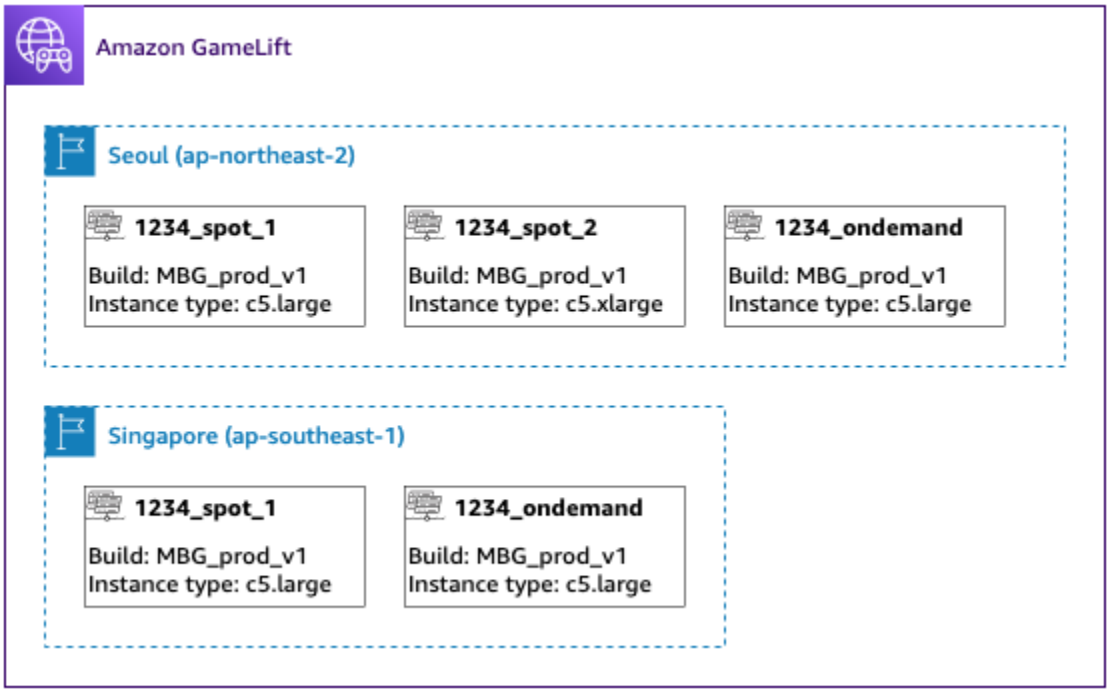
有关更多信息，请参阅定义队列的范围。

步骤 2：创建竞价型实例集基础设施

使用符合您在步骤 1：定义队列范围中定义的范围的位置和游戏服务器构建或脚本创建实例集。

在本教程中，我们创建了一个双位置基础架构，每个位置至少有一个竞价型实例集和一个按需型实例集。每个实例集都部署相同的游戏服务器构建。此外，我们预计首尔地区的玩家流量会更大，因此我们在那里增加了更多的竞价型实例集。

下图显示了竞价型实例集基础设施示例，其中 3 个实例集位于 ap-northeast-2（首尔）位置，2 个实例集位于 ap-southeast-1（新加坡）位置。两个实例集中的所有实例都使用构建 MBG_prod_V1。ap-northeast-2 中的实例集包含以下实例集配置：实例类型为 c5.large 的实例集 1234_spot_1、实例类型为 c5.xlarge 的实例集 1234_spot_2 和实例类型为 c5.large 的 1234_ondemand 实例集。ap-northeast-1 中的实例集包含以下实例集配置：实例类型为 c5.large 的实例集 1234_spot_1 和实例类型为 c5.large 的 1234_ondemand 实例集。

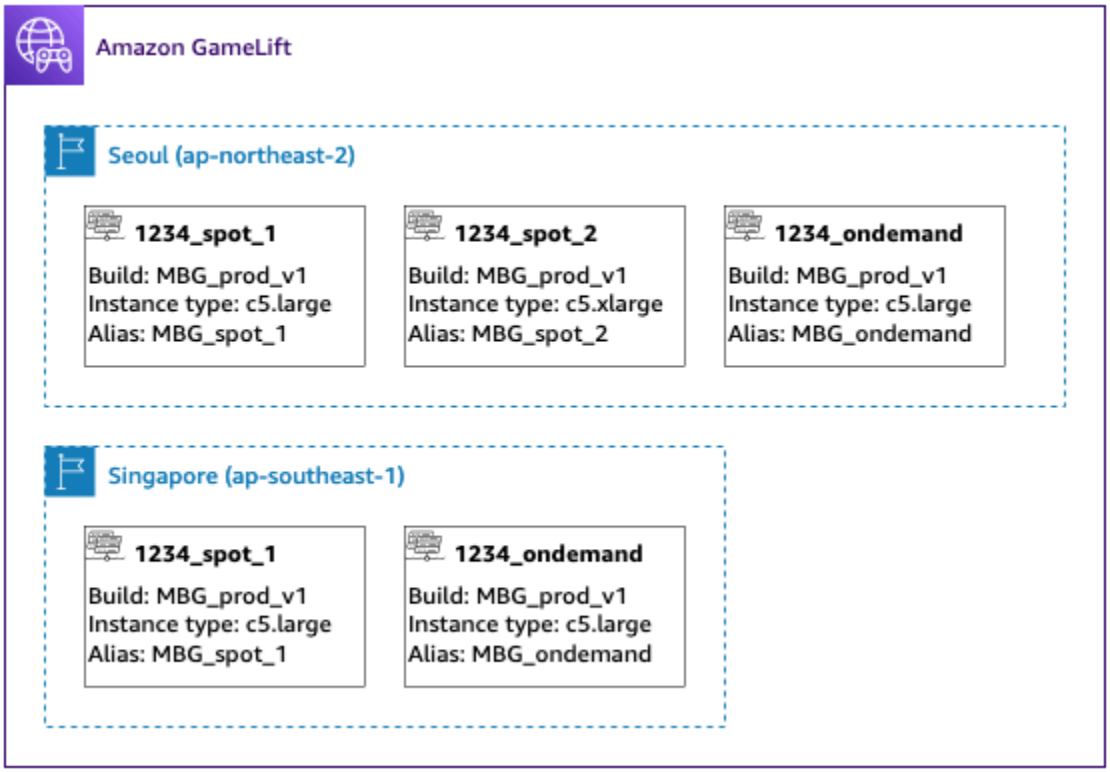


步骤 3：为每个实例集分配别名

为基础架构中的每个实例集创建一个新别名。别名抽象实例集身份，使定期更换实例集变得高效。有关创建别名的更多信息，请参阅向 Amazon GameLift 舰队添加别名。

我们的实例集基础设施有五个实例集，因此我们使用路由策略创建了五个别名。我们在亚太地区（首尔）区域中需要用到三个别名，在亚太地区（新加坡）区域中需要用到两个别名。

下图显示了第二步中描述的竞价型实例集基础设施，并在每个实例集中添加了别名。实例集 1234_spot_1 的别名是 MBG_spot_1，实例集 1234_spot_2 的别名是 MBG_spot_2，实例集 1234_ondemand 的别名是 MBG_ondemand。



有关更多信息，请参阅建立多位置队列。

第 4 步：创建包含目的地的队列

创建游戏会话队列并添加您的实例集目的地。有关创建队列的更多信息，请参阅创建游戏会话队列。

创建队列时：

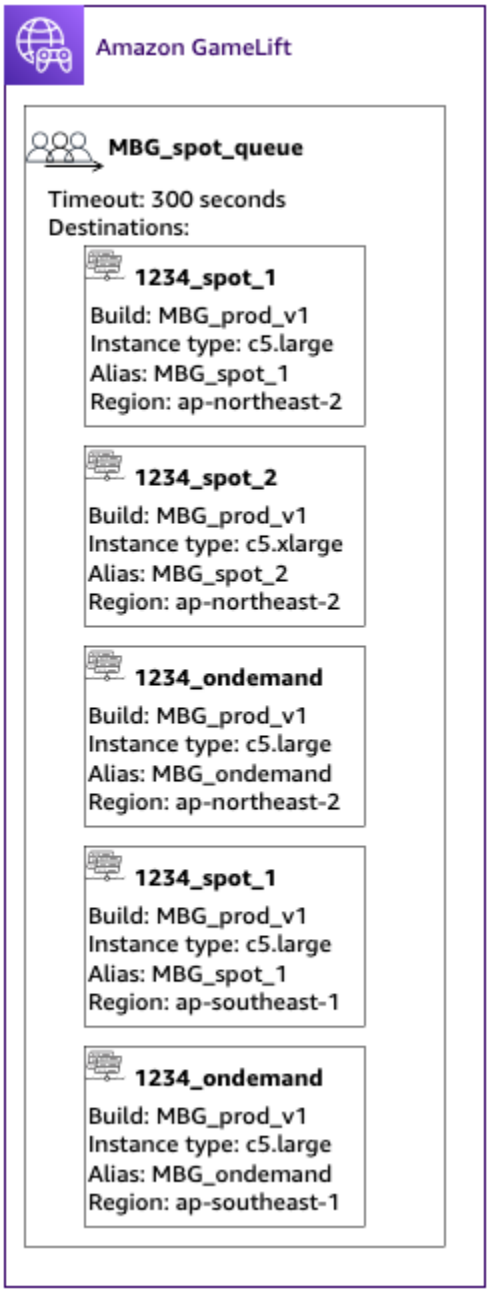
- 默认超时值为 10 分钟。稍后，您可以测试队列超时如何影响玩家进入游戏的等待时间。
- 暂时跳过有关玩家延迟策略的部分。在下一步骤中，我们将介绍这一点。
- 对队列中的实例集进行优先排序。在使用竞价型实例集时，我们建议采用以下任一方法：
  - 如果您的基础设施使用主位置且实例集位于第二个位置进行备份，请先按位置排列实例集的优先级，然后再按实例集类型进行优先排序。

- 如果您的基础设施同时使用多个位置，请按实例集类型确定实例集的优先级，将竞价型实例集放在队列的首位。

在本教程中，我们使用名称 **MBG_spot_queue** 创建一个新队列，并添加所有五个实例集的别名。然后，我们首先按位置排列放置的优先顺序，然后按实例集类型确定放置的优先顺序。

基于此配置，该队列始终尝试将新的游戏会话放入首尔的竞价型实例集中。当这些实例集已满时，队列会使用首尔按需型实例集的可用容量作为备用。如果所有三支首尔实例集都不可用，Amazon GameLift 会将游戏会话安排在新加坡实例集上。

下图显示了超时时间为 300 秒的队列和按优先顺序排列的目的地。目的地按以下顺序排列：ap-northeast-2 中的 1234_spot_1、ap-northeast-2 中的 1234_spot_2、ap-northeast-2 中的 1234_ondemand、ap-southeast-1 中的 1234_ondemand 和 ap-southeast-1 中的 1234_ondemand。



步骤 5：向队列添加延迟限制

我们的游戏在游戏会话放置请求中包含延迟信息。我们还有一个玩家聚会功能，可以为一群玩家创建游戏会话。我们可以让玩家再等一会儿才能进入具有理想游戏体验的游戏。我们的游戏测试显示了以下观察结果：

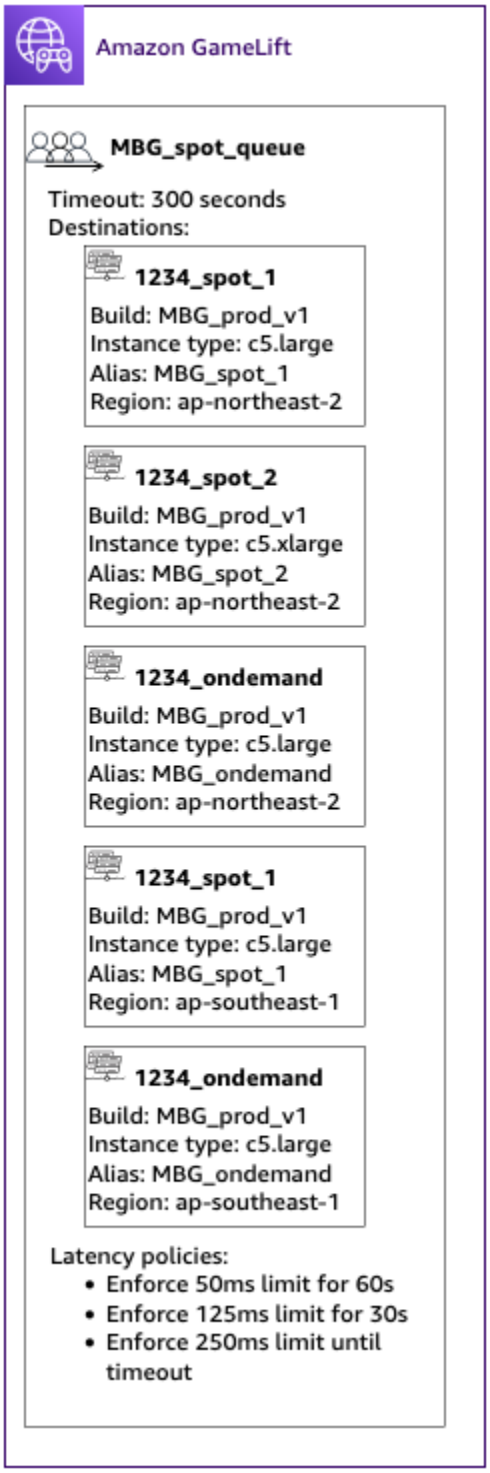
- 延迟低于 50 毫秒是理想的。
- 延迟超过 250 毫秒时无法玩游戏。

- 玩家在约 1 分钟内就会变得不耐烦。

对于我们的队列，超时时间为 300 秒，我们添加了限制允许延迟的策略声明。策略声明逐渐允许更大的延迟值，延迟时间最长可达 250 毫秒。

根据此政策，我们的队列会寻找第一分钟延迟为理想的展示位置（低于 50 毫秒），然后放宽限制。队列不会在玩家延迟等于 250 毫秒或更高的位置进行放置。

下图显示了步骤四中的队列，其中添加了玩家延迟策略。玩家延迟策略规定，对 60 秒强制执行 50 毫秒的限制，对 30 秒强制执行 125 毫秒的限制，并在超时之前强制执行 250 毫秒的限制。



总结

恭喜您！以下是您完成的事项：

- 您的游戏会话队列限定为一部分玩家群体。

- 您的队列可以有效地使用竞价型实例集，并且在竞价型实例中断发生时具有弹性。
- 您的队列会优先考虑实例集以获得出色玩家体验。
- 队列有延迟限制，可以保护玩家免受糟糕的游戏体验的影响。

现在，您可以使用队列为其服务的玩家放置游戏会话。向这些玩家提出游戏会话放置请求时，请在请求中引用此游戏会话队列名称。有关提出游戏会话放置请求的更多信息，请参阅创建游戏会话或为实时服务器集成游戏客户端。

后续步骤：

- 自行设计队列。
- 创建队列。
- 在游戏客户端中使用队列。

使用 AWS CloudFormation 管理资源

您可以使用 AWS CloudFormation 来管理 Amazon GameLift 资源。在 AWS CloudFormation 中，您可以创建对每个资源构建模型的模板，然后使用该模板创建资源。要更新资源，您可以对模板进行更改，然后使用 AWS CloudFormation 实施更新。您可以将资源按照称为堆栈和堆栈集的逻辑组排列。

使用 AWS CloudFormation 维护您的 Amazon GameLift 托管资源提供了一种更有效的方式来管理 AWS 资源集。您可以使用版本控制来跟踪模板随时间推移的更改，并协调由多个团队成员进行的更新。您还可以重复使用模板。例如，在跨区域部署游戏时，您可以在各个区域中使用相同的模板创建相同的资源。您还可以使用这些模板在另一个分区中部署相同的资源集。

有关 AWS CloudFormation 的更多信息，请参阅《AWS CloudFormation 用户指南》。要查看 Amazon GameLift 资源的模板信息，请参阅 Amazon GameLift 资源类型参考。

最佳实操

有关使用 AWS CloudFormation 的详细指导，请参阅《AWS CloudFormation 用户指南》中的AWS CloudFormation 最佳实操。此外，这些最佳实操与 Amazon GameLift 有特殊关联。

- 仅通过 AWS CloudFormation 一致地管理您的资源。如果您在 AWS CloudFormation 外部更改资源，则资源将与资源模板失去同步。
- 使用 AWS CloudFormation 堆栈和堆栈集高效管理多个资源。

  - 使用堆栈来管理连接的资源组。例如，一个堆栈，其中包含构建、引用该构建的实例集和引用该实例集的别名。如果您更新模板以替换构建，AWS CloudFormation 还会替换连接到构建的实例集。然后，AWS CloudFormation 更新现有别名以指向新实例集。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的使用堆栈。
  - 如果您要跨多个区域或 AWS 账户部署相同的堆栈，请使用 AWS CloudFormation 堆栈集。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的使用堆栈。
- 如果您使用竞价型实例，请包括按需型实例集作为备份。我们建议您在模板中设置每个区域有两个实例集，一个实例集包含竞价型实例，另一个则包含按需型实例。
- 当您在多个区域中管理资源时，将您的区域特定资源和全局资源分组到单独的堆栈中。
- 将您的全局资源置于使用它的服务附近。队列和对战配置等资源往往会接收来自特定源）的大量请求。通过将资源放在靠近这些请求源的位置，您可以最大限度地减少请求行程时间并提高整体性能。
- 将您的对战配置放在与使用它的游戏会话队列相同的区域中。
- 为堆栈中的每个实例集创建一个单独的别名。

使用 AWS CloudFormation 堆栈

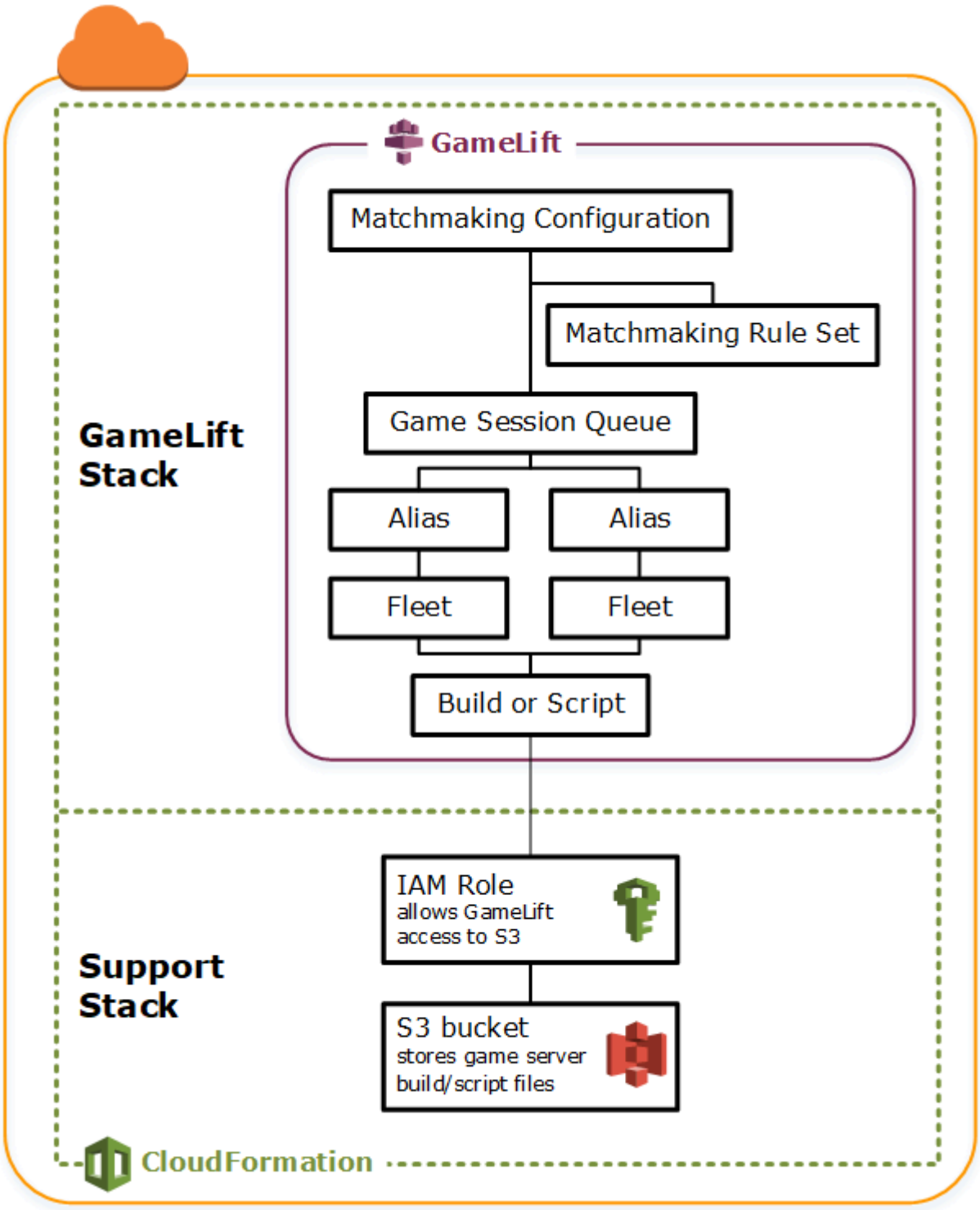
以下是在为 Amazon GameLift 资源设置 AWS CloudFormation 堆栈时建议使用的结构。您的最佳堆栈结构根据您在一个还是多个位置部署游戏而异。

用于单个位置的堆栈

要在单个位置管理 Amazon GameLift 资源，我们建议采用双栈结构：

- 支持堆栈 – 此堆栈包含您的 Amazon GameLift 资源所依赖的资源。此堆栈至少应包括您存储自定义游戏服务器或 Realtime 脚本文件的 S3 存储桶。堆栈还应包含 IAM 角色，在创建 Amazon GameLift 构建或脚本资源时，该角色向 Amazon GameLift 授予从 S3 存储桶检索文件的权限。此堆栈还可能包含用于您游戏的其他 AWS 资源，例如 DynamoDB 表、Amazon Redshift 集群和 Lambda 函数。
- Amazon GameLift 堆栈 – 此堆栈包含您的所有 Amazon GameLift 资源，包括构建或脚本、实例集集合、别名和游戏会话队列。AWS CloudFormation 使用存储在 S3 存储桶位置中的文件创建构建或脚本资源，并将构建或脚本部署到一个或多个实例集资源。每个实例集都应该具有一个对应的别名。游戏会话队列引用部分或全部实例集别名。如果您使用 FlexMatch 进行对战，则此堆栈还包含对战配置和规则集。

下图说明了用于在单个 AWS 区域中部署资源的双堆栈结构。

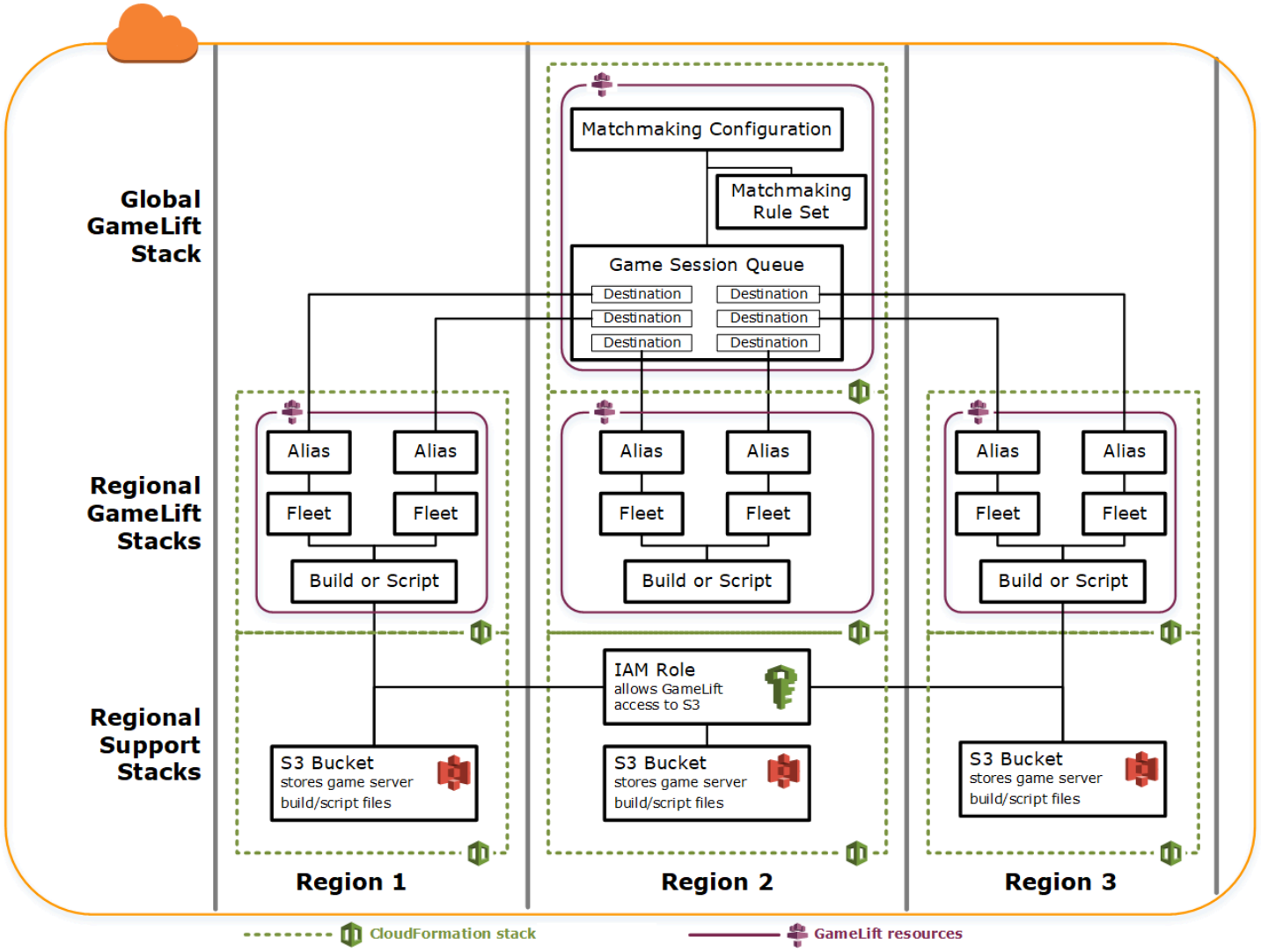


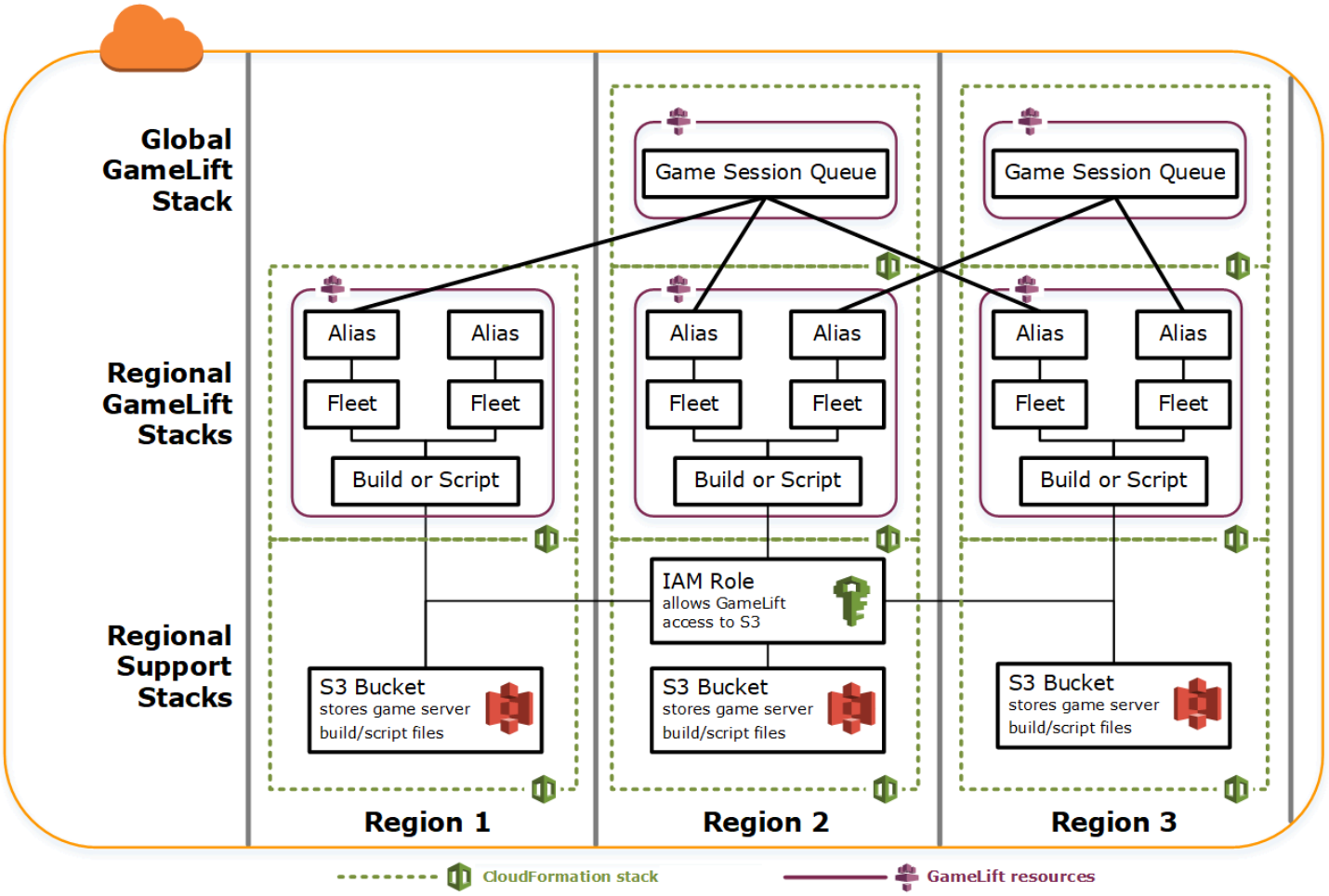
适用于多个区域的堆栈

在多个区域中部署游戏时，请注意资源如何跨区域进行交互。部分资源（例如 Amazon GameLift 实例集）只能引用同一区域中的其他资源。其他资源（例如 Amazon GameLift 队列）与区域无关。要管理多个区域中的 Amazon GameLift 资源，我们建议使用以下结构。

- 区域支持堆栈 – 这些堆栈包含您的 Amazon GameLift 资源所依赖的资源。此堆栈必须包括您存储自定义游戏服务器或 Realtime 脚本文件的 S3 存储桶。此堆栈还可能包含针对您游戏的其他 AWS 资源，例如 DynamoDB 表、Amazon Redshift 集群和 Lambda 函数。这些资源中有许多是特定于区域的，因此您必须在每个区域创建它们。Amazon GameLift 还需要一个 IAM 角色，允许访问受支持资源。由于 IAM 角色与区域无关，因此您只需要一个角色资源，放置在任意区域中，并在所有其他支持堆栈中引用。
- 区域 Amazon GameLift 堆栈 – 此堆栈包含部署游戏的各个区域中必须存在的 Amazon GameLift 资源，包括构建或脚本、实例集集合和别名。AWS CloudFormation 使用 S3 存储桶位置中的文件创建构建或脚本，并将构建或脚本部署到一个或多个实例集资源。每个实例集都应该具有一个对应的别名。游戏会话队列引用部分或全部实例集别名。您可以维护一个模板，用于描述这种类型的堆栈并将其用于在每个区域中创建相同的资源集。
- 全局 Amazon GameLift 堆栈 – 此堆栈包含您的游戏会话队列和对战资源。这些资源可以位于任意区域中，通常放置在相同区域。队列可以引用位于任意区域中的实例集或别名。要在不同的区域中放置其他队列，请创建另外的全局堆栈。

下图说明了在多个 AWS 区域中部署资源的多堆栈结构。第一张图显示了单个游戏会话队列的结构。第二张图显示了多个队列的结构。


Global GameLift Stack
Regional GameLift Stacks
Regional Support Stacks
Matchmaking Configuration
Matchmaking Rule Set
Game Session Queue
Destination
Alias
Fleet
Build or Script
IAM Role
allows GameLift access to S3
S3 Bucket
stores game server build/script files
Region 1
Region 2
Region 3
CloudFormation stack
GameLift resources



更新构建

Amazon GameLift 构建是不可变的，就像构建与实例集之间的关系。因此，当您更新托管资源以使用一组新的游戏构建文件时，必须执行以下操作：

- 使用一组新文件创建新构建（替换）。
- 创建一个新的实例集集合以部署新游戏构建（替换）。
- 重定向别名以指向新实例集（无中断更新）。

有关更多信息，请参阅《AWS CloudFormation 用户指南》中的堆栈资源的更新行为。

自动部署构建更新

更新包含相关构建、实例集和别名资源的堆栈时，默认 AWS CloudFormation 行为是自动执行序列中的这些步骤。通过先将新构建文件上传到新的 S3 位置即可触发此更新。然后，您修改 AWS

CloudFormation 构建模板以指向新 S3 位置。使用新 S3 位置更新堆栈时，这将触发以下 AWS CloudFormation 序列：

1. 从 S3 检索新文件、验证文件并创建新 Amazon GameLift 构建。
2. 在队列模板中更新构建引用，该引用将触发新实例集创建。
3. 新实例集处于活动状态之后，更新别名中的实例集引用，这将触发将别名更新以指向新实例集。
4. 删除旧实例集。
5. 删除旧构建。

如果您的游戏会话队列使用实例集别名，则玩家流量会在更新别名之后自动切换到新实例集。在游戏会话结束时，旧实例集中的玩家逐渐全部退出。在玩家流量波动时，Auto-scaling 处理在每个实例集集合中添加和删除实例的任务。或者，您可以指定所需的初始实例计数，以便快速增加切换并在稍后启用 Auto-scaling。

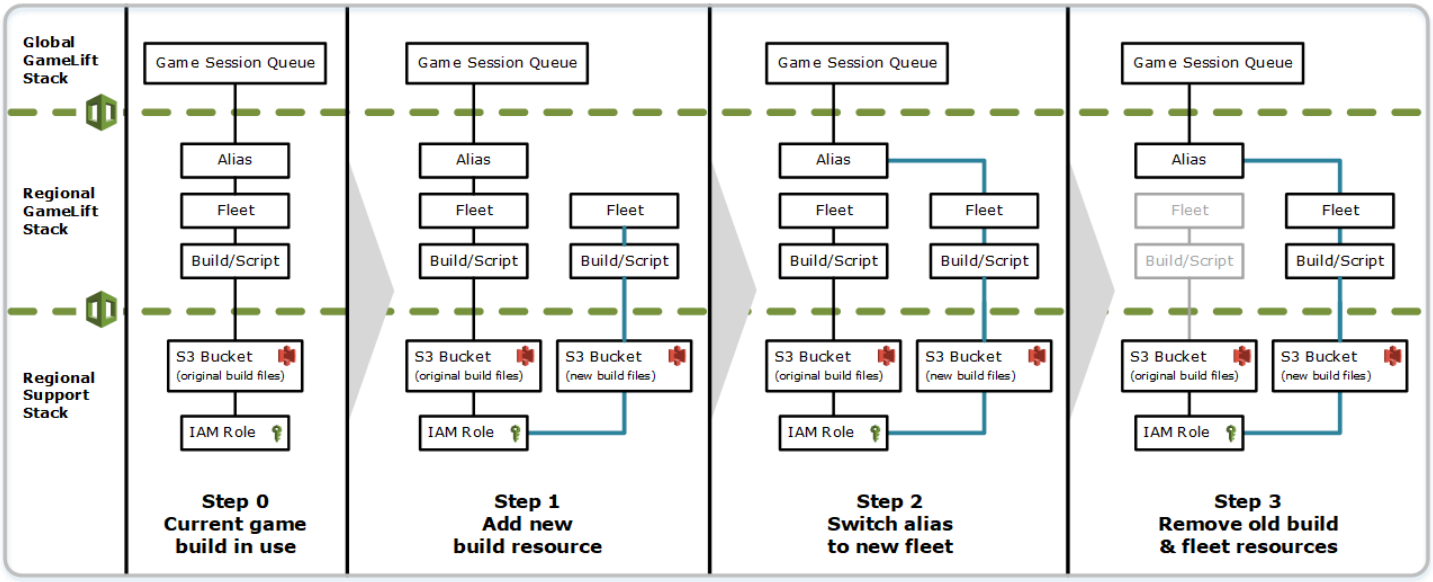
您还可以让 AWS CloudFormation 保留资源而不是删除资源。有关更多信息，请参阅《AWS CloudFormation API 参考》中的 RetainResources。

手动部署构建更新

在希望更好地控制什么时候为玩家上线新实例集时，您可以利用一些选项。您可以选择使用 Amazon GameLift 控制台或 CLI 来手动管理别名。或者，您无需更新构建模板以替换构建和实例集，而是可以向模板添加第二个构建和实例集定义集合。当您更新模板时，AWS CloudFormation 创建第二个构建资源和相应的实例集。由于未替换现有资源，因此这些资源不会被删除，别名仍然指向原始实例集。

这种方法的主要优点在于向您提供了灵活性。您可以为构建的新版本创建单独的资源，测试新资源，并控制何时为玩家上线新实例集。这种方法的一个潜在缺点是，在很短的一段时间内，它在各个区域中需要两倍的资源。

下图阐明了此过程。



回滚的工作原理

执行资源更新时，如果任何步骤未成功完成，AWS CloudFormation 将自动启动回滚。此过程反向执行序列中的各个步骤，删除新创建的资源。

如果您需要手动触发回滚，请将构建模板的 S3 位置键更改回原始位置并更新堆栈。此时创建新 Amazon GameLift 构建和实例集，在该实例集处于活动状态之后，别名切换回新实例集。如果您要单独管理别名，则需要将其切换为指向新实例集。

有关如何处理失败或卡住的回滚的更多信息，请参阅《AWS CloudFormation 用户指南》中的继续回滚更新。

Amazon GameLift 的 VPC 对等连接

本主题提供有关如何在 Amazon GameLift 托管游戏服务器与其他非 Amazon GameLift 资源之间建立 VPC 对等连接的指导。使用 Amazon 虚拟私有云 (VPC) 对等连接支持您的游戏服务器与其他 AWS 资源（例如 Web 服务或存储库）直接私下通信。您可以使用在 AWS 上运行且由您有权访问的 AWS 账户托管的任何资源建立 VPC 对等连接。

> **Note**
>
> VPC 对等连接是一项高级特征。要了解使您的游戏服务器能够与您的其他 AWS 资源建立直接、私密通信的首选选项，请参阅与您的实例集中的其他 AWS 资源进行通信。

如果您已熟悉 Amazon VPC 和 VPC 对等连接，请了解设置与 Amazon GameLift 游戏服务器对等的方式略有不同。您不具备游戏服务器的 VPC 的访问权限（该权限由 Amazon GameLift 服务控制），因此您无法直接为其请求 VPC 对等连接。而应首先对非 Amazon GameLift 资源预授权 VPC，以接受来自 Amazon GameLift 服务的对等连接请求。然后触发 Amazon GameLift 请求您刚刚授权的 VPC 对等。Amazon GameLift 负责创建对等连接、设置路由表和配置连接等任务。

为现有实例集设置 VPC 对等连接

1. 获取 AWS 账户 ID 和凭证。

   您需要以下 AWS 账户的 ID 和登录凭证。您可以通过登录 AWS Management Console并查看您的账户设置来找到 AWS 账户 ID。要获取凭证，请转到 IAM 控制台。

   - 您用于管理 Amazon GameLift 游戏服务器的 AWS 账户。
   - 您用于管理非 Amazon GameLift 资源的 AWS 账户。

   如果您对 Amazon GameLift 和非 Amazon GameLift 资源使用相同账户，则仅需要该账户的 ID 和凭证。

2. 为每个 VPC 获取标识符。

   为要对等的两个 VPC 获取以下信息：

   - 您的 Amazon GameLift 游戏服务器的 VPC – 这是您的 Amazon GameLift 实例集 ID。您的游戏服务器部署在 EC2 实例的实例集上的 Amazon GameLift 中。实例集都自动放置在自己的 VPC 中，后者由 Amazon GameLift 服务进行托管。您没有 VPC 的直接访问权限，因此使用实例集 ID 标识。
   - 您的非 Amazon GameLift AWS 资源的 VPC – 您可以使用在 AWS 上运行且由您有权访问的 AWS 账户托管的任何资源建立 VPC 对等连接。如果您还没有为这些资源创建 VPC，请参阅 Amazon VPC 入门。创建 VPC 后，您可以通过登录 Amazon VPC 的 AWS Management Console并查看您的 VPC 来找到 VPC ID。

   > **Note**
   >
   > 在设置对等连接时，两个 VPC 必须位于相同区域中。您的 Amazon GameLift 实例集游戏服务器的 VPC 与实例集在相同区域中。

3. 授权 VPC 对等连接。

   在此步骤中，您预授权将来从 Amazon GameLift 将用于游戏服务器的 VPC 与用于非 Amazon GameLift 资源的 VPC 建立对等连接的请求。此操作将更新您的 VPC 的安全组。

   要授权 VPC 对等连接，请调用 Amazon GameLift 服务 API CreateVpcPeeringAuthorization() 或使用 AWS CLI 命令 `create-vpc-peering-authorization`。使用管理您的非 Amazon GameLift 资源的账户发出此调用。确定以下信息：

   - 对等 VPC ID – 这是针对非 Amazon GameLift 资源的 VPC。
   - Amazon GameLift AWS 账户 ID – 用于管理 Amazon GameLift 实例集的账户。

   授权 VPC 对等连接后，授权将在 24 小时内保持有效，除非您将其撤销。您可以使用以下操作管理您的 VPC 对等连接授权：

   - DescribeVpcPeeringAuthorizations() (AWS CLI `describe-vpc-peering-authorizations`)。
   - DeleteVpcPeeringAuthorization() (AWS CLI `delete-vpc-peering-authorization`)。

4. 请求对等连接。

   使用有效授权，您可以请求 Amazon GameLift 建立对等连接。

   要请求 VPC 对等连接，请调用 Amazon GameLift 服务 API CreateVpcPeeringConnection() 或使用 AWS CLI 命令 `create-vpc-peering-connection`。使用管理您的 Amazon GameLift 游戏服务器的账户发出此调用。使用以下信息标识要建立对等连接的两个 VPC：

   - 对等 VPC ID 和 AWS 账户 ID – 这是针对您的非 Amazon GameLift 资源和您用于管理这些资源的账户的 VPC。VPC ID 必须与有效对等授权上的 ID 匹配。
   - 实例集 ID – 此信息标识您的 Amazon GameLift 游戏服务器的 VPC。

5. 跟踪对等连接状态。

   请求 VPC 对等连接是一个异步操作。要跟踪对等请求的状态并处理成功或失败的案例，请使用以下选项之一：

   - 使用 `DescribeVpcPeeringConnections()` 持续轮询。此操作将检索 VPC 对等连接记录，包括请求的状态。如果已成功创建对等连接，则连接记录也包含分配给 VPC 的私有 IP 地址的 CIDR 块。

- 使用 DescribeFleetEvents() 处理与 VPC 对等连接关联的实例集事件，包括成功事件和失败事件。

建立对等连接后，您可以立即使用以下操作对其进行管理：

- DescribeVpcPeeringConnections() (AWS CLI `describe-vpc-peering-connections`)。
- DeleteVpcPeeringConnection() (AWS CLI `delete-vpc-peering-connection`)。

使用新实例集设置 VPC 对等连接

您可以创建一个新 Amazon GameLift 实例集，同时请求 VPC 对等连接。

1. 获取 AWS 账户 ID 和凭证。

   您需要以下两个 AWS 账户的 ID 和登录凭证。您可以通过登录 AWS Management Console并查看您的账户设置来找到 AWS 账户 ID。要获取凭证，请转到 IAM 控制台。

   - 您用于管理 Amazon GameLift 游戏服务器的 AWS 账户。
   - 您用于管理非 Amazon GameLift 资源的 AWS 账户。

   如果您对 Amazon GameLift 和非 Amazon GameLift 资源使用相同账户，则仅需要该账户的 ID 和凭证。

2. 获取您的非 Amazon GameLift AWS 资源的 VPC ID。

   如果您还没有为这些资源创建 VPC，请现在创建（请参阅 Amazon VPC 入门）。请确保您在计划创建新实例集的同一区域中创建新的 VPC。如果管理非 Amazon GameLift 资源所用的账户或用户/用户组与您用于 Amazon GameLift 的 AWS 账户或用户/用户组不同，则在下一步中请求授权时，您需要使用这些账户凭证。

   创建 VPC 后，您可以在 Amazon VPC 控制台中通过查看 VPC 来查找 VPC ID。

3. 使用非 Amazon GameLift 资源授权 VPC 对等连接。

   当 Amazon GameLift 创建新实例集和对应的 VPC 时，它还会发送请求，以便与您的非 Amazon GameLift 资源的 VPC 建立对等连接。您需要预先对该请求进行授权。此步骤将更新您的 VPC 的安全组。

使用管理您的非 Amazon GameLift 资源的账户凭证，调用 Amazon GameLift 服务 API CreateVpcPeeringAuthorization() 或使用 AWS CLI 命令 `create-vpc-peering-authorization`。确定以下信息：

- 对等 VPC ID – 这是针对非 Amazon GameLift 资源的 VPC。
- Amazon GameLift AWS 账户 ID – 用于管理 Amazon GameLift 实例集的账户 ID。

授权 VPC 对等连接后，授权将在 24 小时内保持有效，除非您将其撤销。您可以使用以下操作管理您的 VPC 对等连接授权：

- DescribeVpcPeeringAuthorizations() (AWS CLI `describe-vpc-peering-authorizations`)。
- DeleteVpcPeeringAuthorization() (AWS CLI `delete-vpc-peering-authorization`)。

4. 按照使用 AWS CLI 创建新实例集的说明操作。包括以下其他参数：

- peer-vpc-aws-account-id – 您用于管理非 Amazon GameLift 资源的 VPC 的账户 ID。
- peer-vpc-id – 您的非 GameLift 账户的 VPC 的 ID。

使用 VPC 对等参数成功调用 create-fleet 后将会生成一个新实例集和一个新 VPC 对等请求。该实例集的状态设置为 New，并且将启动实例集激活过程。对等连接请求的状态设置为 initiating-request。通过调用 describe-vpc-peering-connections，您可以跟踪对等请求的成功或失败。

在同时请求新实例集和 VPC 对等连接时，两个操作要么成功，要么失败。如果某个实例集在创建过程中失败，则不会建立 VPC 对等连接。同样，如果 VPC 对等连接由于任何原因失败，则新实例集将无法从 Activating 状态变为 Active 状态。

> **Note**
>
> 新的 VPC 对等连接直到实例集准备好变为活动状态才能完成。这意味着连接不可用，且无法在游戏服务器构建安装过程中使用。

以下示例创建一个新实例集，并在预先建立的 VPC 和新实例集的 VPC 之间创建对等连接。预先建立的 VPC 由您的非 Amazon GameLift AWS 账户 ID 和 VPC ID 组合唯一标识。

```
$ 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"
```

可复制版本：

```
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 对等连接问题疑难解答

如果您在为 Amazon GameLift 游戏服务器建立 VPC 对等连接时遇到问题，请考虑以下常见的根本原因：

- 未找到对所请求连接的授权：

  - 检查非 Amazon GameLift VPC 的 VPC 授权状态。它可能不存在或可能已过期。
  - 检查您尝试建立对等连接的两个 VPC 的区域。如果它们不在同一个区域中，则无法建立对等连接。
- 您的两个 VPC 的 CIDR 块（请参阅无效的 VPC 对等连接配置）重叠。分配给对等的 VPC 的 IPv4 CIDR 块无法重叠。系统会自动为 Amazon GameLift 实例集的 VPC 分配 CIDR 块且您无法更改此块，因此，您需要为非 Amazon GameLift 资源更改 VPC 的 CIDR 块。要解决此问题，请执行以下操作：
  - 通过调用 `DescribeVpcPeeringConnections()` 查找 Amazon GameLift 实例集的此 CIDR 块。
  - 转到 Amazon VPC 控制台，查找非 Amazon GameLift 资源的 VPC，然后更改 CIDR 块，以使它们不会重叠。
- 新实例集未激活（当请求与新实例集建立 VPC 对等连接时）。如果新实例集未能进入活动状态，则没有要与之建立对等连接的 VPC，因此对等连接无法成功。

在控制台中查看您的游戏数据

托管 Amazon GameLift 服务持续收集活动游戏的数据，以帮助您了解玩家行为和性能。借助 Amazon GameLift 控制台，您可以查看、管理和分析构建、实例集、游戏会话和玩家会话的这些信息。

主题

- 查看您当前的亚马逊 GameLift 状态
- 查看您的构建
- 查看您的脚本
- 查看您的实例集
- 查看实例集详细信息
- 查看游戏和玩家会话中的数据
- 查看您的别名
- 查看队列

查看您当前的亚马逊 GameLift 状态

A mazon GameLift 控制面板提供以下内容的视图：

- 处于就绪、已初始化和失败状态的构建数量。选择查看构建，了解有关您当前区域中构建的详细信息。
- 处于所有状态的实例集数量。选择查看实例集，了解有关您当前区域中实例集的详细信息。
- 您当前的资源。
- 新特征和服务公告。

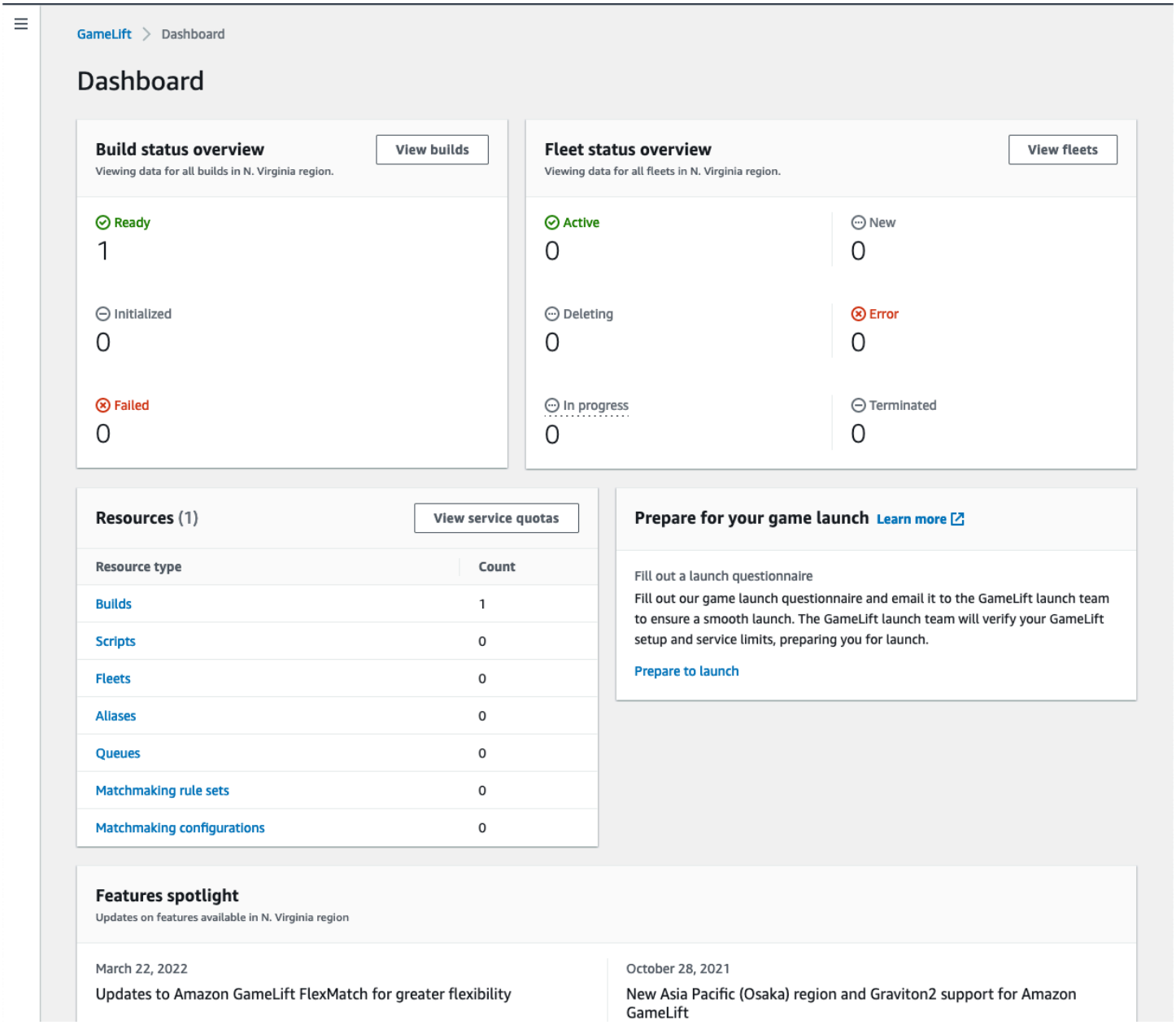
打开 Amazon GameLift 控制面板

- 在 Amazon GameLift 控制台的导航窗格中，选择控制面板。

从控制面板中，您可以执行以下操作：

- 选择准备上线并填写相应的发布问卷，为游戏发布做好准备。
- 通过选择查看服务限额来申请增加服务限额，为启动做准备或响应启动。

- 通过选择特征亮点中的链接，查看有关新特征的博客文章和详细信息。



查看您的构建

在 Amazon GameLift 控制台的构建页面上，您可以查看并管理您上传到 Amazon GameLift 的所有游戏服务器构建的相关信息。在导航窗格中，选择托管，构建。

构建页面显示每个构建的以下信息：

> **Note**
>
> 构建页面仅显示您当前 AWS 区域的构建。

- 名称 – 已上传构建的名称。
- 状态 - 构建的状态。显示以下三种状态消息的一种：
  - 已初始化 – 上传尚未开始或仍在进行中。
  - 准备就绪 – 构建已准备就绪，可以创建实例集。
  - 失败 – Amazon GameLift 收到二进制文件之前构建已过期。
- 创建时间 – 您将构建上传到 Amazon GameLift 的日期和时间。
- 构建 ID – 分配给上传构建的唯一 ID。
- 版本 – 已上传构建的版本标签。
- 操作系统 – 运行构建的操作系统。构建操作系统决定了实例集实例上 Amazon GameLift 安装的操作系统。
- 大小 – 上传到 Amazon GameLift 的构建文件的大小 (MB)。
- 实例集 – 在构建时部署的实例集数量。

在此页面中，您可以执行以下任意操作：

- 查看构建详细信息。选择构建的名称以打开其构建详细信息页面。
- 从构建创建新实例集。选择一个构建，然后选择创建实例集。
- 筛选和排序构建列表。使用表顶部的控件。
- 删除构建。选择构建，然后选择删除。

构建详细信息

在构建页面上，选择构建的名称以打开其构建详细信息页面。详细信息页面的概览部分与构建页面显示相同的构建摘要信息。实例集部分显示了使用该构建创建的实例集列表，包括与实例集页面相同的摘要信息。

查看您的脚本

在 Amazon GameLift 控制台的脚本页面上，您可以查看并管理您上传到 Amazon GameLift 的所有实时服务器脚本的信息。在导航窗格中，选择托管，脚本。

脚本页面显示每个脚本的以下信息：

> **Note**
>
> 脚本页面仅显示您当前 AWS 区域的脚本。

- 名称 – 已上传脚本的名称。
- ID – 分配给上传脚本的唯一 ID。
- 版本 – 已上传脚本的版本标签。
- 大小 – 上传到 Amazon GameLift 的脚本文件的大小 (MB)。
- 创建时间 – 您将脚本上传到 Amazon GameLift 的日期和时间。
- 实例集 – 脚本部署的实例集数量。

在此页面中，您可以执行以下任意操作：

- 查看脚本详细信息。选择构建的名称以打开其脚本详细信息页面。
- 从脚本创建新实例集。选择一个脚本，然后选择创建实例集。
- 筛选和排序脚本列表。使用表顶部的控件。
- 删除脚本。选择脚本，然后选择删除。

脚本详细信息

在脚本页面上，选择脚本的名称以打开其构建详细信息页面。详细信息页面的概览部分与构建页面显示相同的脚本摘要信息。实例集部分显示了使用该脚本创建的实例集列表，包括与实例集页面相同的摘要信息。

查看您的实例集

您可以查看您的 AWS 账户下在 Amazon GameLift 上为托管游戏而创建的所有实例集的信息。该列表显示了您当前所在区域创建的实例集。您可以从实例集页面上创建新的实例集，或查看实例集的更多详

细信息。实例集的详细信息页面包含使用信息、指标、游戏会话数据和玩家会话数据。您也可以编辑实例集记录或删除实例集。

要查看实例集页面，请从导航窗格中选择实例集。

默认情况下，实例集页面显示以下摘要信息：您可以通过选择设置（齿轮）按钮来自定义显示的信息。

- 名称 – 易记的实例集名称。
- 状态 – 实例集的状态，可以是以下状态之一：新建、正在下载、正在构建和活动。
- 创建时间 – 实例集的创建日期和时间。
- 计算类型 – 用于托管游戏的计算类型。实例集可以是托管 EC2 实例集或 Anywhere 实例集。
- 实例类型 – Amazon EC2 实例类型，可决定实例集实例的计算容量。
- 活动实例 – 实例集中正在使用的 EC2 实例数。
- 所需实例 - 要保持活动状态的 EC2 实例的数量。
- 游戏会话数 – 实例集中当前正在运行的活动游戏会话数。该数据有五分钟的延迟。

查看实例集详细信息

在控制面板或实例集页面中选择实例集名称可访问实例集详细信息页面。

在实例集详细信息页面上，您可以执行以下操作：

- 更新实例集的属性、端口设置和运行时配置。
- 添加或删除实例集位置。
- 更改实例集容量设置。
- 设置或更改目标跟踪自动扩缩。
- 删除实例集。

详细信息

实例集设置

- 实例集 ID – 分配给实例集的唯一标识符。
- 名称 – 实例集的名称。
- ARN – 分配给此实例集的标识符。实例集的 ARN 将其标识为 Amazon GameLift 资源并指定区域和 AWS 账户。

- 描述 – 实例集的简要可识别描述。
- 状态 – 实例集的当前状态，可以是新建、正在下载、正在构建和活动。
- 创建时间 – 创建实例集的日期和时间。
- 终止时间 – 实例集终止的日期和时间。如果实例集仍处于活动状态，则此字段为空。
- 实例集类型 – 指示实例集使用按需型实例还是竞价型实例。
- EC2 类型 – 创建实例集时选择的 Amazon EC2 实例类型。
- 实例角色 – 用来管理对您的其他 AWS 资源访问的 AWS IAM 角色（如果在实例集创建期间提供）。
- TLS 证书 – 是启用还是禁用实例集以使用 TLS 证书对游戏服务器进行身份验证和加密所有客户端/服务器通信。
- 指标组 – 用于聚合多个实例集的指标的组。
- 游戏扩展保护策略 – 实例集中游戏会话保护的当前设置。
- 每位玩家的最大游戏会话数 – 玩家在策略期限内可以创建的最大会话数。
- 策略期限 – 在重置玩家创建的会话数量之前要等待多长时间。

构建详细信息

构建详细信息部分显示实例集上托管的构建。选择构建名称可查看完整的构建详细信息页面。

运行时配置

运行时配置部分显示要在每个实例上启动的服务器进程。它包括游戏服务器可执行文件的路径和可选的启动参数。

游戏会话激活

游戏会话激活部分显示同时启动的服务器进程的数量以及该进程在终止之前需要等待多长时间才能激活。

EC2 端口设置

端口部分显示实例集的连接权限，包括 IP 地址和端口设置范围。

指标

指标部分显示一段时间内实例集指标的图形化表示。有关使用 Amazon GameLift 指标的更多信息，请参阅使用 Amazon CloudWatch 监控 Amazon GameLift。

事件

事件选项卡提供实例集中已发生的所有事件的日志，包括事件代码、消息和时间戳。《Amazon GameLift API 参考》中的事件描述。

扩展

扩展选项卡包含与实例集容量相关的信息，包括当前状态和一段时间内容量的变化。它还提供了用于更新容量限制和管理自动扩缩的工具。

扩展容量

查看每个实例集位置的当前实例集容量设置。有关更改限额和容量的更多信息，请参阅扩展 Amazon GameLift 托管容量。

- AWS 位置 – 部署实例集实例的位置的名称。
- 状态 – 实例集位置的托管状态。位置状态必须为 ACTIVE 才能托管游戏。
- 最小大小 – 必须在该位置部署的最小实例数。
- 所需实例 – 维护该位置的活动实例的目标数量。当活动实例和所需实例不相同时，将启动扩展事件以根据需要启动或关闭实例，直到活动实例等于所需的实例。
- 最大大小 – 可以在该位置部署的最大实例数。
- 可用 – 实例的服务限额减去正在使用的实例数量。此值告诉您可以添加至该位置的最大实例数。

自动扩缩策略

本节介绍有关应用于实例集的自动扩缩策略的信息。您可以设置或更新基于目标的策略。此处显示了实例集基于规则的策略，这些策略必须使用 AWS 软件开发工具包或 CLI 进行定义。有关扩展的更多信息，请参阅使用 Amazon GameLift 自动扩缩实例集容量。

扩展历史记录

查看容量随时间变化的图表。

Locations

位置选项卡列出了部署实例集实例的所有位置。位置包括实例集的所在主地区和已添加的任何远程位置。您可以直接在此选项卡中添加或删除位置。

- 位置 – 部署实例集实例的位置的名称。
- 状态 – 实例集位置的托管状态。位置状态跟踪激活该位置中第一个实例的过程。位置状态必须为 ACTIVE 才能托管游戏。
- 活动实例 – 在实例集位置上运行服务器进程的实例数量。
- 活动服务器 – 能够在实例集位置托管游戏会话的游戏服务器进程的数量。
- 游戏会话 – 实例集位置中实例上的活动游戏会话数。
- 玩家会话 – 代表个人玩家参与实例集所在位置的游戏会话的玩家会话数量。

游戏会话

游戏会话选项卡列出了过去和当前在实例集上托管的游戏会话，包括一些详细信息。选择一个游戏会话 ID 可访问游戏会话的更多信息，包括玩家会话。有关玩家会话的更多信息，请参阅查看游戏和玩家会话中的数据。

查看游戏和玩家会话中的数据

您可以查看有关游戏会话以及单个玩家的信息。有关游戏会话和玩家会话的更多信息，请参阅如何将玩家接入游戏。

查看游戏会话和玩家数据

1. 在 Amazon GameLift 控制台的导航窗格中，选择实例集。
2. 从实例集列表中选择托管游戏会话的实例集。
3. 选择游戏会话选项卡。此选项卡将列出实例集中托管的所有游戏会话及其摘要信息。
4. 选择一个游戏会话可查看有关该游戏会话的更多信息，以及连接到游戏的玩家列表。

详细信息

概述

此部分显示您的游戏会话信息的摘要。

- 状态 – 游戏会话状态。
  - 激活 – 实例正在启动游戏会话。
  - 活动 – 游戏会话正在运行且可以接收玩家（取决于会话的玩家创建策略）。

  - 已终止 – 游戏会话已结束。
- ARN – 游戏会话的 Amazon 资源名称。
- 名称 – 为游戏会话生成的名称。
- 位置 – Amazon GameLift 托管游戏会话的位置。
- 创建时间 – Amazon GameLift 创建流会话的日期和时间。
- 结束时间 – 游戏会话结束的日期和时间。
- DNS 名称 – 游戏会话的主机名。
- IP 地址 – 为游戏会话指定的 IP 地址。
- 端口 – 用于连接到游戏会话的端口号。
- 创建者 ID – 发起游戏会话的玩家的唯一标识符。
- 玩家会话创建策略 – 指示游戏会话是否接受新玩家。
- 游戏扩展保护策略 – 要在 Amazon GameLift 在实例集中启动的所有新实例上设置的游戏会话保护类型。

游戏数据

格式良好的数据可在开始时发送到您的游戏会话。

游戏属性

影响游戏会话的键和值对属性。

对战数据

FlexMatch 对战构建器 JSON。要查看和编辑对战构建器，请选择查看对战配置。有关 FlexMatch 对战的更多信息，请参阅构建对战构建器。

玩家会话

每个游戏会话中将收集以下玩家会话数据：

- 玩家会话 ID – 分配给玩家会话的标识符。
- 玩家 ID – 玩家的唯一标识符。单击此 ID 获取更多玩家信息。
- 状态 – 玩家会话的状态。有以下可能状态：
  - 已预留 – 玩家会话已预留，但玩家尚未连接。

  - 活动 – 玩家会话已连接到游戏服务器。
  - 已完成 – 玩家会话已结束；玩家不再处于连接状态。
  - 超时 – 玩家连接失败。
- 开始时间 – 玩家连接到游戏会话的时间。
- 结束时间 – 玩家从游戏会话断开连接的时间。
- 玩家数据 – 玩家会话创建期间提供的有关玩家的信息。

玩家信息

查看关于选定玩家的更多信息，包括玩家在当前区域的所有实例集中曾连接过的全部游戏的列表。此信息包括每个玩家会话的状态、开始和结束时间以及总连接时间。可选择查看相关游戏会话及实例集的数据。

查看您的别名

别名页面显示有关您在当前区域中创建的实例集别名的信息。要查看别名页面，请在导航窗格中选择别名。

在别名页面上，可以执行以下操作：

- 创建新别名。选择创建别名。
- 筛选和排序别名表。使用表顶部的控件。
- 查看别名详细信息。选择一个别名可打开别名详细信息页面。
- 删除别名。选择一个别名，然后选择删除。

别名详细信息

别名详细信息页面中显示有关别名的信息。

在此页面上，您可以：

- 编辑别名。选择编辑。
- 查看与别名关联的实例集。
- 删除别名。选择删除。

别名详细信息包括：

- ID – 用于标识别名的唯一编号。
- 描述 – 别名的描述。
- ARN – 别名的 Amazon Resource Name。
- 创建 – 别名的创建日期和时间。
- 上次更新时间 – 上次更新别名的日期和时间。
- 路由类型 – 别名的路由选项，可以选择以下选项之一：
  - 简单 – 将玩家流量路由到指定的实例集 ID。您可以随时更新别名的实例集 ID。
  - 终端 – 将消息传回客户端。例如，您能够将使用过期客户端的玩家指向可以获得升级的位置。
- 标签 - 用于标识别名的键和值对。

查看队列

您可以查看有关所有现有游戏会话放置队列的信息。队列页面显示在您当前区域创建的队列。在 Queues 页面上，您可以创建新队列、删除现有队列或打开选定队列的详细信息页面。队列详细信息页面包括队列的配置和指标数据。有关队列的更多信息，请参阅《Amazon GameLift 开发人员指南》中的设置游戏会话置放的 GameLift 队列。

Queues 页面将显示每个队列的以下摘要信息：

- 队列名称 为队列分配的名称。对新游戏会话的请求通过此名称指定队列。
- 队列超时 游戏会话放置请求在超时前保留在队列中的最长时间（以秒为单位）。
- 队列中的目标 队列配置中列出的实例集的数量。 Amazon GameLift 会在队列中的任何实例集上放置新的游戏会话。

查看队列详细信息

您可以访问任何队列的详细信息，包括队列配置和指标。要打开队列详细信息页面，请转到 Queues 主页面并选择一个队列名称。

队列详细信息页面显示一个摘要表以及包含附加信息的选项卡。在此页上，您可以执行以下操作：

- 更新队列的配置、目标和玩家延迟策略的列表。选择编辑。
- 删除队列 删除队列后，对引用该队列名称的新游戏会话的所有请求都将失败。选择删除。

> **Note**
>
> 要恢复已删除的队列，请使用已删除队列的名称创建一个新队列。

详细信息

概述

概述部分显示队列的 Amazon 资源名称 (ARN) 和超时时间。在 Amazon GameLift 的其他操作或区域中引用队列时，您可以使用 ARN。队列超时 游戏会话放置请求在超时前保留在队列中的最长时间（以秒为单位）。

事件通知

事件通知部分列出了 Amazon GameLift 向其发布事件通知的 SNS 主题以及添加到该队列创建的所有事件中的事件数据。

标签

标签表显示了用于标记资源的键和值。有关标记的更多信息，请参阅标记 资源（ ）。

指标

Metrics 选项卡显示一段时间内队列指标的图表化表示。

队列指标包括一系列描述整个队列上的放置活动以及按区域细分的成功放置的信息。您可以使用区域数据来了解您的游戏托管位置。区域放置指标可以帮助发现整体队列设计中的问题。

指标还可用于 Amazon CloudWatch。有关这些指标的说明，请参阅 Amazon GameLift 队列指标。

目标

Destinations 选项卡显示为队列列出的所有实例集或别名。

当 搜索可用资源的目标以托管新的游戏会话时，它会按照此处列出的顺序进行搜索。只要列出的第一个目标有容量，就会将新游戏会话放置在那里。通过提供玩家延迟数据，您可以让个别游戏会话放置请求覆盖默认顺序。此数据告知 搜索平均玩家延迟最低的可用目标。有关更多信息，请参阅设计您的架构。

会话放置

玩家延迟策略

玩家延迟策略部分显示队列使用的所有策略。下表将按照强制执行的顺序列出策略。

Locations

“位置” 部分显示了此队列可以将游戏会话置于的位置。

优先级

“优先级” 部分显示队列评估游戏会话详细信息的顺序。

位置顺序

位置顺序部分显示队列在放置游戏会话时使用的默认顺序。如果您尚未定义其他类型的优先级，则队列将使用此顺序。

监控亚马逊 GameLift

如果您将 Amazon GameLift FleetiQ 用作亚马逊 EC2 的独立功能，请参阅亚马逊 EC2 用户指南中的亚马逊 EC 2 中的安全。

监控是维护 Amazon GameLift 和其他 AWS 解决方案的可靠性、可用性和性能的重要组成部分。Amazon 的指标有三个主要用途 GameLift：监控系统运行状况和设置警报、跟踪游戏服务器性能和使用情况，以及使用手动或自动缩放来管理容量。

AWS 提供以下监控工具，用于监视 Amazon GameLift，在出现问题时进行报告，并在适当时自动采取措施：

- 亚马逊 GameLift 控制台
- 亚马逊 CloudWatch--您可以实时监控亚马逊 GameLift 指标，以及您在 AWS 服务上运行的其他 AWS 资源和应用程序的指标。 CloudWatch 提供了一套监控功能，包括用于创建自定义仪表板的工具，以及设置警报的功能，以便在指标达到指定阈值时发出通知或采取行动。
- AWS CloudTrail— 捕获由您的账户或代表您的 AWS 账户为亚马逊和其他 AWS 服务进行的所有 API 调用 GameLift 和相关事件。数据将作为日志文件传送到您指定的 Amazon S3 存储桶。您可以识别哪些用户和帐户拨打了电话 AWS、发出呼叫的源 IP 地址以及呼叫发生的时间。
- 游戏会话日志 – 您可以将游戏会话的自定义服务器消息输出到存储在 Amazon S3 中的日志文件中。

主题

- 使用 Amazon CloudWatch 监控 Amazon GameLift
- 使用 AWS CloudTrail 记录 Amazon GameLift API 调用
- 在 Amazon GameLift 中记录服务器消息

使用 Amazon CloudWatch 监控 Amazon GameLift

您可以使用 Amazon CloudWatch 监控 Amazon GameLift，这是一项 AWS 服务，可收集原始数据，并将数据处理为便于读取的近乎实时的指标。这些统计数据将保留 15 个月，以便从历史角度了解使用 Amazon GameLift 托管游戏服务器的性能。您可以设置用于监测特定阈值的警报，并在达到相应阈值时发送通知或执行操作。有关更多信息，请参阅 Amazon CloudWatch 用户指南。

以下各表列出了 Amazon GameLift 的指标和维度。在 CloudWatch 中可用的所有指标在 Amazon GameLift 控制台（使用一组可自定义的图形提供数据）中同样可用。要访问游戏的 CloudWatch 指标，请使用AWS Management Console、AWS CLI 或 CloudWatch API。

如果指标没有位置，则使用起始位置。

Amazon GameLift 指标的维度

Amazon GameLift 支持按以下维度来筛选指标。

维度	描述
Location	筛选实例集部署位置的指标。如果指标没有位置，则使用起始位置。
FleetId	筛选单个实例集的指标。该维度可用于实例、服务器进程、游戏会话和玩家会话的所有实例集指标。
MetricGroup	筛选实例集集合的指标。通过将实例集指标组名称添加到实例集的属性（请参阅 UpdateFleetAttributes()）向该指标组添加实例集。该维度可用于实例、服务器进程、游戏会话和玩家会话的所有实例集指标。
QueueName	筛选单个队列的指标。该维度只用于游戏会话队列的指标。
ConfigurationName	筛选用于单个对战配置的指标。此维度仅与用于对战配置的指标一起使用。
ConfigurationName-RuleName	筛选用于对战配置与对战规则的相交处的指标。此维度仅与用于对战规则的指标一起使用。
InstanceType	EC2 实例类型名称的筛选指标，如“c4.large”。此维度与竞价型实例的指标一起使用。
OperatingSystem	筛选实例的操作系统的指标。此维度与竞价型实例的指标一起使用。
GameServerGroup	筛选用于游戏服务器组的 FleetIQ 指标。

Amazon GameLift 实例集指标

AWS/GameLift 命名空间包含以下与整个实例集或实例集组活动相关的指标。实例集与托管 Amazon GameLift 解决方案一起使用。Amazon GameLift 服务每分钟向 CloudWatch 发送一次指标。

实例

指标	描述
ActiveInstances	具有 ACTIVE 状态的实例 (表示它们正在运行活动服务器进程)。计数包括空闲实例和托管一个或多个游戏会话的实例。该指标用于衡量当前的总计实例容量。该指标可与自动扩展功能配合使用。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：位置
DesiredInstances	Amazon GameLift 在实例集中设法维护的活动实例的目标数量。配合自动扩展使用时，该值基于当前有效的扩展策略确定。不使用自动扩展时，该值需手动设置。查看实例集指标组数据时，该指标不可用。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：位置
IdleInstances	当前托管零 (0) 个游戏会话的活动实例。该指标用于衡量可用但未用的容量。该指标可与自动扩展功能配合使用。 单位：计数

指标	描述
	相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：位置
MaxInstances	实例集允许的最大实例数。实例集的最大实例数决定了手动或自动扩展期间的容量上限。查看实例集指标组数据时，该指标不可用。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：位置
MinInstances	实例集允许的最小实例数。实例集的最小实例数决定了手动或自动缩减期间的容量下限。查看实例集指标组数据时，该指标不可用。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：位置
PercentIdleInstances	处于空闲状态的所有活动实例的百分比 (计算公式为：`IdleInstances / ActiveInstances`)。该指标可与自动扩展功能配合使用。 单位：百分比 相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：位置

指标	描述
RecycledInstances	已回收和替换的竞价型实例的数量。Amazon GameLift 会回收当前未托管游戏会话且中断概率很高的竞价型实例。 单位：计数 相关的 CloudWatch 统计数据：Sum、Average、Minimum、Maximum 维度：位置
InstanceInterruptions	已中断的竞价型实例的数量。 单位：计数 相关的 CloudWatch 统计数据：Sum、Average、Minimum、Maximum 维度：位置
CPUUtilization	EC2 指标。对于 Amazon GameLift，该指标表示实例集所在地所有活动实例的硬件性能。Amazon EC2 用于运行实例的物理 CPU 时间的百分比，包括运行用户代码和 Amazon EC2 代码所花费的时间。由于旧设备模拟、非旧设备配置、中断密集型工作负载、实时迁移和实时更新等因素，操作系统中的工具显示的百分比可能与 CloudWatch 不同。 单位：百分比
NetworkIn	EC2 指标。对于 Amazon GameLift，该指标表示实例集所在地所有活动实例的硬件性能。实例在所有网络接口上收到的字节数。该指标确认单个实例上向应用程序传入的网络流量。 单位：字节

指标	描述
NetworkOut	EC2 指标。对于 Amazon GameLift，该指标表示实例集所在地所有活动实例的硬件性能。实例在所有网络接口上发送的字节数。该指标确认单个实例上向应用程序传出的网络流量。 单位：字节
DiskReadBytes	EC2 指标。对于 Amazon GameLift，该指标表示实例集所在地所有活动实例的硬件性能。从可供实例使用的所有实例存储卷读取的字节数。该指标用来确定应用程序从实例的硬盘读取的数据量。它可以用于确定应用程序的速度。 单位：字节
DiskWriteBytes	EC2 指标。对于 Amazon GameLift，该指标表示实例集所在地所有活动实例的硬件性能。向可供实例使用的所有实例存储卷写入的字节数。该指标用来确定应用程序向实例的硬盘写入的数据量。它可以用于确定应用程序的速度。 单位：字节
DiskReadOps	EC2 指标。对于 Amazon GameLift，该指标表示实例集所在地所有活动实例的硬件性能。在指定时间段内从可供实例使用的所有实例存储卷完成的读取操作数。要计算该周期的每秒平均 I/O 操作数 (IOPS)，请将该周期的总操作数除以总秒数。 单位：计数

指标	描述
DiskWriteOps	EC2 指标。对于 Amazon GameLift，该指标表示实例集所在地所有活动实例的硬件性能。在指定时间段内向可供实例使用的所有实例存储卷完成的写入操作数。要计算该周期的每秒平均 I/O 操作数 (IOPS)，请将该周期的总操作数除以总秒数。 单位：计数

服务器进程

指标	描述
ActiveServerProcesses	具有 ACTIVE 状态的服务器进程 (表示它们正在运行并且能够托管游戏会话)。计数包括空闲服务器进程和托管游戏会话的进程。该指标用于衡量当前的总计服务器处理能力。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：位置
HealthyServerProcesses	报告运行正常的活动服务器进程。该指标有助于跟踪实例集游戏服务器的整体运行状况。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：位置
PercentHealthyServerProcesses	报告运行正常的所有活动服务器进程的百分比 (计算公式为：HealthyServerProcesses / ActiveServerProcesses)。

指标	描述
	单位：百分比 相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：位置
ServerProcessAbnormalTerminations	自上次报告以来因异常情况而被关闭的服务器进程。该指标包括 Amazon GameLift 服务发起的终止。当服务器进程停止响应、持续报告运行状况检查失败或无法完全终止（通过调用 ProcessEnding()）时，就会出现这种情况。 单位：计数 相关的 CloudWatch 统计数据：Sum、Average、Minimum、Maximum 维度：位置
ServerProcessActivations	自上次报告以来，从 ACTIVATING 成功转换为 ACTIVE 状态的服务器进程。服务器进程必须处于活动状态才能托管游戏会话。 单位：计数 相关的 CloudWatch 统计数据：Sum、Average、Minimum、Maximum 维度：位置

指标	描述
ServerProcessTerminations	自上次报告以来关闭的服务器进程。这包括由于任何原因转换到 TERMINATED 状态的所有服务器进程，包括正常和异常进程终止。 单位：计数 相关的 CloudWatch 统计数据：Sum、Average、Minimum、Maximum 维度：位置

游戏会话

指标	描述
ActivatingGameSessions	具有 ACTIVATING 状态的游戏会话 (表示它们正在启动)。游戏会话在进入活动状态前无法托管玩家。如果该数字的值在一段时间内一直很高，可能说明游戏会话无法从 ACTIVATING 转换为 ACTIVE 状态。该指标可与自动扩展功能配合使用。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：位置
ActiveGameSessions	具有 ACTIVE 状态的游戏会话 (表示它们能够托管玩家，并且正在托管零个或多个玩家)。该指标用于衡量当前被托管的游戏会话的总数。该指标可与自动扩展功能配合使用。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum

指标	描述
	维度：位置
AvailableGameSessions	当前未用于托管游戏会话并且可以毫不延迟地启动新的游戏会话以启动新的服务器进程或实例的主动、运行正常的服务器进程。该指标可与自动扩展功能配合使用。 **Note** 对于限制并发游戏会话激活的实例集，请使用指标 ConcurrentActivatableGameSessions 。该指标可以更准确地表示可以在没有任何延迟的情况下启动的新游戏会话数量。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：位置

指标	描述
ConcurrentActivatableGameSessions	当前未用于托管游戏会话且可以立即启动新游戏会话的主动、运行正常的服务器进程。 该指标与 AvailableGameSessions 的不同之处在于：它不计算由于游戏会话激活限制而当前无法激活新游戏会话的服务器进程。（参见实例集 RuntimeConfiguration 可选设置 MaxConcurrentGameSessionActivations）。对于不限制游戏会话激活的实例集，此指标与 AvailableGameSessions 相同。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：位置
PercentAvailableGameSessions	所有活动服务器进程 (运行正常或不正常) 上当前未使用的游戏会话槽的百分比 (计算公式为：AvailableGameSessions / [ActiveGameSessions + AvailableGameSessions + unhealthy server processes])。该指标可与自动扩展功能配合使用。 单位：百分比 相关的 CloudWatch 统计数据：Average 维度：位置

指标	描述
GameSessionInterruptions	已中断的竞价型实例上的游戏会话的数量。 单位：计数 相关的 CloudWatch 统计数据：Sum、Average、Minimum、Maximum 维度：位置

玩家会话

指标	描述
CurrentPlayerSessions	具有 ACTIVE 状态 (玩家已连接到活动游戏会话) 或 RESERVED 状态 (已在游戏会话中为玩家分配槽，但玩家尚未连接) 的玩家会话。该指标可与自动扩展功能配合使用。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum
PlayerSessionActivations	自上次报告以来，从 RESERVED 状态转换为 ACTIVE 状态的玩家会话。当玩家成功连接到活动的游戏会话时，就会出现这种情况。 单位：计数 相关的 CloudWatch 统计数据：Sum、Average、Minimum、Maximum

Amazon GameLift 队列指标

Amazon GameLift 命名空间包含以下与整个游戏会话放置队列中的活动有关的指标。队列与托管 Amazon GameLift 解决方案一起使用。Amazon GameLift 服务每分钟向 CloudWatch 发送一次指标。

指标	描述
AverageWaitTime	队列中具有 PENDING 状态的游戏会话放置请求等待执行的平均时长。 单位：秒 相关的 CloudWatch 统计数据：Average、Minimum、Maximum、Sum 维度：位置
FirstChoiceNotViable	成功放入游戏会话，但不是首选实例集，因为该实例集被视为不可行（例如，具有较高中断率的竞价型实例集）。该指标基于成本，而不是延迟。首选实例集是队列中列出的第一个实例集，或者当放置请求包含玩家延迟数据时，FleetIQ 优选级选择的第一个实例集。如果没有可行的竞价型实例集，则可以选择该区域中的任何实例集。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum、Sum
FirstChoiceOutOfCapacity	成功放入游戏会话，但不是首选实例集，因为该实例集没有可用资源。首选实例集是队列中列出的第一个实例集，或者当放置请求包含玩家延迟数据时，您定义的 FleetIQ 优选级选择的第一个实例集。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum、Sum
LowestLatencyPlacement	游戏会话成功放入为玩家提供队列最低延迟的区域。此指标仅当放置请求中包含玩家延迟数据时发出。 单位：计数

指标	描述
	相关的 CloudWatch 统计数据：Average、Minimum、Maximum、Sum
LowestPricePlacement	游戏会话成功放入选定区域的队列价格最低的实例集。此实例集可以是竞价型实例集或按需型实例（如果队列中没有竞价型实例集）。此指标仅当放置请求中包含玩家延迟数据时发出。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum、Sum
Placement <region name>	游戏会话成功放入位于指定区域中的实例集。此指标按区域细分 PlacementsSucceeded 指标。 单位：计数 相关的 CloudWatch 统计数据：Sum
PlacementsCanceled	自上次报告以来，在超时前被取消的游戏会话放置请求。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum、Sum
PlacementsFailed	自上次报告以来，因任何原因失败的游戏会话放置请求。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum、Sum

指标	描述
PlacementsStarted	自上次报告以来，添加到队列中的新的游戏会话放置请求。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum、Sum
PlacementsSucceeded	自上次报告以来，产生了新游戏会话的游戏会话放置请求。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum、Sum
PlacementsTimedOut	自上次报告以来，达到队列超时限制而未执行的游戏会话放置请求。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum、Sum
QueueDepth	队列中状态为 PENDING 的游戏会话放置请求的数量。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum、Sum 维度：位置

Amazon GameLift 对战指标

Amazon GameLift 命名空间包括用于对战配置和对战规则的 FlexMatch 活动的指标。FlexMatch 对战与托管的 Amazon GameLift 解决方案一起使用。Amazon GameLift 服务每分钟向 CloudWatch 发送一次指标。

有关对战活动序列的更多信息，请参阅 Amazon GameLift FlexMatch 的工作原理。

对战配置

指标	描述
CurrentTickets	当前正在处理或等待处理的对战请求。 单位：计数 相关的 CloudWatch 统计数据：Average、Minimum、Maximum、Sum
MatchAcceptancesTimedOut	对于要求接受的对战配置，潜在对战游戏从上次报告后在接受过程中超时。 单位：计数 相关的 CloudWatch 统计数据：Sum
MatchesAccepted	对于要求接受的对战配置，是上次报告后被接受的潜在对战游戏。 单位：计数 相关的 CloudWatch 统计数据：Sum
MatchesCreated	上次报告后创建的潜在对战游戏。 单位：计数 相关的 CloudWatch 统计数据：Sum
MatchesPlaced	上次报告后成功放入游戏会话中的对战游戏。 单位：计数

指标	描述
	相关的 CloudWatch 统计数据：Sum
MatchesRejected	对于要求接受的对战配置，是上次报告后至少被一位玩家拒绝的潜在对战游戏。 单位：计数 相关的 CloudWatch 统计数据：Sum
PlayersStarted	上次报告后在对战票证中添加的玩家。 单位：计数 相关的 CloudWatch 统计数据：Sum
TicketsFailed	上次报告后未成功完成对战游戏而发出的对战请求。 单位：计数 相关的 CloudWatch 统计数据：Sum
TicketsStarted	上次报告后创建的新对战请求。 单位：计数 相关的 CloudWatch 统计数据：Sum
TicketsTimedOut	上次报告后达到超时限制的对战请求。 单位：计数 相关的 CloudWatch 统计数据：Sum
TimeToMatch	对于上次报告前放入潜在对战游戏的对战请求，是票证创建和潜在对战游戏创建之间的时间量。 单位：秒 相关的 CloudWatch 统计数据：Data Samples、Average、Minimum、Maximum

指标	描述
TimeToTicketCancel	对于上次报告前取消的对战请求，是票证创建和取消之间的时间量。 单位：秒 相关的 CloudWatch 统计数据：Data Samples、Average、Minimum、Maximum
TimeToTicketSuccess	对于上次报告前成功的对战请求，是票证创建和成功的对战游戏放置之间的时间量。 单位：秒 相关的 CloudWatch 统计数据：Data Samples、Average、Minimum、Maximum

对战规则

指标	描述
RuleEvaluationsPassed	上次报告后在对战时通过的规则评估。此指标仅限前 50 条规则。 单位：计数 相关的 CloudWatch 统计数据：Sum
RuleEvaluationsFailed	上次报告后在对战时未通过的规则评估。此指标仅限前 50 条规则。 单位：计数 相关的 CloudWatch 统计数据：Sum

FleetIQ 的 Amazon GameLift 指标

Amazon GameLift 命名空间包含用于 FleetIQ 游戏服务器组和游戏服务器活动的指标，作为用于游戏托管的 FleetIQ 独立解决方案的一部分。Amazon GameLift 服务每分钟向 CloudWatch 发送一次指标。参阅 Amazon EC2 Auto Scaling 用户指南中的使用 Amazon CloudWatch 监控自动扩缩组和实例。

指标	描述
AvailableGameServers	可用于运行游戏执行但当前未被玩游戏占用的游戏服务器数量。此数字包括已认领但仍处于 AVAILABLE（可用）状态的游戏服务器。 单位：计数 相关的 CloudWatch 统计数据：Sum 维度：GameServerGroup
UtilizedGameServers	当前被游戏占用的游戏服务器数量。此数字包括处于 UTILIZED 状态的游戏服务器。 单位：计数 相关的 CloudWatch 统计数据：Sum 维度：GameServerGroup
DrainingAvailableGameServers	当前不支持玩游戏且计划终止的实例上的游戏服务器数量。这些游戏服务器属于为响应新的认领请求而认领的最低优先级。 单位：计数 相关的 CloudWatch 统计数据：Sum 维度：GameServerGroup
DrainingUtilizedGameServers	当前支持玩游戏且计划终止的实例上的游戏服务器数量。

指标	描述
	单位：计数 相关的 CloudWatch 统计数据：Sum 维度：GameServerGroup
PercentUtilizedGameServers	当前支持游戏执行的游戏服务器所占的部分。此指标表示当前正在使用的游戏服务器容量。它可用于促使制定相应的自动扩缩策略，以便可以动态添加和删除实例来与玩家需求匹配。 单位：百分比 相关的 CloudWatch 统计数据：Average、Minimum、Maximum 维度：GameServerGroup
GameServerInterruptions	由于竞价型实例可用性有限而中断的竞价型实例上的游戏服务器数量。 单位：计数 相关的 CloudWatch 统计数据：Sum 维度：GameServerGroup、InstanceType
InstanceInterruptions	由于可用性有限而中断的竞价型实例数量。 单位：计数 相关的 CloudWatch 统计数据：Sum 维度：GameServerGroup、InstanceType

使用 AWS CloudTrail 记录 Amazon GameLift API 调用

Amazon GameLift 与 AWS CloudTrail 集成，后者是一项服务，提供 Amazon GameLift 中由用户、角色或 AWS 服务所采取操作的记录。CloudTrail 将 Amazon GameLift 的所有 API 调用作为事件捕获。

捕获调用中包括通过 Amazon GameLift 控制台的调用和对 Amazon GameLift API 操作的代码调用。如果您创建跟踪，则可以使 CloudTrail 事件持续传送到 Amazon S3 存储桶（包括 Amazon GameLift 的事件）。如果您不配置跟踪，则仍可在 CloudTrail 控制台中的 Event history（事件历史记录）中查看最新事件。使用通过 CloudTrail 收集的信息，您可以确定向 Amazon GameLift 发出了什么请求、发出请求时使用的 IP 地址、何人发出的请求、请求的发出时间以及其他详细信息。

要了解有关 CloudTrail 的更多信息，请参阅《AWS CloudTrail 用户指南》。

CloudTrail 中的 Amazon GameLift 信息

在您创建 AWS 账户 时，将在该账户上启用 CloudTrail。当 Amazon GameLift 中发生活动时，该活动将记录在 CloudTrail 事件中，并与其他 AWS 服务事件一起保存在事件历史记录中。您可以在 AWS 账户中查看、搜索和下载最新事件。有关更多信息，请参阅使用 CloudTrail 事件历史记录查看事件。

要持续记录 AWS 账户中的事件（包括 Amazon GameLift 的事件），请创建跟踪。通过跟踪记录，CloudTrail 可将日志文件传送至 Simple Storage Service（Amazon S3）存储桶。预设情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有AWS 区域。此跟踪记录在 AWS 分区中记录所有区域中的事件，并将日志文件传送至您指定的 Simple Storage Service（Amazon S3）桶。此外，您可以配置其他 AWS 服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取行动。有关更多信息，请参阅下列内容：

- 创建跟踪概览
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知
- 从多个区域接收 CloudTrail 日志文件和从多个账户接收 CloudTrail 日志文件

所有 Amazon GameLift 操作都由 CloudTrail 记录，并记录在 Amazon GameLift API 参考中。例如，对 `CreateGameSession`、`CreatePlayerSession` 和 `UpdateGameSession` 操作的调用会在 CloudTrail 日志文件中生成条目。

每个事件或日志条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 AWS Identity and Access Management（IAM）用户凭证发出的。
- 请求是使用角色还是联合身份用户的临时安全凭证发出的。
- 请求是否由其它 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

了解 Amazon GameLift 日志文件条目

跟踪是一种配置，可用于将事件作为日志文件传送到您指定的 Amazon S3 桶。CloudTrail 日志文件包含一个或多个日志条目。一个事件表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间、请求参数等方面的信息。CloudTrail 日志文件不是公用 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序显示。

以下示例显示一个 CloudTrail 日志条目，该条目演示 `CreateFleet` 和 `DescribeFleetAttributes` 操作。

```
{
    "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"
        },
    ]
}
```

在 Amazon GameLift 中记录服务器消息

您可以在日志文件中捕获来自 Amazon GameLift 服务器的自定义服务器消息。配置日志的方式取决于您使用的是自定义服务器还是实时服务器（请参阅本章中的相应小节）。

主题

- 记录服务器消息（自定义服务器）
- 记录服务器消息（实时服务器）

记录服务器消息（自定义服务器）

您可以在日志文件中捕获来自您的 Amazon GameLift 自定义服务器的自定义服务器消息。要了解有关实时服务器日志记录的信息，请参阅记录服务器消息（实时服务器）。

> **Important**
>
> 每个游戏会话的日志文件大小有限制（参见中的 Amazon GameLift 终端节点和配额 AWS 一般参考）。游戏会话结束后，亚马逊会将服务器日志 GameLift 上传到亚马逊简单存储服务 (Amazon S3) Service。Amazon GameLift 不会上传超过限制的日志。日志的增长速度可能非常快，并且会超过大小限制。您应该监控日志，将日志输出限制为仅显示必要的消息。

为自定义服务器配置日志记录

使用 Amazon GameLift 定制服务器，您可以编写自己的代码来执行日志记录，并将其配置为服务器进程配置的一部分。Amazon GameLift 使用您的日志配置来识别在每个游戏会话结束时必须上传到 Amazon S3 的文件。

以下说明阐述了如何使用简化的代码示例配置日志记录：

C++

配置日志记录 (C++)

1. 创建字符串矢量，这些字符串是游戏服务器日志文件的目录路径。

```
std::string serverLog("serverOut.log");        // Example server log file
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);
```

2. 提供您的矢量作为ProcessParameters对象LogParameters的矢量。

```
Aws::GameLift::Server::ProcessParameters processReadyParameter =
 Aws::GameLift::Server::ProcessParameters(
    std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
    std::bind(&Server::onProcessTerminate, this),
    std::bind(&Server::OnHealthCheck, this),
    std::bind(&Server::OnUpdateGameSession, this),
    listenPort,
    Aws::GameLift::Server::LogParameters(logPaths));
```

3. 在调用 ProcessReady() 时提供ProcessParameters对象。

```
Aws::GameLift::GenericOutcome outcome =
   Aws::GameLift::Server::ProcessReady(processReadyParameter);
```

有关更完善的示例，请参阅ProcessReady()。

C#

配置日志记录 (C#)

1. 创建字符串列表，这些字符串是游戏服务器日志文件的目录路径。

```
List<string> logPaths = new List<string>();
logPaths.Add("C:\\game\\serverOut.txt");     // Example of a log file that the
 game server writes
```

2. 提供您的列表作为您的ProcessParameters对象。LogParameters

```
var processReadyParameter = new ProcessParameters(
    this.OnGameSession,
    this.OnProcessTerminate,
    this.OnHealthCheck,
    this.OnGameSessionUpdate,
    port,
    new LogParameters(logPaths));
```

3. 在调用 ProcessReady() 时提供ProcessParameters对象。

```
var processReadyOutcome =
   GameLiftServerAPI.ProcessReady(processReadyParameter);
```

有关更完善的示例，请参阅ProcessReady()。

写入日志

您的日志文件在服务器进程启动后就会存在。您可以使用任意方法写入日志来写入文件。要捕获服务器的所有标准输出和错误输出，请将输出流重新映射到日志文件，如以下示例所示：

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"));
```

获取服务器日志

游戏会话结束后，Amazon GameLift 会自动将日志存储在 Amazon S3 存储桶中，并将其保留 14 天。要获取游戏会话日志的位置，可以使用 GetGameSessionLogUrlAPI 操作。要下载日志，请使用操作返回的 URL。

记录服务器消息（实时服务器）

您可以在日志文件中捕获来自实时服务器的自定义服务器消息。要了解有关自定义服务器日志记录的信息，请参阅记录服务器消息（自定义服务器）。

您可以将不同类型的消息输出到日志文件中（请参阅在服务器脚本中记录消息）。除了您的自定义消息外，实时服务器还使用相同的消息类型输出系统消息并写入相同的日志文件。您可以调整实例集的日志记录级别，以减少服务器生成的日志消息量（请参阅调整日志记录级别）。

> **Important**
>
> 每个游戏会话的日志文件大小有限制（参见中的 Amazon GameLift 终端节点和配额 AWS 一般参考）。游戏会话结束后，亚马逊会将服务器日志 GameLift 上传到亚马逊简单存储服务 (Amazon S3) Service。Amazon GameLift 不会上传超过限制的日志。日志的增长速度可能非常快，并且会超过大小限制。您应该监控日志，将日志输出限制为仅显示必要的消息。

在服务器脚本中记录消息

您可以在实时服务器脚本中输出自定义消息。可以使用以下步骤将服务器消息发送到日志文件：

1. 创建一个变量来保存对记录器对象的引用。

```
var logger;
```

2. 在 `init()` 函数中，从会话对象中获取记录器并将其分配给您的记录器变量。

```
function init(rtSession) {
    session = rtSession;
    logger = session.getLogger();
}
```

3. 在记录器上调用相应的函数以输出消息。

   调试消息

```
logger.debug("This is my debug message...");
```

   信息性消息

```
logger.info("This is my info message...");
```

警告消息

```
logger.warn("This is my warn message...");
```

错误消息

```
logger.error("This is my error message...");
```

致命错误消息

```
logger.fatal("This is my fatal error message...");
```

客户体验致命错误消息

```
logger.cxfatal("This is my customer experience fatal error message...");
```

有关脚本中日志语句的示例，请参阅实时服务器脚本示例。

日志文件中的输出指示消息的类型（DEBUG、INFO、WARN、ERROR、FATAL、CXFATAL），如示例日志中的以下几行所示：

```
09 Sep 2021 11:46:32,970 [INFO] (gamelift.js) 215: Calling GameLiftServerAPI.InitSDK...
09 Sep 2021 11:46:32,993 [INFO] (gamelift.js) 220: GameLiftServerAPI.InitSDK succeeded
09 Sep 2021 11:46:32,993 [INFO] (gamelift.js) 223: Waiting for Realtime server to
 start...
09 Sep 2021 11:46:33,15 [WARN] (index.js) 204: Connection is INSECURE. Messages will be
 sent/received as plaintext.
```

获取服务器日志

游戏会话结束后，亚马逊 GameLift 会自动将日志存储在 Amazon S3 中并将其保留 14 天。您可以使用 GetGameSessionLogUrl API 调用来获取游戏会话的日志位置。使用 API 调用返回的 URL 下载日志。

调整日志记录级别

日志的增长速度可能非常快，并且会超过大小限制。您应该监控日志，将日志输出限制为仅显示必要的消息。对于实时服务器，您可以通过在实例集的运行时配置中以 `loggingLevel:LOGGING_LEVEL` 形式提供一个参数来调整日志记录级别，其中 *`LOGGING_LEVEL`* 是以下值之一：

1. `debug`
2. `info`（默认值）
3. `warn`
4. `error`
5. `fatal`
6. `cxfatal`

此列表按从最不严重（`debug`）到最严重（`cxfatal`）的顺序排列。您设置了单个 `loggingLevel`，服务器将只记录该严重性级别或更高严重级别的消息。例如，设置 `loggingLevel:error` 将使实例集中的所有服务器仅向日志写入 `error`、`fatal` 和 `cxfatal` 消息。

可以在创建实例集时或运行之后为实例集设置日志记录级别。在实例集运行后更改其日志记录级别只会影响更新后创建的游戏会话日志。任何现有游戏会话的日志都不会受到影响。如果您在创建实例集时未设置日志记录级别，则您的服务器会将日志记录级别默认设置为 `info`。有关设置日志记录级别的说明，请参阅以下部分。

创建实时服务器实例集时设置日志记录级别（控制台）

按照创建 Amazon GameLift 托管实例集中的说明创建您的实例集，并添加以下内容：

- 在进程管理步骤的服务器进程分配子步骤中，提供日志记录级别键-值对（例如 `loggingLevel:error`）作为启动参数的值。使用非字母数字字符（逗号除外）将日志记录级别与任何其他参数（例如 `loggingLevel:error +map Winter444`）分开。

创建实时服务器实例集时设置日志记录级别（AWS CLI）

按照创建 Amazon GameLift 托管实例集中的说明创建您的实例集，并添加以下内容：

- 在 `create-fleet` 的 `--runtime-configuration` 参数中，提供日志记录级别键-值对（例如 `loggingLevel:error`）作为`Parameters`的值。使用非字母数字字符（逗号除外）将日志记录级别与任何其他参数分开。请参见以下示例：

```
--runtime-configuration "GameSessionActivationTimeoutSeconds=60,
                    MaxConcurrentGameSessionActivations=2,
                    ServerProcesses=[{LaunchPath=/local/game/myRealtimeLaunchScript.js,
                                    Parameters=loggingLevel:error +map Winter444,
                                    ConcurrentExecutions=10}]"
```

设置正在运行的实时服务器实例集的日志记录级别（控制台）

按照中的说明使用 Amazon GameLift 控制台更新您的舰队，并添加以下内容：更新实例集配置

- 在编辑实例集页面的服务器进程分配下，提供日志记录级别键-值对（例如 `loggingLevel:error`）作为启动参数的值。使用非字母数字字符（逗号除外）将日志记录级别与任何其他参数（例如 `loggingLevel:error +map Winter444`）分开。

设置正在运行的实时服务器实例集的日志记录级别（AWS CLI）

按照更新实例集配置中的说明使用 AWS CLI 更新您的实例集，并添加以下内容：

- 在 `update-runtime-configuration` 的 `--runtime-configuration` 参数中，提供日志记录级别键-值对（例如 `loggingLevel:error`）作为Parameters的值。使用非字母数字字符（逗号除外）将日志记录级别与任何其他参数分开。请参见以下示例：

```
--runtime-configuration "GameSessionActivationTimeoutSeconds=60,
                    MaxConcurrentGameSessionActivations=2,
                    ServerProcesses=[{LaunchPath=/local/game/myRealtimeLaunchScript.js,
                                    Parameters=loggingLevel:error +map Winter444,
                                    ConcurrentExecutions=10}]"
```

Amazon 的安全 GameLift

如果您将 Amazon GameLift FleetiQ 用作亚马逊 EC2 的独立功能，请参阅亚马逊 EC2 用户指南中的亚马逊 EC 2 中的安全。

云安全 AWS 是重中之重。为了满足对安全性最敏感的组织的需求，我们打造了具有超高安全性的数据中心和网络架构。作为 AWS 的客户，您也可以从这些数据中心和网络架构受益。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。作为AWS 合规计划合规计划合规计划合的一部分，第三方审计师定期测试和验证我们安全的有效性。要了解适用于亚马逊的合规计划 GameLift，请按合规计划查看AWS 范围内按合。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还应对其他因素负责，包括数据的敏感性、贵公司的要求以及适用的法律AWS 和法规。

本文档可帮助您了解在使用 Amazon 时如何应用分担责任模型 GameLift。以下主题向您展示如何配置 Amazon GameLift 以满足您的安全与合规目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 Amazon GameLift 资源。

主题

- 亚马逊的数据保护 GameLift
- Amazon 的身份和访问管理 GameLift
- Amazon GameLift 的日志记录和监控
- Amazon 合规性验证 GameLift
- Amazon 的弹性 GameLift
- Amazon 的基础设施安全 GameLift
- Amazon 中的配置和漏洞分析 GameLift
- Amazon 安全最佳实践 GameLift

亚马逊的数据保护 GameLift

如果您将 Amazon GameLift FleetiQ 用作亚马逊 EC2 的独立功能，请参阅亚马逊 EC2 用户指南中的亚马逊 EC 2 中的安全。

分 AWS 担责任模式适用于亚马逊的数据保护 GameLift。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础设施上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS 安全性博客 上的 AWS 责任共担模式和 GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证 (MFA)。
- 使用 SSL/TLS 与资源通信。 AWS 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-2 版》。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您 AWS 服务 使用控制台、API GameLift 或 AWS SDK 与 Amazon 或其他机构合作的情况。 AWS CLI在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

亚马逊 GameLift特定数据的处理方式如下：

- 您上传到亚马逊的游戏服务器版本和脚本存储 GameLift 在 Amazon S3 中。上传此数据后，客户无法直接访问此数据。授权用户可以获得临时访问权限来上传文件，但无法直接查看或更新 Amazon S3 中的文件。要删除脚本和版本，请使用 Amazon GameLift 控制台或服务 API。
- 游戏会话日志数据会在游戏会话完成后在 Amazon S3 中存储一段有限的时间。授权用户可以通过亚马逊 GameLift 控制台中的链接下载日志数据或调用服务 API 来访问日志数据。
- 指标和事件数据存储在亚马逊 GameLift 中，可以通过亚马逊 GameLift 控制台或通过调用服务 API 进行访问。可以在实例集、实例、游戏会话放置、对战票证、游戏会话和玩家会话中检索数据。也可以通过 Amazon CloudWatch 和 Ev CloudWatch ents 访问数据。

- 客户提供的数据存储在 Amazon GameLift 中。授权用户可以通过调用服务 API 来访问数据。潜在的敏感数据可能包括玩家数据、玩家会话和游戏会话数据（包括连接信息）、对战构建器数据等。

> **Note**
>
> 如果您在请求中提供自定义玩家 ID，则预计这些值为匿名 UUID，并且不包含识别出的任何玩家信息。

有关数据保护的更多信息，请参阅《AWS 安全性博客》上的 AWS 责任共担模式和 GDPR 博客文章。

静态加密

Amazon GameLift 特定数据的静态加密处理方式如下：

- 游戏服务器构建和脚本存储在具有服务器端加密的 Amazon S3 存储桶中。
- 客户提供的数据以加密格式存储 GameLift 在 Amazon 中。

传输中加密

与 Amazon GameLift API 的连接通过安全 (SSL) 连接建立，并使用AWS 签名版本 4 进行身份验证（通过 AWS CLI 或 AWS SDK 连接时，会自动处理签名）。使用用于建立连接的安全凭证的 IAM 定义访问策略来管理身份验证。

游戏客户端和游戏服务器之间的直接通信如下：

- 对于托管在亚马逊 GameLift 资源上的自定义游戏服务器，通信不涉及亚马逊 GameLift 服务。客户须自行负责对此通信进行加密。您可以使用启用 TLS 的实例集，让游戏客户端在连接时在游戏服务器上进行身份验证，并对游戏客户端和游戏服务器之间的所有通信进行加密。
- 对于启用了 TLS 证书生成功能的实时服务器，游戏客户端与使用实时客户端软件开发工具包的实时服务器之间的流量在传输时会进行加密。TCP 流量使用 TLS 1.2 进行加密，而 UDP 流量使用 DTLS 1.2 进行加密。

互联网络流量隐私保护

您可以安全地远程访问您的 Amazon GameLift 实例。对于使用 Linux 的实例，SSH 为远程访问提供了一个安全的通信通道。对于运行 Windows 的实例，请使用远程桌面协议 (RDP) 客户端。使用 Amazon

GameLift FleetiQ，使用 Systems Manager 会话 AWS 管理器和运行命令对您的实例的远程访问使用 TLS 1.2 进行加密，创建连接的请求使用 Sigv4 进行签名。有关连接托管 Amazon GameLift 实例的帮助，请参阅远程连接到 Amazon GameLift 舰队实例。

Amazon 的身份和访问管理 GameLift

AWS Identity and Access Management（IAM）是一项 AWS 服务，可以帮助管理员安全地控制对 AWS 资源的访问。IAM 管理员控制谁可以通过身份验证（登录）和授权（拥有权限）使用 Amazon GameLift 资源。IAM 是一项无需额外费用即可使用的 AWS 服务。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问
- 亚马逊如何 GameLift 使用 IAM
- Amazon 基于身份的政策示例 GameLift
- 对 Amazon GameLift 身份和访问进行故障排除

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在亚马逊上所做的工作 GameLift。

服务用户 — 如果您使用 Amazon GameLift 服务完成工作，则您的管理员会为您提供所需的凭证和权限。当您使用更多的 Amazon GameLift 功能来完成工作时，您可能需要额外的权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 Amazon 中的某项功能 GameLift，请参阅对 Amazon GameLift 身份和访问进行故障排除。

服务管理员 — 如果您负责公司的亚马逊 GameLift 资源，则可能拥有对亚马逊的完全访问权限 GameLift。您的工作是确定您的服务用户应该访问哪些亚马逊 GameLift 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要详细了解贵公司如何在 Amazon 上使用 IAM GameLift，请参阅亚马逊如何 GameLift 使用 IAM。

IAM 管理员 — 如果您是 IAM 管理员，则可能需要详细了解如何编写策略来管理对 Amazon 的访问权限 GameLift。要查看您可以在 IAM 中使用的 GameLift 基于身份的 Amazon 策略示例，请参阅。Amazon 基于身份的政策示例 GameLift

使用身份进行身份验证

身份验证是使用身份凭证登录AWS的方法。您必须作为AWS 账户根用户、IAM 用户或通过代入 IAM 角色进行身份验证（登录到AWS）。

您可以使用通过身份源提供的凭证以联合身份登录到AWS。AWS IAM Identity Center(IAM Identity Center) 用户、公司的单点登录身份验证以及 Google 或 Facebook 凭证都是联合身份的示例。当以联合身份登录时，管理员以前使用 IAM 角色设置了身份联合验证。当使用联合身份验证访问AWS时，就是在间接代入角色。

根据用户类型，可以登录AWS Management Console或AWS访问门户。有关登录到 AWS 的更多信息，请参阅《AWS 登录 用户指南》中的 如何登录到您的 AWS 账户。

如果以编程方式访问AWS，则AWS将提供软件开发工具包（SDK）和命令行界面 (CLI)，以便使用凭证以加密方式签署请求。如果不使用AWS工具，则必须自行对请求签名。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的签署 AWS API 请求。

无论使用何种身份验证方法，您都可能需要提供其他安全信息。例如，AWS建议使用多重身份验证（MFA）来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证 和《IAM 用户指南》中的 在 AWS 中使用多重身份验证（MFA）。

AWS 账户 根用户

创建AWS 账户时，最初使用的是一个对账户中所有AWS 服务和资源拥有完全访问权限的登录身份。此身份称为AWS 账户根用户，使用创建账户时所用的电子邮件地址和密码登录，即可获得该身份。强烈建议不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实操，要求人类用户（包括需要管理员访问权限的用户）结合使用联合身份验证和身份提供程序，以使用临时凭证来访问 AWS 服务。

联合身份是来自企业用户目录、Web 身份提供程序、AWS Directory Service、Identity Center 目录的用户，或任何使用通过身份来源提供的凭证来访问 AWS 服务的用户。当联合身份访问 AWS 账户时，他们担任角色，而角色提供临时凭证。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和组，也可以连接并同步到自己的身份源中的一组用户和组以跨所有 AWS 账户 和应用程序使

用。有关 IAM Identity Center 的信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？

IAM 用户和组

IAM 用户 是 AWS 账户内对某个人员或应用程序具有特定权限的一个身份。在可能的情况下，建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果有一些特定的使用场景需要长期凭证以及 IAM 用户，我们建议轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组 是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。可以使用群组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，可能具有一个名为 IAMAdmins 的群组，并为该群组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人担任。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

IAM 角色是 AWS 账户 中具有特定权限的身份。它类似于 IAM 用户，但与特定人员不关联。可以通过切换角色，在AWS Management Console中暂时代入 IAM 角色。您可以调用 AWS CLI 或 AWS API 操作或使用自定义网址以担任角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 - 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供商创建角色。如果使用 IAM Identity Center，则需要配置权限集。为控制身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限 - IAM 用户或角色可担任 IAM 角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 - 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户存取权限的主要方式。但是，对于某些AWS 服务，可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅 IAM 用户指南 中的 IAM 角色与基于资源的策略有何不同。

- 跨服务访问 – 某些 AWS 服务使用其他 AWS 服务中的特征。例如，在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Simple Storage Service（Amazon S3）中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话：当您使用 IAM 用户或角色在 AWS 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用主体调用 AWS 服务 的权限，结合请求的 AWS 服务，向下游服务发出请求。只有在服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
  - 服务相关角色 – 服务相关角色是与AWS 服务关联的一种服务角色。服务可以担任代表您执行操作的角色。服务相关角色显示在AWS 账户中，并由该服务拥有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 Amazon EC2 上运行的应用程序 – 可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时凭证。这优先于在 EC2 实例中存储访问密钥。要将AWS角色分配给 EC2 实例并使其对该实例的所有应用程序可用，可以创建一个附加到实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅《IAM 用户指南》中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

将创建策略并将其附加到AWS身份或资源，以控制AWS中的访问。策略是AWS中的对象；在与身份或资源相关联时，策略定义它们的权限。在主体（用户、根用户或角色会话）发出请求时，AWS将评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略在AWS中存储为 JSON 文档。有关 JSON 策略文档的结构和内容的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概览。

管理员可以使用AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件 下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。然后，管理员可以向角色添加 IAM 策略，并且用户可以代入角色。

IAM 策略定义操作的权限，无关乎使用哪种方法执行操作。例如，假设有一个允许 `iam:GetRole` 操作的策略。具有该策略的用户可以从AWS Management Console、AWS CLI或AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管式策略是可以附加到AWS 账户中的多个用户、组和角色的独立策略。托管式策略包括AWS托管式策略和客户托管式策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。必须在基于资源的策略中指定主体。主体可以包括账户、用户、角色、联合用户或AWS 服务。

基于资源的策略是位于该服务中的内联策略。不能在基于资源的策略中使用来自 IAM 的AWS托管式策略。

访问控制列表 (ACL)

访问控制列表 (ACL) 控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3、AWS WAF和 Amazon VPC 是支持 ACL 的服务示例。要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表 (ACL) 概览。

其他策略类型

AWS支持额外的、不太常用的策略类型。这些策略类型可以设置更常用的策略类型所授予的最大权限。

- 权限边界 – 权限边界是一个高级功能，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的

交集。在 Principal 字段中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的 IAM 实体的权限边界。

- 服务控制策略 (SCP) - SCP 是 JSON 策略，指定了组织或组织单位 (OU) 在 AWS Organizations 中的最大权限。AWS Organizations 服务可以分组和集中管理您的企业拥有的多个 AWS 账户 账户。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中实体（包括每个AWS 账户根用户）的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略 - 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解 AWS 如何确定在涉及多种策略类型时是否允许请求，请参阅《IAM 用户指南》中的策略评估逻辑。

亚马逊如何 GameLift 使用 IAM

在使用 IAM 管理亚马逊访问权限之前 GameLift，请先了解亚马逊可以使用哪些 IAM 功能 GameLift。

您可以在 Amazon 上使用的 IAM 功能 GameLift

IAM 特征	亚马逊 GameLift 支持
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACL	否
ABAC（策略中的标签）	是

IAM 特征	亚马逊 GameLift 支持
临时凭证	是
主体权限	是
服务角色	是
服务相关角色	否

要全面了解 Amazon GameLift 和其他AWS服务如何使用大多数 IAM 功能，请参阅 IAM 用户指南中与 IAM 配合使用的AWS服务。

Amazon 基于身份的政策 GameLift

支持基于身份的策略	是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

Amazon 基于身份的政策示例 GameLift

要查看 Amazon GameLift 基于身份的政策示例，请参阅。Amazon 基于身份的政策示例 GameLift

Amazon 内部基于资源的政策 GameLift

支持基于资源的策略	否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条

件下执行。必须在基于资源的策略中指定主体。主体可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户存取，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当主体和资源处于不同的 AWS 账户 中时，则信任账户中的 IAM 管理员还必须授予主体实体（用户或角色）对资源的访问权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅IAM 用户指南中的 IAM 角色与基于资源的策略有何不同。

针对亚马逊的政策行动 GameLift

支持策略操作	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

有关亚马逊 GameLift 操作的列表，请参阅《服务授权参考》 GameLift中亚马逊定义的操作。

Amazon 中的策略操作在操作前 GameLift 使用以下前缀：

```
gamelift
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "gamelift:action1",
      "gamelift:action2"
         ]
```

您也可以使用通配符 (*) 指定多个操作。例如，要指定以单词 Describe 开头的所有操作，包括以下操作：

```
"Action": "gamelift:Describe*"
```

要查看 Amazon GameLift 基于身份的政策示例，请参阅。Amazon 基于身份的政策示例 GameLift

Amazon 的政策资源 GameLift

支持策略资源	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

ResourceJSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实操，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

有关亚马逊 GameLift 资源类型及其 ARN 的列表，请参阅《服务授权参考》 GameLift中亚马逊定义的资源。要了解您可以使用哪些操作来指定每种资源的 ARN，请参阅 Amazon 定义的操作。 GameLift

某些 Amazon GameLift 资源具有 ARN 值，这允许使用 IAM 策略管理资源的访问权限。Amazon GameLift 舰队资源有一个 ARN，其语法如下：

```
arn:${Partition}:gamelift:${Region}:${Account}:fleet/${FleetId}
```

有关 ARN 格式的更多信息，请参阅《AWS 一般参考》中的 Amazon 资源名称 (ARN)。

例如，要在语句中指定 fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa 实例集，请使用以下 ARN：

```
"Resource": "arn:aws:gamelift:us-west-2:123456789012:fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
```

要指定属于特定账户的所有实例集，请使用通配符 (*)：

```
"Resource": "arn:aws:gamelift:us-west-2:123456789012:fleet/*"
```

要查看 Amazon GameLift 基于身份的政策示例，请参阅。Amazon 基于身份的政策示例 GameLift

Amazon 的政策条件密钥 GameLift

支持特定于服务的策略条件键	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您要为单个条件键指定多个值，则 AWS 使用逻辑 OR 运算来评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

AWS 支持全局条件键和特定于服务的条件键。要查看所有 AWS 全局条件键，请参阅IAM 用户指南中的 AWS 全局条件上下文键。

有关亚马逊 GameLift 条件密钥的列表，请参阅《服务授权参考》 GameLift中的 “亚马逊条件密钥”。要了解您可以使用条件键的操作和资源，请参阅 Amazon 定义的操作 GameLift。

要查看 Amazon GameLift 基于身份的政策示例，请参阅。Amazon 基于身份的政策示例 GameLift

Amazon 中的 ACL GameLift

支持 ACL	否

访问控制列表 (ACL) 控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

ABAC 与 Amazon GameLift

支持 ABAC（策略中的标签）	是

基于属性的访问权限控制 (ABAC) 是一种授权策略，该策略基于属性来定义权限。在 AWS 中，这些属性称为标签。您可以将标签附加到 IAM 实体（用户或角色）以及 AWS 资源。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，需要使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息，请参阅《IAM 用户指南》中的什么是 ABAC？。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

有关基于资源标签限制对资源的访问的基于身份的策略示例，请参阅根据标签查看 Amazon GameLift 车队。

在 Amazon 上使用临时证书 GameLift

支持临时凭证	是

某些 AWS 服务 在使用临时凭证登录时无法正常工作。有关更多信息，包括 AWS 服务与临时凭证配合使用，请参阅《IAM 用户指南》中的使用 IAM 的 AWS 服务。

如果您不使用用户名和密码而用其他方法登录到AWS Management Console，则使用临时凭证。例如，当您使用贵公司的单点登录 (SSO) 链接访问AWS时，该过程将自动创建临时凭证。当您以用户身份登录控制台，然后切换角色时，还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的切换到角色（控制台）。

您可以使用 AWS CLI 或者 AWS API 创建临时凭证。之后，您可以使用这些临时凭证访问 AWS。AWS 建议您动态生成临时凭证，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

Amazon 的跨服务委托人权限 GameLift

支持转发访问会话 (FAS)	是

当您使用 IAM 用户或角色在 AWS 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用主体调用 AWS 服务 的权限，结合请求的 AWS 服务，向下游服务发出请求。只有在服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

Amazon 的服务角色 GameLift

支持服务角色	是

服务角色是由一项服务代入、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会中断 Amazon 的 GameLift功能。仅当 Amazon GameLift 提供相关指导时，才可编辑服务角色。

允许您的亚马逊 GameLift托管的游戏服务器访问其他AWS资源，例如AWS Lambda函数或亚马逊 DynamoDB 数据库。由于游戏服务器托管在亚马逊 GameLift 管理的舰队上，因此您需要一个服务角色来 GameLift 限制亚马逊对您的其他AWS资源的访问权限。有关更多信息，请参阅与您的实例集中的其他 AWS 资源进行通信。

Amazon 的服务相关角色 GameLift

支持服务相关角色	否

服务相关角色是一种与 AWS 服务相关的服务角色。服务可以担任代表您执行操作的角色。服务相关角色显示在AWS 账户中，并由该服务拥有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅《IAM 用户指南》中的与 IAM 搭配使用的 AWS 服务。在表中查找服务相关角色列中包含 Yes 的服务。选择是，查看该服务的服务相关角色文档。

Amazon 基于身份的政策示例 GameLift

默认情况下，用户和角色无权创建或修改 Amazon GameLift 资源。他们也无法使用 AWS Management Console、AWS Command Line Interface（AWS CLI）或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。然后，管理员可以向角色添加 IAM 策略，并且用户可以担任角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅IAM 用户指南中的 创建 IAM policy。

有关 Amazon GameLift 定义的操作和资源类型（包括每种资源类型的 ARN 格式）的详细信息，请参阅《服务授权参考》 GameLift中的 Amazon 操作、资源和条件密钥。

主题

- 策略最佳实践
- 使用亚马逊 GameLift 控制台
- 允许用户查看他们自己的权限
- 允许玩家访问游戏会话
- 允许访问一个 Amazon GameLift 队列
- 根据标签查看 Amazon GameLift 车队
- 在 Amazon S3 中访问游戏构建文件

策略最佳实践

基于身份的策略决定了是否有人可以在您的账户中创建、访问或删除亚马逊 GameLift 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- AWS 托管式策略及转向最低权限许可入门 - 要开始向用户和工作负载授予权限，请使用 AWS 托管式策略来为许多常见使用场景授予权限。可以在AWS 账户中找到这些策略。我们建议通过定义特定于您的使用场景的AWS客户托管式策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略或工作职能的 AWS 托管式策略。
- 应用最低权限 - 在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。

- 使用 IAM 策略中的条件进一步限制访问权限 - 您可以向策略添加条件来限制对操作和资源的访问。例如，可以编写策略条件来指定必须使用 SSL 发送所有请求。如果通过特定AWS 服务（例如AWS CloudFormation）使用服务操作，还可以使用条件来授予对服务操作的访问权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性 - IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Access Analyzer 策略验证。
- 需要多重身份验证 (MFA) - 如果您所处的场景要求您的 AWS 账户 中有 IAM 用户或根用户，请启用 MFA 来提高安全性。要在调用 API 操作时需要 MFA，请将 MFA 条件添加到策略中。有关更多信息，请参阅《IAM 用户指南》中的配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实践的更多信息，请参阅 IAM 用户指南中的 IAM 中的安全最佳实践。

使用亚马逊 GameLift 控制台

要访问 Amazon GameLift 控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您的 Amazon GameLift 资源的详细信息AWS 账户。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

为确保这些实体仍然可以使用 Amazon GameLift 控制台，请使用以下示例和中的语法向用户和群组添加权限管理员权限示例。有关更多信息，请参阅管理 Amazon 的用户权限 GameLift。

GameLift 通过AWS CLI或 AWS API 操作与 Amazon 合作的用户不需要最低控制台权限。相反，您可以将访问权限限制为仅限用户需要执行的操作。例如，代表游戏客户端的玩家用户需要访问权限才能请求游戏会话、让玩家进入游戏以及执行其他任务。

有关使用所有 Amazon GameLift 控制台功能所需的权限的信息，请参阅中的管理员权限语法管理员权限示例。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联策略和托管式策略。此策略包括在控制台上完成此操作或者以编程方式使用 AWS CLI 或 AWS API 所需的权限。

```
{
    "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": "*"
        }
    ]
}
```

允许玩家访问游戏会话

要让玩家进入游戏会话，游戏客户端和后端服务需要权限。有关这些场景的策略示例，请参阅玩家用户权限示例。

允许访问一个 Amazon GameLift 队列

以下示例为用户提供了访问特定 Amazon GameLift 队列的权限。

此策略授予用户通过以下操作添加、更新和删除队列目标的权限：gamelift:UpdateGameSessionQueue、gamelift:DeleteGameSessionQueue、和 gamelift:DescribeGameSessionQueues。如下所示，此策略使用Resource元素限制对单个队列的访问：gamesessionqueue/examplequeue123。

```
{
```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ViewSpecificQueueInfo",
         "Effect":"Allow",
         "Action":[
            "gamelift:DescribeGameSessionQueues"
         ],
         "Resource":"arn:aws:gamelift:::gamesessionqueue/examplequeue123"
      },
      {
         "Sid":"ManageSpecificQueue",
         "Effect":"Allow",
         "Action":[
            "gamelift:UpdateGameSessionQueue",
            "gamelift:DeleteGameSessionQueue"
         ],
         "Resource":"arn:aws:gamelift:::gamesessionqueue/examplequeue123"
      }
   ]
}
```

根据标签查看 Amazon GameLift 车队

您可以使用基于身份的策略中的条件根据标签控制对 Amazon GameLift 资源的访问权限。此示例说明如何创建一个策略，该策略允许在 Owner 标签与用户的用户名匹配时允许查看实例集。此策略还授予在控制台上完成此操作的必要权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListFleetsInConsole",
            "Effect": "Allow",
            "Action": "gamelift:ListFleets",
            "Resource": "*"
        },
        {
            "Sid": "ViewFleetIfOwner",
            "Effect": "Allow",
            "Action": "gamelift:DescribeFleetAttributes",
            "Resource": "arn:aws:gamelift:*:*:fleet/*",
            "Condition": {
```

```
                "StringEquals": {"gamelift:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

在 Amazon S3 中访问游戏构建文件

将游戏服务器与亚马逊集成后 GameLift，将构建文件上传到 Amazon S3。 GameLift 要让 Amazon 访问构建文件，请使用以下策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::bucket-name/object-name"
        }
    ]
}
```

有关上传 Amazon GameLift 游戏文件的更多信息，请参阅[将自定义服务器构建上传到 Amazon GameLift]。

对 Amazon GameLift 身份和访问进行故障排除

使用以下信息来帮助您诊断和修复在使用 Amazon GameLift 和 AWS Identity and Access Management (IAM) 时可能遇到的常见问题。

主题

- [我无权在 Amazon 上执行任何操作 GameLift]
- [我无权执行 iam：PassRole]
- [我想允许我以外的人访问我的 AWS 账户 Amazon GameLift 资源]

我无权在 Amazon 上执行任何操作 GameLift

如果 AWS Management Console 告诉您，您无权执行某个操作，请联系您的 AWS 账户管理员寻求帮助。管理员是向您提供登录凭证的人。

当 `mateojackson` IAM 用户尝试使用控制台查看有关队列的详细信息，但不具有 `gamelift:DescribeGameSessionQueues` 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 gamelift:DescribeGameSessionQueues on resource: examplequeue123
```

在这种情况下，Mateo 请求他的管理员更新其策略，以允许他使用 `gamelift:DescribeGameSessionQueues` 操作读取 `examplequeue123` 资源的访问权限。

我无权执行 iam：PassRole

如果您收到错误消息，说您无权执行该`iam:PassRole`操作，则必须更新您的政策，以允许您将角色传递给亚马逊 GameLift。

有些 AWS 服务 允许将现有角色传递到该服务，而不是创建新服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为的 IAM 用户`marymajor`尝试使用控制台在 Amazon 中执行操作时，会出现以下示例错误 GameLift。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果需要帮助，请联系AWS管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人访问我的 AWS 账户 Amazon GameLift 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACL) 的服务，可以使用这些策略向人员授予对资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon 是否 GameLift 支持这些功能，请参阅亚马逊如何 GameLift 使用 IAM。
- 要了解如何为您拥有的 AWS 账户 中的资源提供访问权限，请参阅 IAM 用户指南 中的为您拥有的另一个 AWS 账户 中的 IAM 用户提供访问权限。
- 要了解如何为第三方 AWS 账户提供您资源的访问权限，请参阅《IAM 用户指南》中的为第三方拥有的 AWS 账户提供访问权限。
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。

Amazon GameLift 的日志记录和监控

监控是保持 Amazon GameLift 和您的 AWS 解决方案的可靠性、可用性和性能的重要方面。您应从 AWS 解决方案的所有部分收集监控数据，以便更轻松地调试出现的多点故障。

AWS 和 Amazon GameLift 提供了若干工具来监控您的游戏托管资源并响应潜在事件。

Amazon CloudWatch 警报

使用 Amazon CloudWatch 警报，您可以在指定时间段内监控某个指标。如果指标超过给定阈值，则会向 Amazon SNS 主题或 AWS Auto Scaling 策略发送通知。当状态发生变化并在指定数量的时间段内保持时，会触发 CloudWatch 警报，而不是在处于特定状态时触发警报。有关更多信息，请参阅使用 Amazon CloudWatch 监控 Amazon GameLift。

AWS CloudTrail 日志

CloudTrail 提供了用户、角色或 AWS 服务在 Amazon GameLift 中所执行操作的记录。使用通过 CloudTrail 收集的信息，您可以确定向 Amazon GameLift 发出了什么请求、发出请求时使用的 IP 地址、何人发出的请求、请求的发出时间以及其他详细信息。有关更多信息，请参阅使用 AWS CloudTrail 记录 Amazon GameLift API 调用。

Amazon 合规性验证 GameLift

Amazon GameLift 不在任何合 AWS 规计划的范围内。

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- 安全与合规性快速入门指南 — 这些部署指南讨论了架构注意事项，并提供了部署以安全性和合规性为重点 AWS 的基准环境的步骤。
- 在 A@@ mazon Web Services 上构建 HIPAA 安全与合规架构 — 本白皮书描述了各公司如何使用 AWS 来创建符合 HIPAA 资格的应用程序。

> **Note**
>
> 并非所有 AWS 服务 人都符合 HIPAA 资格。有关更多信息，请参阅符合 HIPAA 要求的服务参考。

- AWS 合规资源AWS — 此工作簿和指南集可能适用于您所在的行业和所在地区。
- AWS 客户合规指南 — 从合规角度了解责任共担模式。这些指南总结了保护的最佳实践， AWS 服务并将指南映射到跨多个框架（包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO)）的安全控制。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- AWS Security Hub— 这 AWS 服务 提供了您内部安全状态的全面视图 AWS。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控件的列表，请参阅 Security Hub 控件参考。
- Amazon GuardDuty — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 通过满足某些合规性框架规定的入侵检测要求，可以帮助您满足各种合规性要求，例如 PCI DSS。
- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

Amazon 的弹性 GameLift

如果您将 Amazon GameLift FleetiQ 用作亚马逊 EC2 的独立功能，请参阅亚马逊 EC2 用户指南中的亚马逊 EC 2 中的安全。

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。利用可用区，您可以设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

除了 AWS 全球基础设施外，Amazon 还 GameLift 提供以下功能来帮助满足您的数据弹性需求：

- 多区域队列 — Amazon GameLift 游戏会话队列用于使用可用托管资源来放置新的游戏会话。跨多个区域的队列能够在发生区域中断时重定向游戏会话放置。有关创建游戏会话队列的更多信息和最佳实操，请参阅设计游戏会话队列。
- 自动扩展容量-使用 Amazon GameLift 扩展工具维护托管资源的运行状况和可用性。这些工具提供了一系列选项，让您可以根据游戏和玩家的需求调整实例集容量。有关扩展的更多信息，请参阅扩展 Amazon GameLift 托管容量。
- 跨实例分配 — Amazon 根据队列规模将传入流量 GameLift分配到多个实例。作为最佳实操，生产中的游戏应该有多个实例来维护可用性，以防实例变得运行状况不佳或无响应。
- Amazon S3 存储 — 上传到亚马逊 GameLift 的游戏服务器版本和脚本使用标准存储在 Amazon S3 中，标准存储类使用多个数据中心复制来提高弹性。游戏会话日志也使用标准存储类别存储在 Amazon S3 中。

Amazon 的基础设施安全 GameLift

如果您将 Amazon GameLift FleetiQ 用作亚马逊 EC2 的独立功能，请参阅亚马逊 EC2 用户指南中的亚马逊 EC 2 中的安全。

作为一项托管服务，亚马逊 GameLift 受到《亚马逊网络服务：安全流程概述》白皮书中描述的 AWS 全球网络安全程序的保护。

您可以使用 AWS 已发布的 API 调用 GameLift 通过网络访问亚马逊。客户端必须支持传输层安全性（TLS）1.2 或更高版本。建议使用 TLS 1.3 或更高版本。客户端还必须支持具有完全向前保密（PFS）的密码套件，例如 Ephemeral Diffie-Hellman（DHE）或 Elliptic Curve Ephemeral Diffie-Hellman（ECDHE）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

Amazon GameLift 服务将所有舰队放置在亚马逊虚拟私有云 (VPC) 中，这样每个舰队都存在于云中逻辑上隔离的区域中 AWS 。您可以使用 Amazon GameLift 策略来控制来自特定 VPC 终端节点或特

定 VPC 的访问。实际上，这可以将对给定 Amazon GameLift 资源的网络访问与 AWS 网络中的特定 VPC 隔离开来。创建实例集时，需要指定端口号和 IP 地址的范围。这些范围限制了入站流量访问实例集 VPC 上托管游戏服务器的方式。选择实例集访问设置时，请使用标准安全最佳实操。

Amazon 中的配置和漏洞分析 GameLift

如果您将 Amazon GameLift FleetiQ 用作亚马逊 EC2 的独立功能，请参阅亚马逊 EC2 用户指南中的亚马逊 EC 2 中的安全。

配置和 IT 控制是 AWS 和您（我们的客户）之间的共同责任。有关更多信息，请参阅责任 AWS 共担模型。 AWS 处理基本的安全任务，例如客户机操作系统 (OS) 和数据库修补、防火墙配置和灾难恢复。这些流程已通过相应第三方审核和认证。有关详细信息，请参阅以下资源：Amazon Web Services：安全流程概述（白皮书）。

以下安全最佳实践还涉及在 Amazon 中进行配置和漏洞分析 GameLift：

- 客户负责管理部署到用于游戏托管的 Amazon GameLift 实例的软件。具体来说：
  - 应维护客户提供的游戏服务器应用程序软件，包括更新和安全补丁。要更新游戏服务器软件，请将新版本上传到 Amazon GameLift，为其创建新队列，然后将流量重定向到新队列。
  - 基本 Amazon 系统映像 (AMI)（包括操作系统）仅在创建新实例集时更新。要修补、更新和保护作为 AMI 一部分的操作系统和其他应用程序，请定期回收利用实例集，而不考虑游戏服务器更新。
- 客户应考虑使用最新的 SDK 版本定期更新他们的游戏，包括 AWS 软件开发工具包、亚马逊 GameLift 服务器软件开发工具包和适用于实时服务器的亚马逊 GameLift客户端 SDK。

Amazon 安全最佳实践 GameLift

如果您将 Amazon GameLift FleetiQ 用作亚马逊 EC2 的独立功能，请参阅亚马逊 EC2 用户指南中的亚马逊 EC 2 中的安全。

Amazon GameLift 提供了许多安全功能，供您在制定和实施自己的安全策略时考虑。以下最佳实操是一般准则，并不代表完整的安全解决方案。这些最佳实操可能不适合您的环境或不满足您的环境要求，请将其视为有用的考虑因素而不是惯例。

不要打开互联网端口

我们强烈建议不要开放互联网端口，因为这样做会带来安全风险。例如，如果您使用以下UpdateFleetPortSettings方式打开远程桌面端口：

```
{
  "FleetId": "<fleet identifier>",
  "InboundPermissionAuthorizations": [
      {
        "FromPort": 3389,
        "IpRange": "0.0.0.0/0",
        "Protocol": "RDP",
        "ToPort": 3389
      }
  ]
}
```

那么你就允许互联网上的任何人访问该实例。

而是使用特定 IP 地址或地址范围打开端口。例如，像这样：

```
{
  "FleetId": "<fleet identifier>",
  "InboundPermissionAuthorizations": [
      {
        "FromPort": 3389,
        "IpRange": "54.186.139.221/32",
        "Protocol": "TCP",
        "ToPort": 3389
      }
  ]
}
```

了解更多信息

有关如何更安全地使用 Amazon 的 GameLift 更多信息，请参阅AWS Well-Architected Tool 安全支柱。 。

Amazon GameLift 参考指南

本节包含使用 Amazon GameLift 的参考文档。

主题

- Amazon GameLift 服务 API 参考（AWS 软件开发工具包）
- Amazon GameLift 实时服务器参考
- Amazon GameLift 服务器软件开发工具包参考
- 游戏会话放置事件

Amazon GameLift 服务 API 参考（AWS 软件开发工具包）

本主题提供了基于任务的 API 操作列表，这些操作可用于 Amazon GameLift 管理的托管解决方案，包括托管自定义游戏服务器和实时服务器。这些操作被打包到 `aws.gamelift` 命名空间中的 AWS 软件开发工具包中。下载 AWS 软件开发工具包或查看 Amazon GameLift API 参考文档。

该 API 包括两组用于游戏托管的操作：

- 设置和管理 Amazon GameLift 托管资源
- 开始游戏会话并加入玩家行列

Amazon GameLift 服务 API 还包含用于其他 Amazon GameLift 工具和解决方案的操作。有关 FleetIQ API 的列表，请参阅 FleetIQ API 操作。有关用于对战的 FlexMatch API 列表，请参阅 FlexMatch API 操作。

设置和管理 Amazon GameLift 托管资源

调用这些操作为您的游戏服务器配置托管资源，扩展容量以满足玩家需求、访问性能和利用率指标，等等。这些 API 操作与 Amazon GameLift 上托管的游戏服务器配合使用，包括实时服务器。您可以使用 Amazon GameLift 控制台处理大多数资源管理任务，也可以使用 AWS Command Line Interface (AWS CLI) 工具或 AWS 软件开发工具包对服务进行调用。

准备游戏服务器以进行部署

上传并配置游戏的游戏服务器代码，为在托管资源上部署和启动做好准备。

管理自定义游戏服务器构建

- upload-build – 从本地路径上传构建文件并创建新的 Amazon GameLift 构建资源。此操作只能作为 AWS CLI 命令使用，是上传游戏服务器构建的最常用方法。
- CreateBuild – 使用 Amazon S3 存储桶中存储的文件创建新构建。
- ListBuilds – 获取上传到 Amazon GameLift 区域的所有构建的列表。
- DescribeBuild – 检索与某个构建关联的信息。
- UpdateBuild – 更改构建元数据，包括构建名称和版本。
- DeleteBuild – 从 Amazon GameLift 中移除一个构建。

管理实时服务器配置脚本

- CreateScript – 上传 JavaScript 文件并创建新的 Amazon GameLift 脚本资源。
- ListBuilds – 获取上传到 Amazon GameLift 区域的所有实时脚本的列表。
- DescribeScript – 检索与实时脚本关联的信息。
- UpdateScript – 更改脚本元数据并上传修订后的脚本内容。
- DeleteScript – 从 Amazon GameLift 中删除实时脚本。

设置用于托管的计算资源

配置托管资源并将其与游戏服务器构建或实时配置脚本一起构建。

创建和管理实例集

- CreateFleet – 配置和部署新的 Amazon GameLift 计算资源实例集来运行您的游戏服务器。部署后，游戏服务器将按照配置自动启动，随时可以托管游戏会话。
- ListFleets – 获取 Amazon GameLift 区域中所有实例集的列表。
- DeleteFleet – 终止不再运行游戏服务器或托管玩家的实例集。
- 查看/更新实例集位置。
  - CreateFleetLocations – 将远程位置添加到支持多个位置的现有实例集中
  - DescribeFleetLocationAttributes – 获取实例集所有远程位置的列表并查看每个位置的当前状态。
  - DeleteFleetLocations – 从支持多个位置的实例集中移除远程位置。
- 查看/更新实例集配置。

- DescribeFleetAttributes/UpdateFleetAttributes – 查看或更改实例集的元数据和设置，用于游戏会话保护和资源创建限制。
- DescribeFleetPortSettings/UpdateFleetPortSettings – 查看或更改实例集允许的入站权限（IP 地址和端口设置范围）。
- DescribeRuntimeConfiguration/UpdateRuntimeConfiguration – 查看或更改在实例集中的每个实例上运行的服务器进程（以及数量）。

管理实例集容量

- DescribeEC2InstanceLimits – 检索当前 AWS 账户允许的最大实例数和当前使用量级别。
- DescribeFleetCapacity – 检索实例集主区域当前容量设置。
- DescribeFleetLocationCapacity – 检索多位置实例集中每个位置的当前容量设置。
- UpdateFleetCapacity – 手动调整实例集容量设置。
- 设置自动扩缩：
  - PutScalingPolicy – 启用基于目标的自动扩缩，创建自定义自自动扩缩策略，或者更新现有策略。
  - DescribeScalingPolicies – 检索现有的自动扩缩策略。
  - DeleteScalingPolicy – 删除自动扩缩策略并阻止其影响实例集的容量。
  - StartFleetActions – 重新启动实例集的自动扩缩策略。
  - StopFleetActions – 暂停实例集的自动扩缩策略。

监控实例集活动。

- DescribeFleetUtilization – 检索有关当前在实例集上处于活动状态的服务器进程、游戏会话和玩家数量的统计信息。
- DescribeFleetLocationUssivation – 检索多位置实例集中每个位置的利用率统计信息。
- DescribeFleetEvents – 查看在指定的时间范围内实例集的已记录事件。
- DescribeGameSessions – 检索游戏会话元数据，包括游戏的运行时间和当前玩家数量。

设置队列以实现最佳游戏会话位置

设置多实例集、多区域队列，以使用最佳可用托管资源放置游戏会话，从而实现成本、延迟和恢复能力等方面的优势。

- CreateGameSessionQueue – 创建一个队列，在处理游戏会话位置请求时使用。

- DescribeGameSessionQueues – 检索 Amazon GameLift 区域中定义的游戏会话队列。
- UpdateGameSessionQueue – 更改游戏会话队列的配置。
- DeleteGameSessionQueue – 从该区域删除游戏会话队列。

管理别名

使用别名来表示您的实例集，或创建终端替代目标。别名在将游戏活动从一个实例集转换到另一个实例集时非常有用，例如在游戏服务器构建更新期间。

- CreateAlias – 定义新别名并可以选择将其分配给实例集。
- ListAliases – 获取 Amazon GameLift 区域中定义的所有实例集别名。
- DescribeAlias – 检索有关现有别名的信息。
- UpdateAlias – 更改别名的设置，例如将其从一个实例集重定向到另一个实例集。
- DeleteAlias – 从区域中删除别名。
- ResolveAlias – 获取指定的别名指向的实例集 ID。

访问托管实例

查看有关实例集中各个实例的信息，或请求远程访问指定的实例集实例以进行故障排除。

- DescribeInstances – 获取有关实例集中每个实例的信息，包括实例 ID、IP 地址、位置和状态。
- GetInstanceAccess – 请求远程连接到实例集中的指定实例时所需的访问凭证。

设置 VPC 对等连接

在 Amazon GameLift 托管资源与其他 AWS 资源之间创建和管理 VPC 对等连接。

- CreateVpcPeeringAuthorization – 授权与其中一个 VPC 建立对等连接。
- DescribeVpcPeeringAuthorizations – 检索有效的对等连接授权。
- DeleteVpcPeeringAuthorization – 删除对等连接授权。
- CreateVpcPeeringConnection – 在 Amazon GameLift 实例集的 VPC 和您的一个 VPC 之间建立对等连接。
- DescribeVpcPeeringConnections – 检索与 Amazon GameLift 实例集建立的活动或待处理 VPC 对等连接的信息。
- DeleteVpcPeeringConnection – 删除与 Amazon GameLift 实例集建立的 VPC 对等连接。

开始游戏会话并加入玩家行列

通过游戏客户端服务调用这些操作即可开始新的游戏会话、获取有关现有游戏会话的信息以及加入玩家的游戏会话。这些操作适用于与在 Amazon GameLift 上托管的自定义游戏服务器配合使用。如果您使用的是实时服务器，请使用实时服务器客户端 API (C#) 参考管理游戏会话。

- 为一个或多个玩家启动新游戏会话。
  - StartGameSessionPlacement – 让 Amazon GameLift 查找可用的最佳托管资源并开始新的游戏会话。这是创建新游戏会话的首选方法。它依靠游戏会话队列来跟踪多个地区的托管可用性，并使用 FleetIQ 算法根据玩家延迟、托管成本、位置等对展示位置进行优先排序。
  - DescribeGameSessionPlacement – 获取有关位置请求的详细信息和状态。
  - StopGameSessionPlacement – 取消位置请求。
  - CreateGameSession – 在一个特定实例集上启动一个新游戏会话。此操作使您可以更好地控制从何处开始游戏会话，而不必使用 FleetIQ 来评估放置选项。您必须通过单独的步骤将玩家添加到新游戏会话中。
- 使玩家进入现有游戏会话。查找具有可用玩家位置的正在运行的游戏会话，并为新玩家预留位置。
  - CreatePlayerSession – 预留一个开放位置供一个玩家加入游戏会话。
  - CreatePlayerSessions – 预留多个开放位置供多个玩家加入游戏会话。
- 处理游戏会话和玩家会话数据。管理游戏会话和玩家会话信息。
  - SearchGameSessions – 根据一组搜索条件请求活跃的游戏会话列表。
  - DescribeGameSessions – 检索特定游戏会话的元数据，包括活跃时长和当前玩家数量。
  - DescribeGameSessionDetails – 检索一个或多个游戏会话的元数据，包括游戏会话保护设置。
  - DescribePlayerSessions – 获取有关玩家活动的详细信息，包括状态、游戏时间和玩家数据。
  - UpdateGameSession – 更改游戏会话设置，例如最大玩家数和加入政策。
  - GetGameSessionLogUrl – 获取游戏会话的已保存日志的位置。

Amazon GameLift 实时服务器参考

本节包含 Amazon GameLift 实时服务器软件开发工具包的参考文档。它包括实时客户端 API 以及对配置实时服务器脚本的指导。

主题

- 实时服务器客户端 API (C#) 参考
- Amazon GameLift 实时服务器脚本参考

实时服务器客户端 API (C#) 参考

使用实时客户端 API 准备与 Amazon GameLift 实时服务器配合使用的多人游戏客户端。有关集成过程的更多信息，请参阅准备实时服务器。客户端 API 包含一组同步 API 调用和异步回调，使游戏客户端能够连接到实时服务器，并通过该服务器与其他游戏客户端交换消息和数据。

此 API 在以下库中定义：

Client.cs

- 同步操作
- 异步回调
- 数据类型

设置实时客户端 API

1. 下载 Amazon GameLift 实时客户端软件开发工具包。
2. 构建 C# 软件开发工具包库。找到解决方案文件 `GameLiftRealtimeClientSdkNet45.sln`。有关最低要求和附加构建选项，请参阅 C# 服务器软件开发工具包的 `README.md` 文件。在 IDE 中，加载解决方案文件。要生成软件开发工具包库，请还原 NuGet 程序包并构建解决方案。
3. 将实时客户端库添加到游戏客户端项目。

实时服务器客户端 API (C#) 参考：操作

使用此 C# 实时客户端 API 参考可帮助您准备多人游戏，以便与 Amazon GameLift 实例集上部署的实时服务器配合使用。有关集成过程的详细信息，请参阅 准备实时服务器。

- 同步操作
- 异步回调
- 数据类型

Client()

初始化新客户端以与实时服务器通信，并确定要使用的连接类型。

语法

```
public Client(ClientConfiguration configuration)
```

参数

clientConfiguration

指定客户端/服务器连接类型的配置详细信息。您可以选择不使用此参数而调用 Client()；但这种方法默认情况下会导致不安全的连接。

类型：ClientConfiguration

必需：否

返回值

返回用于与实时服务器通信的实时客户端的实例。

Connect()

请求与托管游戏会话的服务器进程的连接。

语法

```
public ConnectionStatus Connect(string endpoint, int remoteTcpPort, int listenPort,
 ConnectionToken token)
```

参数

端点

要连接到的游戏会话的 DNS 名称和 IP 地址。端点在 GameSession 对象中指定，它作为对 AWS 软件开发工具包 Amazon GameLift API 操作 StartGameSessionPlacement、CreateGameSession 或 DescribeGameSessions 请求客户端调用的响应返回。

> **Note**
>
> 如果实时服务器在具有 TLS 证书的实例集中运行，则必须使用 DNS 名称。

类型：字符串

必需：是

remoteTcpPort

分配给游戏会话的 TCP 连接的端口号。此信息在 GameSession 对象中指定，作为对 StartGameSessionPlacement、CreateGameSession 或 DescribeGameSession 请求的响应返回。

类型：整数

有效值：1900 到 2000。

必需：是

listenPort

游戏客户端侦听使用 UDP 通道发送的消息的端口号。

类型：整数

有效值：33400 到 33500。

必需：是

token

标识向服务器进程请求游戏客户端的可选信息。

类型：ConnectionToken

必需：是

返回值

返回指示客户端连接状态的 ConnectionStatus 枚举值。

Disconnect()

连接到游戏会话后，断开游戏客户端与游戏会话的连接。

语法

```
public void Disconnect()
```

参数

此操作没有参数。

返回值

此方法不会返回任何内容。

NewMessage()

使用指定的操作代码创建新的消息对象。返回消息对象后，通过指定目标、更新传递方法和根据需要添加数据负载来完成消息内容。完成后，使用 SendMessage() 发送消息。

语法

```
public RTMessage NewMessage(int opCode)
```

参数

opCode

标识游戏事件或操作（例如，玩家移动或服务器通知）的开发人员定义操作代码。

类型：整数

必需：是

返回值

返回包含指定操作代码和默认传递方法的 RTMessage 对象。传递意图参数默认设置为 FAST。

SendMessage()

使用指定的传递方法向玩家或组发送消息。

语法

```
public void SendMessage(RTMessage message)
```

参数

message

指定目标收件人、传递方法和消息内容的消息对象。

类型：RTMessage

必需：是

返回值

此方法不会返回任何内容。

JoinGroup()

将玩家添加到指定组的成员资格。组可以包含连接到游戏的任何玩家。加入后，玩家会收到发送给该组的所有未来消息，并可以向整个组发送消息。

语法

```
public void JoinGroup(int targetGroup)
```

参数

targetGroup

标识要添加玩家的组的唯一 ID。组 ID 由开发人员定义。

类型：整数

必需：是

返回值

此方法不会返回任何内容。由于此请求是使用可靠 (TCP) 传递方法发送的，因此失败的请求会触发回调OnError()。

LeaveGroup()

从指定组的成员资格中删除玩家。不再在该组中后，玩家不会收到发送给该组的消息，并且无法向整个组发送消息。

语法

```
public void LeaveGroup(int targetGroup)
```

参数

targetGroup

标识要删除玩家的组的唯一 ID。组 ID 由开发人员定义。

类型：整数

必需：是

返回值

此方法不会返回任何内容。由于此请求是使用可靠 (TCP) 传递方法发送的，因此失败的请求会触发回调OnError()。

RequestGroupMembership()

请求将指定组中的玩家列表发送给游戏客户端。任何玩家均可请求此信息，无论其是否为该组的成员。作为对此请求的响应，成员资格列表通过OnGroupMembershipUpdated()回调发送给客户端。

语法

```
public void RequestGroupMembership(int targetGroup)
```

参数

targetGroup

标识要获取成员资格信息的组的唯一 ID。组 ID 由开发人员定义。

类型：整数

必需：是

返回值

此方法不会返回任何内容。

实时服务器客户端 API (C#) 参考：异步回调

使用此 C# 实时客户端 API 参考可帮助您准备多人游戏，以便与 Amazon GameLift 实例集上部署的实时服务器配合使用。有关集成过程的详细信息，请参阅 准备实时服务器。

- 同步操作
- 异步回调
- 数据类型

游戏客户端需要实施这些回调方法来响应事件。实时服务器调用这些回调以将游戏相关信息发送到游戏客户端。相同事件的回调也可使用实时服务器脚本中的自定义游戏逻辑实施。请参阅实时服务器的脚本回调。

回调方法在 `ClientEvents.cs` 中定义。

OnOpen()

当服务器进程接受游戏客户端的连接请求并打开连接时调用。

语法

```
public void OnOpen()
```

参数

此方法未采用任何参数。

返回值

此方法不会返回任何内容。

OnClose()

当服务器进程终止与游戏客户端的连接时（例如，在游戏会话结束后）调用。

语法

```
public void OnClose()
```

参数

此方法未采用任何参数。

返回值

此方法不会返回任何内容。

OnError()

当实时客户端 API 请求发生故障时调用。可以自定义此回调以处理各种连接错误。

语法

```
private void OnError(byte[] args)
```

参数

此方法未采用任何参数。

返回值

此方法不会返回任何内容。

OnDataReceived()

当游戏客户端收到来自实时服务器的消息时调用。这是游戏客户端接收消息和通知的主要方法。

语法

```
public void OnDataReceived(DataReceivedEventArgs dataReceivedEventArgs)
```

参数

dataReceivedEventArgs

与消息活动相关的信息。

类型：DataReceivedEventArgs

必需：是

返回值

此方法不会返回任何内容。

OnGroupMembershipUpdated()

当玩家所属组的成员资格已更新时调用。客户端调用 RequestGroupMembership 时也会调用此回调。

语法

```
public void OnGroupMembershipUpdated(GroupMembershipEventArgs groupMembershipEventArgs)
```

参数

groupMembershipEventArgs

与组成员资格活动相关的信息。

类型：GroupMembershipEventArgs

必需：是

返回值

此方法不会返回任何内容。

实时服务器客户端 API (C#) 参考：数据类型

使用此 C# 实时客户端 API 参考可帮助您准备多人游戏，以便与 Amazon GameLift 实例集上部署的实时服务器配合使用。有关集成过程的详细信息，请参阅 准备实时服务器。

- 同步操作
- 异步回调
- 数据类型

ClientConfiguration

有关如何将游戏客户端连接到实时服务器的信息。

目录

ConnectionType

要使用的客户端/服务器连接类型，可以是安全的或不安全的。如果您未指定连接类型，默认值为不安全。

> **Note**
>
> 当使用 TLS 证书连接到安全实例集上的实时服务器时，必须使用值 RT_OVER_WSS_DTLS_TLS12。

类型：一个 `ConnectionType` 枚举值。

必需：否

ConnectionToken

有关请求与实时服务器连接的游戏客户端和/或玩家的信息。

目录

PlayerSessionId

创建新玩家会话时由 Amazon GameLift 颁发的唯一 ID。玩家会话 ID 在 `PlayerSession` 对象中指定，作为对 GameLift API 操作 StartGameSessionPlacement、CreateGameSession、DescribeGameSessionPlacement 或 DescribePlayerSessions 请求客户端调用的响应返回。

类型：字符串

必需：是

payload

连接时将传递给实时服务器的开发人员定义的信息。这包括可能用于自定义登录机制的任何任意数据。例如，在允许客户端连接之前，负载可提供实时服务器脚本要处理的身份验证信息。

类型：字节数组

必需：否

RTMessage

消息的内容和传递信息。消息必须指定目标玩家或目标组。

目录

opCode

标识游戏事件或操作（例如，玩家移动或服务器通知）的开发人员定义操作代码。消息的操作代码为所提供的数据负载提供上下文。使用 NewMessage() 创建的消息已具有操作代码集，但它可以随时进行更改。

类型：整数

必需：是

targetPlayer

标识作为所发送消息预期收件人的玩家的唯一 ID。目标可以是服务器本身（使用服务器 ID）或其他玩家（使用玩家 ID）。

类型：整数

必需：否

targetGroup

标识作为所发送消息预期收件人的组的唯一 ID。组 ID 由开发人员定义。

类型：整数

必需：否

deliveryIntent

指示使用可靠 TCP 连接还是使用快速 UDP 通道发送消息。使用NewMessage()创建的消息。

类型：DeliveryIntent 枚举

有效值：FAST | RELIABLE

必需：是

payload

消息内容。此信息根据游戏客户端基于随附操作代码处理的需要进行结构化。它可能包含游戏状态数据或者在游戏客户端之间或在游戏客户端与实时服务器之间进行通信所需的其他信息。

类型：字节数组

必需：否

DataReceivedEventArgs

通过OnDataReceived()回调提供的数据。

目录

发件人

标识发起消息的实体的唯一 ID（玩家 ID 或服务器 ID）。

类型：整数

必需：是

opCode

标识游戏事件或操作（例如，玩家移动或服务器通知）的开发人员定义操作代码。消息的操作代码为所提供的数据负载提供上下文。

类型：整数

必需：是

data

消息内容。此信息根据游戏客户端基于随附操作代码处理的需要进行结构化。它可能包含游戏状态数据或者在游戏客户端之间或在游戏客户端与实时服务器之间进行通信所需的其他信息。

类型：字节数组

必需：否

GroupMembershipEventArgs

通过OnGroupMembershipUpdated()回调提供的数据。

目录

发件人

标识请求组成员资格更新的玩家的唯一 ID。

类型：整数

必需：是

opCode

标识游戏事件或操作的开发人员定义操作代码。

类型：整数

必需：是

groupId

标识作为所发送消息预期收件人的组的唯一 ID。组 ID 由开发人员定义。

类型：整数

必需：是

playerId

作为指定组当前成员的玩家 ID 列表。

类型：整数数组

必需：是

枚举

为实时客户端软件开发工具包定义的枚举定义如下：

ConnectionStatus

- CONNECTED – 游戏客户端仅通过 TCP 连接连接到实时服务器。无论传递意图如何，所有消息都通过 TCP 发送。
- CONNECTED_SEND_FAST – 游戏客户端通过 TCP 和 UDP 连接连接到实时服务器。但是，通过 UDP 接收消息的功能尚未验证；因此，发送到游戏客户端的所有消息均使用 TCP。
- CONNECTED_SEND_AND_RECEIVE_FAST – 游戏客户端通过 TCP 和 UDP 连接连接到实时服务器。游戏客户端可以使用 TCP 或 UDP 发送和接收消息。
- CONNECTING – 游戏客户端已发送连接请求，实时服务器正在进行处理。
- DISCONNECTED_CLIENT_CALL – 游戏客户端与实时服务器断开连接以响应来自游戏客户端的Disconnect()请求。
- DISCONNECTED – 游戏客户端因客户端断开连接调用以外的原因与实时服务器断开连接。

ConnectionType

- RT_OVER_WSS_DTLS_TLS12 – 安全连接类型。

  适用于在 GameLift 实例集上运行并生成 TLS 证书的实时服务器。当使用安全连接时，TCP 流量使用 TLS 1.2 进行加密，而 UDP 流量使用 DTLS 1.2 进行加密。
- RT_OVER_WS_UDP_UNSECURED – 非安全连接类型。

- RT_OVER_WEBSOCKET – 非安全连接类型。此值不再是首选。

DeliveryIntent

- FAST – 使用 UDP 通道传递。
- RELIABLE – 使用 TCP 连接传递。

Amazon GameLift 实时服务器脚本参考

使用这些资源在实时脚本中构建自定义逻辑。

主题

- 实时服务器的脚本回调
- 实时服务器接口

实时服务器的脚本回调

通过在脚本中实施这些回调，您可提供自定义逻辑来响应事件。

init

初始化 Server 并接收 Realtime Server 接口。

语法

```
init(rtsession)
```

onMessage

当收到的消息发送到服务器时调用。

语法

```
onMessage(gameMessage)
```

onHealthCheck

调用它可设置游戏会话运行状况。默认情况下，运行状况是正常（或 `true`）。可以实施此回调来执行自定义运行状况检查并返回状态。

语法

```
onHealthCheck()
```

onStartGameSession

在新游戏会话启动时调用，并传入一个游戏会话对象。

语法

```
onStartGameSession(session)
```

onProcessTerminate

在 服务终止服务器进程时调用。这可以用作从游戏会话中完全退出的触发器。无需调用 `processEnding()`.

语法

```
onProcessTerminate()
```

onPlayerConnect

当玩家请求连接并通过初始验证时调用。

语法

```
onPlayerConnect(connectMessage)
```

onPlayerAccepted

当接受玩家连接时调用。

语法

```
onPlayerAccepted(player)
```

onPlayerDisconnect

当玩家通过发送断开连接请求或通过其他方式断开与游戏会话的连接时调用。

语法

```
onPlayerDisconnect(peerId)
```

onProcessStarted

当启动服务器进程时调用。此回调允许脚本执行准备托管游戏会话所需的任何自定义任务。

语法

```
onProcessStarted(args)
```

onSendToPlayer

当服务器上从一个玩家接收的消息要传递给另一个玩家时调用。此进程在传递消息之前运行。

语法

```
onSendToPlayer(gameMessage)
```

onSendToGroup

当服务器上从一个玩家接收的消息要传递给一个组时调用。此进程在传递消息之前运行。

语法

```
onSendToGroup(gameMessage))
```

onPlayerJoinGroup

当玩家发送加入组的请求时调用。

语法

```
onPlayerJoinGroup(groupId, peerId)
```

onPlayerLeaveGroup

当玩家发送离开组的请求时调用。

语法

```
onPlayerLeaveGroup(groupId, peerId)
```

实时服务器接口

当实时脚本初始化时，将返回到实时服务器的接口。本主题介绍了可通过接口使用的属性和方法。了解有关编写实时脚本的更多信息，并查看创建实时脚本中的详细脚本示例。

该实时接口提供对以下对象的访问：

- 会话
- 玩家
- 游戏消息
- 配置

实时会话对象

使用这些方法来访问与服务器相关的信息和执行与服务器相关的操作。

getPlayers()

检索当前连接到游戏的玩家的对等连接 ID 列表。返回一组玩家对象。

语法

```
rtSession.getPlayers()
```

broadcastGroupMembershipUpdate()

触发向玩家组传输更新后的组成员资格列表。指定要广播的成员资格 (groupIdToBroadcast) 和接收更新的组 (targetGroupId)。组 ID 必须为正整数或“-1”以表示所有组。有关用户定义的组 ID 的示例，请参阅实时服务器脚本示例。

语法

```
rtSession.broadcastGroupMembershipUpdate(groupIdToBroadcast, targetGroupId)
```

getServerId()

检索服务器的唯一对等连接 ID 标识符，这用于将消息路由到服务器。

语法

```
rtSession.getServerId()
```

getAllPlayersGroupId()

检索默认组的组 ID，该组包含当前连接到游戏会话的所有玩家。

语法

```
rtSession.getAllPlayersGroupId()
```

ProcessEnding()

触发实时服务器来终止游戏服务器。必须从实时脚本调用此函数才能从游戏会话中完全退出。

语法

```
rtSession.processEnding()
```

GetGameSessionId()

检索当前正在运行的游戏会话的唯一 ID。

语法

```
rtSession.getGameSessionId()
```

getLogger()

检索用于日志记录的接口。使用此项来记录将捕获到游戏会话日志中的语句。日志记录程序支持使用“info”、“warn”和“error”语句。例如：`logger.info("<string>")`。

语法

```
rtSession.getLogger()
```

sendMessage()

通过 UDP 通道将使用 `newTextGameMessage` 或 `newBinaryGameMessage` 创建的消息从实时服务器发送给玩家收件人。使用玩家的对等连接 ID 确认接收方。

语法

```
rtSession.sendMessage(gameMessage, targetPlayer)
```

sendGroupMessage()

通过 UDP 通道将使用 newTextGameMessage 或 newBinaryGameMessage 创建的消息从实时服务器发送给玩家组中的所有玩家。组 ID 必须为正整数或“-1”以表示所有组。有关用户定义的组 ID 的示例，请参阅实时服务器脚本示例。

语法

```
rtSession.sendGroupMessage(gameMessage, targetGroup)
```

sendReliableMessage()

通过 TCP 通道将使用 newTextGameMessage 或 newBinaryGameMessage 创建的消息从实时服务器发送给玩家收件人。使用玩家的对等连接 ID 确认接收方。

语法

```
rtSession.sendReliableMessage(gameMessage, targetPlayer)
```

sendReliableGroupMessage()

通过 TCP 通道将使用 newTextGameMessage 或 newBinaryGameMessage 创建的消息从实时服务器发送给玩家组中的所有玩家。组 ID 必须为正整数或“-1”以表示所有组。有关用户定义的组 ID 的示例，请参阅实时服务器脚本示例。

语法

```
rtSession.sendReliableGroupMessage(gameMessage, targetGroup)
```

newTextGameMessage()

创建包含文本的新消息，使用 SendMessage 函数将该消息从服务器发送到玩家接收方。消息格式类似于在 Realtime 客户端开发工具包中使用的格式（请参阅 RTMessage）。返回 gameMessage 对象。

语法

```
rtSession.newTextGameMessage(opcode, sender, payload)
```

newBinaryGameMessage()

创建包含二进制数据的新消息，该消息使用 SendMessage 函数，从服务器发送到玩家接收方。消息格式类似于在 Realtime 客户端开发工具包中使用的格式（请参阅 RTMessage）。返回 gameMessage 对象。

语法

```
rtSession.newBinaryGameMessage(opcode, sender, binaryPayload)
```

玩家对象

访问与玩家相关的信息。

player.peerId

游戏客户端连接到实时服务器并加入游戏会话时分配给游戏客户端的唯一 ID。

player.playerSessionId

游戏客户端连接到实时服务器并加入游戏会话时引用的玩家会话 ID。

游戏消息对象

使用这些方法访问实时服务器接收到的消息。从游戏客户端收到的消息具有 RTMessage 结构。

getPayloadAsText()

以文本形式获取游戏消息有效载荷。

语法

```
gameMessage.getPayloadAsText()
```

gameMessage.opcode

消息中包含的操作代码。

gameMessage.payload

消息中包含的有效负载。可以是文本或二进制数据。

gameMessage.sender

发送消息的游戏客户端的对等连接 ID。

gameMessage.reliable

布尔值，指示通过 TCP (true) 还是 UDP (false) 发送消息。

配置对象

配置对象可用于覆盖默认配置。

configuration.maxPlayers

实时服务器可以接受的最大客户端/服务器连接数。

默认值为 32。

configuration.pingIntervalTime

服务器尝试向所有连接的客户端发送 ping 命令以验证连接是否正常的时间间隔（以毫秒为单位）。

默认值为 3000ms。

Amazon GameLift 服务器软件开发工具包参考

本节包含 Amazon GameLift 服务器软件开发工具包的参考文档。使用服务器软件开发工具包集成您的自定义游戏服务器，以便与 Amazon GameLift 服务进行通信。

主题

- 适用于 C++ 的软件开发工具包 参考
- 适用于 C# 的 Amazon GameLift 服务器软件开发工具包 参考
- 适用于 Go 的 Amazon GameLift 服务器软件开发工具包 参考
- Unreal Engine 的 Amazon GameLift 服务器软件开发工具包 参考

适用于 C++ 的软件开发工具包 参考

您可以使用这份 Amazon GameLift C++ 服务器软件开发工具包 参考来帮助您为多人游戏做好准备，以便在 Amazon GameLift 上使用。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

主题

- 适用于 C++ 的 Amazon GameLift 服务器软件开发工具包 5.x 参考
- Amazon GameLift C++ 服务器软件开发工具包 3.x 参考

适用于 C++ 的 Amazon GameLift 服务器软件开发工具包 5.x 参考

此 C++ 服务器 API 参考可帮助您准备通过 使用多人游戏。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

> **Note**
>
> 本主题介绍使用 C++ 标准库 () 构建时可以使用的 Amazon GameLift C++ API。`std`具体而言，本文档适用于使用该`-DGAMELIFT_USE_STD=1`选项编译的代码。

主题

- 亚马逊 GameLift 服务器 SDK (C++) 5.x 参考：操作
- 亚马逊 GameLift 服务器软件开发工具包 (C++) 参考：数据类型

亚马逊 GameLift 服务器 SDK (C++) 5.x 参考：操作

您可以使用此 Amazon GameLift C++ 服务器 SDK 参考来帮助您准备在亚马逊上使用的多人游戏 GameLift。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

> **Note**
>
> 本主题介绍使用 GameLift C++ 标准库 (`std`) 构建时可以使用的亚马逊 C++ API。具体而言，本文档适用于使用该`-DGAMELIFT_USE_STD=1`选项编译的代码。

操作

- GetSdkVersion()
- InitSDK()
- InitSDK()
- ProcessReady()
- ProcessReadyAsync()

- ProcessEnding()
- ActivateGameSession()
- UpdatePlayerSessionCreationPolicy()
- GetGameSessionId()
- GetTerminationTime()
- AcceptPlayerSession()
- RemovePlayerSession()
- DescribePlayerSessions()
- StartMatchBackfill()
- StopMatchBackfill()
- GetComputeCertificate()
- GetFleetRoleCredentials()
- Destroy

GetSdkVersion()

返回当前内置到服务器进程中的开发工具包的版本号。

语法

```
Aws::GameLift::AwsStringOutcome Server::GetSdkVersion();
```

返回值

如果成功，返回以 the section called “AwsStringOutcome” 对象返回当前软件开发工具包的版本。返回的字符串仅包含版本号 (例如：如果不成功，将返回错误消息。

示例

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
 Aws::GameLift::Server::GetSdkVersion();
```

InitSDK()

为托管 EC2 队列初始化 Amazon GameLift 开发工具包。在启动时调用此方法，然后再进行任何其他与 Amazon GameLift 相关的初始化。此方法从主机环境读取服务器参数，以设置服务器与 Amazon GameLift 服务之间的通信。

语法

```
Server::InitSDKOutcome Server::initSdkOutcome = InitSDK();
```

返回值

返回一个the section called “initsdk结果”对象，该对象指示服务器进程是否已准备好调用ProcessReady()。

示例

```
//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
Aws::GameLift::Server::InitSDKOutcome initSdkOutcome =
  Aws::GameLift::Server::InitSDK();
```

InitSDK()

为Anywhere队列初始化 Amazon GameLift 软件开发工具包。在启动时调用此方法，然后再进行任何其他与 Amazon GameLift 相关的初始化。此方法需要明确的服务器参数来设置服务器和 Amazon GameLift 服务之间的通信。

语法

```
Server::InitSDKOutcome Server::initSdkOutcome = InitSDK(serverParameters);
```

参数

ServerParameters

要在 Amazon GameLift Anywhere 舰队上初始化游戏服务器，请使用以下信息构造一个`ServerParameters`对象：

- WebSocket 用于连接到您的游戏服务器的 URL。
- 用于托管游戏服务器的进程的 ID。
- 托管游戏服务器进程的计算的 ID。
- 包含您的亚马逊 GameLift Anywhere计算的亚马逊 GameLift 舰队的 ID。
- Amazon GameLift 操作生成的授权令牌。

返回值

返回一个the section called “initsdk结果”对象，该对象指示服务器进程是否已准备好调用ProcessReady()。

> **Note**
>
> 如果对部署到 InitSDK() Anywhere 队列的游戏版本调用失败，请检查创建编译资源时使用的ServerSdkVersion参数。您必须将此值明确设置为正在使用的服务器软件开发工具包 版本。此参数的默认值为 4.x，这不兼容。要解决此问题，请创建新版本并将其部署到新队列。

示例

亚马逊 GameLift Anywhere示例

```
//Define the server parameters
std::string websocketUrl = "wss://us-west-1.api.amazongamelift.com";
std::string processId = "PID1234";
std::string fleetId = "arn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa";
std::string hostId = "HardwareAnywhere";
std::string authToken = "1111aaaa-22bb-33cc-44dd-5555eeee66ff";
Aws::GameLift::Server::Model::ServerParameters serverParameters =
  Aws::GameLift::Server::Model::ServerParameters(webSocketUrl, authToken, fleetId,
 hostId, processId);

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
Aws::GameLift::Server::InitSDKOutcome initSdkOutcome =
 Aws::GameLift::Server::InitSDK(serverParameters);
```

ProcessReady()

通知 Amaz GameLift on 服务器进程已准备好托管游戏会话。调用后调用InitSDK()此方法。每个进程只能调用一次此方法。

语法

```
GenericOutcome ProcessReady(const Aws::GameLift::Server::ProcessParameters
&processParameters);
```

参数

processParameters

ProcessParameters 对象，用于传输有关服务器进程的以下信息：

- 游戏服务器代码中实现的回调方法的名称，Amazon GameLift 服务调用这些方法与服务器进程进行通信。
- 服务器进程正在侦听的端口号。
- 您希望 Ama GameLift zon 捕获和存储的所有游戏会话特定文件的路径。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例说明 ProcessReady() 调用和委派函数实施。

```
// 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()

通知 Amazon GameLift 服务服务器进程已准备好托管游戏会话。在服务器进程准备好托管游戏会话后，应调用此方法。这些参数指定 Amazon 在某些情况下 GameLift 要调用的回调函数名称。游戏服务器代码必须执行这些函数。

此调用为异步操作。要进行同步调用，请使用 ProcessReady()。有关更多信息，请参阅初始化服务器进程。

语法

```
GenericOutcomeCallable ProcessReadyAsync(
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

参数

processParameters

ProcessParameters 对象，用于传输有关服务器进程的以下信息：

- 游戏服务器代码中实现的回调方法的名称，Amazon GameLift 服务调用这些方法与服务器进程进行通信。
- 服务器进程正在侦听的端口号。
- 您希望 Ama GameLift zon 捕获和存储的所有游戏会话特定文件的路径。

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
// 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()

通知 Amaz GameLift on 服务器进程即将终止。在完成所有其他清理任务（包括关闭活动游戏会话）之后和终止该进程之前，调用此方法。根据的结果`ProcessEnding()`，进程以成功 (0) 或错误 (-1) 退出并生成队列事件。如果流程因错误而终止，则生成的舰队事件为`SERVER_PROCESS_TERMINATED_UNHEALTHY`。

语法

```
Aws::GameLift::GenericOutcome processEndingOutcome =
 Aws::GameLift::Server::ProcessEnding();
```

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例在使用成功或错误退出代码终止服务器进程`Destroy()`之前调`ProcessEnding()`用和。

```
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()

通知 Amazon GameLift 服务器进程已激活游戏会话，现在可以接收玩家连接了。此操作应作为`onStartGameSession()`回调函数的一部分，在所有游戏会话初始化已完成之后调用。

语法

```
Aws::GameLift::GenericOutcome activateGameSessionOutcome =
 Aws::GameLift::Server::ActivateGameSession();
```

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例显示了 ActivateGameSession() 作为 onStartGameSession() 委派函数的一部分调用。

```
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()

更新当前游戏会话接受新玩家会话的能力。可将游戏会话设置为接受或拒绝所有新的玩家会话。

语法

```
GenericOutcome
 UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCreationPolicy
 newPlayerSessionPolicy);
```

参数

playerCreationSession政策

类型：一个 PlayerSessionCreationPolicy 枚举值。

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例将当前游戏会话的接入策略设为接受所有玩家。

```
Aws::GameLift::GenericOutcome outcome =

 Aws::GameLift::Server::UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCr
```

GetGameSessionId()

检索当前由服务器进程托管的游戏会话的 ID (如果服务器进程处于活动状态)。

对于未通过游戏会话激活的空闲进程，该调用将返回the section called “GameLiftError”。

语法

```
AwsStringOutcome GetGameSessionId()
```

参数

此操作没有参数。

返回值

如果成功，以 the section called “AwsStringOutcome” 对象返回游戏会话 ID。如果不成功，将返回错误消息。

对于未通过游戏会话激活的空闲进程，调用返回 `Success = True` 和 `GameSessionId = ""`。

示例

```
Aws::GameLift::AwsStringOutcome sessionIdOutcome =
 Aws::GameLift::Server::GetGameSessionId();
```

GetTerminationTime()

如果终止时间可用，则返回安排服务器进程关闭的时间。服务器进程在收到 Amazon 的`onProcessTerminate()`回调后采取行动 GameLift。Amazon GameLift 打电话`onProcessTerminate()`的原因如下：

- 当服务器进程报告运行状况不佳或未对 Amazon 做出响应时 GameLift。
- 在缩减事件期间终止实例时。
- 当实例因竞价型实例中断而终止时。

语法

```
AwsDateTimeOutcome GetTerminationTime()
```

返回值

如果成功，则以AwsDateTimeOutcome对象形式返回终止时间。该值是终止时间，以此后经过的刻度表示。0001 00:00:00例如，日期时间值等2020-09-13 12:26:40 -000Z于637355968000000000刻度。如果没有可用的终止时间，将返回一条错误消息。

如果流程未收到 ProcessParameters. OnProcessTerminate() 回调，返回错误消息。有关关闭服务器进程的更多信息，请参阅回应服务器进程关闭通知。

示例

```
Aws::GameLift::AwsLongOutcome TermTimeOutcome =
 Aws::GameLift::Server::GetTerminationTime();
```

AcceptPlayerSession()

通知 Amazon GameLift 具有指定玩家会话 ID 的玩家已连接到服务器进程，需要验证。Amazon 会 GameLift 验证玩家会话 ID 是否有效。玩家会话经过验证后，Amazon 会将玩家位置的状态从 “已保留” GameLift 更改为 “活跃”。

语法

```
GenericOutcome AcceptPlayerSession(String playerSessionId)
```

参数

playerSessionId

创建新玩家会话 GameLift 时由 Amazon 颁发的唯一 ID。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例处理的连接请求包括验证和拒绝无效的玩家会话 ID。

```
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()

通知 Amazon GameLift 有玩家已断开与服务器进程的连接。作为回应，Amazon将玩家位置 GameLift 更改为可用。

语法

```
GenericOutcome RemovePlayerSession(String playerSessionId)
```

参数

playerSessionId

创建新玩家会话 GameLift 时由 Amazon 颁发的唯一 ID。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
Aws::GameLift::GenericOutcome disconnectOutcome =
 Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

DescribePlayerSessions()

检索玩家会话数据，包括设置、会话元数据和玩家数据。使用此方法获取有关以下内容的信息：

- 单人游戏会话
- 游戏会话中的所有玩家会话
- 与单个玩家 ID 关联的所有玩家会话

语法

```
DescribePlayerSessionsOutcome DescribePlayerSessions(DescribePlayerSessionsRequest
 describePlayerSessionsRequest)
```

参数

DescribePlayerSessionsRequest

the section called “DescribePlayerSessionsRequest” 对象描述要检索的玩家会话。

返回值

如果成功，将返回一个 the section called “DescribePlayerSessionsOutcome” 对象，包含一组与请求参数相匹配的玩家会话对象。

示例

此示例演示了将所有玩家会话主动连接到指定游戏会话的请求。通过省略限制值NextToken并将其设置为 10，Amazon 会 GameLift 返回与请求匹配的前 10 条玩家会话记录。

```
// 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()

发送请求以为使用 FlexMatch 创建的游戏会话中的开放位置查找新玩家。有关更多信息，请参阅FlexMatch 回填功能。

此操作为异步操作。如果匹配了新玩家，Amazon会使用回调函数 GameLift 提供更新的匹配器数据OnUpdateGameSession()。

服务器进程每次只能具有一个活动的匹配回填请求。要发送新请求，请先调用 StopMatchBackfill() 以取消原始请求。

语法

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest
 startBackfillRequest);
```

参数

StartMatchBackfillRequest

用于传达以下信息的 StartMatchBackfillRequest 对象：

- 要分配给回填请求的票证 ID。此信息是可选的；如果未提供编号，Amazon GameLift 将生成一个。
- 要将请求发送到的对战构建器。需要完整的配置 ARN。该值位于游戏会话的对战构建器数据中。
- 要回填的游戏会话的 ID。
- 游戏会话的当前玩家的可用对战数据。

返回值

返回带有匹配回填票证 ID 的 StartMatchBackfillOutcome 对象或包含错误消息的失败。

示例

```
// 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()

取消使用 创建的活动对战回填请求。有关更多信息，请参阅FlexMatch回填功能。

语法

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

参数

StopMatchBackfillRequest

标识要取消的配对门票的 StopMatchBackfillRequest 对象：

- 要分配给回填请求的票证 ID
- 回填请求所发送到的对战构建器
- 与回填请求关联的游戏会话

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
// 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()

检索 TLS 证书的路径，该证书用于加密您的亚马逊 GameLift Anywhere计算资源与亚马逊之间的网络连接。 GameLift当您将计算设备注册到 Amazon GameLift Anywhere 队列时，您可以使用证书路径。有关更多信息，请参阅RegisterCompute。

语法

```
GetComputeCertificateOutcome Server::GetComputeCertificate()
```

返回值

返回 the section called “GetComputeCertificateOutcome”。

示例

```
Aws::GameLift::GetComputeCertificateOutcome certificate =
 Aws::GameLift::Server::GetComputeCertificate();
```

GetFleetRoleCredentials()

检索授权 Amazon GameLift 与其他AWS 服务角色互动的 IAM 角色证书。有关更多信息，请参阅与您的实例集中的其他 AWS 资源进行通信。

语法

```
GetFleetRoleCredentialsOutcome GetFleetRoleCredentials(GetFleetRoleCredentialsRequest
 request);
```

参数

GetFleetRoleCredentialsRequest

返回值

返回 the section called “GetFleetRoleCredentialsOutcome” 对象。

示例

```
// 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);
```

此示例说明如何使用可选RoleSessionName值为凭证会话分配名称以进行审计。如果您不提供角色会话名称，则使用默认值“*[fleet-id]-[host-id]*”。

```
// 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

从内存中释放 Amazon GameLift 游戏服务器 SDK。作为最佳实操，请在终止进程之后ProcessEnding()和之前调用此方法。如果您使用的是 Anywhere 队列，并且没有在每次游戏会

话后终止服务器进程，请先致电Destroy()然后InitSDK()重新初始化，然后再通知 A GameLift mazon 该进程已准备好与之托管游戏会话。ProcessReady()

语法

```
GenericOutcome Aws::GameLift::Server::Destroy();
```

参数

没有参数。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
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);
}
```

亚马逊 GameLift 服务器软件开发工具包 (C++) 参考：数据类型

您可以使用此 Amazon GameLift C++ 服务器 SDK 参考来帮助您准备在亚马逊上使用的多人游戏 GameLift。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

> **Note**
>
> 本主题介绍在使用 GameLift C++ 标准库 (std) 进行构建时可以使用的亚马逊 C++ API。具体而言，本文档适用于使用该-DDGAMELIFT_USE_STD=1选项编译的代码。

数据类型

- LogParameters
- ProcessParameters
- UpdateGameSession
- GameSession
- ServerParameters
- StartMatchBackfillRequest
- 玩家
- DescribePlayerSessionsRequest
- StopMatchBackfillRequest
- AttributeValue
- GetFleetRoleCredentialsRequest
- AwsLongOutcome
- AwsStringOutcome
- DescribePlayerSessionsOutcome
- DescribePlayerSessionsResult
- GenericOutcome
- GenericOutcomeCallable
- PlayerSession
- StartMatchBackfillOutcome
- StartMatchBackfillResult
- GetComputeCertificateOutcome
- GetComputeCertificateResult
- GetFleetRoleCredentialsOutcome
- GetFleetRoleCredentialsResult
- initsdk结果
- GameLiftError
- 枚举

LogParameters

一个对象，用于标识游戏会话期间生成的文件，您希望 Amazon GameLift 在游戏会话结束后上传和存储这些文件。游戏服务器在ProcessReady()调用中 GameLift 作为ProcessParameters对象的一部分提供LogParameters给 Amazon。

属性	描述
LogPaths	您希望 Amazon GameLift 存储以备将来访问的游戏服务器日志文件的目录路径列表。服务器进程在每个游戏会话期间生成这些文件。文件路径和名称在游戏服务器中定义并存储在根游戏构建目录中。 日志路径必须是绝对的。例如，如果您的游戏构建在类似于 `MyGame\sessionLogs\` 的路径中存储游戏会话日志，则日志路径将为 (在 Windows 实例上) 或 (在 Linux 实例上)。 类型：`std:vector<std::string>` 必需：否

ProcessParameters

此数据类型包含在 a 中发送给 Amazon GameLift 的一组参数ProcessReady()。

属性	描述
LogParameters	一个对象，其中包含游戏会话期间生成的文件的目录路径。Amazon 会 GameLift 复制并存储文件以备将来访问。 类型：`Aws::GameLift::Server:: LogParameters` 必需：否

OnHealthCheck	Amazon 为向服务器 GameLift 进程请求健康状态报告而调用的回调函数。Amazon 每 60 秒 GameLift 调用一次此函数，并等待 60 秒等待响应。TRUE如果运行状况良好，则服务器进程会返回，FALSE如果不健康，则返回。如果未返回任何响应，Amazon 会将服务器进程 GameLift 记录为运行状况不佳。 类型：`std::function<bool()> onHealthCheck` 必需：否
OnProcessTerminate	Amazon 为强制关闭服务器进程而 GameLift 调用的回调函数。调用此函数后，Amazon 会等待 5 分钟， GameLift 等待服务器进程关闭并通过ProcessEnding()调用进行响应，然后再关闭服务器进程。 类型：`std::function<void()> onProcessTerminate` 必需：是
OnRefreshConnection	Amazon 为刷新与游戏 GameLift 服务器的连接而调用的回调函数的名称。 类型：`void OnRefreshConnectionDelegate()` 必需：是

OnStartGameSession	Amazon 为激活新游戏 GameLift 会话而调用的回调函数。Amazon GameLift 调用此函数是为了响应客户请求CreateGameSession。回调函数传递一个GameSession对象，如亚马逊 GameLift API 参考中所定义。 类型：`const std::function<void (Aws::GameLift::Model::GameSession)> onStartGameSession` 必需：是
OnUpdateGameSession	Amazon GameLift 调用的回调函数，用于将更新的游戏会话对象传递给服务器进程。当已处理匹配回填请求以提供更新的匹配器数据时，Amazon会 GameLift 调用此函数。它传递一个GameSession对象、一个状态更新 (`updateReason`) 和匹配回填票证 ID。 类型：`std::function<void(Aws::GameLift::Server::Model::UpdateGameSession)> onUpdateGameSession` 必需：否
端口	服务器进程用于侦听新玩家连接的端口号。该值必须在部署此游戏服务器构建的任意实例集上所配置的端口范围内。此端口号包含在游戏会话和玩家会话对象中，游戏会话在连接到服务器进程时使用。 类型：`Integer` 必需：是

UpdateGameSession

此数据类型更新为游戏会话对象，其中包括更新游戏会话的原因以及相关的回填票证 ID（如果使用回填来填充游戏会话中的玩家会话）。

属性	描述
GameSession	由亚马逊 GameLift API 定义的GameSession对象。该GameSession 对象包含描述游戏会话的属性。 类型：`Aws::GameLift::Server::GameSession` 必需：是
UpdateReason	更新游戏会话的原因。 类型：`Aws::GameLift::Server::UpdateReason` 必需：是
BackfillTicketId	尝试更新游戏会话的回填票证的 ID。 类型：`std::string` 必需：否

GameSession

此数据类型提供游戏会话的详细信息。

属性	描述
GameSessionId	会话的唯一标识符。游戏会话ARN具有以下格式:. `arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token>`

属性	描述
	类型：std::string 必需：否
名称	游戏会话的描述性标签。 类型：std::string 必需：否
FleetId	运行游戏会话的集的唯一标识符。 类型：std::string 必需：否
MaximumPlayerSessionCount	机群中已连接到游戏会话的玩家数量。 类型：int 必需：否
端口	游戏会话的端口号。要连接到 Amazon GameLift 游戏服务器，应用程序需要 IP 地址和端口号。 类型：in 必需：否
IpAddress	游戏会话的 IP 地址。要连接到 Amazon GameLift 游戏服务器，应用程序需要 IP 地址和端口号。 类型：std::string 必需：否

属性	描述
GameSessionData	一组自定义游戏会话属性，采用单个字符串值格式。 类型：`std::string` 必需：否
MatchmakerData	有关用于创建游戏会话的对战过程的信息，采用 JSON 语法，格式为字符串。除了使用的对战配置外，它还包含分配到对战的所有玩家的数据，包括玩家属性和队伍分配。 类型：`std::string` 必需：否
GameProperties	游戏会话的一组自定义属性，采用键值对格式。这些属性将通过启动新游戏会话的请求传递到游戏服务器进程。 类型：`std :: vector < GameProperty >` 必需：否

属性	描述
DnsName	分配给正在运行游戏会话的实例的 DNS 标识符。CRL 采用以下格式： • 支持 TLS 的实例集:. `<unique identifier>.<region identifier>.amazongamelift.com` • 未启用 TLS 的实例集:. `ec2-<unique identifier>.compute.amazonaws.com` 连接到在支持 TLS 的队列上运行的游戏会话时，必须使用 DNS 名称，而不是 IP 地址。 类型：`std::string` 必需：否

ServerParameters

用于维护 Amazon GameLift Anywhere 舰队上的游戏服务器与 Amazon GameLift 服务之间连接的信息。使用启动新的服务器进程时会使用此信息InitSDK()。对于托管在 Amazon GameLift 托管 EC2 实例上的服务器，请使用空对象。

属性	描述
webSocketUrl	当您获取`GameLiftServerSdkEndpoint` 亚马逊 GameLift Anywhere计算资源时，`RegisterCompute` Amazon 会 GameLift 返回。 类型：`std::string` 必需：是
ProcessID	注册到托管游戏的服务器进程的唯一标识符。

属性	描述
	类型：`std::string` 必需：是
hostId	`HostID`是在您注册计算机时`ComputeNa me` 使用的。有关更多信息，请参阅RegisterC ompute。 类型：`std::string` 必需：是
FleetId	计算注册到的实例集的唯一标识符。有关更多信息，请参阅RegisterCompute。 类型：`std::string` 必需：是
AuthToken	由亚马逊生成的身份验证令牌 GameLift ，用于向亚马逊 GameLift验证您的服务器。有关更多信息，请参阅GetComputeAuthToken。 类型：`std::string` 必需：是

StartMatchBackfillRequest

用于创建对战回填请求的信息。游戏服务器通过StartMatchBackfill()电话将此信息传达给 Amazon GameLift 。

属性	描述
GameSessionArn	游戏会话的唯一标识符。此 API 操作 `GetGameSessionId` 返回采用 ARN 格式的标识符。

属性	描述
	类型：`std::string` 必需：是
MatchmakingConfigurationArn	采用 ARN 格式的唯一标识符，适用于用于此请求的对战构建器。要查找用于创建原始游戏会话的对战构建器，请查看对战构建器数据属性中的游戏会话对象。在使用对战构建器数据中了解有关对战构建器数据的更多信息。 类型：`std::string` 必需：是
玩家	一组表示当前正在游戏会话中的所有玩家的数据。对战构建器使用此信息搜索与当前玩家非常匹配的新玩家。 类型：`std::vector<Player>` 必需：是
TicketId	对战或匹配回填请求票证的唯一标识符。如果您不提供值，Amazon GameLift 会生成一个值。使用此标识符可跟踪匹配回填票证状态或取消请求 (如需要)。 类型：`std::string` 必需：否

玩家

此数据类型代表对战中的玩家。发起对战请求时，玩家会有玩家 ID、属性，可能还有延迟数据。比赛结束后，Amazon 会 GameLift 添加球队信息。

属性	描述
LatencyInMS	一组以毫秒为单位的值，表示玩家在连接到某个位置时所经历的延迟量。 如果使用此属性，则仅匹配列出的位置的玩家。如果对战构建器具有评估玩家延迟的规则，玩家必须报告延迟才能进行匹配。 类型：`Dictionary<string,int>` 必需：否
PlayerAttributes	键/值对的集合，其中包含用于对战的玩家信息。玩家属性键必须与配对规则集中 PlayerAttributes 使用的属性密钥相匹配。 有关玩家属性的更多信息，请参阅Attribute Value。 类型：`std::map<std::string,AttributeValue>` 必需：否
PlayerId	玩家的唯一标识符。 类型：`std::string` 必需：否
团队	玩家在对战中被分配到的团队的名称。在对战规则集中定义团队名称。 类型：`std::string` 必需：否

DescribePlayerSessionsRequest

一个对象，用于指定要检索哪些玩家会话。服务器进程通过DescribePlayerSessions()调用 Amazon 来提供这些信息 GameLift。

属性	描述
GameSessionId	游戏会话的唯一标识符。使用此参数可请求指定游戏会话的所有玩家会话。 游戏会话 ID 的格式为`arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>` 。GameSessionID 是自定义 ID 字符串或 类型：`std::string` 必需：否
PlayerSessionId	玩家会话的唯一标识符。使用此参数请求单个特定的玩家会话。 类型：`std::string` 必需：否
PlayerId	玩家的唯一标识符。使用此参数可请求指定游戏会话的所有玩家会话。请参阅 生成玩家 ID。 类型：`std::string` 必需：否
PlayerSessionStatusFilter	用于筛选结果的玩家会话状态。可能的玩家会话状态包括以下内容： • RESERVED - 已收到玩家会话请求，但玩家尚未连接到服务器进程和/或进行验证。 • ACTIVE - 服务器进程已验证玩家，当前已连接。

属性	描述
	• COMPLETED - 玩家连接已断开。 • TIMEDOUT - 收到了玩家会话请求，但玩家未连接和/或在超时限制 (60 秒) 内验证。 类型：`std::string` 必需：否
NextToken	令牌指示结果的下一个连续页面的开头。要指定结果集的开始，请不要指定值。如果指定玩家会话 ID，将忽略此参数。 类型：`std::string` 必需：否
限制	要返回的最大结果数量。如果指定玩家会话 ID，将忽略此参数。 类型：`int` 必需：否

StopMatchBackfillRequest

用于取消对战回填请求的信息。游戏服务器通过StopMatchBackfill()呼叫将此信息传送给 Amazon GameLift 服务。

属性	描述
GameSessionArn	与被取消的请求关联的唯一游戏会话标识符。 类型：`char[]` 必需：否
MatchmakingConfigurationArn	此请求发送到的对战构建器的唯一标识符。

属性	描述
	类型：char[] 必需：否
TicketId	要取消的回填请求票证的唯一标识符。 类型：char[] 必需：否

AttributeValue

在玩家属性密钥-值对中使用。此对象允许您使用任何有效的数据类型来指定属性值：字符串、数字、字符串数组或数据映射。每个AttributeValue对象必须只使用一个可用属性：S、NSL、或SDM。

属性	描述
AttrType	指定属性值的类型。可能的属性值类型包括： • NONE • string • DOUBLE • 字符串列表 • 字符串_DOUBLE_MAP 必需：否
S	表示字符串属性值。 类型：std::string 必需：否
否	表示属性值。 类型：double

属性	描述
	必需：否
sl	表示字符串属性值的数组。 类型：`std::vector<std::string>` 必需：否
SDM	表示字符串键和双精度值的字典。 类型：`std::map<std::string, double>` 必需：否

GetFleetRoleCredentialsRequest

这种数据类型使游戏服务器只能有限地访问您的其他AWS资源。有关更多信息，请参阅为亚马逊设置 IAM 服务角色 GameLift。

属性	描述
RoleArn	可以扩展对资源的有限访问权限的服务角色的 Amazon 资源名称（ARN）。AWS 类型：`std::string` 必需：否
RoleSessionName	可用于唯一标识会话的角色会AWS Security Token ServiceAssumeRole话名称。此名称会显示在审计日志中，例如中的日志 CloudTrail。 类型：`std::string` 必需：否

AwsLongOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型：`long` 必需：否
ResultWithOwnership	操作的结果，转换为右值，以便调用代码可以拥有该对象的所有权。 类型：`long&&` 必需：否
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “GameLiftError” 必需：否

AwsStringOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型：`std::string`

属性	描述
	必需：否
ResultWithOwnership	操作的结果，转换为右值，以便调用代码可以拥有该对象的所有权。 类型：`long&&` 必需：否
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “GameLiftError” 必需：否

DescribePlayerSessionsOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型：the section called “DescribePlayerSessionsResult” 必需：否
ResultWithOwnership	操作的结果，转换为右值，以便调用代码可以拥有该对象的所有权。

属性	描述
	类型：Aws::GameLift::Server::Model::DescribePlayerSessionsResult&& 必需：否
成功	操作是否成功。 类型：bool 必需：是
错误	操作失败时发生的错误。 类型：the section called “GameLiftError” 必需：否

DescribePlayerSessionsResult

一组对象，其中包含与请求相匹配的每个玩家会话的属性。

属性	描述
NextToken	令牌指示结果的下一个连续页面的开头。使用之前的调用返回此操作的令牌。要指定结果集的开始，请不要指定值。如果指定玩家会话 ID，将忽略此参数。 类型：std::string 必需：是
PlayerSessions	类型：IList<the section called “PlayerSession” > 必填

属性	描述
ResultWithOwnership	操作的结果，转换为右值，以便调用代码可以拥有该对象的所有权。 类型：`std::string&&` 必需：否
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “GameLiftError” 必需：否

GenericOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “GameLiftError” 必需：否

GenericOutcomeCallable

这种数据类型是一种异步的通用结果。它具有以下属性：

属性	描述
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “GameLiftError” 必需：否

PlayerSession

此数据类型表示 Amazon GameLift 传递给游戏服务器的玩家会话。有关更多信息，请参阅PlayerSession。

属性	描述
CreationTime	类型：`long` 必需：否
FleetId	类型：`std::string` 必需：否
GameSessionId	类型：`std::string` 必需：否
IpAddress	类型：`std::string` 必需：否

属性	描述
PlayerData	类型：`std::string` 必需：否
PlayerId	类型：`std::string` 必需：否
PlayerSessionId	类型：`std::string` 必需：否
端口	类型：`int` 必需：否
Status	用于筛选结果的玩家会话状态。如果提供 PlayerId 了 o PlayerSessionId r，则 PlayerSessionStatusFilter 对响应没有影响。 类型：`PlayerSessionStatus` 枚举。可能的值包括： • ACTIVE (处于活动状态) • COMPLETED • NOT_SET • 预留 • 超时 必需：否
TerminationTime	类型：`long` 必需：否

属性	描述
DnsName	类型：`std::string` 必需：否

StartMatchBackfillOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型：the section called “StartMatchBackfillResult” 必需：否
ResultWithOwnership	操作的结果，转换为右值，以便调用代码可以拥有该对象的所有权。 类型：`StartMatchBackfillResult&&` 必需：否
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “GameLiftError” 必需：否

StartMatchBackfillResult

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
TicketId	对战配置的唯一标识符。如果此处未指定票证 ID，Amazon GameLift 将以 UUID 的形式生成一个。使用此标识符跟踪对战回填单状态并检索匹配结果。 类型：`std::string` 必需：否

GetComputeCertificateOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型：the section called “GetComputeCertificateResult” 必需：否
ResultWithOwnership	操作的结果，转换为右值，以便调用代码可以拥有该对象的所有权。 类型：`Aws::GameLift::Server::Model::GetComputeCertificateResult&&` 必需：否
成功	操作是否成功。 类型：`bool`

属性	描述
	必需：是
错误	操作失败时发生的错误。 类型：the section called “GameLiftError” 必需：否

GetComputeCertificateResult

计算机上 TLS 证书的路径和计算机的主机名。

属性	描述
CertificatePath	计算资源上 TLS 证书的路径。使用 Amazon GameLift 托管队列时，此路径包含： • `certificate.pem` ：最终用户证书。完整的证书链是`certificateChain.pem` 附加到此证书的组合。 • `certificateChain.pem` ：包含根证书和中间证书的证书链。 • 根证书。 • `privateKey.pem` ：最终用户证书的私钥。 类型：`std::string` 必需：否
ComputeName	您的计算资源的名称。 类型：`std::string` 必需：否

GetFleetRoleCredentialsOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型：the section called “GetFleetRoleCredentialsResult” 必需：否
ResultWithOwnership	操作的结果，转换为右值，以便调用代码可以拥有该对象的所有权。 类型：`Aws::GameLift::Server::Model::GetFleetRoleCredentialsResult` 必需：否
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “GameLiftError” 必需：否

GetFleetRoleCredentialsResult

属性	描述
AccessKeyId	用于对您的AWS资源进行身份验证和提供访问权限的访问密钥 ID。

属性	描述
	类型：`string` 必需：否
AssumedRoleId	服务角色所属的用户 ID。 类型：`string` 必需：否
AssumedRoleUserArn	用户应承担的角色的 Amazon 资源名称 (ARN)。 类型：`string` 必需：否
过期	您的会话凭证到期之前的时间。 类型：`DateTime` 必需：否
SecretAccessKey	指定 S3 验证的访问密钥 ID。 类型：`string` 必需：否
SessionToken	用于识别与您的AWS资源交互的当前活动会话的令牌。 类型：`string` 必需：否
成功	操作是否成功。 类型：`bool` 必需：是

属性	描述
错误	操作失败时发生的错误。 类型：the section called “GameLift 错误” 必需：否

initsdk结果

> **Note**
>
> `InitSDKOutcome`仅在使用`std`标志构建软件开发工具包 时才会返回。如果您使用`nostd`标志进行构建，则会改the section called “GenericOutcome”为返回。

属性	描述
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “GameLiftError” 必需：否

GameLiftError

属性	描述
ErrorType	错误的类型。 类型：**`GameLiftErrorType`** 枚举。

属性	描述
	必需：否
ErrorName	错误类型的名称。 类型：`std::string` 必需：否
ErrorMessage	错误消息。 类型：`std::string` 必需：否

枚举

为亚马逊 GameLift 服务器软件开发工具包 (C++) 定义的枚举定义如下：

GameLiftErrorType

表示错误类型的字符串值。有效值包括：

- 错误的请求_异常
- GAMESESSION_ID_NOT_SET – 尚未设置游戏会话 ID。
- 内部服务异常
- LOCAL_CONNECTION_FAILED — 与亚马逊的本地连接失败。 GameLift
- NETWORK_NOT_INITIALIZED – 网络尚未初始化。
- SERVICE_CALL_FAILED – 对服务的调用失败。AWS
- 网络套接字连接失败
- 禁止网络套接字连接失败
- WEBSOCKET_CONNECT_FAILURE_INVALID_UR
- WEBSOCKET_CONNECT_FAILURE_
- 已初始化 — Amazon GameLift 服务器或客户端已使用初始化 () 进行初始化。
- FLEET_MISMATCH – 目标实例集与 GameSession 或 PlayerSession 的实例集不匹配。
- GAMELIFT_CLIENT_NOT_INITIALIZED — 亚马逊客户端尚未初始化。 GameLift

- GAMELIFT_SERVER_NOT_INITIALIZED — 亚马逊服务器尚未初始化。 GameLift
- GAME_SESSION_ENDED_FAILED — A GameLift mazon Server SDK 无法联系服务以报告游戏会话已结束。
- GAME_SESSION_NOT_READY — GameLift 亚马逊服务器游戏会话未激活。
- GAME_SESSION_READY_FAILED — A GameLift mazon Server SDK 无法联系服务以报告游戏会话已准备就绪。
- INITI@@ ALIZATION_MISMATCH – 在 Server::In itialization () 之后调用了客户端方法，反之亦然。
- NOT_INITIAL IZED — Amazon GameLift 服务器或客户端尚未使用初始化 () 进行初始化。
- NO_TARGET_ALIASID_SET – 尚未设置目标 AliasID。
- NO_TARGET_FLEET_SET – 尚未设置目标实例集。
- PROCESS@@ _ENDING_FAILED — A GameLift mazon Server SDK 无法联系服务部门报告流程即将结束。
- PROCESS_NOT_AC TIVE — 服务器进程尚未处于活动状态，未绑定到 GameSession，也无法接受或处理。 PlayerSessions
- PROCESS_NOT_READ Y – 服务器进程尚未准备好激活。
- PROCESS@@ _READY_FAILED — Amazon S GameLift erver SDK 无法联系服务部门报告流程已准备就绪。
- SDK_VERSION_DETECTION_FAILED – 软件开发工具包 版本检测失败。
- STX_CALL_FAILED – 对 xstX 服务器后端组件的调用失败。
- STX_INITIALIZATION_FAILED – xSTX 服务器后端组件无法初始化。
- E@@ XPRENTED_PLAYER_SESSION – 服务器遇到了未注册的玩家会话。
- 网络套接字连接失败
- 禁止网络套接字连接失败
- WEBSOCKET_CONNECT_FAILURE_INVALID_UR
- WEBSOCKET_CONNECT_FAILURE_
- WEBSOCKET_RETRIABLE_SEND_MESSAGE_FAILURE — 向服务发送消息时可重试失败。 GameLift WebSocket
- WEBSOCKET_SEND_MESSAGE_FAILURE — 无法向服务发送消息。 GameLift WebSocket
- MATCH_BACKFILL_REQUEST_VALIDATION – 请求验证失败。
- PLAYER_SESSION_REQUEST_VALIDATION – 请求验证失败。

PlayerSessionCreationPolicy

用于指示游戏会话是否接受新玩家的字符串值。有效值包括：

- ACCEPT_ALL - 接受所有新玩家会话。
- DENY_ALL - 拒绝所有新玩家会话。
- NOT_SET – 游戏会话未设置为接受或拒绝新玩家会话。

Amazon GameLift C++ 服务器软件开发工具包 3.x 参考

您可以使用这份 Amazon GameLift C++ 服务器软件开发工具包 3.x 参考来帮助您为多人游戏做好准备，以便在 Amazon GameLift 上使用。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

主题

- Amazon GameLift 服务器软件开发工具包 (C++) 参考：操作
- Amazon GameLift 服务器软件开发工具包 (C++) 参考：数据类型

Amazon GameLift 服务器软件开发工具包 (C++) 参考：操作

您可以使用这份 Amazon GameLift C++ 服务器软件开发工具包 参考来帮助您为多人游戏做好准备，以便在 Amazon GameLift 上使用。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

操作

- AcceptPlayerSession()
- ActivateGameSession()
- DescribePlayerSessions()
- GetGameSessionId()
- GetInstanceCertificate()
- GetSdkVersion()
- GetTerminationTime()
- InitSDK()
- ProcessEnding()
- ProcessReady()

- ProcessReadyAsync()
- RemovePlayerSession()
- StartMatchBackfill()
- StopMatchBackfill()
- TerminateGameSession()
- UpdatePlayerSessionCreationPolicy()
- Destroy

AcceptPlayerSession()

通知 服务，具有所指定玩家会话 ID 的玩家已连接到服务器进程，需要进行验证。 需要确保玩家会话 ID 有效，即该玩家 ID 在游戏会话中有保留的玩家位置。通过验证后， 将玩家位置的状态从 RESERVED 更改为 ACTIVE。

语法

```
GenericOutcome AcceptPlayerSession(const std::string& playerSessionId);
```

参数

PlayerSessionId

Amazon GameLift 服务为响应调用软件开发工具包 Amazon GameLift API 操作 createPlayerSessi AWS on 而颁发的唯一身份证。连接到服务器进程时，游戏客户端会引用此 ID。

类型：std:: string

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例演示了处理连接请求的函数，包括验证和拒绝无效的玩家会话 ID。

```
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()

通知 服务，服务器进程已启动游戏会话，现在已准备好接收玩家连接。此操作应作为 onStartGameSession() 回调函数的一部分，在所有游戏会话初始化已完成之后调用。

语法

```
GenericOutcome ActivateGameSession();
```

参数

此操作没有参数。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例显示作为 onStartGameSession() 回调函数的一部分调用 ActivateGameSession()。

```
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()

检索玩家会话数据，包括设置、会话元数据和玩家数据。使用此操作获取单个玩家会话的信息、游戏会话中所有玩家会话的信息或者与单个玩家 ID 相关联的所有玩家会话的信息。

语法

```
DescribePlayerSessionsOutcome DescribePlayerSessions (
    const Aws::GameLift::Server::Model::DescribePlayerSessionsRequest
 &describePlayerSessionsRequest);
```

参数

describePlayerSessionsRequest

DescribePlayerSessionsRequest 对象描述要检索的玩家会话。

必需：是

返回值

如果成功，将返回一个 DescribePlayerSessionsOutcome 对象，包含一组与请求参数相匹配的玩家会话对象。玩家会话对象与 API PlayerSession 数据类型具有相同的结构。

示例

此示例演示了将所有玩家会话主动连接到指定游戏会话的请求。通过忽略 NextToken 和将 值设置为 10， 将返回符合请求条件的前 10 条玩家会话记录。

```
// 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()

检索当前由服务器进程托管的游戏会话的唯一标识符 (如果服务器进程处于活动状态)。该标识符以 ARN 格式返回：arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>。

对于尚未通过游戏会话激活的空闲进程，该调用返回 Success = True 和 GameSessionId ="" (空字符串) 。

语法

```
AwsStringOutcome GetGameSessionId();
```

参数

此操作没有参数。

返回值

如果成功，以 AwsStringOutcome 对象返回游戏会话 ID。如果不成功，将返回错误消息。

示例

```
Aws::GameLift::AwsStringOutcome sessionIdOutcome =
    Aws::GameLift::Server::GetGameSessionId();
```

GetInstanceCertificate()

检索与队列及其实例关联的 PEM 编码 TLS 证书的文件位置。 AWS Certificate Manager当您在证书配置设置为 GENERATED 的情况下创建新队列时，将生成此证书。使用此证书可与游戏客户端建立安全连接并加密客户端/服务器通信。

语法

```
GetInstanceCertificateOutcome GetInstanceCertificate();
```

参数

此操作没有参数。

返回值

如果成功，则返回一个 GetInstanceCertificateOutcome 对象，该对象包含实例集的 TLS 证书文件的位置（该证书文件存储在实例上）。从证书链中提取的根证书文件也存储在实例上。如果不成功，将返回错误消息。

有关证书和证书链数据的更多信息，请参阅 AWS Certificate Manager API 参考中的获取证书响应元素。

示例

```
Aws::GameLift::GetInstanceCertificateOutcome certificateOutcome =
```

```
    Aws::GameLift::Server::GetInstanceCertificate();
```

GetSdkVersion()

返回正在使用中的软件开发工具包的当前版本号。

语法

```
AwsStringOutcome GetSdkVersion();
```

参数

此操作没有参数。

返回值

如果成功，返回以 AwsStringOutcome 对象返回当前软件开发工具包的版本。返回的字符串仅包含版本号 (例如：如果不成功，将返回错误消息。

示例

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
    Aws::GameLift::Server::GetSdkVersion();
```

GetTerminationTime()

如果终止时间可用，则返回安排服务器进程关闭的时间。收到来自 服务的 回调后，服务器进程将执行此操作。Amazon GameLift 可能出于以下原因onProcessTerminate()调用：(1) 当服务器进程报告运行状况不佳或未响应 Amazon GameLift 时，(2) 在缩小规模事件期间终止实例时，或 (3) 由于竞价中断而终止实例时。

如果该进程已收到 onProcessTerminate() 回调，则返回的值是估计的终止时间 (纪元秒)。如果进程未收到onProcessTerminate()回调，则会返回一条错误消息。了解有关关闭服务器进程的更多信息。

语法

```
AwsLongOutcome GetTerminationTime();
```

参数

此操作没有参数。

返回值

如果成功，则以AwsLongOutcome对象形式返回终止时间。该值是终止时间，以自 0001 00:00:00 起经过的报价表示。例如，日期时间值 2020-09-13 12:26:40-000Z 等于 637355968000000000 滴答作响。如果没有可用的终止时间，将返回一条错误消息。

示例

```
Aws::GameLift::AwsLongOutcome TermTimeOutcome =
    Aws::GameLift::Server::GetTerminationTime();
```

InitSDK()

初始化 Amazon GameLift 软件开发工具包。应在启动之后、进行任何其他 相关的初始化之前调用此方法。

语法

```
InitSDKOutcome InitSDK();
```

参数

此操作没有参数。

返回值

如果成功，返回 InitSdkOutcome 对象，指示服务器进程已准备好调用 ProcessReady()。

示例

```
Aws::GameLift::Server::InitSDKOutcome initOutcome =
    Aws::GameLift::Server::InitSDK();
```

ProcessEnding()

通知 服务，该服务器进程正在关闭。应在所有其他清除任务 (包括关闭所有活动游戏会话) 之后调用此方法。此方法应退出，退出代码为 0；非零退出代码将导致生成一条事件消息，提示进程未完全退出。

当该方法以代码为 0 退出后，您可以使用成功的退出代码终止该进程。您也可以退出，并返回错误代码。如果您退出时出现错误代码，则队列事件将指示进程异常终止（SERVER_PROCESS_TERMINATED_UNHEALTHY）。

语法

```
GenericOutcome ProcessEnding();
```

参数

此操作没有参数。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
Aws::GameLift::GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
if (outcome.Success)
    exit(0);  // exit with success
// otherwise, exit with error code
exit(errorCode);
```

ProcessReady()

通知 服务，服务器进程已准备好托管游戏会话。在成功调用 InitSDK() 并完成了服务器进程托管游戏会话所需的全部设置任务后，应调用此方法。每个进程只能调用一次此方法。

此调用为同步操作。要进行异步调用，请使用 ProcessReadyAsync()。有关更多信息，请参阅 初始化服务器进程。

语法

```
GenericOutcome ProcessReady(
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

参数

processParameters

ProcessParameters 对象，用于传输有关服务器进程的以下信息：

- 回调方法的名称，在游戏服务器代码中实现， 服务调用它来与服务器进程通信。
- 服务器进程正在侦听的端口号。
- 您希望 捕获和存储的任何游戏会话特定文件的路径。

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例演示了 ProcessReady() 调用和回调函数的实现。

```
// 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()

通知 服务，服务器进程已准备好托管游戏会话。在服务器进程准备好托管游戏会话后，应调用此方法。该方法的参数用于指定 在特定环境下应调用的回调函数的名称。游戏服务器代码必须执行这些函数。

此调用为异步操作。要进行同步调用，请使用 ProcessReady()。有关更多信息，请参阅 初始化服务器进程。

语法

```
GenericOutcomeCallable ProcessReadyAsync(
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

参数

processParameters

ProcessParameters 对象，用于传输有关服务器进程的以下信息：

- 回调方法的名称，在游戏服务器代码中实现， 服务调用它来与服务器进程通信。
- 服务器进程正在侦听的端口号。
- 您希望 捕获和存储的任何游戏会话特定文件的路径。

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
// 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()

通知 服务，具有所指定玩家会话 ID 的玩家已从服务器进程断开连接。作为响应， 将玩家位置更改为可用，这使得该玩家位置可以分配给新玩家。

语法

```
GenericOutcome RemovePlayerSession(
    const std::string& playerSessionId);
```

参数

PlayerSessionId

Amazon GameLift 服务为响应调用软件开发工具包 Amazon GameLift API 操作 createPlayerSessi AWS on 而颁发的唯一身份证。连接到服务器进程时，游戏客户端会引用此 ID。

类型：std:: string

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
Aws::GameLift::GenericOutcome disconnectOutcome =
    Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

StartMatchBackfill()

发送请求以为使用 创建的游戏会话中的开放位置查找新玩家。另请参阅 AWS 软件开发工具包操作 AWSStartMatchBackfill()。通过此操作，托管游戏会话的游戏服务器进程可以发出匹配回填请求。在中了解有关 FlexMatch 回填功能的更多信息。

此操作为异步操作。如果成功匹配了新玩家，则 服务会使用回调函数 提供更新的对战构建器数据。

服务器进程每次只能具有一个活动的匹配回填请求。要发送新请求，请先调用 StopMatchBackfill() 以取消原始请求。

语法

```
StartMatchBackfillOutcome StartMatchBackfill (
    const Aws::GameLift::Server::Model::StartMatchBackfillRequest
 &startBackfillRequest);
```

参数

StartMatchBackfillRequest

一个 StartMatchBackfillRequest 对象，用于传递以下信息：

- 要分配给回填请求的票证 ID。此信息是可选的；如果未提供任何 ID，则 将自动生成一个 ID。
- 要将请求发送到的对战构建器。需要完整的配置 ARN。可从游戏会话的对战构建器数据中获得此值。
- 正在进行回填的游戏会话的 ID。
- 游戏会话的当前玩家的可用对战数据。

必需：是

返回值

返回带有对战回填票证的 StartMatchBackfillOutcome 对象或包含错误消息的失败。使用 AWS 软件开发工具包操作 AWSDescribeMatchmaking() 可跟踪票证状态。

示例

```
// 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()

取消使用 StartMatchBackfill() 创建的活动对战回填请求。另请参阅 AWS 软件开发工具包操作 AWSStopMatchmaking()。在中了解有关 FlexMatch 回填功能的更多信息。

语法

```
GenericOutcome StopMatchBackfill (
    const Aws::GameLift::Server::Model::StopMatchBackfillRequest &stopBackfillRequest);
```

参数

StopMatchBackfillRequest

一个 StopMatchBackfillRequest 对象，用于识别要取消的对战票证：

- 分配给被取消的回填请求的票证 ID
- 回填请求所发送到的对战构建器
- 与回填请求关联的游戏会话

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
// 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()

4.0.1 相反，服务器进程应在游戏会话结束ProcessEnding()后调用。

通知 Amazon GameLift 服务服务器进程已结束当前游戏会话。当服务器进程保持活动状态并准备托管新游戏会话时，将调用此操作。只有在游戏会话终止程序完成后才应调用它，因为它会向 Amazon GameLift 发出信号，表明服务器进程可以立即用于托管新的游戏会话。

如果服务器进程将在游戏会话停止后关闭，则不会调用此操作。取而代之的是，调用ProcessEnding()表示游戏会话和服务器进程都将结束。

语法

```
GenericOutcome TerminateGameSession();
```

参数

此操作没有参数。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

UpdatePlayerSessionCreationPolicy()

更新当前游戏会话接受新玩家会话的能力。可将游戏会话设置为接受或拒绝所有新的玩家会话。另请参阅 AWS 软件开发工具包操作 AWSUpdateGameSession()。

语法

```
GenericOutcome UpdatePlayerSessionCreationPolicy(
    Aws::GameLift::Model::PlayerSessionCreationPolicy newPlayerSessionPolicy);
```

参数

newPlayerSessionPolicy

用于指示游戏会话是否接受新玩家的字符串值。

类型： Aws::GameLift::Model::PlayerSessionCreationPolicy 枚举。有效值包括：

- ACCEPT_ALL - 接受所有新玩家会话。
- DENY_ALL - 拒绝所有新玩家会话。

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例将当前游戏会话的接入策略设为接受所有玩家。

```
Aws::GameLift::GenericOutcome outcome =
 Aws::GameLift::Server::UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCr
```

Destroy

在游戏服务器初始化期间清理 initSDK () 分配的内存。在结束游戏服务器进程后使用此方法，以避免浪费服务器内存。

语法

```
GenericOutcome Aws::GameLift::Server::Destroy();
```

参数

没有参数。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例在游戏服务器进程结束后清理 InitSDK 分配的内存。

```
if (Aws::GameLift::Server::ProcessEnding().IsSuccess()) {
  Aws::GameLift::Server::Destroy();
  exit(0);
}
```

Amazon GameLift 服务器软件开发工具包 (C++) 参考：数据类型

您可以使用这份 Amazon GameLift C++ 服务器软件开发工具包 参考来帮助您为多人游戏做好准备，以便在 Amazon GameLift 上使用。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

此 API 在 `GameLiftServerAPI.h`、`LogParameters.h` 和 `ProcessParameters.h` 中定义。

- 操作
- 数据类型

DescribePlayerSessionsRequest

此数据类型用于指定检索哪些玩家会话。您可以按如下方式使用它：

- 提供 PlayerSessionId 以请求特定的玩家会话。
- 提供 GameSessionId 以请求指定游戏会话中的所有玩家会话。
- 提供 PlayerId 以请求指定玩家的所有玩家会话。

对于大型玩家会话集合，请使用分页参数以在有序数据块中检索结果。

目录

GameSessionId

游戏会话的唯一标识符。使用此参数可请求指定游戏会话的所有玩家会话。游戏会话 ID 的格式如下所示：`arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>`。<ID string> 的值为自定义 ID 字符串（如果在创建游戏会话时指定了一个）或者是生成的字符串。

类型：字符串

必需：否

限制

要返回的最大结果数量。使用此参数和 NextToken 以一组连续页面的格式获取结果。如果指定玩家会话 ID，将忽略此参数。

类型：整数

必需：否

NextToken

令牌指示结果的下一个连续页面的开头。使用之前的调用返回此操作的令牌。要指定结果集的开始，请不要指定值。如果指定玩家会话 ID，将忽略此参数。

类型：字符串

必需：否

PlayerId

玩家的唯一标识符。玩家 ID 由开发人员定义。请参阅生成玩家 ID。

类型：字符串

必需：否

PlayerSessionId

玩家会话的唯一标识符。

类型：字符串

必需：否

PlayerSessionStatusFilter

用于筛选结果的玩家会话状态。可能的玩家会话状态包括以下内容：

- RESERVED - 已收到玩家会话请求，但玩家尚未连接到服务器进程和/或进行验证。
- ACTIVE - 服务器进程已验证玩家，当前已连接。
- COMPLETED - 玩家连接已断开。
- TIMEDOUT - 收到了玩家会话请求，但玩家未连接和/或在超时限制 (60 秒) 内验证。

类型：字符串

必需：否

LogParameters

此数据类型用于标识您希望 在游戏会话结束时上传并存储游戏会话期间生成的哪些文件。此信息在 调用中传递给 服务。

目录

logPaths

您希望 用于存储供以后访问的游戏服务器日志文件的目录路径。这些文件将在每个游戏会话期间生成。文件路径和名称在游戏服务器中定义并存储在根游戏构建目录中。日志路径必须是绝对的。例如，如果您的游戏构建在类似于 `MyGame\sessionlogs\` 的路径中存储游戏会话日志，则日志路径将为 `c:\game\MyGame\sessionLogs` (在 Windows 实例上) 或 `/local/game/MyGame/sessionLogs` (在 Linux 实例上)。

类型：std:vector<std::string>

必需：否

ProcessParameters

此数据类型包含一组在 调用中发送到 服务的参数。

目录

port

服务器进程用于侦听新玩家连接的端口号。该值必须在部署此游戏服务器构建的任意实例集上所配置的端口范围内。此端口号包含在游戏会话和玩家会话对象中，游戏会话在连接到服务器进程时使用。

类型：整数

必需：是

logParameters

包含游戏会话日志文件目录路径列表的对象。

类型：Aws::GameLift::Server::LogParameters

必需：否

onStartGameSession

服务调用用于激活新游戏会话的回调函数的名称。 调用此函数响应客户端请求 CreateGameSession。回调函数传递 GameSession 对象（在 服务 API 参考 中定义）。

类型：`const std::function<void(Aws::GameLift::Model::GameSession)> onStartGameSession`

必需：是

onProcessTerminate

服务调用以强制关闭服务器进程的回调函数的名称。调用此函数之后， 等待五分钟以便服务器进程关闭，然后使用 调用来进行响应。如果没有收到响应，它会关闭该服务器进程。

类型：`std::function<void()> onProcessTerminate`

必需：否

onHealthCheck

服务调用以从服务器进程请求运行状态报告的回调函数的名称。 每隔 60 秒调用此函数。调用此函数后， 将等待 60 秒接收响应，如果未收到响应，则会将服务器进程记录为不正常。

类型：`std::function<bool()> onHealthCheck`

必需：否

onUpdateGameSession

Amazon GameLift 服务为将更新的游戏会话对象传递给服务器进程而调用的回调函数的名称。为了提供更新的匹配数据而处理了匹配回填请求，Amazon GameLift 会调用此函数。它会传递一个 GameSession 对象、状态更新 (`updateReason`) 以及对战回填票证 ID。

类型：`std::function<void(Aws::GameLift::Server::Model::UpdateGameSession)> onUpdateGameSession`

必需：否

StartMatchBackfillRequest

此数据类型用于发送对战回填请求。此信息在 调用中传递给 服务。

目录

GameSessionArn

游戏会话的唯一标识符。此 API 操作 GetGameSessionId() 返回采用 ARN 格式的标识符。

类型：字符串

必需：是

MatchmakingConfigurationArn

采用 ARN 格式的唯一标识符，适用于用于此请求的对战构建器。要查找用于创建原始游戏会话的对战构建器，请查看对战构建器数据属性中的游戏会话对象。使用对战构建器数据详细了解 Word 中的对战构建器数据。

类型：字符串

必需：是

玩家

一组表示当前正在游戏会话中的所有玩家的数据。对战构建器使用此信息搜索与当前玩家非常匹配的新玩家。有关玩家对象格式的描述，请参阅 Amazon GameLift API 参考指南。要查找玩家属性、ID 和团队任务，请查看对战构建器数据属性中的游戏会话对象。如果对战构建器使用了延迟，则收集当前区域中更新的延迟并将其包含在每个玩家的数据中。

类型：std:vector<Player>

必需：是

TicketId

对战或匹配回填请求票证的唯一标识符。如果此处未提供任何值，则 Amazon GameLift 将生成一个采用 UUID 形式的值。使用此标识符可跟踪匹配回填票证状态或取消请求 (如需要)。

类型：字符串

必需：否

StopMatchBackfillRequest

此数据类型用于取消对战回填请求。此信息在 调用中传递给 服务。

目录

GameSessionArn

与被取消的请求关联的唯一游戏会话标识符。

类型：字符串

必需：是

MatchmakingConfigurationArn

此请求发送到的对战构建器的唯一标识符。

类型：字符串

必需：是

TicketId

要取消的回填请求票证的唯一标识符。

类型：字符串

必需：是

适用于 C# 的 Amazon GameLift 服务器软件开发工具包 参考

您可以使用这份 Amazon GameLift C# 服务器软件开发工具包 参考来帮助您为多人游戏做好准备，以便在 Amazon GameLift 上使用。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

主题

- 适用于 C# 和 Unity 的 Amazon GameLift 服务器软件开发工具包 5.x 参考
- 适用于 C# 的 Amazon GameLift 服务器软件开发工具包 4.x 参考

适用于 C# 和 Unity 的 Amazon GameLift 服务器软件开发工具包 5.x 参考

您可以使用这份 Amazon GameLift C# 服务器软件开发工具包 5.x 参考来帮助您为多人游戏做好准备，以便在 Amazon GameLift 上使用。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器；有关使用适用于 Unity 的 C# 服务器软件开发工具包插件的信息，请参阅将 Amazon GameLift 集成到 Unity 项目中。适用于 C# 的 Amazon GameLift 服务器软件开发工具包 5.x 支持.NET 4.6 和.NET 6。

主题

- 适用于 C# 和 Unity 的亚马逊 GameLift 服务器软件开发工具包参考：操作

- 适用于 C# 和 Unity 的 Amazon GameLift 服务器软件开发工具包 参考：数据类型

适用于 C# 和 Unity 的亚马逊 GameLift 服务器软件开发工具包参考：操作

此 Amazon GameLift C# 服务器 SDK 参考可帮助您准备多人游戏以供亚马逊使用 GameLift。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器；有关使用适用于 Unity 的 C# 服务器软件开发工具包插件的信息，请参阅将 Amazon GameLift 集成到 Unity 项目中。

操作

- GetSdkVersion()
- InitSDK()
- InitSDK()
- ProcessReady()
- ProcessEnding()
- ActivateGameSession()
- UpdatePlayerSessionCreationPolicy()
- GetGameSessionId()
- GetTerminationTime()
- AcceptPlayerSession()
- RemovePlayerSession()
- DescribePlayerSessions()
- StartMatchBackfill()
- StopMatchBackfill()
- GetComputeCertificate()
- GetFleetRoleCredentials()
- Destroy

GetSdkVersion()

返回当前内置到服务器进程中的开发工具包的版本号。

语法

```
AwsStringOutcome GetSdkVersion();
```

返回值

如果成功，返回以 the section called “awsString结果” 对象返回当前软件开发工具包的版本。返回的字符串仅包含版本号 (例如：如果不成功，将返回错误消息。

示例

```
var getSdkVersionOutcome = GameLiftServerAPI.GetSdkVersion();
```

InitSDK()

为托管 EC2 队列初始化 Amazon GameLift 开发工具包。在启动时调用此方法，然后再进行任何其他与 Amazon GameLift 相关的初始化。此方法从主机环境读取服务器参数，以设置服务器与 Amazon GameLift 服务之间的通信。

语法

```
GenericOutcome InitSDK();
```

返回值

如果成功，则返回一个 InitSdkOutcome 对象，表示服务器进程已准备好调用ProcessReady()。

示例

```
//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
GenericOutcome initSDKOutcome = GameLiftServerAPI.InitSDK();
```

InitSDK()

为Anywhere队列初始化 Amazon GameLift 软件开发工具包。在启动时调用此方法，然后再进行任何其他与 Amazon GameLift 相关的初始化。此方法需要明确的服务器参数来设置服务器和 Amazon GameLift 服务之间的通信。

语法

```
GenericOutcome InitSDK(ServerParameters serverParameters);
```

参数

服务器参数

要在 Amazon GameLift Anywhere 舰队上初始化游戏服务器，请使用以下信息构造一个ServerParameters对象：

- WebSocket 用于连接到您的游戏服务器的 URL。
- 用于托管游戏服务器的进程的 ID。
- 托管游戏服务器进程的计算的 ID。
- 包含您的亚马逊 GameLift Anywhere计算的亚马逊 GameLift 舰队的 ID。
- Amazon GameLift 操作生成的授权令牌。

返回值

如果成功，则返回一个 InitSdkOutcome 对象，表示服务器进程已准备好调用ProcessReady()。

> **Note**
>
> 如果对部署到 InitSDK() Anywhere 队列的游戏版本调用失败，请检查创建编译资源时使用的ServerSdkVersion参数。您必须将此值明确设置为正在使用的服务器软件开发工具包 版本。此参数的默认值为 4.x，这不兼容。要解决此问题，请创建新版本并将其部署到新队列。

示例

```
//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()

通知 Amaz GameLift on 服务器进程已准备好托管游戏会话。调用后调用InitSDK()此方法。每个进程只能调用一次此方法。

语法

```
GenericOutcome ProcessReady(ProcessParameters processParameters)
```

参数

ProcessParameters

`ProcessParameters`对象保存有关服务器进程的信息。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例说明 调用和委派函数实施。

```
// 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()

通知 Amaz GameLift on 服务器进程即将终止。在完成所有其他清理任务（包括关闭活动游戏会话）之后和终止该进程之前，调用此方法。根据的结果`ProcessEnding()`，进程

以成功 (0) 或错误 (-1) 退出并生成队列事件。如果流程因错误而终止，则生成的舰队事件为SERVER_PROCESS_TERMINATED_UNHEALTHY。

语法

```
GenericOutcome ProcessEnding()
```

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例在使用成功或错误退出代码终止服务器进程Destroy()之前调ProcessEnding()用和。

```
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()

通知 Amazon GameLift 服务器进程已激活游戏会话，现在可以接收玩家连接了。此操作应作为 onStartGameSession() 回调函数的一部分，在所有游戏会话初始化已完成之后调用。

语法

```
GenericOutcome ActivateGameSession()
```

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例显示了 ActivateGameSession() 作为 onStartGameSession() 委派函数的一部分调用。

```
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()

更新当前游戏会话接受新玩家会话的能力。可将游戏会话设置为接受或拒绝所有新的玩家会话。

语法

```
GenericOutcome UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy
 playerSessionPolicy)
```

参数

playerSessionPolicy

用于指示游戏会话是否接受新玩家的字符串值。

有效值包括：

- ACCEPT_ALL - 接受所有新玩家会话。
- DENY_ALL - 拒绝所有新玩家会话。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例将当前游戏会话的接入策略设为接受所有玩家。

```
GenericOutcome updatePlayerSessionPolicyOutcome =

 GameLiftServerAPI.UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy.ACCEPT_ALL);
```

GetGameSessionId()

检索当前由服务器进程托管的游戏会话的 ID (如果服务器进程处于活动状态)。

对于未通过游戏会话激活的空闲进程，该调用将返回the section called “GameLift 错误”。

语法

```
AwsStringOutcome GetGameSessionId()
```

返回值

如果成功，以 the section called “awsString结果” 对象返回游戏会话 ID。如果不成功，将返回错误消息。

示例

```
AwsStringOutcome getGameSessionIdOutcome = GameLiftServerAPI.GetGameSessionId();
```

GetTerminationTime()

如果终止时间可用，则返回安排服务器进程关闭的时间。服务器进程在收到 Amazon 的`onProcessTerminate()`回调后采取此操作 GameLift。Amazon GameLift 打电话`onProcessTerminate()`的原因如下：

- 当服务器进程报告运行状况不佳或没有响应 Amazon 时 GameLift。
- 在缩减事件期间终止实例时。
- 当实例因竞价型实例中断而终止时。

语法

```
AwsDateTimeOutcome GetTerminationTime()
```

返回值

如果成功，则以the section called “awsDateTime 结果”对象形式返回终止时间。该值是终止时间，以此后经过的刻度表示。`0001 00:00:00`例如，日期时间值等`2020-09-13 12:26:40 -000Z`于`637355968000000000`刻度。如果没有可用的终止时间，将返回一条错误消息。

示例

```
AwsDateTimeOutcome getTerminationTimeOutcome = GameLiftServerAPI.GetTerminationTime();
```

AcceptPlayerSession()

通知 Amazon GameLift 具有指定玩家会话 ID 的玩家已连接到服务器进程，需要验证。Amazon 会 GameLift 验证玩家会话 ID 是否有效。玩家会话经过验证后，Amazon 会将玩家位置的状态从 “已保留” GameLift 更改为 “活跃”。

语法

```
GenericOutcome AcceptPlayerSession(String playerSessionId)
```

参数

playerSessionId

创建新玩家会话 GameLift 时颁发的唯一 ID。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例演示了处理连接请求的函数，包括验证和拒绝无效的玩家会话 ID。

```
void ReceiveConnectingPlayerSessionID (Connection connection, String playerSessionId)
{
  GenericOutcome acceptPlayerSessionOutcome =
 GameLiftServerAPI.AcceptPlayerSession(playerSessionId);
  if(acceptPlayerSessionOutcome.Success)
  {
    connectionToSessionMap.emplace(connection, playerSessionId);
    connection.Accept();
  }
  else
  {
    connection.Reject(acceptPlayerSessionOutcome.Error.ErrorMessage);
  }
```

```
}
```

RemovePlayerSession()

通知 Amazon GameLift 有玩家已断开与服务器进程的连接。作为回应，Amazon将玩家位置 GameLift 更改为可用。

语法

```
GenericOutcome RemovePlayerSession(String playerSessionId)
```

参数

playerSessionId

创建新玩家会话 GameLift 时由 Amazon 颁发的唯一 ID。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
GenericOutcome removePlayerSessionOutcome =
 GameLiftServerAPI.RemovePlayerSession(playerSessionId);
```

DescribePlayerSessions()

检索玩家会话数据，包括设置、会话元数据和玩家数据。使用此操作获取单个玩家会话的信息、游戏会话中所有玩家会话的信息或者与单个玩家 ID 相关联的所有玩家会话的信息。

语法

```
DescribePlayerSessionsOutcome DescribePlayerSessions(DescribePlayerSessionsRequest
 describePlayerSessionsRequest)
```

参数

DescribePlayerSessionsRequest

the section called “DescribePlayerSessionsRequest” 对象描述要检索的玩家会话。

返回值

如果成功，将返回一个 the section called “描述玩家会话结果” 对象，包含一组与请求参数相匹配的玩家会话对象。

示例

此示例演示了将所有玩家会话主动连接到指定游戏会话的请求。通过省略限制值NextToken并将其设置为 10，Amazon GameLift 将返回与请求匹配的前 10 条玩家会话记录。

```
// 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()

发送请求以为使用 FlexMatch 创建的游戏会话中的开放位置查找新玩家。有关更多信息，请参阅 FlexMatch回填功能。

此操作为异步操作。如果匹配了新玩家，Amazon会使用回调函数 GameLift 提供更新的匹配器数据OnUpdateGameSession()。

服务器进程每次只能具有一个活动的匹配回填请求。要发送新请求，请先调用 StopMatchBackfill() 以取消原始请求。

语法

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest
 startBackfillRequest);
```

参数

StartMatchBackfillRequest

StartMatchBackfillRequest对象包含有关回填请求的信息。

返回值

返回带有匹配回填票证 ID 的 StartMatchBackfillOutcome 对象或包含错误消息的失败。

示例

```
// 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()

取消使用 创建的活动对战回填请求。有关更多信息，请参阅FlexMatch 回填功能。

语法

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

参数

StopMatchBackfillRequest

一个 StopMatchBackfillRequest 对象，提供有关您要停止的对战票证的详细信息。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
// 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()

检索用于加密游戏服务器和游戏客户端之间网络连接的 TLS 证书的路径。当您将计算设备注册到 Amazon GameLift Anywhere 队列时，您可以使用证书路径。有关更多信息，请参阅RegisterCompute。

语法

```
GetComputeCertificateOutcome GetComputeCertificate();
```

返回值

返回一个包含以下内容的 GetComputeCertificateResponse 对象：

- CertificatePath：计算资源上的 TLS 证书的路径。使用 Amazon GameLift 托管队列时，此路径包含：
  - `certificate.pem`：最终用户证书。完整的证书链是`certificateChain.pem`附加到此证书的组合。
  - `certificateChain.pem`：包含根证书和中间证书的证书链。
  - 根证书。
  - `privateKey.pem`：最终用户证书的私钥。
- ComputeName：您的计算资源的名称。

示例

```
GetComputeCertificateOutcome getComputeCertificateOutcome =
 GameLiftServerAPI.GetComputeCertificate();
```

GetFleetRoleCredentials()

检索授权 Amazon GameLift 与其他AWS 服务角色互动的 IAM 角色证书。有关更多信息，请参阅与您的实例集中的其他 AWS 资源进行通信。

语法

```
GetFleetRoleCredentialsOutcome GetFleetRoleCredentials(GetFleetRoleCredentialsRequest
 request);
```

参数

获取实例集角色凭证申请

角色证书，用于将有限的AWS资源访问权限扩展到游戏服务器。

返回值

返回 the section called “获取实例集角色凭证结果” 对象。

示例

```
// form the fleet credentials request
```

```
GetFleetRoleCredentialsRequest getFleetRoleCredentialsRequest = new
 GetFleetRoleCredentialsRequest(){
  RoleArn = "arn:aws:iam::123456789012:role/service-role/exampleGameLiftAction"
};
GetFleetRoleCredentialsOutcome GetFleetRoleCredentialsOutcome credentials =
 GetFleetRoleCredentials(getFleetRoleCredentialsRequest);
```

Destroy

从内存中释放 Amazon GameLift 游戏服务器 SDK。作为最佳实操，请在终止进程之后ProcessEnding()和之前调用此方法。如果您使用的是 Anywhere 队列，并且没有在每次游戏会话后终止服务器进程，请先致电Destroy()然后InitSDK()重新初始化，然后再通知 A GameLift mazon 该进程已准备好与之托管游戏会话。ProcessReady()

语法

```
GenericOutcome Destroy()
```

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
// 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);
  }
```

适用于 C# 和 Unity 的 Amazon GameLift 服务器软件开发工具包 参考：数据类型

此 C# 服务器 API 参考可帮助您准备使用 的多人游戏。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器；有关使用适用于 Unity 的 C# 服务器软件开发工具包插件的信息，请参阅将 Amazon GameLift 集成到 Unity 项目中。

数据类型

- LogParameters
- ProcessParameters
- UpdateGameSession
- GameSession
- 服务器参数
- StartMatchBackfillRequest
- 玩家
- DescribePlayerSessionsRequest
- StopMatchBackfillRequest
- 获取实例集角色凭证申请
- AttributeValue
- awsString结果
- 通用结果
- 描述玩家会话结果
- 描述 PlayerSessions 结果
- PlayerSession
- StartMatchBackfillOutcome
- 开始匹配回填结果
- 获取计算证书结果
- 获取计算证书结果
- 获取实例集角色凭证结果
- 获取 FleeTroleCredentials 结果
- awsDateTime 结果
- GameLift 错误

- 枚举

LogParameters

此数据类型用于标识您希望 在游戏会话结束时上传并存储游戏会话期间生成的哪些文件。游戏服务器通过通话与 LogParameters to Amazon GameLi ProcessReady() ft 进行通信。

属性	描述
LogPaths	您希望 用于存储供以后访问的游戏服务器日志文件的目录路径列表。服务器进程在每个游戏会话期间生成这些文件。文件路径和名称在游戏服务器中定义并存储在根游戏构建目录中。 日志路径必须是绝对的。例如，如果您的游戏构建在类似于 MyGame\sessionLogs\ 的路径中存储游戏会话日志，则日志路径将为 (在 Windows 实例上) 或 (在 Linux 实例上)。 类型： List<String> 必需：否

ProcessParameters

此数据类型包含一组在 调用中发送到 服务的参数。

属性	描述
LogParameters	包含游戏会话日志文件目录路径列表的对象。 类型： Aws::GameLift::Server:: LogParameters 必需：是
onHealthCheck	服务调用以从服务器进程请求运行状态报告的回调函数的名称。 每隔 60 秒调用此函数。调用此

	函数后，将等待 60 秒接收响应，如果未收到响应，则会将服务器进程记录为不正常。 类型：`void OnHealthCheckDelegate()` 必需：是
onProcessTerminate	服务调用以强制关闭服务器进程的回调函数的名称。调用此函数之后，等待五分钟以便服务器进程关闭，然后使用 调用响应，再关闭服务器进程。 类型：`void OnProcessTerminateDelegate()` 必需：是
onStartGameSession	Amazon GameLift 为激活新游戏会话而调用的回调函数的名称。 调用此函数响应客户端请求 CreateGameSession。回调函数采用 GameSession 对象（在 服务 API 参考 中定义）。 类型：`void OnStartGameSessionDelegate( GameSession )` 必需：是
onUpdateGameSession	Amazon GameLift 为将更新的游戏会话对象传递给服务器进程而调用的回调函数的名称。当处理匹配回填请求以提供更新的对战构建器数据时，Amazon GameLift 会调用此函数。它会传递一个 GameSession 对象、状态更新 (`updateReason`) 以及对战回填票证 ID。 类型：void onUpdateGamesessionDelegate () UpdateGameSession 必需：否

端口	服务器进程用于侦听新玩家连接的端口号。该值必须在部署此游戏服务器构建的任意实例集上所配置的端口范围内。此端口号包含在游戏会话和玩家会话对象中，游戏会话在连接到服务器进程时使用。 类型： `Integer` 必需：是

UpdateGameSession

更新了游戏会话对象的信息，包括更新游戏会话的原因。如果更新与匹配回填操作相关，则此数据类型包括回填票证 ID。

属性	描述
GameSession	由 Amazon GameLift API 定义的 GameSession 对象。该`GameSession` 对象包含描述游戏会话的属性。 类型： `GameSession GameSession()` 必需：是
更新原因	更新游戏会话的原因。 类型： `UpdateReason UpdateReason()` 必需：是
backfillTicketID	尝试更新游戏会话的回填票证的 ID。 类型： `String` 必需：是

GameSession

游戏会话的详细信息。

属性	描述
GameSessionId	会话的唯一标识符。游戏会话ARN具有以下格式:. `arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token>` 类型： `String` 必需：否
名称	游戏会话的描述性标签。 类型： `String` 必需：否
FleetId	运行游戏会话的集的唯一标识符。 类型： `String` 必需：否
最大玩家会话数	机群中已连接到游戏会话的玩家数量。 类型： `Integer` 必需：否
端口	游戏会话的端口号。要连接到 Amazon GameLift 游戏服务器，应用程序需要 IP 地址和端口号。 类型： `Integer` 必需：否

属性	描述
IpAddress	游戏会话的 IP 地址。要连接到 Amazon GameLift 游戏服务器，应用程序需要 IP 地址和端口号。 类型： `String` 必需：否
GameSessionData	一组自定义游戏会话属性，采用单个字符串值格式。 类型： `String` 必需：否
MatchmakerData	有关用于创建游戏会话的对战过程的信息，采用 JSON 语法，格式为字符串。除了使用的对战配置外，它还包含分配到对战的所有玩家的数据，包括玩家属性和队伍分配。 类型： `String` 必需：否
GameProperties	游戏会话的一组自定义属性，采用键值对格式。这些属性将通过启动新游戏会话的请求传递到游戏服务器进程。 类型： `Dictionary<string, string>` 必需：否

属性	描述
DnsName	分配给正在运行游戏会话的实例的 DNS 标识符。CRL 采用以下格式： • 支持 TLS 的实例集:. `<unique identifier>.<region identifier>.amazongamelift.com` • 未启用 TLS 的实例集:. `ec2-<unique identifier>.compute.amazonaws.com` 连接到在支持 TLS 的队列上运行的游戏会话时，必须使用 DNS 名称，而不是 IP 地址。 类型： `String` 必需：否

服务器参数

用于维护 Amazon GameLift Anywhere 服务器与 Amazon GameLift 服务之间连接的信息。使用启动新的服务器进程时会使用此信息InitSDK()。对于托管在 Amazon GameLift 托管 EC2 实例上的服务器，请使用空对象。

属性	描述
WebSockeTurl	当 `RegisterCompute` 作为 Amazon GameLift Anywhere 的一员时 `GameLiftServerSdkEndpoint` 返回。 类型： `String` 必需：是
ProcessId	注册到托管游戏的服务器进程的唯一标识符。 类型： `String`

属性	描述
	必需：是
HostId	托管游戏的服务器进程的主机的唯一标识符。主机 ID 是您注册计算机时使用的计算名称。有关更多信息，请参阅 Reg isterCompute 类型： `String` 必需：是
FleetId	计算注册到的队列的实例集 ID。有关更多信息，请参阅 Register Compute。 类型： `String` 必需：是
AuthToken	由 Amazon GameLift 生成的身份验证令牌，用于向 Amazon GameLift 验证您的服务器。有关更多信息，请参阅 getComputeaut hToken。 类型： `String` 必需：是

StartMatchBackfillRequest

用于创建对战回填请求的信息。游戏服务器通过 StartMatchBackfill() 调用将这些信息传达给 Amazon GameLift。

属性	描述
GameSessionArn	游戏会话的唯一标识符。此 API 操作 `GetGameSessionId` 返回采用 ARN 格式的标识符。 类型： `String`

属性	描述
	必需：是
MatchmakingConfigurationArn	采用 ARN 格式的唯一标识符，适用于用于此请求的对战构建器。要查找用于创建原始游戏会话的对战构建器，请查看对战构建器数据属性中的游戏会话对象。在使用对战构建器数据中了解有关对战构建器数据的更多信息。 类型： `String` 必需：是
玩家	一组表示当前正在游戏会话中的所有玩家的数据。对战构建器使用此信息搜索与当前玩家非常匹配的新玩家。 类型： `List<Player>` 必需：是
TicketId	对战或匹配回填请求票证的唯一标识符。如果不提供值，Amazon GameLift 会生成值。使用此标识符可跟踪匹配回填票证状态或取消请求 (如需要)。 类型： `String` 必需：否

玩家

代表对战中的玩家。当对战请求开始时，玩家会有玩家 ID、属性，可能还有延迟数据。对战结束后，Amazon GameLift 会添加团队信息。

属性	描述
Latencyinms	一组以毫秒为单位的值，表示玩家在连接到某个位置时所经历的延迟量。

属性	描述
	如果使用此属性，则仅匹配列出的位置的玩家。如果对战构建器具有评估玩家延迟的规则，玩家必须报告延迟才能进行匹配。 类型： `Dictionary<string, int>` 必需：否
玩家属性	键/值对的集合，其中包含用于对战的玩家信息。玩家属性键必须与对战规则集中使用的玩家属性相匹配。 有关玩家属性的更多信息，请参阅 AttributeValue。 类型： `Dictionary<string, AttributeValue` 必需：否
PlayerId	玩家的唯一标识符。 类型： `String` 必需：否
团队	玩家在对战中被分配到的团队的名称。在对战规则集中定义团队名称。 类型： `String` 必需：否

DescribePlayerSessionsRequest

此数据类型用于指定检索哪些玩家会话。它可以通过多种方式使用：(1) 提供 PlayerSessionId 来请求特定的玩家会话；(2) 提供 GameSessionId 以在指定游戏会话中请求所有玩家会话；或 (3) 提供

PlayerId 以请求指定玩家的所有玩家会话。对于大型玩家会话集合，请使用分页参数以连续页面格式检索结果。

属性	描述
GameSessionId	游戏会话的唯一标识符。使用此参数可请求指定游戏会话的所有玩家会话。游戏会话 ID 的格式如下所示：`arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>` 。<ID string> 的值为自定义 ID 字符串（如果在创建游戏会话时指定了一个）或者是生成的字符串。 类型： `String` 必需：否
PlayerSessionId	玩家会话的唯一标识符。 类型： `String` 必需：否
PlayerId	玩家的唯一标识符。请参阅生成玩家 ID。 类型： `String` 必需：否
PlayerSessionStatusFilter	用于筛选结果的玩家会话状态。可能的玩家会话状态包括以下内容： • RESERVED - 已收到玩家会话请求，但玩家尚未连接到服务器进程和/或进行验证。 • ACTIVE - 服务器进程已验证玩家，当前已连接。 • COMPLETED - 玩家连接已断开。 • TIMEDOUT - 收到了玩家会话请求，但玩家未连接和/或在超时限制 (60 秒) 内验证。

属性	描述
	类型： String 必需：否
NextToken	令牌指示结果的下一个连续页面的开头。要指定结果集的开始，请不要指定值。如果指定玩家会话 ID，将忽略此参数。 类型： String 必需：否
限制	要返回的最大结果数量。如果指定玩家会话 ID，将忽略此参数。 类型： int 必需：否

StopMatchBackfillRequest

用于取消对战回填请求的信息。游戏服务器通过电话将这些信息传达给 Amazon GameLift 服务。StopMatchBackfill()

属性	描述
GameSessionArn	与被取消的请求关联的唯一游戏会话标识符。 类型： string 必需：是
MatchmakingConfigurationArn	此请求发送到的对战构建器的唯一标识符。 类型： string 必需：是
TicketId	要取消的回填请求票证的唯一标识符。

属性	描述
	类型： `string` 必需：是

获取实例集角色凭证申请

这种数据类型使游戏服务器只能有限地访问您的其他AWS资源。有关更多信息，请参阅为亚马逊设置 IAM 服务角色 GameLift。

属性	描述
RoleArn	可以扩展对资源的有限访问权限的服务角色的 Amazon 资源名称（ARN）。AWS 类型： `string` 必需：是
roleSessionName	描述角色证书使用情况的会话名称。 类型： `string` 必需：否

AttributeValue

在玩家属性密钥-值对中使用。此对象允许您使用任何有效的数据类型来指定属性值：字符串、数字、字符串数组或数据映射。每个`AttributeValue`对象只能使用一个可用属性。

属性	描述
属性类型	指定属性值的类型。 类型：一个 枚举值。 必需：否
S	表示字符串属性值。

属性	描述
	类型： `string` 必需：是
N	表示属性值。 类型： `double` 必需：是
sl	表示字符串属性值的数组。 类型： `string[]` 必需：是
SDM	表示字符串键和双精度值的字典。 类型： `Dictionary<string, double>` 必需：是

awsString结果

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型： `string` 必需：否
成功	操作是否成功。 类型： `bool` 必需：是

属性	描述
错误	操作失败时发生的错误。 类型： the section called “GameLift 错误” 必需：否

通用结果

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
成功	操作是否成功。 类型： `bool` 必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLift 错误” 必需：否

描述玩家会话结果

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型： the section called “描述 PlayerSessions 结果” 必需：否
成功	操作是否成功。

属性	描述
	类型： bool 必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLift 错误” 必需：否

描述 PlayerSessions 结果

属性	描述
NextToken	令牌指示结果的下一个连续页面的开头。要指定结果集的开始，请不要指定值。如果指定玩家会话 ID，将忽略此参数。 类型： string 必需：是
玩家会话	一组对象，其中包含与请求相匹配的每个玩家会话的属性。 类型： IList<the section called “PlayerSession” > 必填
成功	操作是否成功。 类型： bool 必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLift 错误”

属性	描述
	必需：否

PlayerSession

属性	描述
CreationTime	类型： `long` 必需：是
FleetId	类型： `string` 必需：是
GameSessionId	类型： `string` 必需：是
IpAddress	类型： `string` 必需：是
PlayerData	类型： `string` 必需：是
PlayerId	类型： `string` 必需：是
PlayerSessionId	类型： `string` 必需：是
端口	类型： `int` 必需：是
状态	类型：**`PlayerSessionStatus`** 枚举。

属性	描述
	必需：是
终止时间	类型： `long` 必需：是
DnsName	类型： `string` 必需：是

StartMatchBackfillOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型： the section called “开始匹配回填结果” 必需：否
成功	操作是否成功。 类型： `bool` 必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLift 错误” 必需：否

开始匹配回填结果

属性	描述
TicketId	类型： `string` 必需：是

获取计算证书结果

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型： the section called “获取计算证书结果” 必需：否
成功	操作是否成功。 类型： `bool` 必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLift 错误” 必需：否

获取计算证书结果

计算机上 TLS 证书的路径和计算机的主机名。

属性	描述
证书路径	类型： `string`

属性	描述
	必需：是
计算名称	类型： `string` 必需：是

获取实例集角色凭证结果

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型： the section called “获取 FleeTrole Credentials 结果” 必需：否
成功	操作是否成功。 类型： `bool` 必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLift 错误” 必需：否

获取 FleeTroleCredentials 结果

属性	描述
AccessKeyId	用于对您的AWS资源进行身份验证和提供访问权限的访问密钥 ID。

属性	描述
	类型： `string` 必需：否
假设角色ID	服务角色所属的用户 ID。 类型： `string` 必需：否
假设角色UserRarn	用户应承担的角色的 Amazon 资源名称 (ARN)。 类型： `string` 必需：否
过期	您的会话凭证到期之前的时间。 类型： `DateTime` 必需：否
SecretAccessKey	指定 S3 验证的访问密钥 ID。 类型： `string` 必需：否
SessionToken	用于识别与您的AWS资源交互的当前活动会话的令牌。 类型： `string` 必需：否
成功	操作是否成功。 类型： `bool` 必需：是

属性	描述
错误	操作失败时发生的错误。 类型： the section called “GameLift 错误” 必需：否

awsDateTime 结果

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型： `DateTime` 必需：否
成功	操作是否成功。 类型： `bool` 必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLift 错误” 必需：否

GameLift 错误

属性	描述
ErrorType	错误的类型。 类型：**`GameLiftErrorType`** 枚举。

属性	描述
	必需：否
错误名称	错误类型的名称。 类型： `string` 必需：否
ErrorMessage	错误消息。 类型： `string` 必需：否

枚举

为 Amazon GameLift 服务器软件开发工具包 (C#) 定义的枚举定义如下：

属性类型

- NONE
- string
- DOUBLE
- 字符串列表
- 字符串_DOUBLE_MAP

GameLift 错误类型

表示错误类型的字符串值。有效值包括：

- SERVICE_CALL_FAILED – 对服务的调用失败。AWS
- LOCAL_CONNECTION_FAILED – 与 Amazon GameLift 的本地连接失败。
- NETWORK_NOT_INITIALIZED – 网络尚未初始化。
- GAMESESSION_ID_NOT_SET – 尚未设置游戏会话 ID。
- 错误的请求_异常
- 内部服务异常
- 已初始化 – Amazon GameLift 服务器或客户端已使用初始化 () 进行初始化。

- FLEET_MISMATCH – 目标实例集与 GameSession 或 PlayerSession 的实例集不匹配。
- GAMELIFT_CLIENT_NOT_INITIALIZED – Amazon GameLift 客户端尚未初始化。
- GAMELIFT_SERVER_NOT_INITIALIZED – Amazon GameLift 服务器尚未初始化。
- GAME_SESSION_ENDED_FAILED – Amazon GameLift Server软件开发工具包 无法联系服务部门报告游戏会话已结束。
- GAME_SESSION_NOT_READY – Amazon GameLift 服务器游戏会话未激活。
- GAME_SESSION_READY_FAILED – Amazon GameLift Server软件开发工具包 无法联系服务部门报告游戏会话已准备就绪。
- INITI@@ ALIZATION_MISMATCH – 在 Server::In itialization () 之后调用了客户端方法，反之亦然。
- NOT_INITIALIZED – Amazon GameLift 服务器或客户端尚未使用初始化 () 进行初始化。
- NO_TARGET_ALIASID_SET – 尚未设置目标 AliasID。
- NO_TARGET_FLEET_SET – 尚未设置目标实例集。
- PROCESS@@ _ENDING_FAILED – Amazon GameLift Server软件开发工具包 无法联系服务部门报告流程即将结束。
- PROCESS_NOT_ACTI VE – 服务器进程尚未激活，未绑定到游戏会话，也无法接受或处理 PlayerSession。
- PROCESS_NOT_READ Y – 服务器进程尚未准备好激活。
- PROCESS@@ _READY_FAILED – Amazon GameLift Server软件开发工具包 无法联系服务部门报告流程已准备就绪。
- SDK_VERSION_DETECTION_FAILED –软件开发工具包 版本检测失败。
- STX_CALL_FAILED – 对 xstX 服务器后端组件的调用失败。
- STX_INITIALIZATION_FAILED – xSTX 服务器后端组件无法初始化。
- E@@ XPRENTED_PLAYER_SESSION – 服务器遇到了未注册的玩家会话。
- 网络套接字连接失败
- 禁止网络套接字连接失败
- WEBSOCKET_CONNECT_FAILURE_INVALID_UR
- WEBSOCKET_CONNECT_FAILURE_
- WEBSOCKET_RETRIABLE_SEND_MESSAGE_FAILURE – 向 GameLift 服务 WebSocket 发送消息时可重试失败。

- WEBSOCKET_SEND_MESSAGE_FAILURE – 无法向 GameLift 服务 WebSocket 发送消息。
- MATCH_BACKFILL_REQUEST_VALIDATION – 请求验证失败。
- PLAYER_SESSION_REQUEST_VALIDATION – 请求验证失败。

类型：PlayerSessionCreationPolicy 枚举。

用于指示游戏会话是否接受新玩家的字符串值。有效值包括：

- ACCEPT_ALL - 接受所有新玩家会话。
- DENY_ALL - 拒绝所有新玩家会话。
- NOT_SET – 游戏会话未设置为接受或拒绝新玩家会话。

玩家会话状态

- ACTIVE (处于活动状态)
- COMPLETED
- NOT_SET
- 预留
- 超时

适用于 C# 的 Amazon GameLift 服务器软件开发工具包 4.x 参考

此 C# 服务器 API 参考可帮助您准备使用 的多人游戏。有关集成过程的详细信息，请参阅 将 Amazon GameLift 添加到您的游戏服务器。

主题

- Amazon GameLift 服务器软件开发工具包 (C#) 参考：操作
- Amazon GameLift 服务器软件开发工具包 (C#) 参考：数据类型

Amazon GameLift 服务器软件开发工具包 (C#) 参考：操作

您可以使用这份 Amazon GameLift C# 服务器软件开发工具包 参考来帮助您为多人游戏做好准备，以便在 Amazon GameLift 上使用。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

- 操作
- 数据类型

AcceptPlayerSession()

通知 服务，具有所指定玩家会话 ID 的玩家已连接到服务器进程，需要进行验证。 需要确保玩家会话 ID 有效，即该玩家 ID 在游戏会话中有保留的玩家位置。通过验证后， 将玩家位置的状态从 RESERVED 更改为 ACTIVE。

语法

```
GenericOutcome AcceptPlayerSession(String playerSessionId)
```

参数

PlayerSessionId

创建新玩家会话时由 Amazon GameLift 颁发的唯一标识。玩家会话 ID 在 PlayerSession 对象中指定，作为对 GameLift API 操作 StartGameSessionPlacement、CreateGameSession、DescribeGameSessionPlacement 或 DescribePlayerSessions 请求客户端调用的响应返回。

类型：字符串

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例演示了处理连接请求的函数，包括验证和拒绝无效的玩家会话 ID。

```
void ReceiveConnectingPlayerSessionID (Connection connection, String playerSessionId){
    var acceptPlayerSessionOutcome =
 GameLiftServerAPI.AcceptPlayerSession(playerSessionId);
     if(acceptPlayerSessionOutcome.Success)
    {
        connectionToSessionMap.emplace(connection, playerSessionId);
        connection.Accept();
    }
     else
    {
        connection.Reject(acceptPlayerSessionOutcome.Error.ErrorMessage);    }
```

```
}
```

ActivateGameSession()

通知 服务，服务器进程已激活游戏会话，现在已准备好接收玩家连接。此操作应作为 `onStartGameSession()` 回调函数的一部分，在所有游戏会话初始化已完成之后调用。

语法

```
GenericOutcome ActivateGameSession()
```

参数

此操作没有参数。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例显示了 `ActivateGameSession()` 作为 `onStartGameSession()` 委派函数的一部分调用。

```
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()

检索玩家会话数据，包括设置、会话元数据和玩家数据。使用此操作获取单个玩家会话的信息、游戏会话中所有玩家会话的信息或者与单个玩家 ID 相关联的所有玩家会话的信息。

语法

```
DescribePlayerSessionsOutcome DescribePlayerSessions(DescribePlayerSessionsRequest
 describePlayerSessionsRequest)
```

参数

describePlayerSessionsRequest

DescribePlayerSessionsRequest 对象描述要检索的玩家会话。

必需：是

返回值

如果成功，将返回一个 DescribePlayerSessionsOutcome 对象，包含一组与请求参数相匹配的玩家会话对象。玩家会话对象与 API PlayerSession 数据类型具有相同的结构。

示例

此示例演示了将所有玩家会话主动连接到指定游戏会话的请求。通过省略 NextToken 和将 Limit 值设置为 10，将返回与请求匹配的前 10 个玩家会话记录。

```
// 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()

检索当前由服务器进程托管的游戏会话的 ID (如果服务器进程处于活动状态)。

对于尚未通过游戏会话激活的空闲进程，该调用返回 Success = True 和 GameSessionId ="" (空字符串)。

语法

```
AwsStringOutcome GetGameSessionId()
```

参数

此操作没有参数。

返回值

如果成功，以 AwsStringOutcome 对象返回游戏会话 ID。如果不成功，将返回错误消息。

示例

```
var getGameSessionIdOutcome = GameLiftServerAPI.GetGameSessionId();
```

GetInstanceCertificate()

检索与队列及其实例关联的 PEM 编码 TLS 证书的文件位置。 AWS Certificate Manager当您在证书配置设置为 GENERATED 的情况下创建新队列时，将生成此证书。使用此证书可与游戏客户端建立安全连接并加密客户端/服务器通信。

语法

```
GetInstanceCertificateOutcome GetInstanceCertificate();
```

参数

此操作没有参数。

返回值

如果成功，则返回一个 GetInstanceCertificateOutcome 对象，该对象包含实例集的 TLS 证书文件的位置（该证书文件存储在实例上）。从证书链中提取的根证书文件也存储在实例上。如果不成功，将返回错误消息。

有关证书和证书链数据的更多信息，请参阅 AWS Certificate Manager API 参考中的获取证书响应元素。

示例

```
var getInstanceCertificateOutcome = GameLiftServerAPI.GetInstanceCertificate();
```

GetSdkVersion()

返回当前内置到服务器进程中的开发工具包的版本号。

语法

```
AwsStringOutcome GetSdkVersion()
```

参数

此操作没有参数。

返回值

如果成功，返回以 AwsStringOutcome 对象返回当前软件开发工具包的版本。返回的字符串仅包含版本号 (例如：如果不成功，将返回错误消息。

示例

```
var getSdkVersionOutcome = GameLiftServerAPI.GetSdkVersion();
```

GetTerminationTime()

如果终止时间可用，则返回安排服务器进程关闭的时间。收到来自 服务的 回调后，服务器进程将执行此操作。Amazon GameLift 可能出于以下原因onProcessTerminate()调用：(1) 运行状况不佳（服务器进程已报告端口运行状况或未对 Amazon GameLift 做出响应），(2) 在缩小规模事件期间终止实例时，或 (3) 由于现货实例中断而终止实例时。

如果该进程已收到 onProcessTerminate() 回调，则返回的值是估计的终止时间 (纪元秒)。如果进程未收到onProcessTerminate()回调，则会返回一条错误消息。了解有关关闭服务器进程的更多信息。

语法

```
AwsDateTimeOutcome GetTerminationTime()
```

参数

此操作没有参数。

返回值

如果成功，则以AwsDateTimeOutcome对象形式返回终止时间。该值是终止时间，以自 0001 00:00:00 起经过的报价表示。例如，日期时间值 2020-09-13 12:26:40-000Z 等于 637355968000000000 滴答作响。如果没有可用的终止时间，将返回一条错误消息。

示例

```
var getTerminationTimeOutcome = GameLiftServerAPI.GetTerminationTime();
```

InitSDK()

初始化 Amazon GameLift 软件开发工具包。应在启动之后、进行任何其他 相关的初始化之前调用此方法。

语法

```
InitSDKOutcome InitSDK()
```

参数

此操作没有参数。

返回值

如果成功，返回 InitSdkOutcome 对象，指示服务器进程已准备好调用 ProcessReady()。

示例

```
var initSDKOutcome = GameLiftServerAPI.InitSDK();
```

ProcessEnding()

通知 服务，该服务器进程正在关闭。应在所有其他清除任务 (包括关闭所有活动游戏会话) 之后调用此方法。此方法应退出，退出代码为 0；非零退出代码将导致生成一条事件消息，提示进程未完全退出。

当该方法以代码为 0 退出后，您可以使用成功的退出代码终止该进程。您也可以退出，并返回错误代码。如果您退出时出现错误代码，则队列事件将指示进程异常终止（SERVER_PROCESS_TERMINATED_UNHEALTHY）。

语法

```
GenericOutcome ProcessEnding()
```

参数

此操作没有参数。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();
if (processReadyOutcome.Success)
   Environment.Exit(0);
// otherwise, exit with error code
Environment.Exit(errorCode);
```

ProcessReady()

通知 服务，服务器进程已准备好托管游戏会话。在成功调用 InitSDK() 并完成了服务器进程托管游戏会话所需的全部设置任务后，应调用此方法。每个进程只能调用一次此方法。

语法

```
GenericOutcome ProcessReady(ProcessParameters processParameters)
```

参数

processParameters

ProcessParameters 对象，用于传输有关服务器进程的以下信息：

- 回调方法的名称，在游戏服务器代码中实现， 服务调用它来与服务器进程通信。
- 服务器进程正在侦听的端口号。
- 您希望 捕获和存储的任何游戏会话特定文件的路径。

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例说明 ProcessReady() 调用和委派函数实施。

```
// 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()

通知 服务，具有所指定玩家会话 ID 的玩家已从服务器进程断开连接。作为响应， 将玩家位置更改为可用，这使得该玩家位置可以分配给新玩家。

语法

```
GenericOutcome RemovePlayerSession(String playerSessionId)
```

参数

PlayerSessionId

创建新玩家会话时由 Amazon GameLift 颁发的唯一标识。玩家会话 ID 在 PlayerSession 对象中指定，作为对 GameLift API 操作 StartGameSessionPlacement、CreateGameSession、DescribeGameSessionPlacement 或 DescribePlayerSessions 请求客户端调用的响应返回。

类型：字符串

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
Aws::GameLift::GenericOutcome disconnectOutcome =
    Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

StartMatchBackfill()

发送请求以为使用 创建的游戏会话中的开放位置查找新玩家。另请参阅 AWS 软件开发工具包操作 AWSStartMatchBackfill()。通过此操作，托管游戏会话的游戏服务器进程可以发出匹配回填请求。在中了解有关 FlexMatch 回填功能的更多信息。

此操作为异步操作。如果成功匹配了新玩家，则 服务会使用回调函数 提供更新的对战构建器数据。

服务器进程每次只能具有一个活动的匹配回填请求。要发送新请求，请先调用 StopMatchBackfill() 以取消原始请求。

语法

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest
 startBackfillRequest);
```

参数

StartMatchBackfillRequest

一个 StartMatchBackfillRequest 对象，用于传递以下信息：

- 要分配给回填请求的票证 ID。此信息是可选的；如果未提供任何 ID，则 将自动生成一个 ID。
- 要将请求发送到的对战构建器。需要完整的配置 ARN。可从游戏会话的对战构建器数据中获得此值。
- 正在进行回填的游戏会话的 ID。
- 游戏会话的当前玩家的可用对战数据。

必需：是

返回值

返回带有匹配回填票证 ID 的 StartMatchBackfillOutcome 对象或包含错误消息的失败。

示例

```
// 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()

取消使用 StartMatchBackfill() 创建的活动对战回填请求。另请参阅 AWS 软件开发工具包操作 AWSStopMatchmaking()。在中了解有关 FlexMatch 回填功能的更多信息。

语法

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

参数

StopMatchBackfillRequest

一个 StopMatchBackfillRequest 对象，用于识别要取消的对战票证：

- 分配给被取消的回填请求的票证 ID
- 回填请求所发送到的对战构建器
- 与回填请求关联的游戏会话

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
// 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()

4.0.1 相反，服务器进程应在游戏会话结束ProcessEnding()后调用。

通知 Amazon GameLift 服务服务器进程已结束当前游戏会话。当服务器进程保持活动状态并准备托管新游戏会话时，将调用此操作。只有在游戏会话终止程序完成后才应调用它，因为它会向 Amazon GameLift 发出信号，表明服务器进程可以立即用于托管新的游戏会话。

如果服务器进程将在游戏会话停止后关闭，则不会调用此操作。取而代之的是，调用ProcessEnding()表示游戏会话和服务器进程都将结束。

语法

```
GenericOutcome TerminateGameSession()
```

参数

此操作没有参数。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例说明了游戏会话结束时的服务器进程。

```
// 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()

更新当前游戏会话接受新玩家会话的能力。可将游戏会话设置为接受或拒绝所有新的玩家会话。(另请参阅 https://docs.aws.amazon.com/gamelift/latest/apireference/API_UpdateGameSession.html 服务 API 参考 中的 UpdateGameSession() 操作)。

语法

```
GenericOutcome UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy
 playerSessionPolicy)
```

参数

newPlayerSessionPolicy

用于指示游戏会话是否接受新玩家的字符串值。

类型：PlayerSessionCreationPolicy 枚举。有效值包括：

- ACCEPT_ALL - 接受所有新玩家会话。
- DENY_ALL - 拒绝所有新玩家会话。

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例将当前游戏会话的接入策略设为接受所有玩家。

```
var updatePlayerSessionCreationPolicyOutcomex =

 GameLiftServerAPI.UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy.ACCEPT_ALL);
```

Amazon GameLift 服务器软件开发工具包 (C#) 参考：数据类型

您可以使用这份 Amazon GameLift C# 服务器软件开发工具包 参考来帮助您为多人游戏做好准备，以便在 Amazon GameLift 上使用。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

- 操作
- 数据类型

LogParameters

此数据类型用于标识您希望 在游戏会话结束时上传并存储游戏会话期间生成的哪些文件。此信息在 调用中传递给 服务。

目录

logPaths

您希望 用于存储供以后访问的游戏服务器日志文件的目录路径列表。这些文件由服务器进程在每个游戏会话期间生成；文件路径和名称在游戏服务器中定义并存储在根游戏构建目录中。日志路径必须是绝对的。例如，如果您的游戏构建在类似于 `MyGame\sessionlogs\` 的路径中存储游戏会话日志，则日志路径将为 `c:\game\MyGame\sessionLogs` (在 Windows 实例上) 或 `/local/game/MyGame/sessionLogs` (在 Linux 实例上)。

类型：List<String>

必需：否

DescribePlayerSessionsRequest

此数据类型用于指定检索哪些玩家会话。它可以通过多种方式使用：(1) 提供 PlayerSessionId 来请求特定的玩家会话；(2) 提供 GameSessionId 以在指定游戏会话中请求所有玩家会话；或 (3) 提供 PlayerId 以请求指定玩家的所有玩家会话。对于大型玩家会话集合，请使用分页参数以连续页面格式检索结果。

目录

GameSessionId

游戏会话的唯一标识符。使用此参数可请求指定游戏会话的所有玩家会话。游戏会话 ID 的格式如下所示：`arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>`。<ID string> 的值为自定义 ID 字符串（如果在创建游戏会话时指定了一个）或者是生成的字符串。

类型：字符串

必需：否

限制

要返回的最大结果数量。使用此参数和 NextToken 以一组连续页面的格式获取结果。如果指定玩家会话 ID，将忽略此参数。

类型：整数

必需：否

NextToken

令牌指示结果的下一个连续页面的开头。使用之前的调用返回此操作的令牌。要指定结果集的开始，请不要指定值。如果指定玩家会话 ID，将忽略此参数。

类型：字符串

必需：否

PlayerId

玩家的唯一标识符。玩家 ID 由开发人员定义。请参阅生成玩家 ID。

类型：字符串

必需：否

PlayerSessionId

玩家会话的唯一标识符。

类型：字符串

必需：否

PlayerSessionStatusFilter

用于筛选结果的玩家会话状态。可能的玩家会话状态包括以下内容：

- RESERVED - 已收到玩家会话请求，但玩家尚未连接到服务器进程和/或进行验证。
- ACTIVE - 服务器进程已验证玩家，当前已连接。
- COMPLETED - 玩家连接已断开。
- TIMEDOUT - 收到了玩家会话请求，但玩家未连接和/或在超时限制 (60 秒) 内验证。

类型：字符串

必需：否

ProcessParameters

此数据类型包含一组在 调用中发送到 服务的参数。

目录

port

服务器进程侦听新玩家连接的端口号。该值必须在部署此游戏服务器构建的任意实例集上所配置的端口范围内。此端口号包含在游戏会话和玩家会话对象中，游戏会话在连接到服务器进程时使用。

类型：整数

必需：是

logParameters

包含游戏会话日志文件目录路径列表的对象。

类型：`Aws::GameLift::Server::LogParameters`

必需：是

onStartGameSession

服务调用用于激活新游戏会话的回调函数的名称。 调用此函数响应客户端请求 CreateGameSession。回调函数采用 GameSession 对象（在 服务 API 参考 中定义）。

类型：`void OnStartGameSessionDelegate(GameSession gameSession)`

必需：是

onProcessTerminate

服务调用以强制关闭服务器进程的回调函数的名称。调用此函数之后， 等待五分钟以便服务器进程关闭，然后使用 调用响应，再关闭服务器进程。

类型：`void OnProcessTerminateDelegate()`

必需：是

onHealthCheck

服务调用以从服务器进程请求运行状态报告的回调函数的名称。 每隔 60 秒调用此函数。调用此函数后， 将等待 60 秒接收响应，如果未收到响应，则会将服务器进程记录为不正常。

类型：`bool OnHealthCheckDelegate()`

必需：是

onUpdateGameSession

Amazon GameLift 服务为将更新的游戏会话对象传递给服务器进程而调用的回调函数的名称。为了提供更新的匹配数据而处理了匹配回填请求，Amazon GameLift 会调用此函数。它会传递一个 GameSession 对象、状态更新 (updateReason) 以及对战回填票证 ID。

类型：`void OnUpdateGameSessionDelegate ( UpdateGameSession updateGameSession )`

必需：否

StartMatchBackfillRequest

此数据类型用于发送对战回填请求。此信息在 调用中传递给 服务。

目录

GameSessionArn

游戏会话的唯一标识符。此 API 操作 GetGameSessionId() 返回采用 ARN 格式的标识符。

类型：字符串

必需：是

MatchmakingConfigurationArn

采用 ARN 格式的唯一标识符，适用于用于此请求的对战构建器。要查找用于创建原始游戏会话的对战构建器，请查看对战构建器数据属性中的游戏会话对象。在使用对战构建器数据中了解有关对战构建器数据的更多信息。

类型：字符串

必需：是

玩家

一组表示当前正在游戏会话中的所有玩家的数据。对战构建器使用此信息搜索与当前玩家非常匹配的新玩家。有关玩家对象格式的描述，请参阅 Amazon GameLift API 参考指南。要查找玩家属性、ID 和团队任务，请查看对战构建器数据属性中的游戏会话对象。如果对战构建器使用了延迟，则收集当前区域中更新的延迟并将其包含在每个玩家的数据中。

类型：Player[ ]

必需：是

TicketId

对战或匹配回填请求票证的唯一标识符。如果此处未提供任何值，则 Amazon GameLift 将生成一个采用 UUID 形式的值。使用此标识符可跟踪匹配回填票证状态或取消请求 (如需要)。

类型：字符串

必需：否

StopMatchBackfillRequest

此数据类型用于取消对战回填请求。此信息在 调用中传递给 服务。

目录

GameSessionArn

与被取消的请求关联的唯一游戏会话标识符。

类型：字符串

必需：是

MatchmakingConfigurationArn

此请求发送到的对战构建器的唯一标识符。

类型：字符串

必需：是

TicketId

要取消的回填请求票证的唯一标识符。

类型：字符串

必需：是

适用于 Go 的 Amazon GameLift 服务器软件开发工具包 参考

您可以使用这份 Amazon GameLift Go 服务器软件开发工具包 参考来帮助您为多人游戏做好准备，以便在 Amazon GameLift 上使用。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

主题

- Amazon GameLift 服务器 SDK (Go) 参考：操作
- Amazon GameLift 服务器 SDK (Go) 参考：数据类型

Amazon GameLift 服务器 SDK (Go) 参考：操作

您可以使用此 Amazon GameLift Go 服务器 SDK 参考来帮助您为多人游戏做好准备，以便在亚马逊上使用 GameLift。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

`GameLiftServerAPI.go`定义了 Go 服务器软件开发工具包 操作。

操作

- GetSdkVersion()
- InitSDK()
- ProcessReady()
- ProcessEnding()
- ActivateGameSession()
- UpdatePlayerSessionCreationPolicy()
- GetGameSessionId()
- GetTerminationTime()
- AcceptPlayerSession()
- RemovePlayerSession()
- DescribePlayerSessions()
- StartMatchBackfill()
- StopMatchBackfill()
- GetComputeCertificate()
- GetFleetRoleCredentials()

- Destroy

GetSdkVersion()

返回当前内置到服务器进程中的开发工具包的版本号。

语法

```
func GetSdkVersion() (string, error)
```

返回值

如果成功，返回以 对象返回当前软件开发工具包的版本。返回的字符串仅包含版本号 (例如：如果不成功，将返回错误消息。

示例

```
version, err := server.GetSdkVersion()
```

InitSDK()

初始化 Amazon GameLift 软件开发工具包。在启动时调用此方法，然后再进行任何其他与 Amazon GameLift 相关的初始化。此方法在服务器和 Amazon GameLift 服务之间设置通信。

语法

```
func InitSDK(params ServerParameters) error
```

参数

ServerParameters

要在 Amazon GameLift Anywhere 舰队上初始化游戏服务器，请使用以下信息构造一个`ServerParameters`对象：

- WebSocket 用于连接到您的游戏服务器的 URL。
- 用于托管游戏服务器的进程的 ID。
- 托管游戏服务器进程的计算的 ID。
- 包含您的亚马逊 GameLift Anywhere计算的亚马逊 GameLift 舰队的 ID。
- Amazon GameLift 操作生成的授权令牌。

要在 Amazon GameLift 托管 EC2 队列上初始化游戏服务器，请构造一个不带参数的`ServerParameters`对象。通过此次调用，Amazon GameLift 代理为您设置计算环境并自动连接到 Amazon GameLift 服务。

返回值

如果成功，则返回`nil`错误，表示服务器进程已准备好调用ProcessReady()。

> **Note**
>
> 如果对部署到 `InitSDK()` Anywhere 队列的游戏版本调用失败，请检查创建编译资源时使用的`ServerSdkVersion`参数。您必须将此值明确设置为正在使用的服务器软件开发工具包 版本。此参数的默认值为 4.x，这不兼容。要解决此问题，请创建新版本并将其部署到新队列。

示例

亚马逊 GameLift Anywhere示例

```
//Define the server parameters
serverParameters := ServerParameters {
  WebSocketURL: "wss://us-west-1.api.amazongamelift.com",
  ProcessID: "PID1234",
  HostID: "HardwareAnywhere",
  FleetID: "aarn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa",
  AuthToken: "1111aaaa-22bb-33cc-44dd-5555eeee66ff"
}

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
err := server.InitSDK(serverParameters)
```

亚马逊 GameLift 托管 EC2 示例

```
//Define the server parameters
serverParameters := ServerParameters {}

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
```

```
err := server.InitSDK(serverParameters)
```

ProcessReady()

通知 Amaz GameLift on 服务器进程已准备好托管游戏会话。调用后调用InitSDK()此方法。每个进程只能调用一次此方法。

语法

```
func ProcessReady(param ProcessParameters) error
```

参数

ProcessParameters

ProcessParameters 对象，用于传输有关服务器进程的以下信息：

- 游戏服务器代码中实现的回调方法的名称，Amazon GameLift 服务调用这些方法与服务器进程进行通信。
- 服务器进程正在侦听的端口号。
- 包含您希望 Ama GameLift zon 捕获和存储的任何游戏会话特定文件的路径LogParameters的数据类型。

返回值

如果方法失败，则返回带有错误消息的错误。如果刷新成功，则返回 `nil`。

示例

此示例说明 ProcessReady() 调用和委派函数实施。

```
// 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()

通知 Amaz GameLift on 服务器进程即将终止。在完成所有其他清理任务（包括关闭活动游戏会话）之后和终止该进程之前，调用此方法。根据的结果`ProcessEnding()`，进程以成功 (0) 或错误 (-1) 退出并生成队列事件。如果流程因错误而终止，则生成的舰队事件为`SERVER_PROCESS_TERMINATED_UNHEALTHY`。

语法

```
func ProcessEnding() error
```

返回值

返回 0 错误代码或定义的错误代码。

示例

```
// 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()

通知 Amazon GameLift 服务器进程已激活游戏会话，现在可以接收玩家连接了。此操作应作为`onStartGameSession()`回调函数的一部分，在所有游戏会话初始化已完成之后调用。

语法

```
func ActivateGameSession() error
```

返回值

如果方法失败，则返回带有错误消息的错误。

示例

此示例显示了 ActivateGameSession() 作为 onStartGameSession() 委派函数的一部分调用。

```
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()

更新当前游戏会话接受新玩家会话的能力。可将游戏会话设置为接受或拒绝所有新的玩家会话。

语法

```
func UpdatePlayerSessionCreationPolicy(policy model.PlayerSessionCreationPolicy) error
```

参数

playerSessionCreation政策

用于指示游戏会话是否接受新玩家的字符串值。

有效值包括：

- ACCEPT_ALL - 接受所有新玩家会话。
- DENY_ALL - 拒绝所有新玩家会话。

返回值

如果发生故障，则返回带有错误消息的错误。

示例

此示例将当前游戏会话的接入策略设为接受所有玩家。

```
err := server.UpdatePlayerSessionCreationPolicy(model.AcceptAll)
```

GetGameSessionId()

检索当前由服务器进程托管的游戏会话的 ID (如果服务器进程处于活动状态)。

语法

```
func GetGameSessionID() (string, error)
```

参数

此操作没有参数。

返回值

如果成功，以 对象返回游戏会话 ID。对于尚未通过游戏会话激活的空闲进程，该调用会返回一个空字符串和`nil`错误。

示例

```
gameSessionID, err := server.GetGameSessionID()
```

GetTerminationTime()

如果终止时间可用，则返回安排服务器进程关闭的时间。服务器进程在收到 Amazon 的`onProcessTerminate()`回调后采取此操作 GameLift。Amazon GameLift 打电话`onProcessTerminate()`的原因如下：

- 当服务器进程报告运行状况不佳或没有响应 Amazon 时 GameLift。
- 在缩减事件期间终止实例时。
- 当实例因竞价型实例中断而终止时。

语法

```
func GetTerminationTime() (int64, error)
```

返回值

如果成功，则返回服务器进程计划关闭和`nil`错误终止的时间戳（以纪元秒为单位）。该值是终止时间，以经过的刻度表示。`0001 00:00:00`例如，日期时间值等`2020-09-13 12:26:40 -000Z`于`637355968000000000`刻度。如果没有可用的终止时间，将返回一条错误消息。

示例

```
terminationTime, err := server.GetTerminationTime()
```

AcceptPlayerSession()

通知 Amazon GameLift 具有指定玩家会话 ID 的玩家已连接到服务器进程，需要验证。Amazon 会 GameLift 验证玩家会话 ID 是否有效。玩家会话经过验证后，Amazon 会将玩家位置的状态从 GameLift 更改RESERVED为ACTIVE。

语法

```
func AcceptPlayerSession(playerSessionID string) error
```

参数

playerSessionId

创建新玩家会话 GameLift 时由 Amazon 颁发的唯一 ID。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例处理的连接请求包括验证和拒绝无效的玩家会话 ID。

```
func ReceiveConnectingPlayerSessionID(conn Connection, playerSessionID string) {
    err := server.AcceptPlayerSession(playerSessionID)
    if err != nil {
        connection.Accept()
    } else {
        connection.Reject(err.Error())
    }
}
```

RemovePlayerSession()

通知 Amazon GameLift 有玩家已断开与服务器进程的连接。作为回应，Amazon将玩家位置 GameLift 更改为 可用。

语法

```
func RemovePlayerSession(playerSessionID string) error
```

参数

playerSessionId

创建新玩家会话 GameLift 时由 Amazon 颁发的唯一 ID。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
err := server.RemovePlayerSession(playerSessionID)
```

DescribePlayerSessions()

检索玩家会话数据，包括设置、会话元数据和玩家数据。使用此方法获取有关以下内容的信息：

- 单人游戏会话
- 游戏会话中的所有玩家会话
- 与单个玩家 ID 关联的所有玩家会话

语法

```
func DescribePlayerSessions(req request.DescribePlayerSessionsRequest)
 (result.DescribePlayerSessionsResult, error) {
 return srv.describePlayerSessions(&req)
}
```

参数

DescribePlayerSessionsRequest

DescribePlayerSessionsRequest 对象描述要检索的玩家会话。

返回值

如果成功，将返回一个 DescribePlayerSessionsResult 对象，包含一组与请求参数相匹配的玩家会话对象。

示例

此示例演示了将所有玩家会话主动连接到指定游戏会话的请求。通过省略限制值NextToken并将其设置为 10，Amazon 会 GameLift 返回与请求匹配的前 10 条玩家会话记录。

```
// 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()

发送请求以为使用 FlexMatch 创建的游戏会话中的开放位置查找新玩家。有关更多信息，请参阅FlexMatch 回填功能。

此操作为异步操作。如果匹配了新玩家，Amazon会使用回调函数 GameLift 提供更新的匹配器数据OnUpdateGameSession()。

服务器进程每次只能具有一个活动的匹配回填请求。要发送新请求，请先调用 StopMatchBackfill() 以取消原始请求。

语法

```
func StartMatchBackfill(req request.StartMatchBackfillRequest)
 (result.StartMatchBackfillResult, error)
```

参数

StartMatchBackfillRequest

StartMatchBackfillRequest 对象传达以下信息：

- 要分配给回填请求的票证 ID。此信息是可选的；如果未提供身份证，Amazon GameLift 会生成一个。
- 要将请求发送到的对战构建器。需要完整的配置 ARN。该值位于游戏会话的对战构建器数据中。
- 要回填的游戏会话的 ID。
- 游戏会话的当前玩家的可用对战数据。

返回值

返回带有匹配回填票证 ID 的 StartMatchBackfillOutcome 对象或包含错误消息的失败。

示例

```
// 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()

取消使用 创建的活动对战回填请求。有关更多信息，请参阅FlexMatch回填功能。

语法

```
func StopMatchBackfill(req request.StopMatchBackfillRequest) error
```

参数

StopMatchBackfillRequest

标识要取消的配对门票的 StopMatchBackfillRequest 对象：

- 要分配给回填请求的票证 ID
- 回填请求所发送到的对战构建器
- 与回填请求关联的游戏会话

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
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()

检索用于加密游戏服务器和游戏客户端之间网络连接的 TLS 证书的路径。当您将计算设备注册到 Amazon GameLift Anywhere 队列时，您可以使用证书路径。有关更多信息，请参阅RegisterCompute。

语法

```
func GetComputeCertificate() (result.GetComputeCertificateResult, error)
```

返回值

返回包含以下内容的GetComputeCertificateResult对象：

- CertificatePath：计算资源上的 TLS 证书的路径。使用 Amazon GameLift 托管队列时，此路径包含：

- certificate.pem：最终用户证书。完整的证书链是certificateChain.pem附加到此证书的组合。
- certificateChain.pem：包含根证书和中间证书的证书链。
- 根证书。
- privateKey.pem：最终用户证书的私钥。

- ComputeName：您的计算资源的名称。

示例

```
tlsCertificate, err := server.GetFleetRoleCredentials(getFleetRoleCredentialsRequest)
```

GetFleetRoleCredentials()

检索您创建的服务角色证书，这些证书用于将权限扩展到您的其他角色AWS 服务以访问亚马逊 GameLift。这些凭据允许您的游戏服务器使用您的AWS资源。有关更多信息，请参阅为亚马逊设置 IAM 服务角色 GameLift。

语法

```
func GetFleetRoleCredentials(
  req request.GetFleetRoleCredentialsRequest,
) (result.GetFleetRoleCredentialsResult, error) {
  return srv.getFleetRoleCredentials(&req)
}
```

参数

GetFleetRoleCredentialsRequest

角色证书，用于将有限的AWS资源访问权限扩展到游戏服务器。

返回值

返回包含以下内容的GetFleetRoleCredentialsResult对象：

- AssumedRoleUserArn -服务角色所属用户的亚马逊资源名称 (ARN)。
- AssumedRoleId -服务角色所属的用户的 ID。
- AccessKeyId -用于对您的AWS资源进行身份验证和提供访问权限的访问密钥 ID。

- SecretAccessKey -用于身份验证的私有访问密钥 ID。
- SessionToken -用于识别与您的AWS资源交互的当前活动会话的令牌。
- 过期-您的会话凭证到期之前的时间。

示例

```
// form the customer credentials request
getFleetRoleCredentialsRequest := request.NewGetFleetRoleCredentials()
getFleetRoleCredentialsRequest.RoleArn = "arn:aws:iam::123456789012:role/service-role/
exampleGameLiftAction"

credentials, err := server.GetFleetRoleCredentials(getFleetRoleCredentialsRequest)
```

Destroy

从内存中释放 Amazon GameLift 游戏服务器 SDK。作为最佳实操，请在终止进程之后ProcessEnding()和之前调用此方法。如果您使用的是 Anywhere 队列，并且没有在每次游戏会话后终止服务器进程，请先致电Destroy()然后InitSDK()重新初始化，然后再通知 A GameLift mazon 该进程已准备好与之托管游戏会话。ProcessReady()

语法

```
func Destroy() error {
 return srv.destroy()
}
```

返回值

如果方法失败，则返回带有错误消息的错误。

示例

```
// operations to end game sessions and the server process
defer func() {
  err := server.ProcessEnding()
  server.Destroy()
  if err != nil {
    fmt.Println("ProcessEnding() failed. Error: ", err)
    os.Exit(-1)
  } else {
```

```
    os.Exit(0)
  }
}
```

Amazon GameLift 服务器 SDK (Go) 参考：数据类型

您可以使用此 Amazon GameLift Go 服务器 SDK 参考来帮助您为多人游戏做好准备，以便在亚马逊上使用 GameLift。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

数据类型

- LogParameters
- ProcessParameters
- UpdateGameSession
- GameSession
- ServerParameters
- StartMatchBackfillRequest
- 玩家
- DescribePlayerSessionsRequest
- StopMatchBackfillRequest
- GetFleetRoleCredentialsRequest

LogParameters

一个对象，用于标识游戏会话期间生成的文件，您希望 Amazon GameLift 在游戏会话结束后上传和存储这些文件。游戏服务器在ProcessReady()调用中 GameLift 作为`ProcessParameters`对象的一部分提供`LogParameters`给 Amazon。

属性	描述
`LogPaths`	您希望 Amazon GameLift 存储以备将来访问的游戏服务器日志文件的目录路径列表。服务器进程在每个游戏会话期间生成这些文件。文件路径和名称在游戏服务器中定义并存储在根游戏构建目录中。 日志路径必须是绝对的。例如，如果您的游戏构建在类似于 `MyGame\sessionLogs\` 的路径中存储游戏会话日志，则日志路径将为 (在 Windows 实例上) 或 (在 Linux 实例上)。

	类型：[]string 必需：否

ProcessParameters

描述服务器进程与 Amazon 之间通信的对象 GameLift。服务器进程通过调用 Amazon 将此信息提供给 Amazon GameLift ProcessReady()。

属性	描述
LogParameters	一个对象，其中包含游戏会话期间生成的文件的目录路径。Amazon 会 GameLift 复制并存储文件以备将来访问。 类型：LogParameters 必需：否
OnHealthCheck	Amazon 为向服务器 GameLift 进程请求健康状态报告而调用的回调函数。Amazon 每 60 秒 GameLift 调用一次此函数，并等待 60 秒等待响应。TRUE如果运行状况良好，则服务器进程会返回，FALSE如果不健康，则返回。如果未返回任何响应，Amazon 会将服务器进程 GameLift 记录为运行状况不佳。 类型：OnHealthCheck func() bool 必需：否
OnProcessTerminate	Amazon 为强制关闭服务器进程而 GameLift 调用的回调函数。调用此函数后，Amazon 会等待 5 分钟， GameLift 等待服务器进程关闭并通过ProcessEnding()调用进行响应，然后再关闭服务器进程。 类型：OnProcessTerminate func() 必需：是
OnStartGameSession	Amazon GameLift 调用的回调函数，用于将更新的游戏会话对象传递给服务器进程。当已处理匹配回填请求以提供更新的匹配器数据时，Amazon 会 GameLift 调用此函数。它传递一个GameSession对象、一个状态更新 (updateReason) 和匹配回填票证 ID。

	类型：`OnStartGameSession func (model.GameSession )` 必需：是
`OnUpdateGameSession`	Amazon GameLift 调用的回调函数，用于将更新的游戏会话信息传递给服务器进程。Amazon在处理匹配回填请求后 GameLift 调用此函数，以提供更新的匹配器数据。 类型：`OnUpdateGameSession func (model.UpdateGameSession)` 必需：否
`Port`	服务器进程用于侦听新玩家连接的端口号。该值必须在部署此游戏服务器构建的任意实例集上所配置的端口范围内。此端口号包含在游戏会话和玩家会话对象中，游戏会话在连接到服务器进程时使用。 类型：`int` 必需：是

UpdateGameSession

游戏会话对象的更新，包括更新游戏会话的原因，以及相关的回填票证 ID（如果使用回填填填填充游戏会话中的玩家会话）。

属性	描述
GameSession	由亚马逊 GameLift API 定义的GameSession对象。该`GameSession` 对象包含描述游戏会话的属性。 类型：`GameSession GameSession()` 必需：是
UpdateReason	更新游戏会话的原因。 类型：`UpdateReason UpdateReason()` 必需：是

属性	描述
BackfillTicketId	尝试更新游戏会话的回填票证的 ID。 类型：String 必需：否

GameSession

游戏会话的详细信息。

属性	描述
GameSessionId	会话的唯一标识符。arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token> 事件具有以下 Amazon 资源名称 (ARN) 格式。 类型：String 必需：否
名称	游戏会话的描述性标签。 类型：String 必需：否
FleetId	运行游戏会话的集的唯一标识符。 类型：String 必需：否
MaximumPlayerSessionCount	机群中已连接到游戏会话的玩家数量。 类型：Integer 必需：否

属性	描述
端口	游戏会话的端口号。要连接到 Amazon GameLift 游戏服务器，应用程序需要 IP 地址和端口号。 类型：`Integer` 必需：否
IpAddress	游戏会话的 IP 地址。要连接到 Amazon GameLift 游戏服务器，应用程序需要 IP 地址和端口号。 类型：`String` 必需：否
GameSessionData	一组自定义游戏会话属性，采用单个字符串值格式。 类型：`String` 必需：否
MatchmakerData	有关用于创建游戏会话的对战过程的信息，采用 JSON 语法，格式为字符串。除了使用的对战配置外，它还包含分配到对战的所有玩家的数据，包括玩家属性和队伍分配。 类型：`String` 必需：否
GameProperties	游戏会话的一组自定义属性，采用键值对格式。这些属性将通过启动新游戏会话的请求传递到游戏服务器进程。 类型：`map[string] string` 必需：否

属性	描述
DnsName	分配给正在运行游戏会话的实例的 DNS 标识符。CRL 采用以下格式： • 支持 TLS 的实例集:. `<unique identifier>.<region identifier>.amazongamelift.com` • 未启用 TLS 的实例集:. `ec2-<unique identifier>.compute.amazonaws.com` 连接到在支持 TLS 的队列上运行的游戏会话时，必须使用 DNS 名称，而不是 IP 地址。 类型：`String` 必需：否

ServerParameters

用于维护亚马逊 GameLift Anywhere服务器和亚马逊 GameLift 服务之间连接的信息。使用启动新的服务器进程时会使用此信息InitSDK()。对于托管在 Amazon GameLift 托管 EC2 实例上的服务器，请使用空对象。

属性	描述
`WebSocketURL`	当您获取`GameLiftServerSdkEndpoint` 亚马逊 GameLift Anywhere计算资源时 `RegisterCompute` ，Amazon 会 GameLift 返回。 类型：`string` 必需：是
`ProcessID`	注册到托管游戏的服务器进程的唯一标识符。 类型：`string` 必需：是
`HostID`	托管新服务器进程的计算资源的唯一标识符。

属性	描述
	HostID是在您注册计算机时ComputeName 使用的。有关更多信息，请参阅RegisterCompute。 类型：string 必需：是
FleetID	计算注册到的实例集的唯一标识符。有关更多信息，请参阅RegisterCompute。 类型：string 必需：是
AuthToken	由亚马逊生成的身份验证令牌 GameLift ，用于向亚马逊 GameLift验证您的服务器。有关更多信息，请参阅GetComputeAuthToken。 类型：string 必需：是

StartMatchBackfillRequest

用于创建对战回填请求的信息。游戏服务器通过StartMatchBackfill()电话将此信息传达给 Amazon GameLift 。

属性	描述
GameSessionArn	游戏会话的唯一标识符。此 API 操作 GetGameSessionId 返回采用 ARN 格式的标识符。 类型：String 必需：是
MatchmakingConfigurationArn	采用 ARN 格式的唯一标识符，适用于用于此请求的对战构建器。要查找用于创建原始游戏会话的对战构建器，请查看对战构建器数据属性中的游戏会话对象。有关对战构建器数据的更多信息，请参阅使用对战构建器数据。

属性	描述
	类型：`String` 必需：是
玩家	一组表示当前正在游戏会话中的所有玩家的数据。对战构建器使用此信息搜索与当前玩家非常匹配的新玩家。 类型：`[]model.Player` 必需：是
TicketId	对战或匹配回填请求票证的唯一标识符。如果您不提供值，Amazon GameLift 会生成一个值。使用此标识符可跟踪匹配回填票证状态或取消请求 (如需要)。 类型：`String` 必需：否

玩家

表示对战中的玩家的对象。当对战请求开始时，玩家会有玩家 ID、属性，可能还有延迟数据。比赛结束后，Amazon 会 GameLift 添加球队信息。

属性	描述
LatencyInMS	一组以毫秒为单位的值，表示玩家在连接到某个位置时所经历的延迟量。 如果使用此属性，则仅匹配列出的位置的玩家。如果对战构建器具有评估玩家延迟的规则，玩家必须报告延迟才能进行匹配。 类型：`map[string] int` 必需：否
PlayerAttributes	键/值对的集合，其中包含用于对战的玩家信息。玩家属性键必须与配对规则集中 PlayerAttributes 使用的属性密钥相匹配。 有关玩家属性的更多信息，请参阅AttributeValue。

属性	描述
	类型：map[string] AttributeValue 必需：否
PlayerId	玩家的唯一标识符。 类型：String 必需：否
团队	玩家在对战中被分配到的团队的名称。您可以在对战规则集中定义团队名称。 类型：String 必需：否

DescribePlayerSessionsRequest

一个对象，用于指定要检索哪些玩家会话。服务器进程通过DescribePlayerSessions()调用 Amazon 来提供这些信息 GameLift。

属性	描述
GameSessionID	游戏会话的唯一标识符。使用此参数可请求指定游戏会话的所有玩家会话。 游戏会话 ID 的格式为arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string> 。GameSessionID 是自定义 ID 字符串或生成的字符串。 类型：String 必需：否
PlayerSessionID	玩家会话的唯一标识符。使用此参数请求单个特定的玩家会话。 类型：String 必需：否

属性	描述
PlayerID	玩家的唯一标识符。使用此参数可请求指定游戏会话的所有玩家会话。请参阅生成玩家 ID。 类型：String 必需：否
PlayerSessionStatusFilter	用于筛选结果的玩家会话状态。可能的玩家会话状态包括以下内容： • RESERVED - 已收到玩家会话请求，但玩家尚未连接到服务器进程和/或进行验证。 • ACTIVE - 服务器进程已验证玩家，当前已连接。 • COMPLETED - 玩家连接已断开。 • TIMEDOUT - 收到了玩家会话请求，但玩家未连接和/或在超时限制 (60 秒) 内验证。 类型：String 必需：否
NextToken	令牌指示结果的下一个连续页面的开头。要指定结果集的开始，请不要指定值。如果指定玩家会话 ID，将忽略此参数。 类型：String 必需：否
Limit	要返回的最大结果数量。如果指定玩家会话 ID，将忽略此参数。 类型：int 必需：否

StopMatchBackfillRequest

用于取消对战回填请求的信息。游戏服务器通过StopMatchBackfill()呼叫将此信息传送给 Amazon GameLift 服务。

属性	描述
GameSessionArn	与被取消的请求关联的唯一游戏会话标识符。 类型：`string` 必需：否
MatchmakingConfigurationArn	此请求发送到的对战构建器的唯一标识符。 类型：`string` 必需：否
TicketId	要取消的回填请求票证的唯一标识符。 类型：`string` 必需：否

GetFleetRoleCredentialsRequest

将对AWS资源的有限访问权限扩展到游戏服务器的角色证书。有关更多信息，请参阅为亚马逊设置 IAM 服务角色 GameLift。

属性	描述
RoleArn	将有限访问权限扩展到您的AWS资源的服务角色的 ARN。 类型：`string` 必需：是
RoleSessionName	描述角色证书使用情况的会话名称。 类型：`string` 必需：是

Unreal Engine 的 Amazon GameLift 服务器软件开发工具包 参考

此 服务器 API 参考可帮助您准备 Unreal Engine 游戏项目用于 。有关集成过程的详细信息，请参阅 将 Amazon GameLift 添加到您的游戏服务器。

此 API 在 `GameLiftServerSDK.h` 和 `GameLiftServerSDKModels.h` 中定义。

设置 Unreal Engine 插件并查看代码示例将 Amazon GameLift 集成到虚幻引擎项目中。

主题

- Amazon GameLift Unreal Engine 服务器软件开发工具包 5.x 参考
- Amazon GameLift Unreal Engine 服务器软件开发工具包 3.x 参考

Amazon GameLift Unreal Engine 服务器软件开发工具包 5.x 参考

您可以使用这份 Amazon GameLift Unreal Engine 服务器软件开发工具包 5.x 参考来帮助您为多人游戏做好准备，以便在 Amazon GameLift 上使用。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器；有关使用 Unreal 软件开发工具包服务器插件的信息，请参阅将 Amazon GameLift 集成到虚幻引擎项目中。

主题

- 亚马逊 GameLift 服务器 SDK（虚幻）5.x 参考：操作
- 亚马逊 GameLift 服务器 SDK（虚幻）参考：数据类型

亚马逊 GameLift 服务器 SDK（虚幻）5.x 参考：操作

你可以使用这份亚马逊 GameLift 虚幻服务器 SDK 参考来帮助你准备好在亚马逊上使用的多人游戏 GameLift。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器；有关使用 Unreal 软件开发工具包服务器插件的信息，请参阅将 Amazon GameLift 集成到虚幻引擎项目中。

操作

- GetSdkVersion()
- InitSDK()
- InitSDK()
- ProcessReady()
- ProcessEnding()

- ActivateGameSession()
- UpdatePlayerSessionCreationPolicy()
- GetGameSessionId()
- GetTerminationTime()
- AcceptPlayerSession()
- RemovePlayerSession()
- DescribePlayerSessions()
- StartMatchBackfill()
- StopMatchBackfill()
- GetComputeCertificate()
- GetFleetRoleCredentials()

> **Note**
>
> 本主题介绍在为虚幻引擎构建时可以使用的Amazon GameLift C++ API。具体而言，本文档适用于使用该-DBUILD_FOR_UNREAL=1选项编译的代码。

GetSdkVersion()

返回当前内置到服务器进程中的开发工具包的版本号。

语法

```
FGameLiftStringOutcome GetSdkVersion();
```

返回值

如果成功，返回以 the section called “F GameLiftStringOutcome” 对象返回当前软件开发工具包的版本。返回的字符串仅包含版本号 (例如：如果不成功，将返回错误消息。

示例

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
 Aws::GameLift::Server::GetSdkVersion();
```

InitSDK()

为托管 EC2 队列初始化 Amazon GameLift 开发工具包。在启动时调用此方法，然后再进行任何其他与 Amazon GameLift 相关的初始化。此方法从主机环境读取服务器参数，以设置服务器与 Amazon GameLift 服务之间的通信。

语法

```
FGameLiftGenericOutcome InitSDK()
```

返回值

如果成功，返回 InitSdkOutcome 对象，指示服务器进程已准备好调用 。

示例

```
//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
FGameLiftGenericOutcome initSdkOutcome = gameLiftSdkModule->InitSDK();
```

InitSDK()

为Anywhere队列初始化 Amazon GameLift 软件开发工具包。在启动时调用此方法，然后再进行任何其他与 Amazon GameLift 相关的初始化。此方法需要明确的服务器参数来设置服务器和 Amazon GameLift 服务之间的通信。

语法

```
FGameLiftGenericOutcome InitSDK(serverParameters)
```

参数

F ServerParameters

要在 Amazon GameLift Anywhere 舰队上初始化游戏服务器，请使用以下信息构造一个`ServerParameters`对象：

- WebSocket 用于连接到游戏服务器的 URL。
- 用于托管游戏服务器的进程的 ID。

- 托管游戏服务器进程的计算的 ID。
- 包含您的亚马逊 GameLift Anywhere计算的亚马逊 GameLift 舰队的 ID。
- Amazon GameLift 操作生成的授权令牌。

返回值

如果成功，返回 InitSdkOutcome 对象，指示服务器进程已准备好调用 。

> **Note**
>
> 如果对部署到 `InitSDK()` Anywhere 队列的游戏版本调用失败，请检查创建编译资源时使用的`ServerSdkVersion`参数。您必须将此值明确设置为正在使用的服务器软件开发工具包 版本。此参数的默认值为 4.x，这不兼容。要解决此问题，请创建新版本并将其部署到新队列。

示例

```
//Define the server parameters
FServerParameters serverParameters;
parameters.m_authToken = "1111aaaa-22bb-33cc-44dd-5555eeee66ff";
parameters.m_fleetId = "arn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa";
parameters.m_hostId = "HardwareAnywhere";
parameters.m_processId = "PID1234";
parameters.m_webSocketUrl = "wss://us-west-1.api.amazongamelift.com";

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
FGameLiftGenericOutcome initSdkOutcome = gameLiftSdkModule->InitSDK(serverParameters);
```

ProcessReady()

通知 Amaz GameLift on 服务器进程已准备好托管游戏会话。调用后调用InitSDK()此方法。每个进程只能调用一次此方法。

语法

```
GenericOutcome ProcessReady(const Aws::GameLift::Server::ProcessParameters
&processParameters);
```

参数

processParameters

F ProcessParameters 对象，用于传输有关服务器进程的以下信息：

- 游戏服务器代码中实现的回调方法的名称，Amazon GameLift 服务调用这些方法与服务器进程进行通信。
- 服务器进程正在侦听的端口号。
- 您希望 Ama GameLift zon 捕获和存储的所有游戏会话特定文件的路径。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例说明 ProcessReady() 调用和委派函数实施。

```
//Calling ProcessReady tells GameLift this game server is ready to receive incoming
 game sessions!
UE_LOG(GameServerLog, Log, TEXT("Calling Process Ready"));
FGameLiftGenericOutcome processReadyOutcome = gameLiftSdkModule-
>ProcessReady(*params);
```

ProcessEnding()

通知 Amaz GameLift on 服务器进程即将终止。在完成所有其他清理任务（包括关闭活动游戏会话）之后和终止该进程之前，调用此方法。根据的结果ProcessEnding()，进程以成功 (0) 或错误 (-1) 退出并生成队列事件。如果进程因错误而终止，则生成的队列事件为SERVER_PROCESS_TERMINATED_UNHEALTHY)。

语法

```
FGameLiftGenericOutcome ProcessEnding()
```

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
//OnProcessTerminate callback. GameLift will invoke this callback before shutting down
 an instance hosting this game server.
//It gives this game server a chance to save its state, communicate with services,
 etc., before being shut down.
//In this case, we simply tell GameLift we are indeed going to shutdown.
params->OnTerminate.BindLambda([=]() {
  UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
  gameLiftSdkModule->ProcessEnding();
});
```

ActivateGameSession()

通知 Amazon GameLift 服务器进程已激活游戏会话，现在可以接收玩家连接了。此操作应作为 onStartGameSession() 回调函数的一部分，在所有游戏会话初始化已完成之后调用。

语法

```
FGameLiftGenericOutcome ActivateGameSession()
```

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例显示了 ActivateGameSession() 作为 onStartGameSession() 委派函数的一部分调用。

```
//When a game session is created, GameLift sends an activation request to the game
 server and passes along the game session object containing game properties and other
 settings.
//Here is where a game server should take action based on the game session object.
//Once the game server is ready to receive incoming player connections, it should
 invoke GameLiftServerAPI.ActivateGameSession()
auto onGameSession = [=](Aws::GameLift::Server::Model::GameSession gameSession)
{
  FString gameSessionId = FString(gameSession.GetGameSessionId());
  UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"), *gameSessionId);
  gameLiftSdkModule->ActivateGameSession();
};
```

UpdatePlayerSessionCreationPolicy()

更新当前游戏会话接受新玩家会话的能力。可将游戏会话设置为接受或拒绝所有新的玩家会话。

语法

```
FGameLiftGenericOutcome UpdatePlayerSessionCreationPolicy(EPlayerSessionCreationPolicy
 policy)
```

参数

playerCreationSession政策

用于指示游戏会话是否接受新玩家的字符串值。

有效值包括：

- ACCEPT_ALL - 接受所有新玩家会话。
- DENY_ALL - 拒绝所有新玩家会话。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例将当前游戏会话的接入策略设为接受所有玩家。

```
FGameLiftGenericOutcome outcome = gameLiftSdkModule-
>UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::EPlayerSessionCreationPolicy::ACCEPT_A
```

GetGameSessionId()

检索当前由服务器进程托管的游戏会话的 ID (如果服务器进程处于活动状态)。

对于未通过游戏会话激活的空闲进程，该调用将返回the section called “F GameLiftError”。

语法

```
FGameLiftStringOutcome GetGameSessionId()
```

参数

此操作没有参数。

返回值

如果成功，以 the section called “F GameLiftStringOutcome” 对象返回游戏会话 ID。如果不成功，将返回错误消息。

对于未通过游戏会话激活的空闲进程，调用返回 `Success = True` 和 `GameSessionId = ""`。

示例

```
//When a game session is created, GameLift sends an activation request to the game
 server and passes along the game session object containing game properties and other
 settings.
//Here is where a game server should take action based on the game session object.
//Once the game server is ready to receive incoming player connections, it should
 invoke GameLiftServerAPI.ActivateGameSession()
auto onGameSession = [=](Aws::GameLift::Server::Model::GameSession gameSession)
{
  FString gameSessionId = FString(gameSession.GetGameSessionId());
  UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"), *gameSessionId);
  gameLiftSdkModule->ActivateGameSession();
};
```

GetTerminationTime()

如果终止时间可用，则返回安排服务器进程关闭的时间。服务器进程在收到 Amazon 的`onProcessTerminate()`回调后采取行动 GameLift。Amazon GameLift 打电话`onProcessTerminate()`的原因如下：

- 当服务器进程报告运行状况不佳或没有响应 Amazon 时 GameLift。
- 在缩减事件期间终止实例时。
- 当实例因竞价型实例中断而终止时。

语法

```
AwsDateTimeOutcome GetTerminationTime()
```

返回值

如果成功，则以AwsDateTimeOutcome对象形式返回终止时间。该值是终止时间，以此后经过的刻度表示。0001 00:00:00例如，日期时间值等2020-09-13 12:26:40 -000Z于637355968000000000刻度。如果没有可用的终止时间，将返回一条错误消息。

如果进程未收到 ProcessParameters.OnProcessTerminate()回调，则会返回一条错误消息。有关关闭服务器进程的更多信息，请参阅回应服务器进程关闭通知。

示例

```
AwsDateTimeOutcome TermTimeOutcome = gameLiftSdkModule->GetTerminationTime();
```

AcceptPlayerSession()

通知 Amazon GameLift 具有指定玩家会话 ID 的玩家已连接到服务器进程，需要验证。Amazon 会 GameLift 验证玩家会话 ID 是否有效。玩家会话经过验证后，Amazon 会将玩家位置的状态从 “已保留” GameLift 更改为 “活跃”。

语法

```
FGameLiftGenericOutcome AcceptPlayerSession(const FString& playerSessionId)
```

参数

playerSessionId

创建新玩家会话 GameLift 时由 Amazon 颁发的唯一 ID。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

此示例处理的连接请求包括验证和拒绝无效的玩家会话 ID。

```
bool GameLiftManager::AcceptPlayerSession(const FString& playerSessionId, const
 FString& playerId)
{
```

```
  #if WITH_GAMELIFT
  UE_LOG(GameServerLog, Log, TEXT("Accepting GameLift PlayerSession: %s . PlayerId:
 %s"), *playerSessionId, *playerId);
  FString gsId = GetCurrentGameSessionId();
  if (gsId.IsEmpty()) {
    UE_LOG(GameServerLog, Log, TEXT("No GameLift GameSessionId. Returning early!"));
    return false;
  }

  if (!gameLiftSdkModule->AcceptPlayerSession(playerSessionId).IsSuccess()) {
    UE_LOG(GameServerLog, Log, TEXT("PlayerSession not Accepted."));
    return false;
  }

  // Add PlayerSession from internal data structures keeping track of connected players
  connectedPlayerSessionIds.Add(playerSessionId);
  idToPlayerSessionMap.Add(playerSessionId, PlayerSession{ playerId,
 playerSessionId });
  return true;
  #else
  return false;
  #endif
}
```

RemovePlayerSession()

通知 Amazon GameLift 有玩家已断开与服务器进程的连接。作为回应，Amazon将玩家位置 GameLift 更改为可用。

语法

```
FGameLiftGenericOutcome RemovePlayerSession(const FString& playerSessionId)
```

参数

playerSessionId

创建新玩家会话 GameLift 时由 Amazon 颁发的唯一 ID。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
bool GameLiftManager::RemovePlayerSession(const FString& playerSessionId)
{
  #if WITH_GAMELIFT
  UE_LOG(GameServerLog, Log, TEXT("Removing GameLift PlayerSession: %s"),
 *playerSessionId);

  if (!gameLiftSdkModule->RemovePlayerSession(playerSessionId).IsSuccess()) {
    UE_LOG(GameServerLog, Log, TEXT("PlayerSession Removal Failed"));
    return false;
  }

  // Remove PlayerSession from internal data structures that are keeping track of
 connected players
  connectedPlayerSessionIds.Remove(playerSessionId);
  idToPlayerSessionMap.Remove(playerSessionId);

  // end the session if there are no more players connected
  if (connectedPlayerSessionIds.Num() == 0) {
    EndSession();
  }

  return true;
  #else
  return false;
  #endif
}
```

DescribePlayerSessions()

检索玩家会话数据，包括设置、会话元数据和玩家数据。使用此方法获取有关以下内容的信息：

- 单人游戏会话
- 游戏会话中的所有玩家会话
- 与单个玩家 ID 关联的所有玩家会话

语法

```
FGameLiftDescribePlayerSessionsOutcome DescribePlayerSessions(const
 FGameLiftDescribePlayerSessionsRequest &describePlayerSessionsRequest)
```

参数

F GameLiftDescribePlayerSessionsRequest

the section called “F GameLiftDescribePlayerSessionsRequest” 对象描述要检索的玩家会话。

返回值

如果成功，将返回一个 the section called “F GameLiftDescribePlayerSessionsOutcome” 对象，包含一组与请求参数相匹配的玩家会话对象。

示例

此示例演示了将所有玩家会话主动连接到指定游戏会话的请求。通过省略限制值NextToken并将其设置为 10，Amazon 会 GameLift 返回与请求匹配的前 10 条玩家会话记录。

```
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()

发送请求以为使用 FlexMatch 创建的游戏会话中的开放位置查找新玩家。有关更多信息，请参阅FlexMatch 回填功能。

此操作为异步操作。如果匹配了新玩家，Amazon会使用回调函数 GameLift 提供更新的匹配器数据OnUpdateGameSession()。

服务器进程每次只能具有一个活动的匹配回填请求。要发送新请求，请先调用 StopMatchBackfill() 以取消原始请求。

语法

```
FGameLiftStringOutcome StartMatchBackfill (FStartMatchBackfillRequest
 &startBackfillRequest);
```

参数

F StartMatchBackfillRequest

传达以下信息的 StartMatchBackfillRequest 对象：

- 要分配给回填请求的票证 ID。此信息是可选的；如果未提供编号，Amazon GameLift 将生成一个。
- 要将请求发送到的对战构建器。需要完整的配置 ARN。该值位于游戏会话的对战构建器数据中。
- 要回填的游戏会话的 ID。
- 游戏会话的当前玩家的可用对战数据。

返回值

返回带有匹配回填票证 ID 的 StartMatchBackfillOutcome 对象或包含错误消息的失败。

示例

```
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()

取消使用 创建的活动对战回填请求。有关更多信息，请参阅FlexMatch回填功能。

语法

```
FGameLiftGenericOutcome StopMatchBackfill (FStopMatchBackfillRequest
 &stopBackfillRequest);
```

参数

F StopMatchBackfillRequest

标识要取消的配对门票的 StopMatchBackfillRequest 对象：

- 要分配给回填请求的票证 ID
- 回填请求所发送到的对战构建器
- 与回填请求关联的游戏会话

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

```
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()

检索 TLS 证书的路径，该证书用于加密您的亚马逊 GameLift Anywhere计算资源与亚马逊之间的网络连接。 GameLift当您将计算设备注册到 Amazon GameLift Anywhere 队列时，您可以使用证书路径。有关更多信息，请参阅RegisterCompute。

语法

```
FGameLiftGetComputeCertificateOutcome FGameLiftServerSDKModule::GetComputeCertificate()
```

返回值

返回一个包含以下内容的GetComputeCertificateResponse对象：

- CertificatePath：计算资源上的 TLS 证书的路径。
- HostName：您的计算资源的主机名。

示例

```
FGameLiftGetComputeCertificateOutcome FGameLiftServerSDKModule::GetComputeCertificate()
{
  #if WITH_GAMELIFT
  auto outcome = Aws::GameLift::Server::GetComputeCertificate();
  if (outcome.IsSuccess()) {
    auto& outres = outcome.GetResult();
    FGameLiftGetComputeCertificateResult result;
    result.m_certificate_path = UTF8_TO_TCHAR(outres.GetCertificatePath());
    result.m_computeName = UTF8_TO_TCHAR(outres.GetComputeName());
    return FGameLiftGetComputeCertificateOutcome(result);
  }
  else {
    return FGameLiftGetComputeCertificateOutcome(FGameLiftError(outcome.GetError()));
  }
  #else
```

```
  return FGameLiftGetComputeCertificateOutcome(FGameLiftGetComputeCertificateResult());
  #endif
}
```

GetFleetRoleCredentials()

检索授权 Amazon GameLift 与其他AWS 服务角色互动的 IAM 角色证书。有关更多信息，请参阅与您的实例集中的其他 AWS 资源进行通信。

语法

```
FGameLiftGetFleetRoleCredentialsOutcome
 FGameLiftServerSDKModule::GetFleetRoleCredentials(const
 FGameLiftGetFleetRoleCredentialsRequest &request)
```

参数

F GameLiftGetFleetRoleCredentialsRequest

返回值

返回 the section called “F GameLiftGetFleetRoleCredentialsOutcome” 对象。

示例

```
FGameLiftGetFleetRoleCredentialsOutcome
 FGameLiftServerSDKModule::GetFleetRoleCredentials(const
 FGameLiftGetFleetRoleCredentialsRequest &request)
{
  #if WITH_GAMELIFT
  Aws::GameLift::Server::Model::GetFleetRoleCredentialsRequest sdkRequest;
  sdkRequest.SetRoleArn(TCHAR_TO_UTF8(*request.m_roleArn));
  sdkRequest.SetRoleSessionName(TCHAR_TO_UTF8(*request.m_roleSessionName));

  auto outcome = Aws::GameLift::Server::GetFleetRoleCredentials(sdkRequest);

  if (outcome.IsSuccess()) {
    auto& outres = outcome.GetResult();
    FGameLiftGetFleetRoleCredentialsResult result;
    result.m_assumedUserRoleArn = UTF8_TO_TCHAR(outres.GetAssumedUserRoleArn());
    result.m_assumedRoleId = UTF8_TO_TCHAR(outres.GetAssumedRoleId());
    result.m_accessKeyId = UTF8_TO_TCHAR(outres.GetAccessKeyId());
    result.m_secretAccessKey = UTF8_TO_TCHAR(outres.GetSecretAccessKey());
    result.m_sessionToken = UTF8_TO_TCHAR(outres.GetSessionToken());
```

```
    result.m_expiration = FDateTime::FromUnixTimestamp(outres.GetExpiration());
    return FGameLiftGetFleetRoleCredentialsOutcome(result);
  }
  else {
    return FGameLiftGetFleetRoleCredentialsOutcome(FGameLiftError(outcome.GetError()));
  }
  #else
  return
 FGameLiftGetFleetRoleCredentialsOutcome(FGameLiftGetFleetRoleCredentialsResult());
  #endif
}
```

亚马逊 GameLift 服务器 SDK（虚幻）参考：数据类型

你可以使用这份亚马逊 GameLift 虚幻服务器 SDK 参考来帮助你准备好在亚马逊上使用的多人游戏 GameLift。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器；有关使用 Unreal 软件开发工具包服务器插件的信息，请参阅将 Amazon GameLift 集成到虚幻引擎项目中。

数据类型

- F ProcessParameters
- UpdateGameSession
- GameSession
- F ServerParameters
- F StartMatchBackfillRequest
- fPlayer
- F GameLiftDescribePlayerSessionsRequest
- F StopMatchBackfillRequest
- F AttributeValue
- F GameLiftGetFleetRoleCredentialsRequest
- F GameLiftLongOutcome
- F GameLiftStringOutcome
- F GameLiftDescribePlayerSessionsOutcome
- F GameLiftDescribePlayerSessionsResult
- F GenericOutcome
- F GameLiftPlayerSession
- F GameLiftGetComputeCertificateOutcome

- F GameLiftGetComputeCertificateResult
- F GameLiftGetFleetRoleCredentialsOutcome
- F GetFleetRoleCredentialsResult
- F GameLiftError
- 枚举

> **Note**
>
> 本主题介绍在为虚幻引擎构建时可以使用的Amazon GameLift C++ API。具体而言，本文档适用于使用该-DBUILD_FOR_UNREAL=1选项编译的代码。

F ProcessParameters

此数据类型包含在 a 中发送给 Amazon GameLift 的一组参数ProcessReady()。

属性	描述
LogParameters	一个对象，其中包含游戏会话期间生成的文件的目录路径。Amazon 会 GameLift 复制并存储文件以备将来访问。 类型：`TArray<FString>` 必需：否
OnHealthCheck	Amazon 为向服务器 GameLift 进程请求健康状态报告而调用的回调函数。Amazon 每 60 秒 GameLift 调用一次此函数，并等待 60 秒等待响应。TRUE如果运行状况良好，则服务器进程会返回，FALSE如果不健康，则返回。如果未返回任何响应，Amazon 会将服务器进程 GameLift 记录为运行状况不佳。 此属性是一个委托函数，定义为`DECLARE_DELEGATE_RetVal(bool, FOnHealthCheck);`

	类型：`FOnHealthCheck` 必需：否
OnProcessTerminate	Amazon 为强制关闭服务器进程而 GameLift 调用的回调函数。调用此函数后，Amazon 会等待 5 分钟， GameLift 等待服务器进程关闭并通过ProcessEnding()调用进行响应，然后再关闭服务器进程。 类型：`FSimpleDelegate` 必需：是
OnStartGameSession	Amazon 为激活新游戏 GameLift 会话而调用的回调函数。Amazon GameLift 调用此函数是为了响应客户请求CreateGameSession。回调函数传递一个GameSession对象，如亚马逊 GameLift API 参考中所定义。 此属性是一个委托函数，定义为 `DECLARE_DELEGATE_OneParam(FOnStartGameSession, Aws::GameLift::Server::Model::GameSession);` 类型：`FOnStartGameSession` 必需：是

OnUpdateGameSession	Amazon GameLift 调用的回调函数，用于将更新的游戏会话对象传递给服务器进程。当已处理匹配回填请求以提供更新的匹配器数据时，Amazon会 GameLift 调用此函数。它传递一个GameSession对象、一个状态更新 (updateReason) 和匹配回填票证 ID。 此属性是一个委托函数，定义为 `DECLARE_DELEGATE_OneParam(FOnUpdateGameSession, Aws::GameLift::Server::Model::UpdateGameSession);` 类型：`FOnUpdateGameSession` 必需：否
端口	服务器进程用于侦听新玩家连接的端口号。该值必须在部署此游戏服务器构建的任意实例集上所配置的端口范围内。此端口号包含在游戏会话和玩家会话对象中，游戏会话在连接到服务器进程时使用。 类型：`int` 必需：是

UpdateGameSession

此数据类型更新为游戏会话对象，其中包括更新游戏会话的原因以及相关的回填票证 ID（如果使用回填来填充游戏会话中的玩家会话）。

属性	描述
GameSession	由亚马逊 GameLift API 定义的GameSession对象。该`GameSession` 对象包含描述游戏会话的属性。

属性	描述
	类型：`Aws::GameLift::Server::GameSession` 必需：否
UpdateReason	更新游戏会话的原因。 类型：`enum class UpdateReason` • 对战数据已更新 • 回填失败 • BACKFILL_TIMED_OUT • 回填已取消 必需：否
BackfillTicketId	尝试更新游戏会话的回填票证的 ID。 类型：`char[]` 必需：否

GameSession

此数据类型提供游戏会话的详细信息。

属性	描述
GameSessionId	会话的唯一标识符。游戏会话ARN具有以下格式:. `arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token>` 类型：`char[]` 必需：否

属性	描述
名称	游戏会话的描述性标签。 类型：char[] 必需：否
FleetId	运行游戏会话的集的唯一标识符。 类型：char[] 必需：否
MaximumPlayerSessionCount	机群中已连接到游戏会话的玩家数量。 类型：int 必需：否
端口	游戏会话的端口号。要连接到 Amazon GameLift 游戏服务器，应用程序需要 IP 地址和端口号。 类型：int 必需：否
IpAddress	游戏会话的 IP 地址。要连接到 Amazon GameLift 游戏服务器，应用程序需要 IP 地址和端口号。 类型：char[] 必需：否
GameSessionData	一组自定义游戏会话属性，采用单个字符串值格式。 类型：char[] 必需：否

属性	描述
MatchmakerData	有关用于创建游戏会话的对战过程的信息，采用 JSON 语法，格式为字符串。除了使用的对战配置外，它还包含分配到对战的所有玩家的数据，包括玩家属性和队伍分配。 类型：`char[]` 必需：否
GameProperties	游戏会话的一组自定义属性，采用键值对格式。这些属性将通过启动新游戏会话的请求传递到游戏服务器进程。 类型：`GameProperty[]` 必需：否
DnsName	分配给正在运行游戏会话的实例的 DNS 标识符。CRL 采用以下格式： • 支持 TLS 的实例集:. `<unique identifier>.<region identifier>.amazongamelift.com` • 未启用 TLS 的实例集:. `ec2-<unique identifier>.compute.amazonaws.com` 连接到在支持 TLS 的队列上运行的游戏会话时，必须使用 DNS 名称，而不是 IP 地址。 类型：`char[]` 必需：否

F ServerParameters

用于维护亚马逊 GameLift Anywhere服务器和亚马逊 GameLift 服务之间连接的信息。使用启动新的服务器进程时会使用此信息InitSDK()。对于托管在 Amazon GameLift 托管 EC2 实例上的服务器，请使用空对象。

属性	描述
webSocketUrl	当您获取GameLiftServerSdkEndpoint 亚马逊 GameLift Anywhere计算资源时，RegisterCompute Amazon 会 GameLift返回。 类型：char[] 必需：是
ProcessID	注册到托管游戏的服务器进程的唯一标识符。 类型：char[] 必需：是
hostId	HostID是在您注册计算机时ComputeName 使用的。有关更多信息，请参阅RegisterCompute。 类型：char[] 必需：是
FleetId	计算注册到的实例集的唯一标识符。有关更多信息，请参阅RegisterCompute。 类型：char[] 必需：是
AuthToken	由亚马逊生成的身份验证令牌 GameLift ，用于向亚马逊 GameLift验证您的服务器。有关更多信息，请参阅GetComputeAuthToken。

属性	描述
	类型：char[] 必需：是

F StartMatchBackfillRequest

用于创建对战回填请求的信息。游戏服务器通过StartMatchBackfill()电话将此信息传达给 Amazon GameLift 。

属性	描述
GameSessionArn	游戏会话的唯一标识符。此 API 操作 GetGameSessionId 返回采用 ARN 格式的标识符。 类型：char[] 必需：是
MatchmakingConfigurationArn	采用 ARN 格式的唯一标识符，适用于用于此请求的对战构建器。要查找用于创建原始游戏会话的对战构建器，请查看对战构建器数据属性中的游戏会话对象。在使用对战构建器数据中了解有关对战构建器数据的更多信息。 类型：char[] 必需：是
玩家	一组表示当前正在游戏会话中的所有玩家的数据。对战构建器使用此信息搜索与当前玩家非常匹配的新玩家。 类型：TArray<FPlayer> 必需：是

属性	描述
TicketId	对战或匹配回填请求票证的唯一标识符。如果您不提供值，Amazon GameLift 会生成一个值。使用此标识符可跟踪匹配回填票证状态或取消请求 (如需要)。 类型：`char[]` 必需：否

fPlayer

此数据类型代表对战中的玩家。发起对战请求时，玩家会有玩家 ID、属性，可能还有延迟数据。比赛结束后，Amazon 会 GameLift 添加球队信息。

属性	描述
LatencyInMS	一组以毫秒为单位的值，表示玩家在连接到某个位置时所经历的延迟量。 如果使用此属性，则仅匹配列出的位置的玩家。如果对战构建器具有评估玩家延迟的规则，玩家必须报告延迟才能进行匹配。 类型：`TMap>FString, int32<` 必需：否
PlayerAttributes	键/值对的集合，其中包含用于对战的玩家信息。玩家属性键必须与配对规则集中 PlayerAttributes 使用的属性密钥相匹配。 有关玩家属性的更多信息，请参阅Attribute Value。 类型：`TMap>FString, FAttributeValue<`

属性	描述
	必需：否
PlayerId	玩家的唯一标识符。 类型：std::string 必需：否
团队	玩家在对战中被分配到的团队的名称。在对战规则集中定义团队名称。 类型：FString 必需：否

F GameLiftDescribePlayerSessionsRequest

一个对象，用于指定要检索哪些玩家会话。服务器进程通过DescribePlayerSessions()调用 Amazon 来提供这些信息 GameLift。

属性	描述
GameSessionId	游戏会话的唯一标识符。使用此参数可请求指定游戏会话的所有玩家会话。 游戏会话 ID 的格式为FString。GameSessionID 是自定义 ID 字符串或 类型：std::string 必需：否
PlayerSessionId	玩家会话的唯一标识符。使用此参数请求单个特定的玩家会话。 类型：FString 必需：否

属性	描述
PlayerId	玩家的唯一标识符。使用此参数可请求指定游戏会话的所有玩家会话。请参阅 生成玩家 ID。 类型：FString 必需：否
PlayerSessionStatusFilter	用于筛选结果的玩家会话状态。可能的玩家会话状态包括以下内容： • RESERVED - 已收到玩家会话请求，但玩家尚未连接到服务器进程和/或进行验证。 • ACTIVE - 服务器进程已验证玩家，当前已连接。 • COMPLETED - 玩家连接已断开。 • TIMEDOUT - 收到了玩家会话请求，但玩家未连接和/或在超时限制 (60 秒) 内验证。 类型：FString 必需：否
NextToken	令牌指示结果的下一个连续页面的开头。要指定结果集的开始，请不要指定值。如果指定玩家会话 ID，将忽略此参数。 类型：FString 必需：否
限制	要返回的最大结果数量。如果指定玩家会话 ID，将忽略此参数。 类型：int 必需：否

F StopMatchBackfillRequest

用于取消对战回填请求的信息。游戏服务器通过StopMatchBackfill()呼叫将此信息传送给 Amazon GameLift 服务。

属性	描述
GameSessionArn	与被取消的请求关联的唯一游戏会话标识符。 类型：FString 必需：是
MatchmakingConfigurationArn	此请求发送到的对战构建器的唯一标识符。 类型：FString 必需：是
TicketId	要取消的回填请求票证的唯一标识符。 类型：FString 必需：是

F AttributeValue

在fPlayer属性密钥-值对中使用。此对象允许您使用任何有效的数据类型来指定属性值：字符串、数字、字符串数组或数据映射。每个AttributeValue对象只能使用一个可用属性。

属性	描述
属性类型	指定属性值的类型。 类型：一个 枚举值。 必需：否
S	表示字符串属性值。 类型：FString

属性	描述
	必需：否
否	表示属性值。 类型：double 必需：否
sl	表示字符串属性值的数组。 类型：TArray<FString> 必需：否
SDM	表示字符串键和双精度值的字典。 类型：TMap<FString, double> 必需：否

F GameLiftGetFleetRoleCredentialsRequest

此数据类型提供角色凭证，可将对AWS资源的有限访问权限扩展到游戏服务器。有关更多信息，请参阅为亚马逊设置 IAM 服务角色 GameLift。

属性	描述
RoleArn	可以扩展对资源的有限访问权限的服务角色的 Amazon 资源名称（ARN）。AWS 类型：FString 必需：否
RoleSessionName	描述角色证书使用情况的会话名称。 类型：FString 必需：否

F GameLiftLongOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型：`long` 必需：否
ResultWithOwnership	操作的结果，转换为右值，以便调用代码可以拥有该对象的所有权。 类型：`long&&` 必需：否
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “F GameLiftError” 必需：否

F GameLiftStringOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型：`FString`

属性	描述
	必需：否
ResultWithOwnership	操作的结果，转换为右值，以便调用代码可以拥有该对象的所有权。 类型：`FString&&` 必需：否
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “F GameLiftError” 必需：否

F GameLiftDescribePlayerSessionsOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型：the section called “F GameLiftDescribePlayerSessionsResult” 必需：否
ResultWithOwnership	操作的结果，转换为右值，以便调用代码可以拥有该对象的所有权。 类型：`FGameLiftDescribePlayerSessionsResult&&`

属性	描述
	必需：否
成功	操作是否成功。 类型：bool 必需：是
错误	操作失败时发生的错误。 类型：the section called “F GameLiftError” 必需：否

F GameLiftDescribePlayerSessionsResult

属性	描述
PlayerSessions	类型：TArray<FGameLiftPlayerSession> 必需：是
NextToken	令牌指示结果的下一个连续页面的开头。要指定结果集的开始，请不要指定值。如果指定玩家会话 ID，将忽略此参数。 类型：FString 必需：否
成功	操作是否成功。 类型：bool 必需：是
错误	操作失败时发生的错误。

属性	描述
	类型：the section called “F GameLiftError” 必需：否

F GenericOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “F GameLiftError” 必需：否

F GameLiftPlayerSession

属性	描述
CreationTime	类型：`long` 必需：是
FleetId	类型：`FString` 必需：是
GameSessionId	类型：`FString` 必需：是

属性	描述
IpAddress	类型：FString 必需：是
PlayerData	类型：FString 必需：是
PlayerId	类型：FString 必需：是
PlayerSessionId	类型：FString 必需：是
端口	类型：int 必需：是
Status	类型：**PlayerSessionStatus** 枚举。 必需：是
TerminationTime	类型：long 必需：是
DnsName	类型：FString 必需：是

F GameLiftGetComputeCertificateOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。

属性	描述
	类型：the section called “F GameLiftGetComputeCertificateResult” 必需：否
ResultWithOwnership	操作的结果，转换为右值，以便调用代码可以拥有该对象的所有权。 类型：`FGameLiftGetComputeCertificateResult&&` 必需：否
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “F GameLiftError” 必需：否

F GameLiftGetComputeCertificateResult

计算机上 TLS 证书的路径和计算机的主机名。

属性	描述
CertificatePath	类型：`FString` 必需：是
ComputeName	类型：`FString` 必需：是

F GameLiftGetFleetRoleCredentialsOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

属性	描述
结果	操作的结果。 类型：the section called “F GetFleetRoleCredentialsResult” 必需：否
ResultWithOwnership	操作的结果，转换为右值，以便调用代码可以拥有该对象的所有权。 类型：`FGameLiftGetFleetRoleCredentialsResult&&` 必需：否
成功	操作是否成功。 类型：`bool` 必需：是
错误	操作失败时发生的错误。 类型：the section called “F GameLiftError” 必需：否

F GetFleetRoleCredentialsResult

属性	描述
AccessKeyId	用于对您的AWS资源进行身份验证和提供访问权限的访问密钥 ID。

属性	描述
	类型：FString 必需：否
AssumedRoleId	服务角色所属的用户 ID。 类型：FString 必需：否
AssumedRoleUserArn	用户应承担的角色的 Amazon 资源名称 (ARN)。 类型：FString 必需：否
过期	您的会话凭证到期之前的时间。 类型：FDateTime 必需：否
SecretAccessKey	指定 S3 验证的访问密钥 ID。 类型：FString 必需：否
SessionToken	用于识别与您的AWS资源交互的当前活动会话的令牌。 类型：FString 必需：否
成功	操作是否成功。 类型：bool 必需：是

属性	描述
错误	操作失败时发生的错误。 类型：the section called “GameLift 错误” 必需：否

F GameLiftError

属性	描述
ErrorType	错误的类型。 类型：**GameLiftErrorType** 枚举。 必需：否
ErrorName	错误类型的名称。 类型：`std::string` 必需：否
ErrorMessage	错误消息。 类型：`std::string` 必需：否

枚举

为亚马逊 GameLift 服务器 SDK（虚幻）定义的枚举定义如下：

F AttributeType

- NONE
- string
- DOUBLE

- 字符串列表
- 字符串_DOUBLE_MAP

GameLiftErrorType

表示错误类型的字符串值。有效值包括：

- SERVICE_CALL_FAILED – 对服务的调用失败。AWS
- LOCAL_CONNECTION_FAILED — 与亚马逊的本地连接失败。 GameLift
- NETWORK_NOT_INITIALIZED – 网络尚未初始化。
- GAMESESSION_ID_NOT_SET – 尚未设置游戏会话 ID。
- 错误的请求_异常
- 内部服务异常
- 已初始化 — Amazon GameLift 服务器或客户端已使用初始化 () 进行初始化。
- FLEET_MISMATCH – 目标实例集与 GameSession 或 PlayerSession 的实例集不匹配。
- GAMELIFT_CLIENT_NOT_INITIALIZED — 亚马逊客户端尚未初始化。 GameLift
- GAMELIFT_SERVER_NOT_INITIALIZED — 亚马逊服务器尚未初始化。 GameLift
- GAME_SESSION_ENDED_FAILED — A GameLift mazon Server SDK 无法联系服务以报告游戏会话已结束。
- GAME_SESSION_NOT_READY — GameLift 亚马逊服务器游戏会话未激活。
- GAME_SESSION_READY_FAILED — A GameLift mazon Server SDK 无法联系服务以报告游戏会话已准备就绪。
- INITI@@ ALIZATION_MISMATCH – 在 Server::In itialization () 之后调用了客户端方法，反之亦然。
- NOT_INITIAL IZED — Amazon GameLift 服务器或客户端尚未使用初始化 () 进行初始化。
- NO_TARGET_ALIASID_SET – 尚未设置目标 AliasID。
- NO_TARGET_FLEET_SET – 尚未设置目标实例集。
- PROCESS@@ _ENDING_FAILED — A GameLift mazon Server SDK 无法联系服务部门报告流程即将结束。
- PROCESS_NOT_AC TIVE — 服务器进程尚未处于活动状态，未绑定到 GameSession，也无法接受或处理。 PlayerSessions
- PROCESS_NOT_READ Y – 服务器进程尚未准备好激活。
- PROCESS@@ _READY_FAILED — Amazon S GameLift erver SDK 无法联系服务部门报告流程已准备就绪。

- SDK_VERSION_DETECTION_FAILED – 软件开发工具包 版本检测失败。
- STX_CALL_FAILED – 对 xstX 服务器后端组件的调用失败。
- STX_INITIALIZATION_FAILED – xSTX 服务器后端组件无法初始化。
- E@@ XPRENTED_PLAYER_SESSION – 服务器遇到了未注册的玩家会话。
- 网络套接字连接失败
- 禁止网络套接字连接失败
- WEBSOCKET_CONNECT_FAILURE_INVALID_UR
- WEBSOCKET_CONNECT_FAILURE_
- WEBSOCKET_RETRIABLE_SEND_MESSAGE_FAILURE — 向服务发送消息时可重试失败。 GameLift WebSocket
- WEBSOCKET_SEND_MESSAGE_FAILURE — 无法向服务发送消息。 GameLift WebSocket
- MATCH_BACKFILL_REQUEST_VALIDATION – 请求验证失败。
- PLAYER_SESSION_REQUEST_VALIDATION – 请求验证失败。

E PlayerSessionCreationPolicy

用于指示游戏会话是否接受新玩家的字符串值。有效值包括：

- ACCEPT_ALL - 接受所有新玩家会话。
- DENY_ALL - 拒绝所有新玩家会话。
- NOT_SET – 游戏会话未设置为接受或拒绝新玩家会话。

E PlayerSessionStatus

- ACTIVE (处于活动状态)
- COMPLETED
- NOT_SET
- 预留
- 超时

Amazon GameLift Unreal Engine 服务器软件开发工具包 3.x 参考

您可以使用这份 Amazon GameLift Unreal Engine 服务器软件开发工具包 3.x 参考来帮助您为多人游戏做好准备，以便在 Amazon GameLift 上使用。有关集成过程的详细信息，请参阅将 Amazon GameLift 添加到您的游戏服务器。

主题

- Unreal Engine 的 Amazon GameLift 服务器软件开发工具包 参考：动作
- Unreal Engine 的 Amazon GameLift 服务器软件开发工具包 参考：数据类型

Unreal Engine 的 Amazon GameLift 服务器软件开发工具包 参考：动作

此 服务器 API 参考可帮助您准备 Unreal Engine 游戏项目用于 。有关集成过程的详细信息，请参阅 将 Amazon GameLift 添加到您的游戏服务器。

此 API 在 `GameLiftServerSDK.h` 和 `GameLiftServerSDKModels.h` 中定义。

设置 Unreal Engine 插件并查看代码示例将 Amazon GameLift 集成到虚幻引擎项目中。

- 操作
- 数据类型

AcceptPlayerSession()

通知 服务，具有所指定玩家会话 ID 的玩家已连接到服务器进程，需要进行验证。 需要确保玩家会话 ID 有效，即该玩家 ID 在游戏会话中有保留的玩家位置。通过验证后， 将玩家位置的状态从 RESERVED 更改为 ACTIVE。

语法

```
FGameLiftGenericOutcome AcceptPlayerSession(const FString& playerSessionId)
```

参数

PlayerSessionId

Amazon GameLift 服务为响应调用软件开发工具包 Amazon GameLift API 操作 createPlayerSessi AWS on 而颁发的唯一身份证。连接到服务器进程时，游戏客户端会引用此 ID。

类型：FString

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

ActivateGameSession()

通知 服务，服务器进程已激活游戏会话，现在已准备好接收玩家连接。此操作应作为 `onStartGameSession()` 回调函数的一部分，在所有游戏会话初始化已完成之后调用。

语法

```
FGameLiftGenericOutcome ActivateGameSession()
```

参数

此操作没有参数。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

DescribePlayerSessions()

检索玩家会话数据，包括设置、会话元数据和玩家数据。使用此操作获取单个玩家会话的信息、游戏会话中所有玩家会话的信息或者与单个玩家 ID 相关联的所有玩家会话的信息。

语法

```
FGameLiftDescribePlayerSessionsOutcome DescribePlayerSessions(const
 FGameLiftDescribePlayerSessionsRequest &describePlayerSessionsRequest)
```

参数

describePlayerSessionsRequest

fdescribePlayer 会话请求 对象描述要检索的玩家会话。

必需：是

返回值

如果成功，将返回一个 fdescribePlayer 会话请求 对象，包含一组与请求参数相匹配的玩家会话对象。玩家会话对象与 API PlayerSession 数据类型具有相同的结构。

GetGameSessionId()

检索当前由服务器进程托管的游戏会话的 ID (如果服务器进程处于活动状态)。

语法

```
FGameLiftStringOutcome GetGameSessionId()
```

参数

此操作没有参数。

返回值

如果成功，以 FGameLiftStringOutcome 对象返回游戏会话 ID。如果不成功，将返回错误消息。

GetInstanceCertificate()

检索与队列及其实例关联的 PEM 编码 TLS 证书的文件位置。 AWS Certificate Manager当您在证书配置设置为 GENERATED 的情况下创建新队列时，将生成此证书。使用此证书可与游戏客户端建立安全连接并加密客户端/服务器通信。

语法

```
FGameLiftGetInstanceCertificateOutcome GetInstanceCertificate()
```

参数

此操作没有参数。

返回值

如果成功，则返回一个 GetInstanceCertificateOutcome 对象，该对象包含实例集的 TLS 证书文件的位置（该证书文件存储在实例上）。从证书链中提取的根证书文件也存储在实例上。如果不成功，将返回错误消息。

有关证书和证书链数据的更多信息，请参阅 AWS Certificate Manager API 参考中的获取证书响应元素。

GetSdkVersion()

返回当前内置到服务器进程中的开发工具包的版本号。

语法

```
FGameLiftStringOutcome GetSdkVersion();
```

参数

此操作没有参数。

返回值

如果成功，返回以 FGameLiftStringOutcome 对象返回当前软件开发工具包的版本。返回的字符串仅包含版本号 (例如：如果不成功，将返回错误消息。

示例

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
    Aws::GameLift::Server::GetSdkVersion();
```

InitSDK()

初始化 Amazon GameLift 软件开发工具包。应在启动之后、进行任何其他 相关的初始化之前调用此方法。

语法

```
FGameLiftGenericOutcome InitSDK()
```

参数

此操作没有参数。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

ProcessEnding()

通知 服务，该服务器进程正在关闭。应在所有其他清除任务 (包括关闭所有活动游戏会话) 之后调用此方法。此方法应退出，退出代码为 0；非零退出代码将导致生成一条事件消息，提示进程未完全退出。

语法

```
FGameLiftGenericOutcome ProcessEnding()
```

参数

此操作没有参数。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

ProcessReady()

通知 服务，服务器进程已准备好托管游戏会话。在成功调用 InitSDK() 并完成了服务器进程托管游戏会话所需的全部设置任务后，应调用此方法。每个进程只能调用一次此方法。

语法

```
FGameLiftGenericOutcome ProcessReady(FProcessParameters &processParameters)
```

参数

FProcessParameters

FProcessParameters 对象，用于传输有关服务器进程的以下信息：

- 回调方法的名称，在游戏服务器代码中实现， 服务调用它来与服务器进程通信。
- 服务器进程正在侦听的端口号。
- 您希望 捕获和存储的任何游戏会话特定文件的路径。

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

示例

请参阅使用 Unreal Engine 插件中的示例代码。

RemovePlayerSession()

通知 服务，具有所指定玩家会话 ID 的玩家已从服务器进程断开连接。作为响应， 将玩家位置更改为可用，这使得该玩家位置可以分配给新玩家。

语法

```
FGameLiftGenericOutcome RemovePlayerSession(const FString& playerSessionId)
```

参数

PlayerSessionId

Amazon GameLift 服务为响应调用软件开发工具包 Amazon GameLift API 操作 createPlayerSessi AWS on 而颁发的唯一身份证。连接到服务器进程时，游戏客户端会引用此 ID。

类型：FString

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

StartMatchBackfill()

发送请求以为使用 创建的游戏会话中的开放位置查找新玩家。另请参阅 AWS 软件开发工具包操作 AWSStartMatchBackfill()。通过此操作，托管游戏会话的游戏服务器进程可以发出匹配回填请求。在中了解有关 FlexMatch 回填功能的更多信息。

此操作为异步操作。如果成功匹配了新玩家，则 服务会使用回调函数 提供更新的对战构建器数据。

服务器进程每次只能具有一个活动的匹配回填请求。要发送新请求，请先调用 StopMatchBackfill() 以取消原始请求。

语法

```
FGameLiftStringOutcome StartMatchBackfill (FStartMatchBackfillRequest
 &startBackfillRequest);
```

参数

FStartMatchBackfillRequest

一个 FStartMatchBackfillRequest 对象，用于传递以下信息：

- 要分配给回填请求的票证 ID。此信息是可选的；如果未提供任何 ID，则 将自动生成一个 ID。
- 要将请求发送到的对战构建器。需要完整的配置 ARN。可从游戏会话的对战构建器数据中获得此值。
- 正在进行回填的游戏会话的 ID。
- 游戏会话的当前玩家的可用对战数据。

必需：是

返回值

如果成功，将返回对战回填票证作为 FGameLiftStringOutcome 对象。如果不成功，将返回错误消息。使用 AWS 软件开发工具包操作 AWSDescribeMatchmaking() 可跟踪票证状态。

StopMatchBackfill()

取消使用 StartMatchBackfill() 创建的活动对战回填请求。另请参阅 AWS 软件开发工具包操作 AWSStopMatchmaking()。在中了解有关 FlexMatch 回填功能的更多信息。

语法

```
FGameLiftGenericOutcome StopMatchBackfill (FStopMatchBackfillRequest
 &stopBackfillRequest);
```

参数

StopMatchBackfillRequest

一个 FStopMatchBackfillRequest 对象，用于识别要取消的对战票证：

- 分配给被取消的回填请求的票证 ID
- 回填请求所发送到的对战构建器
- 与回填请求关联的游戏会话

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

TerminateGameSession()

4.0.1 相反，服务器进程应在游戏会话结束ProcessEnding()后调用。

通知 Amazon GameLift 服务服务器进程已结束当前游戏会话。当服务器进程保持活动状态并准备托管新游戏会话时，将调用此操作。只有在游戏会话终止程序完成后才应调用它，因为它会向 Amazon GameLift 发出信号，表明服务器进程可以立即用于托管新的游戏会话。

如果服务器进程将在游戏会话停止后关闭，则不会调用此操作。取而代之的是，调用ProcessEnding()表示游戏会话和服务器进程都将结束。

语法

```
FGameLiftGenericOutcome TerminateGameSession()
```

参数

此操作没有参数。

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

UpdatePlayerSessionCreationPolicy()

更新当前游戏会话接受新玩家会话的能力。可将游戏会话设置为接受或拒绝所有新的玩家会话。(另请参阅 UpdateGameSession() 服务 API 参考 中的 操作)。

语法

```
FGameLiftGenericOutcome UpdatePlayerSessionCreationPolicy(EPlayerSessionCreationPolicy
 policy)
```

参数

策略

指示游戏会话是否接受新玩家的值。

类型：EPlayerSessionCreationPolicy 枚举。有效值包括：

- ACCEPT_ALL - 接受所有新玩家会话。
- DENY_ALL - 拒绝所有新玩家会话。

必需：是

返回值

返回由成功或失败组成的通用结果，并显示错误消息。

Unreal Engine 的 Amazon GameLift 服务器软件开发工具包 参考：数据类型

此 服务器 API 参考可帮助您准备 Unreal Engine 游戏项目用于 。有关集成过程的详细信息，请参阅 将 Amazon GameLift 添加到您的游戏服务器。

此 API 在 `GameLiftServerSDK.h` 和 `GameLiftServerSDKModels.h` 中定义。

设置 Unreal Engine 插件并查看代码示例将 Amazon GameLift 集成到虚幻引擎项目中。

- 操作
- 数据类型

fdescribePlayer 会话请求

此数据类型用于指定检索哪些玩家会话。您可以按如下方式使用它：

- 提供 PlayerSessionId 以请求特定的玩家会话。
- 提供 GameSessionId 以请求指定游戏会话中的所有玩家会话。
- 提供 PlayerId 以请求指定玩家的所有玩家会话。

对于大型玩家会话集合，请使用分页参数以在有序数据块中检索结果。

目录

GameSessionId

游戏会话的唯一标识符。使用此参数可请求指定游戏会话的所有玩家会话。游戏会话 ID 的格式如下所示：`arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>`。<ID string> 的值为自定义 ID 字符串（如果在创建游戏会话时指定了一个）或者是生成的字符串。

类型：字符串

必需：否

限制

要返回的最大结果数量。使用此参数和 NextToken 以一组连续页面的格式获取结果。如果指定玩家会话 ID，将忽略此参数。

类型：整数

必需：否

NextToken

令牌指示结果的下一个连续页面的开头。使用之前的调用返回此操作的令牌。要指定结果集的开始，请不要指定值。如果指定玩家会话 ID，将忽略此参数。

类型：字符串

必需：否

PlayerId

玩家的唯一标识符。玩家 ID 由开发人员定义。请参阅生成玩家 ID。

类型：字符串

必需：否

PlayerSessionId

玩家会话的唯一标识符。

类型：字符串

必需：否

PlayerSessionStatusFilter

用于筛选结果的玩家会话状态。可能的玩家会话状态包括以下内容：

- RESERVED - 已收到玩家会话请求，但玩家尚未连接到服务器进程和/或进行验证。
- ACTIVE - 服务器进程已验证玩家，当前已连接。
- COMPLETED - 玩家连接已断开。

- TIMEDOUT - 收到了玩家会话请求，但玩家未连接和/或在超时限制 (60 秒) 内验证。

类型：字符串

必需：否

FProcessParameters

此数据类型包含一组在 调用中发送到 服务的参数。

目录

port

服务器进程侦听新玩家连接的端口号。该值必须在部署此游戏服务器构建的任意实例集上所配置的端口范围内。此端口号包含在游戏会话和玩家会话对象中，游戏会话在连接到服务器进程时使用。

类型：整数

必需：是

logParameters

包含游戏会话日志文件目录路径列表的对象。

类型：TArray<FString>

必需：否

onStartGameSession

服务调用用于激活新游戏会话的回调函数的名称。 调用此函数响应客户端请求 CreateGameSession。回调函数采用 GameSession 对象（在 服务 API 参考 中定义）。

类型：FOnStartGameSession

必需：是

onProcessTerminate

服务调用以强制关闭服务器进程的回调函数的名称。调用此函数之后， 等待五分钟以便服务器进程关闭，然后使用 调用响应，再关闭服务器进程。

类型：FSimpleDelegate

必需：否

onHealthCheck

服务调用以从服务器进程请求运行状态报告的回调函数的名称。 每隔 60 秒调用此函数。调用此函数后， 将等待 60 秒接收响应，如果未收到响应，则会将服务器进程记录为不正常。

类型：FOnHealthCheck

必需：否

onUpdateGameSession

Amazon GameLift 服务为将更新的游戏会话对象传递给服务器进程而调用的回调函数的名称。为了提供更新的匹配数据而处理了匹配回填请求，Amazon GameLift 会调用此函数。它会传递一个 GameSession 对象、状态更新 (`updateReason`) 以及对战回填票证 ID。

类型：fonupdateGamesSession

必需：否

FStartMatchBackfillRequest

此数据类型用于发送对战回填请求。此信息在 调用中传递给 服务。

目录

GameSessionArn

游戏会话的唯一标识符。此 API 操作 GetGameSessionId() 返回采用 ARN 格式的标识符。

类型：FString

必需：是

MatchmakingConfigurationArn

采用 ARN 格式的唯一标识符，适用于用于此请求的对战构建器。要查找用于创建原始游戏会话的对战构建器，请查看对战构建器数据属性中的游戏会话对象。在使用对战构建器数据中了解有关对战构建器数据的更多信息。

类型：FString

必需：是

玩家

一组表示当前正在游戏会话中的所有玩家的数据。对战构建器使用此信息搜索与当前玩家非常匹配的新玩家。有关玩家对象格式的描述，请参阅 Amazon GameLift API 参考指南。要查找玩家属性、ID 和团队任务，请查看对战构建器数据属性中的游戏会话对象。如果对战构建器使用了延迟，则收集当前区域中更新的延迟并将其包含在每个玩家的数据中。

类型：TArray<FPlayer>

必需：是

TicketId

对战或匹配回填请求票证的唯一标识符。如果此处未提供任何值，则 Amazon GameLift 将生成一个采用 UUID 形式的值。使用此标识符可跟踪匹配回填票证状态或取消请求 (如需要)。

类型：FString

必需：否

FStopMatchBackfillRequest

此数据类型用于取消对战回填请求。此信息在 调用中传递给 服务。

目录

GameSessionArn

与被取消的请求关联的唯一游戏会话标识符。

类型：FString

必需：是

MatchmakingConfigurationArn

此请求发送到的对战构建器的唯一标识符。

类型：FString

必需：是

TicketId

要取消的回填请求票证的唯一标识符。

类型：FString

必需：是

游戏会话放置事件

A GameLift mazon 会在处理每个游戏会话放置请求时发出事件。您可以将这些事件发布到 Amazon SNS 主题，如请参阅设置游戏会话置放通知。中所述。这些事件还会以近乎实时的方式发送到 Amazon CloudWatch Events，并尽力而为。

本主题描述了游戏会话放置事件的结构，并提供了每种事件类型的示例。有关游戏会话放置请求状态的更多信息，请参阅 Amazon GameLift API 参考GameSessionPlacement中的。

放置事件语法

事件表示为 JSON 对象。事件结构符合 CloudWatch 事件模式，具有相似的顶级字段和特定于服务的详细信息。

顶级字段包括以下内容（有关更多详细信息，请参阅事件模式）：

版本

此字段始终设置为 0（零）。

id

事件的唯一标识符。

detail-type

此值始终为 `GameLift Queue Placement Event`。

源

此值始终为 `aws.gamelift`。

account

用于管理 Amazon 的 AWS 账户 GameLift。

time

事件时间戳

region

正在处理放置请求的地 AWS 区。这是正在使用的游戏会话队列所在的区域。

资源

正在处理放置请求的游戏会话队列的 ARN 值。

PlacementFulfilled

放置请求已成功完成。新的游戏会话已经开始，并且已经为游戏会话放置请求中列出的每位玩家创建了新的玩家会话。玩家连接信息可用。

详细语法：

placementID

分配给游戏会话放置请求的唯一标识符。

port

新游戏会话的端口号。

gameSessionArn

新游戏会话的 ARN 标识符。

ipAddress

游戏会话的 IP 地址。

DNSName

分配给正在运行新游戏会话的实例的 DNS 标识符。根据运行游戏会话的实例是否启用 TLS，值格式会有所不同。在支持 TLS 的实例集上连接到游戏会话时，玩家必须使用 DNS 名称，而不是 IP 地址。

支持 TLS 的实例集:. `<unique identifier>.<region identifier>.amazongamelift.com`

未启用 TLS 的实例集:. `ec2-<unique identifier>.compute.amazonaws.com`

startTime

表示何时将此请求放入队列的时间戳。

endTime

表示此请求何时完成的时间戳。

gameSessionRegion

AWS 主办游戏会话的舰队区域。这与中的区域令牌相对应GameSessionArn。

placedPlayerSessions

为游戏会话放置请求中的每位玩家创建的玩家会话集合。

示例

```
{
  "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-123213123232412412 3123"
      }
    ]
  }
}
```

PlacementCancelled

通过拨打 GameLift 服务电话，放置请求被取消StopGameSessionPlacement。

详细信息：

placementID

分配给游戏会话放置请求的唯一标识符。

startTime

表示何时将此请求放入队列的时间戳。

endTime

表示此请求何时取消的时间戳。

示例

```
{
  "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

在队列的时间限制到期之前，游戏会话放置未成功完成。根据需要，可以重新提交放置请求。

详细信息：

placementID

分配给游戏会话放置请求的唯一标识符。

startTime

表示何时将此请求放入队列的时间戳。

endTime

表示此请求何时取消的时间戳。

示例

```
{
  "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 无法完成游戏会话请求。这通常是由意外的内部错误引起的。根据需要，可以重新提交放置请求。

详细信息：

placementID

分配给游戏会话放置请求的唯一标识符。

startTime

表示何时将此请求放入队列的时间戳。

endTime

表示此请求何时失败的时间戳。

示例

```
{
  "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"
  }
}
```

生成 Amazon GameLift 定价估算值

使用 AWS Pricing Calculator，您可以为 Amazon GameLift 创建估算价格。您不需要具有 AWS 账户或深入了解 AWS 即可使用计算器。

AWS Pricing Calculator 计算器会引导您做出影响服务成本的决策，让您大致了解 Amazon GameLift 可能为您的游戏项目付出多少成本。如果您还不确定计划如何使用 Amazon GameLift，请使用默认值来生成估算值。在计划生产使用时，计算器可以帮助您测试潜在的场景并生成更准确的估算值。

您可以使用 AWS Pricing Calculator 生成以下 Amazon GameLift 托管选项的估算值：

- 估算 Amazon GameLift 托管
- 估算 Amazon GameLift 独立 FlexMatch

估算 Amazon GameLift 托管

此选项提供了在 Amazon GameLift 托管服务器上托管游戏的成本估算，包括服务器实例使用和数据传输的成本。FlexMatch 对战已包含在 Amazon GameLift 托管托管资源的成本中。

如果您在多个 AWS 区域或多个实例类型上托管或计划托管游戏服务器，请为每个区域和实例类型创建估算值。

Amazon GameLift 实例

本节可帮助您估算为玩家托管游戏会话所需的计算资源类型和数量。Amazon GameLift 使用 Amazon Elastic Compute Cloud (Amazon EC2) 实例来管理游戏服务器。在 Amazon GameLift 中，您可以部署一组具有特定实例类型和操作系统的实例。如果您拥有或计划拥有多个实例集，请为每个实例集创建估算值。

要开始使用，请打开 AWS Pricing Calculator 的 配置 Amazon GameLift 页面。添加描述，选择区域，然后选择估计 Amazon GameLift 托管（实例 + 数据传出）。在 Amazon GameLift 实例下，填写以下字段：

- 并发玩家峰值（CCU 峰值）

  这是可同时连接到游戏服务器的最大玩家数。此字段指示 Amazon GameLift 需要多少托管容量才能满足玩家的高峰需求。输入您希望使用所选 AWS 区域的实例托管的每日玩家峰值数量。

例如，如果您想让 1,000 名玩家同时连接到您的游戏，请保留默认值 **1000**。

- 每小时平均 CCU 占每日峰值 CCU 的百分比

  这是在 24 小时内每小时的平均并发玩家数。我们使用该值来估计 Amazon GameLift 需要为您的玩家维持的持续托管容量。如果您不确定要使用哪个百分比值，请保留默认的 **50**% 值。对于玩家需求稳定的游戏，我们建议输入 **70**% 值。

  例如，如果您的游戏的平均每小时 CCU 为 6,000，峰值 CCU 为 10,000，则输入百分比的 **60**% 值。

- 每个实例的游戏会话数

  这是您的每个游戏服务器实例可以同时托管的游戏会话数量。影响此数量的因素包括游戏服务器的资源要求、每个游戏会话中要托管的玩家数以及玩家性能预期。如果您知道游戏的并发游戏会话数，请输入该值。您也可保留默认值 **20**。

- 每个游戏会话的玩家数

  根据您的游戏设计中的定义，这是连接到游戏会话的平均玩家人数。如果游戏模式中有不同数量的玩家，请估计整个游戏中每个游戏会话的平均玩家人数。默认值为 **8**。

- 实例空闲缓冲区百分比

  这是为应对玩家需求的突然激增而需要保留的未使用托管容量的百分比。缓冲区大小是占实例集实例总数的百分比。默认值为 **10**%。

  例如，如果空闲缓冲区为 20%，则支持拥有 100 个活跃实例的玩家的实例集会保持 20 个空闲实例。

- 竞价型实例百分比

  Amazon GameLift 实例集可以组合使用按需型实例和竞价型实例。按需型实例可提供更可靠的可用性，而竞价型实例则提供了一种极具成本效益的替代方案。我们建议使用组合来优化成本节约和可用性。有关 Amazon GameLift 如何使用竞价型实例的信息，请参阅[按需型实例和竞价型实例]。

  在此字段中，输入要在实例集中维护的竞价型实例的百分比。我们建议将竞价型实例百分比设定在 50% 到 85% 之间。默认值为 **50**%。

  例如，如果您部署一个包含 100 个实例的实例集并指定 **40**%，则 Amazon GameLift 将维护 60 个按需型实例和 40 个竞价型实例。

- 实例类型

Amazon GameLift 实例集可以使用一系列 Amazon EC2 实例类型，其计算能力、内存、存储和联网能力各不相同。在配置 Amazon GameLift 实例集时，请选择最适合您的游戏需求的实例类型。有关使用 Amazon GameLift 选择实例类型的信息，请参阅选择 Amazon GameLift 计算资源。

如果您知道自己正在使用或计划在 Amazon GameLift 实例集中使用的实例类型，请选择该类型。如果您不确定选择哪种类型，请考虑选择 c5.large。这是一种具有中等大小和功能的高可用性类型。

- 操作系统

  此字段指定游戏服务器运行的操作系统，可以是 Linux 还是 Windows。默认值是 Linux。

数据传出 (DTO)

本节可帮助您估算游戏客户端和游戏服务器之间的流量成本。数据传输费仅适用于出站流量。入站数据传输不收取任何费用。

在 AWS Pricing Calculator 的配置 Amazon GameLift 页面上，展开 数据传出 (DTO)，然后填写以下字段：

- DTO 估计值类型

  您可以选择通过以下两种方式中的任何一种估计 DTO，具体取决于您如何跟踪游戏的数据传输。

  - 每月（以 GB 为单位）– 如果您跟踪游戏服务器的每月流量，请选择此类型。
  - 每位玩家 – 如果您按玩家跟踪数据传输，请选择此类型。这是默认类型。

    在以下字段中，您可以根据在上一节中计算的玩家时数来估算每位玩家的 DTO。

- 每月 DTO（以 GB 为单位）

  如果您选择每月（以 GB 为单位）DTO 估算类型，请输入每个区域每个实例的估计月度 DTO 使用量（以 GB 为单位）。

- 每位玩家的 DTO

  如果您选择了每位玩家 DTO 估算类型，请输入每位玩家的 DTO 估计使用量（以 KB/sec 为单位）。默认值为 **4**。

配置完 Amazon GameLift 价格估算值后，选择添加到我的估算值。有关在 AWS Pricing Calculator 中创建和管理估算值的更多信息，请参阅《AWS Pricing Calculator 用户指南》中的创建估算、配置服务和添加更多服务。

估算 Amazon GameLift 独立 FlexMatch

此选项提供了使用 FlexMatch 对战作为独立服务同时使用其他游戏服务器解决方案托管游戏的成本估算。这包括带有 FleetIQ 的 Amazon GameLift 自我管理托管和本地托管、点对点或云计算原始数据类型。独立 FlexMatch 的成本取决于所使用的计算能力。

如果您有或计划在不同 AWS 区域拥有多个对战者，请为每个地区创建估算值。

> **Note**
>
> Amazon GameLift FlexMatch 在以下区域可用：美国东部（弗吉尼亚州北部）、美国西部（俄勒冈州）、亚太地区（首尔）、亚太地区（悉尼）、亚太地区（东京)、欧洲地区（法兰克福）、欧洲地区（爱尔兰）。

要开始使用，请打开 AWS Pricing Calculator 的 配置 Amazon GameLift 页面。添加描述，选择一个区域，然后选择估算 Amazon GameLift 独立 FlexMatch。在 Amazon GameLift FlexMatch 下，填写以下字段：

- 并发玩家峰值（CCU 峰值）

  这是可同时连接到游戏服务器并请求对战的最大玩家数。输入您希望在所选区域匹配游戏会话的每日玩家峰值数量。

  例如，如果您想让多达 1,000 名玩家同时匹配，请保留默认值 **1000**。

- 每小时平均 CCU 占每日峰值 CCU 的百分比

  这是在 24 小时内每小时的平均并发玩家数。此值有助于估算您的对战请求量。如果您不确定要使用哪个百分比值，请保留默认的 **50**% 值。对于玩家需求稳定的游戏，我们建议输入 **70**% 值。

  例如，如果您的游戏的平均每小时 CCU 为 6,000，峰值 CCU 为 10,000，则输入百分比的 **60**% 值。

- 每次匹配的玩家人数

  根据您的游戏设计中的定义，这是匹配到游戏会话的平均玩家人数。如果游戏模式中有不同数量的玩家，请估计整个游戏中每个游戏会话的平均玩家人数。默认值为 **8**。

- 游戏时长（以分钟为单位）

这是玩家从头到尾在游戏会话中停留的平均时间。此值有助于确定玩家多久需要一场新匹配。输入玩家的平均游戏时长（以分钟为单位）。默认值为 **1**。

- 对战规则的复杂性

  对战规则的复杂性是指您用来匹配玩家的规则的数量和类型。规则集的复杂程度有助于确定每次匹配所需的计算能力。

  - 较低的复杂性 – 如果您的对战规则集包含的规则很少，使用更简单的规则类型（例如比较规则），并且具有以较少的尝试次数即可成功匹配的规则，请选择此选项。
  - 复杂性更高 – 如果您的对战规则集包含多个规则，使用更复杂的规则类型（例如距离或延迟规则），并且限制性规则会导致更多失败并需要更多匹配尝试，请选择此选项。

  有关规则复杂性和定价的更多信息，请参阅 Amazon GameLift 定价页面上的 Amazon GameLift FlexMatch。

配置完 Amazon GameLift FlexMatch 价格估算值后，选择添加到我的估算值。有关在 AWS Pricing Calculator 中创建和管理估算值的更多信息，请参阅《AWS Pricing Calculator 用户指南》中的创建估算、配置服务和添加更多服务。

限额和支持的区域

有关 AWS Amazon GameLift 服务限额，请参阅 Amazon GameLift 限额。

有关请求放宽 AWS 资源限额的信息，请参阅 AWS 服务限额。

有关 AWS 区域 支持 Amazon GameLift 的列表，请参阅 Amazon GameLift 区域。

Amazon GameLift 发行说明

Amazon GameLift 发行说明详细介绍了与该服务相关的新功能、更新和修复。

开发工具包版本

下表列出了所有包含软件开发工具包 GameLift 版本信息的 Amazon 版本。无需在游戏服务器和客户端集成中使用类似的软件开发工具包。但是，一个软件开发工具包的早期版本可能无法完全支持另一个软件开发工具包的最新功能。

有关 Amazon GameLift 软件开发工具包的更多信息，请参阅Amazon 为开发提供支持 GameLift。

要获取最新的亚马逊 GameLift 软件开发工具包，请访问亚马逊 GameLift软件开发工具包下载网站。

当前版本

服务 SDK
和服务客户端开发工具包

通用于 Unreal Engine 的插件 + 插件

更新

先前版本

服务发布	AWS SDK	服务器软件开发工具包				实时客户端软件开发工具包
		适用于 Unity 的 C# 插件	C++	适用于 Unreal Engine 的 C++ 插件	Go	
2023-12-14	1.11.225 或更高版本	5.1.0	5.1.1	5.1.0	5.0.0	1.2.0
2023-11-02	1.11.193 或更高版本	5.1.0	5.1.1	5.1.0	5.0.0	1.2.0
2023-09-28	1.11.144 或更高版本	5.1.0	5.1.1	5.1.0	5.0.0	1.2.0
2023-08-17	1.11.144 或更高版本	5.1.0	5.1.1	5.1.0	5.0.0	1.2.0
2023-07-27	1.11.111 或更高版本	5.1.0	5.1.0	5.0.2	5.0.0	1.2.0

服务发布	AWS SDK	服务器软件开发工具包				实时客户端软件开发工具包
		适用于 Unity 的 C# 插件	C++	适用于 Unreal Engine 的 C++ 插件	Go	
2023-06-29	1.11.111 或更高版本		5.0.4	5.0.2	5.0.0	1.2.0
2023-06-15	1.11.87 或更高版本		5.0.4	5.0.2	5.0.0	1.2.0
2023-05-25	1.11.87 或更高版本		5.0.3	5.0.2	5.0.0	1.2.0
2023-04-20	1.11.63 或更高版本				5.0.0	1.2.0

服务发布	AWS SDK	服务器软件开发工具包				实时客户端软件开发工具包
		适用于 Unity 的 C# 插件	C++	适用于 Unreal Engine 的 C++ 插件	Go	
2023-04-13	1.10.21 或更高版本				5.0.0	1.2.0
2023-02-09	1.10.21 或更高版本			3.4.0	5.0.0	1.2.0
2023-01-31	1.10.21 或更高版本			3.4.0	5.0.0	1.2.0
2022-12-01	1.10.21 或更高版本			3.4.0		1.2.0
2022-08-25	1.9.333 或更高版本		3.4.2	3.4.0		1.2.0
2021-10-28	1.9.133 或更高版本		3.4.2	3.4.0		1.2.0
2021-06-03	1.8.168 或更高版本		3.4.2	3.4.0		1.2.0

服务发布	AWS SDK	服务器软件开发工具包				实时客户端软件开发工具包
		适用于 Unity 的 C# 插件	C++	适用于 Unreal Engine 的 C++ 插件	Go	
2021-03-23	1.8.168 或更高版本		3.4.1	3.3.3		1.1.0
2021-03-16	1.8.163 或更高版本		3.4.1	3.3.3		1.1.0
2021-02-09	1.8.139 或更高版本		3.4.1	3.3.3		1.1.0
2020-12-22	1.8.95 或更高版本		3.4.1	3.3.3		1.1.0
2020-11-24	1.8.95 或更高版本		3.4.1	3.3.2		1.1.0
2020-11-11	1.8.36 或更高版本		3.4.1	3.3.2		1.1.0
2020-09-17	1.8.36 或更高版本		3.4.1	3.3.2		1.1.0

服务发布	AWS SDK	服务器软件开发工具包				实时客户端软件开发工具包
		适用于 Unity 的 C# 插件	C++	适用于 Unreal Engine 的 C++ 插件	Go	
2020-08-27	1.7.310 或更高版本		3.4.0	3.3.1		1.1.0
2020-04-16	1.7.310 或更高版本		3.4.0	3.3.1		1.1.0
2020-04-02	1.7.310 或更高版本		3.4.0			1.1.0
2019-12-19	1.7.249 或更高版本		3.4.0			1.1.0
2019-11-14	1.7.210 或更高版本		3.4.0			1.1.0
2019-10-24	1.7.210 或更高版本		3.4.0			1.1.0
2019-09-03	1.7.175 或更高版本		3.4.0			1.1.0

服务发布	AWS SDK	服务器软件开发工具包				实时客户端软件开发工具包
		适用于 Unity 的 C# 插件	C++	适用于 Unreal Engine 的 C++ 插件	Go	
2019-07-09	1.7.140 或更高版本		3.3.0			1.0.0
2019-04-25	1.7.91 或更高版本		3.3.0			1.0.0
2019-03-07	1.7.65 或更高版本		3.3.0			
2019-02-07	1.7.45 或更高版本		3.3.0			
2018-12-14	1.6.20 或更高版本		3.3.0			
2018-09-27	1.6.20 或更高版本		3.2.1			
2018-06-14	1.4.47 或更高版本		3.2.1			

服务发布	AWS SDK	服务器软件开发工具包			
		适用于 Unity 的 C# 插件	C++	适用于 Unreal Engine 的 C++ 插件	Go
2018-05-10	1.4.47 或更高版本		3.2.1		
2018-02-15	1.3.58 或更高版本		3.2.1		
2018-02-08	1.3.52 或更高版本		3.2.0		
2017-09-01	1.1.43 或更高版本		3.1.7		
2017-08-16	1.1.31 或更高版本		3.1.7		
2017-05-16	1.0.122 或更高版本		3.1.5		
2017-04-11	1.0.103 或更高版本		3.1.5		

实时客户端软件开发工具包

服务发布	AWS SDK	服务器软件开发工具包				实时客户端软件开发工具包
		适用于 Unity 的 C# 插件	C++	适用于 Unreal Engine 的 C++ 插件	Go	
2017-02-21	1.0.72 或更高版本		3.1.5			
2016-11-18	1.0.31 或更高版本		3.1.0			
2016-10-13	1.0.17 或更高版本		3.1.0			
2016-09-01	0.14.9 或更高版本		3.1.0			
2016-08-04	0.12.16 或更高版本		3.0.7			

发布说明

以下发行说明按时间顺序排列，首先列出最新更新。亚马逊 GameLift 于 2016 年首次发布。要了解早于此处所列的发行说明，请参阅开发工具包版本中的发布日期链接。

2024 年 4 月 24 日：亚马逊 GameLift 推出集装箱船队

Amazon GameLift 现在提供集装箱船队预览，这为您提供了更好的便携性、可扩展性、容错能力和灵活性。

在容器队列中，Amazon EC2 实例托管您的一个或多个容器。这些容器包括您的游戏服务器及其所需的任何内容，包括依赖项和配置。依赖关系的示例包括 SDK 和软件包。在您将集装箱上传到私有的 Amazon 弹性容器注册表后，亚马逊会用集装箱 GameLift 填充您的车队。

要在容器队列中运行，您的游戏服务器必须在 Linux 中运行并与 Server SDK 5.x 集成。在容器队列中，您可以微调对托管资源的控制，以便可以优化 CPU 单位和内存等资源的消耗。您还可以在容器中托管多个游戏服务器，以减少资源的使用。

在容器队列中，您可以获得许多与其他类型的队列相同的好处，例如按需实例类型、扩展（自动和手动）、队列和配对。您还可以获得与其他舰队类型相同的指标，以及一些新的集装箱指标。集装箱舰队可以让您在全球范围内覆盖以下地点的玩家：

- ap-northeast-1
- ap-northeast-2
- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2

要覆盖更多地区和本地区域，请创建多地点集装箱舰队。

了解更多：

- 使用亚马逊 GameLift 容器管理托管，《亚马逊 GameLift 开发者指南》
- CreateContainerGroupDefinition，亚马逊 GameLift API 参考

2024 年 2 月 13 日：亚马逊 GameLift 推出对 SDK 的改进，并简化了虚幻引擎亚马逊 GameLift 插件的安装

更新了 SDK 版本：

- Go Server SDK，版本 5.1.0
- C# 服务器开发工具包，版本 5.1.2
- C++ 服务器开发工具包，版本 5.1.2

我们进行了以下改进：

- 通过在网络中断时添加自动重新连接，提高了 SDK 的可靠性。
- [Go] 现在你可以`InitSDK()`使用或不带服务器参数进行调用。在 Amazon GameLift 托管 EC2 队列上运行的游戏服务器直接从环境变量中读取服务器参数。Amazon GameLift Anywhere 舰队上的游戏服务器必须`InitSDK()`使用服务器参数进行调用。

更新了插件版本：

- 适用于虚幻引擎的 Amazon GameLift 插件，版本 1.1.0
- 适用于 Unity 的亚马逊 GameLift 插件，版本 2.1.0
- 适用于虚幻引擎的 C++ 服务器 SDK 插件，版本 5.1.1
- 适用于 Unity 的 C# 服务器 SDK 插件，版本 5.1.2

我们进行了以下改进：

- [虚幻引擎的Amazon GameLift 插件] 更新了安装说明并简化了包装。该插件现在包含最新版本的虚幻引擎C++服务器SDK。
- 已升级插件以支持最新版本的 GameLift 服务器 SDK。

了解更多：

- 将游戏与亚马逊虚幻引擎 GameLift 插件集成，《亚马逊 GameLift 开发者指南》
- 下载亚马逊 GameLift 插件和 SDK

2023 年 12 月 14 日：亚马逊 GameLift 增加了更新活跃游戏会话游戏属性的功能

你已经能够在创建游戏会话时设置游戏属性，也可以在游戏会话中搜索指定的属性。现在，您还可以在活跃的游戏会话中添加和更新这些属性。

例如，你的玩家在他们想玩的地图上投票。您的游戏客户端调UpdateGameSession用将GameProperty值修改为{"Key": "map", "Value":"jungle"}。然后，您的游戏会在游戏会话中为玩家实现新地图。

游戏管理员还可以使用该SearchGameSessions操作从游戏属性中检索有用的数据。例如，管理员可以列出Status值为ACTIVE且此游戏属性的游戏会话：{"Key": "map", "Value":"desert"}。

了解更多：

- the section called “将 Amazon GameLift 添加到游戏客户端”，《亚马逊 GameLift 开发者指南》
- GameProperty，亚马逊 GameLift API 参考
- UpdateGameSession，亚马逊 GameLift API 参考
- SearchGameSessions，亚马逊 GameLift API 参考

2023 年 11 月 21 日：亚马逊 GameLift 推出对 Terraform 和 Pulumi 等基础设施即代码工具的支持 AWS Cloud Control API

现在，您可以使用基础设施即代码 (IaC) 工具管理整个 Amazon GameLift 资源堆栈。这些工具包括第三方工具 AWS CloudFormation，例如Terraform和Pulumi。有了这项新增的支持，您现在可以专注于开发游戏，并利用 DevOps 策略来处理资源管理、CI/CD 以及向客户部署。

现在，您还可以使用 AWS 云控制 API 配置和配置所有 Amazon GameLift 资源类型。您可以使用亚马逊 GameLift API 或亚马逊 AWS CloudFormation 模板继续使用资源 GameLift。

有关通过 iaC 提供的亚马逊 GameLift 资源的详细信息，请参阅亚马逊 GameLift 资源类型参考亚马逊 GameLift 资源类型参考。

此外，你现在还可以使用新的舰队属性，使用 AWS CloudFormation 模板或 AWS 云控制 API 自动扩展舰队：ScalingPolicies。

Cloud Control API 为开发人员提供了一组标准的 API，用于创建、读取、更新、删除和列出数百种 AWS 服务和多个第三方工具（如 Terraform 和 Pulumi）中的资源 (CRUDL)。

了解更多：

- AWS CloudFormation
- AWS 云控制 API
- AWS CC Terraform 提供商
- Pulumi

2023 年 11 月 16 日：亚马逊 GameLift 更新了 Unity 的独立插件

更新了 SDK 版本：适用于 Unity 的亚马逊 GameLift 插件，版本 2.0.0

适用于 Unity 的亚马逊 GameLift 插件提供了工具和工作流程，可简化在亚马逊上启动和运行 Unity 游戏以进行云托管的步骤 GameLift。Amazon GameLift 是一项完全托管的服务，允许游戏开发者管理和扩展用于基于会话的多人游戏的专用游戏服务器。

在此版本中，Unity 插件已更新为使用最新的亚马逊 GameLift 功能，包括服务器 SDK 版本 5.x，并支持使用 Amazon GameLift Anywhere 进行本地测试。该插件兼容 Unity 版本 Unity 2021.3 LTS 和 2022.3 LTS。

插件的主要功能包括：

- Unity 编辑器中针对以下场景的指导用户界面工作流程：
  - GameLift 使用您的本地工作站作为主机，测试您与 Amazon 的游戏集成。此工作流程可帮助您为本地计算机设置 Amazon GameLift Anywhere 舰队、启动游戏服务器和客户端的实例、通过 Amazon GameLift 请求游戏会话以及加入游戏。
  - 使用 Amazon 托管 EC2 和支持 AWS 资源，为您的集成游戏服务器部署云 GameLift 托管解决方案。此工作流程可帮助您为云托管配置游戏，并提供三种部署方案：
    - 将游戏服务器部署到单个舰队中。
    - 将游戏服务器部署到多个 AWS 区域的一组低成本竞价舰队中。
    - 使用 FlexMatch 匹配器部署游戏服务器。
- 能够设置与 AWS 账户用户关联的用户个人资料并设置默认 AWS 区域。您可以维护多个个人资料，以便在不同的 AWS 账户、账户用户和地区工作。
- 有助于简化 Amazon GameLift 集成和部署流程的特殊便利，包括：
  - 每种托管解决方案都包含支持 AWS 资源，包括提供唯一玩家 ID 和玩家验证的 Amazon Cognito 用户池。这些解决方案还包括用于存储的 Amazon S3 存储桶、Amazon SNS 事件通知、 AWS Lambda 函数和其他资源。
  - 对于Anywhere工作流程，该插件可自动设置所需的服务器参数。
  - 对于 Amazon EC2 工作流程，每个部署解决方案都提供使用 Lambda 函数的内置客户端后端服务。后端服务位于游戏客户端和亚马逊 GameLift服务之间，负责管理对亚马逊 GameLift服务的所有直接调用。
- 集成测试内容，包括用于说明游戏服务器和游戏客户端集成的简单示例多人游戏的资产和代码。
- 包含详细集成指南和示例代码的插件文档。

包括Anywhere和 Amazon EC2 队列在内的所有部署场景都使用 AWS CloudFormation 模板来描述和部署游戏解决方案的 AWS 资源。这些模板包含在 Amazon GameLift 插件下载中。你可以按原样使用它们，也可以为你的游戏自定义它们。

了解更多：

- 适用于 Unity 的亚马逊 GameLift 插件服务器指南 SDK 5.x，《亚马逊 GameLift 开发者指南》
- 从以下地址下载插件 GitHub
- 关于亚马逊 GameLift 托管
- 亚马逊 GameLift 论坛

2023 年 11 月 2 日：亚马逊 GameLift 增加了对共享凭证的支持

更新了 SDK 版本：S AWS DK 1.11.193

新的 Amazon GameLift 共享凭证功能允许部署在托管 EC2 队列上的应用程序与其他 AWS 资源进行交互。此更新会影响您捆绑和部署的应用程序以及与服务器 SDK 版本 5.x 或更高版本集成的游戏服务器二进制文件。（游戏服务器可执行文件已经可以使用服务器软件开发工具包 5.x `GetFleetRoleCredentials()` 操作请求凭证。）

例如，如果您想使用 Amazon CloudWatch 代理部署游戏服务器版本来收集 EC2 实例指标和其他数据，则该代理需要权限才能与您的 CloudWatch 资源进行交互。为此，您必须先设置一个具有 CloudWatch 资源使用权限 AWS Identity and Access Management 的 IAM () 角色，然后在启用 IAM 角色和共享凭证的情况下配置队列。当 Amazon 将您的游戏服务器版本 GameLift 部署到每个 EC2 实例时，它会生成一个共享凭证文件并将其存储在实例上。实例上的所有应用程序都可使用共享凭证。Amazon GameLift 会在实例的整个生命周期内自动刷新临时证书。

您可以使用以下方法在创建托管 EC2 实例集时启用共享凭证：

- 在 Amazon GameLift 控制台舰队创建工作流程中。
- `CreateFleet`使用新参数调用 Amazon GameLift 服务 API 操作时`InstanceRoleCredentialsProvider`。
- 使用参数调用 AWS CLI 操作`aws gamelift create-fleet`时`instance-role-credentials-provider`。

了解更多：

- 《Amazon GameLift 开发者指南》，与您的车队中的其他 AWS 资源进行沟通

- CreateFleet， InstanceRoleCredentialsProvider，亚马逊 GameLift API 参考
- 设置 IAM 服务角色，《亚马逊 GameLift 开发者指南》

2023 年 9 月 28 日：亚马逊 GameLift 发布全新的虚幻引擎独立插件

更新了 SDK 版本：适用于虚幻引擎版本 1.0.0 的 Amazon GameLift 插件

虚幻引擎的亚马逊 GameLift 插件提供了工具和工作流程，可简化您在亚马逊云托管上启动和运行游戏 GameLift 的步骤。Amazon GameLift 是一项完全托管的服务，允许游戏开发者管理和扩展用于基于会话的多人游戏的专用游戏服务器。该插件支持 UE 版本 5.0、5.1 和 5.2。主要特征包括：

- Unreal 编辑器中的引导式用户界面工作流程通过以下路径逐步执行：
  - GameLift 使用您的本地工作站作为主机，测试您与 Amazon 的游戏集成。此工作流程可帮助您为本地计算机设置 Amazon GameLift Anywhere 舰队、启动游戏服务器和客户端的实例、通过 Amazon GameLift 请求游戏会话以及获取新游戏会话的连接信息。
  - 为您的集成游戏服务器部署 Amazon EC2 云托管解决方案。此工作流程可帮助您为云托管配置游戏，并提供三种不同的部署场景：部署到单个舰队、部署到多个区域的一组现货舰队或使用匹配器部署到一组舰队。 FlexMatch 每种部署场景的解决方案都包括 Amazon GameLift 资源和支持 AWS 资源。
- 能够设置与 AWS 账户用户关联的用户配置文件并定义默认 AWS 区域。您可以维护多个个人资料，以便在不同的 AWS 账户、账户用户和地区工作。
- 有助于简化 Amazon GameLift 集成和部署流程的特殊便利，包括：
  - 每种托管解决方案都包括支持 AWS 资源，包括提供唯一玩家 ID 的基本 Amazon Cognito 用户池、用于存储的 Amazon S3 存储桶、Amazon SNS 事件通知和功能。 AWS Lambda
  - 对于 Anywhere 工作流程，该插件使用命令行参数自动设置所需的服务器参数。
  - 对于 Amazon EC2 工作流程，每个部署解决方案都提供使用 Lambda 函数的内置客户端后端服务。后端服务接收来自游戏客户端的请求并将其传递给 Amazon GameLift 服务。
- 集成测试内容，包括启动游戏地图和两张测试地图，其中包含基本蓝图和用户界面元素。
- 包含详细集成指南和示例代码的插件文档。

所有部署方案，包括适用于 Amazon EC2 队列Anywhere和 Amazon EC2 队列，都使用 AWS CloudFormation 模板来描述解决方案。该插件在为您的游戏部署 Amazon GameLift 资源时使用这些模板。这些模板包含在 Amazon GameLift 插件下载中，并且可以编辑。您可以按原样使用它们，也可以根据游戏进行修改。

了解更多：

- 将游戏与适用于虚幻引擎的 Amazon GameLift 插件集成，《亚马逊 GameLift 开发者指南》
- 从以下地址下载插件 GitHub
- 关于亚马逊 GameLift 托管
- 亚马逊 GameLift 论坛

2023 年 8 月 17 日：亚马逊 GameLift 提供搭载 AWS Graviton 处理器的游戏服务器托管

更新了 SDK 版本：S AWS DK 1.11.144

借助 Amazon， GameLift 您现在可以使用带有 AWS Graviton 处理器的 EC2 实例在云端托管游戏。Graviton 实例 AWS 采用基于 ARM64 的处理器设计，可为使用 EC2 的云工作负载提供最佳的性价比，与基于 x86 的同类实例相比，最高可提高 40%。与早期版本相比，最新的 Graviton3 处理器的计算性能提高了 25%。

借助 Amazon GameLift，您现在可以从 AWS Graviton 系列中的以下新实例中进行选择：

- 基于 Graviton2 的实例：c6g、c6gn、r6g、m6g、g5g
- 基于 Graviton3 的实例：c7g、r7g、m7g

了解更多：

- AWS Graviton Proces sor：了解基于 Graviton 的 EC2 实例的好处和实际用途。
- Graviton 入门：概述基于 Graviton 的实例，并根据操作系统、语言和运行时间，深入了解应用程序如何在其上运行。

> **Note**
>
> Graviton Arm 实例需要在 Linux 操作系统上构建亚马逊 GameLift 服务器。C++ 和 C# 需要服务器软件开发工具包 5.1.1 或更高版本。Go 需要服务器软件开发工具包 5.0 或更高版本。这些实例不 out-of-the-box 支持在亚马逊 Linux 2023 (AL2023) 或亚马逊 Linux 2 (AL2) 上安装 Mono。

2023 年 7 月 27 日：亚马逊 GameLift 发布服务器 SDK 5.1.0，增加了对 Unity 开发的支持

更新的软件开发工具包版本：适用于 C++、C#/Unity、Unreal 5.1.0 的服务器软件开发工具包

最新版本的 Amazon GameLift 服务器 SDK 提供了 C++、C# 和虚幻插件的更新，以及与 Unity 游戏引擎配合使用的新插件。游戏开发者将 Amazon GameLift 服务器 SDK 集成到他们部署在亚马逊上托管的游戏服务器中 GameLift。

最新的服务器软件开发工具包版本包含以下更新，其中包括许多客户请求：

- 下载特定语言的 SDK 包 — 更新后的 A mazon GameLift 下载网站包含每种语言的 SDK 包。您可以下载当前或以前的版本。
- 适用于 Unity 的新 C# 服务器软件开发工具包插件 – 适用于 Unity 的新服务器软件开发工具包包含内置的 C# 库，您可以在 Unity 编辑器中使用程序包管理器安装这些库（请参阅新的 Unity 集成指南）。这些库包括所需的依赖关系 UnityNuGet。可以将此插件与适用于 Windows 和 Mac OS 的 Unity 2020.3 LTS、2021.3 LTS 和 2022.3 LTS 一起使用。它支持 Unity 的 .NET 框架和 .NET 标准配置文件，以及 .NET 标准 2.1 和 .NET 4.x。
- 适用于 C# 的整合 .NET 解决方案 – 适用于 C# 的服务器软件开发工具包现在在一个解决方案中支持 .NET 框架 4.6.2（从 4.6.1 升级）和 .NET 6.0。.NET 标准 2.1 在 Unity 构建的库中可用。
- 服务器软件开发工具包 5.1.0 更新
  - [C++、C#、Unreal] 现在可以使用或不带服务器参数调用 `InitSDK()`。在 Amazon GameLift 托管 EC2 队列上运行的游戏服务器直接从环境变量中读取服务器参数。Amazon GameLift Anywhere 舰队上的游戏服务器必须`InitSDK()`使用服务器参数进行调用。
  - [C++、C#、Unreal] 服务器软件开发工具包调用改进了错误消息。
  - [C++ 软件开发工具包] 为了缩短服务器软件开发工具包的构建时间，默认情况下，构建标志 `-DRUN_CLANG_FORMAT` 处于禁用状态。可以通过 `-DRUN_CLANG_FORMAT=1` 启用。
  - [C++ 软件开发工具包] 在不使用标准库（`-DGAMELIFT_USE_STD=0`）的情况下构建库时，`InitSDK()` 不再使用 `std::` 数据类型。
- 扩展服务器软件开发工具包 5.x 文档
  - 更新了 C++、C#/Unity 和 Unreal 的服务器软件开发工具包参考指南，包括扩大了对所有数据类型的覆盖范围。
    - 适用于 C# 和 Unity 的 Amazon GameLift 服务器软件开发工具包 5.x 参考
    - 适用于 C++ 的 Amazon GameLift 服务器软件开发工具包 5.x 参考
    - Amazon GameLift Unreal Engine 服务器软件开发工具包 5.x 参考

- 适用于 Unity 和 Unreal 插件的服务器软件开发工具包 5 集成指南的新版本
  - 将 Amazon GameLift 集成到 Unity 项目中
  - 将 Amazon GameLift 集成到虚幻引擎项目中
- 其他文档更新
  - 修订了亚马逊 GameLift 服务 API 操作文档 GetInstanceAccess，GetComputeAccess并根据正在使用的亚马逊 GameLift 服务器 SDK 版本阐明了远程访问程序。
  - 修改了描述 GameSessionPlacement，以记录展示位置处于 “待定” 状态时游戏会话信息是如何瞬态的。

2023 年 7 月 13 日：亚马逊 GameLift 增加了车队硬件指标

现在，您可以跟踪您的 Amazon GameLift 托管 EC2 队列的硬件性能指标。指标包括 CPU 利用率、网络流量和磁盘读/写活动的 EC2 实例指标。对于 Amazon GameLift，这些指标描述了队列所在地的所有活跃实例。您可以使用中的 Amazon CloudWatch 控制面板查看这些舰队硬件指标 AWS Management Console。您也可以在Amazon GameLift 控制台的舰队详情中查看它们。

了解更多：

- 使用 Amazon CloudWatch 监控 Amazon GameLift（车队指标），《亚马逊 GameLift 开发者指南》

2023 年 6 月 29 日：亚马逊 GameLift 推出对 2023 年亚马逊 Linux 的支持

更新了 SDK 版本：S AWS DK 1.11.111

亚马逊 GameLift 客户现在可以使用亚马逊 Linux 2023 操作系统来托管他们的游戏服务器。AL2023 比 AL2 提供了多项改进，包括安全性。除中国地区外，该操作系统在所有 AWS 区域 地区均可用。

当 Amazon Linux (AL1) 的支持于 2023 年 12 月终止时，客户可以使用较新的 Linux 操作系统并继续收到重要的安全更新。Amazon Linux 2 的支持将持续到 2025 年。

了解更多：

- 亚马逊 GameLift Linux 服务器常见问题解答
- 比较 Amazon Linux 2 和 Amazon Linux 2023
- 亚马逊 GameLift API 参考链接：
  - AWS SDK 操作 `CreateBuild`
  - CLI 命令 `upload-build`

- CLI 命令 `create-build`

2023 年 5 月 25 日：亚马逊 GameLift FleetiQ 添加了过滤器，在耗尽的实例上排除游戏会话放置

更新了 SDK 版本：S AWS DK 1.11.87

如果您使用 Amazon GameLift FleetiQ 进行游戏托管，则现在可以阻止在当前耗尽的实例上放置游戏会话。耗尽的实例会被标记为已关闭，但如果没有其他托管资源可用，仍然可以选择它们来托管新的游戏会话。借助这项新特征，您可以完全排除使用耗尽的实例。

调用 `ClaimGameServer` 查找可用的游戏服务器时，请使用此特征。添加新 `FilterOption` 参数并将允许的实例状态设置为仅限 ACTIVE。作为回应，Amazon GameLift FleetiQ 在搜索和申领可用游戏服务器时仅查看活动实例。

了解更多：

- ClaimGameServer在 Amazon GameLift API 参考中
- 亚马 GameLift 逊 FleetiQ 开发者指南中 FleetiQ 的工作原理

2023 年 5 月 16 日：亚马逊 GameLift 支持为车队添加成本分配标签

Amazon GameLift 客户现在可以使用 AWS Billing 成本分配标签来组织他们的游戏托管成本。您可以为各 GameLift 个 Amazon EC2 队列资源分配成本分配标签，以跟踪您的队列如何影响总体托管成本。

了解更多：

- 管理您的游戏托管成本
- 使用 AWS 成本分配标签，《AWS Billing 用户指南》

2023 年 4 月 20 日：亚马逊 GameLift 推出对 2016 年 Windows Server 的支持

更新了 SDK 版本：S AWS DK 1.11.63

亚马逊 GameLift 客户现在可以使用 Windows Server 2016 操作系统来托管他们的游戏服务器。该操作系统全部可用 AWS 区域。随着微软将于 2023 年 10 月终止对 Windows Server 2012 的支持，客户可以使用更新的 Windows 操作系统并继续获得重要的安全更新。

从今天开始，需要 Windows 运行时环境的新客户在创建用于托管的新游戏服务器构建时必须指定 Windows Server 2016。现有客户可以继续使用 Windows Server 2012 创建新的构建和实例集，但必须在 2023 年 10 月 10 日 Microsoft 终止支持日期之前完成 Windows Server 2016 的迁移。

此次更新包含以下服务更改：

- 使用 Amazon GameLift SDK 或 CLI 命令创建游戏服务器版本时，您现在必须明确设置操作系统。不再有默认值。要在 Windows Server 2016 上部署游戏服务器，请使用值 `WINDOWS_2016`。
- 使用 Amazon GameLift 控制台创建游戏服务器版本时，必须从可用值中选择一个操作系统。如果是拥有活跃 Windows Server 2012 实例集的现有客户，则可以选择 `WINDOWS_2012` 或 `WINDOWS_2016`。

了解更多：

- 亚马逊 GameLift API 参考链接：
  - CLI 命令 `upload-build`
  - CLI 命令 `create-build`
  - AWS SDK 操作 `CreateBuild`
- 适用于 Windows 的亚马逊 2012 GameLift 常见问题解答

2023 年 4 月 13 日：亚马逊 GameLift 推出适用于虚幻引擎的服务器 SDK 5.x

更新的开发工具包版本：适用于 Unreal 的服务器软件开发工具包 5.0.0

最新版本的亚马逊虚幻引擎 GameLift 轻量级插件现在基于亚马逊 GameLift 服务器SDK 5.x。要开始将你的虚幻引擎环境与Amazon集成， GameLift 请参阅以下链接。

了解更多：

- 将 Amazon GameLift 集成到虚幻引擎项目中
- 将 Amazon GameLift 添加到您的游戏服务器
- 适用于 C++ 的 Amazon GameLift 服务器软件开发工具包 5.x 参考

2023 年 3 月 14 日：亚马逊 GameLift 推出全新的主机体验

全新 Amazon GameLift 控制台包括以下改进：

- 改进了导航-新的导航窗格便于在 Amazon GameLift 资源之间导航。
- 亚马逊 GameLift 登录页面 — 新的登录页面提供指向有用文档的链接，显示亚马逊的高级概述 GameLift，并通过文档链接、常见问题和提供支持 AWS re:Post。
- 改进了亚马逊 CloudWatch 指标 — 亚马逊 GameLift 指标现在可在亚马逊 GameLift 控制台和您的控制 CloudWatch面板中使用。此更新还包括性能、利用率和玩家会话的新指标。

了解更多：

- 在控制台中查看您的游戏数据
- 管理 Amazon GameLift 托管资源
- 建立 FlexMatch 媒人

2023 年 2 月 14 日：亚马逊 GameLift 现在支持对亚马逊 SNS 主题进行服务器端加密

SNS 主题的服务器端加密 (SSE) 可加密您的敏感静态数据。SSE 使用 AWS Key Management Service (AWS KMS) 密钥来保护您的 SNS 主题的内容。

了解更多：

- 请参阅设置游戏会话置放通知。
- FlexMatch对接会活动
- 静态加密

2023 年 2 月 9 日：亚马逊 GameLift 服务器 SDK 支持带有 C #10 的.NET 6

更新了软件开发工具包版本：适用于 .NET 6 的服务器软件开发工具包 5.0.0。无需更新软件开发工具包。

如果您使用 Unity 实时开发平台，请继续使用带有.NET 4.6 的亚马逊 GameLift服务器软件开发工具包 5.0.0。Unity 不支持 .NET 6。

了解更多：

- 在亚马逊下载最新版本的亚马逊 GameLift 服务器 SDK GameLift 入门指南
- 适用于 C# 和 Unity 的 Amazon GameLift 服务器软件开发工具包 5.x 参考

2023 年 1 月 31 日：亚马逊 GameLift 服务器 SDK 支持 Go 语言

更新的软件开发工具包版本：适用于 Go 的服务器软件开发工具包 5.0.0

了解更多：

- 在亚马逊下载最新版本的亚马逊 GameLift 服务器 SDK GameLift 入门指南
- 适用于 Go 的 Amazon GameLift 服务器软件开发工具包 参考

2022 年 12 月 1 日：亚马逊 GameLift 推出亚马逊 GameLift Anywhere和亚马逊 GameLift 服务器 SDK 5.0

更新了 SDK 版本： AWS SDK 1.10.21、适用于 C++ 和 C# 的服务器 SDK 5.0.0

亚马逊 GameLift Anywhere使用您的游戏服务器资源来托管亚马逊 GameLift 游戏服务器。您可以使用 Amazon GameLift Anywhere 将自己的计算资源与 Amazon GameLift 托管 EC2 计算集成，从而将您的游戏服务器分配到多种计算类型。您也可以使用 Amazon GameLift Anywhere 对游戏服务器进行迭代测试，无需在每次迭代时都将版本上传到亚马逊 GameLift 。

要点：

- 全新 Amazon GameLift Anywhere 舰队和计算类型
- Amazon GameLift Anywhere 计算资源注册
- 改进了测试迭代周期

Amazon S GameLift erver SDK 5.0.0 引入了对现有服务器软件开发工具包的改进和一种新的资源类型“计算”。服务器 SDK 5.0.0 支持 Amazon GameLift Anywhere 以及使用您自己的计算资源托管游戏服务器。

了解更多：

- Amazon GameLift 服务器软件开发工具包参考
- 实例集位置
- 选择 Amazon GameLift 计算资源
- 创建亚马逊 GameLift Anywhere舰队

2022 年 8 月 25 日：亚马逊 GameLift 推出对 Local Zones 的支持

更新了 SDK 版本：S AWS DK 1.9.333

Amazon GameLift 现已在美国的八个 Local Zones 上市，因此您可以将舰队部署到离玩家更近的地方。通过将本地区域添加到队列中，您可以将所有亚马逊托管 GameLift 功能与本地区域一起使用。

Local Zones 将 AWS 资源和服务扩展到云端边缘，靠近人口众多、工业和信息技术 (IT) 中心。这意味着您可以将需要几毫秒延迟的应用程序部署到离最终用户或本地数据中心更近的地方。

了解更多：

- Local Zones
- 实例集位置
- 创建 Amazon GameLift 托管实例集

2022 年 6 月 28 日：亚马逊 GameLift 推出全新的可选主机体验

全新 Amazon GameLift 控制台包括以下改进：

- 改进了导航-新的导航窗格便于在 Amazon GameLift 资源之间导航。
- 亚马逊 GameLift 登录页面 — 新的登录页面提供指向有用文档的链接，显示亚马逊的高级概述 GameLift，并通过文档链接、常见问题和提供支持 AWS re:Post。
- 改进了亚马逊 CloudWatch 指标 — 亚马逊 GameLift 指标现在可在亚马逊 GameLift 控制台和您的控制 CloudWatch面板中使用。此更新还包括性能、利用率和玩家会话的新指标。

了解更多：

- 在控制台中查看您的游戏数据
- 管理 Amazon GameLift 托管资源
- 建立 FlexMatch 媒人

2022 年 2 月 15 日： FlexMatch 添加复合规则和其他改进

FlexMatch 用户现在可以访问以下功能：

- 复合规则 – 增加了对 40 人或更少玩家的对战的复合对战规则的支持。现在，您可以使用逻辑语句创建复合规则来形成对战。如果您的规则集中没有复合规则，则要形成对战，则规则集中的所有规则

都必须为真。使用复合规则，您可以使用以下逻辑运算符选择要应用的规则：and、or、not、和 xor。

- 灵活的团队选择 – 更新了对战属性表达式，支持选择所有可用队伍的子集。
- 更长的字符串列表 – 将玩家属性值字符串列表中的最大字符串数从 10 增加到 100。

了解更多：

- Amazon GameLift FlexMatch 开发者指南：
  - FlexMatch 规则类型
  - FlexMatch 属性表达式
- AttributeValue: SL

2021 年 10 月 28 日：亚马逊 GameLift 增加了对亚太地区（大阪）地区的多区域队列的支持；亚马逊 FleetiQ 增加了对 Graviton2 GameLift 处理器的支持 AWS

更新了 SDK 版本：S AWS DK 1.9.133

Amazon GameLift 现已在亚太地区（大阪）地区上市。游戏开发者现在可以使用 GameLift 多区域队列在大阪部署实例。

与基于 Intel 的同等计算选项相比，您现在可以使用基于 ARM 的处理器架构的 Graviton2 托管游戏服务器以更低的成本提高性能。

要点：

- Amazon GameLift 现已在亚太地区（大阪）地区上市。
- 现在可以将 Amazon GameLift FleetiQ 游戏服务器组配置为管理 Graviton2 实例系列 c6g、m6g 和 r6g。

了解更多：

- Amazon GameLift 多区域舰队
- CreateGameServerGroup
- AWS 引力子处理器

2021 年 9 月 20 日：亚马逊 GameLift 发布适用于 Unity 的插件

适用于 Unity 1.0.0 版本的亚马逊 GameLift 插件包含库和原生用户界面，可让您更轻松地访问亚马逊 GameLift 资源 GameLift 并将亚马逊集成到您的 Unity 游戏中。您可以使用适用于 Unity 的亚马逊 GameLift 插件来访问亚马逊 GameLift API 并部署适用于常见游戏场景的 AWS CloudFormation 模板。该插件还包括一个适用于示例场景的示例游戏。您可以使用 Amazon L GameLift ocal 查看游戏客户端和游戏服务器之间传递的消息，以了解典型游戏如何与亚马逊 GameLift互动。

适用于 Unity 的插件支持 Unity 2019.4 LTS 和 2020.3 LTS。

要点：

- 构建、运行和修改具有不同场景的示例游戏，或者自行创建游戏。
- 为典型的游戏 AWS CloudFormation 场景部署示例场景，包括仅限身份验证、单区域舰队、带有队列和自定义匹配器的多区域舰队、带有队列和自定义匹配器的竞价舰队，以及. FlexMatch

了解更多：

- 将游戏与适用于 Unity 的 Amazon GameLift 插件集成

2021 年 6 月 30 日： FlexMatch 添加 batchDistance 规则

您可以使用 batchDistance 规则类型来指定字符串或数字属性，从而为每个区段带来诸多优势。

要点：

- 对于大型对战（超过 40 名玩家），现在可以根据技能、模式和地图获得同样的平衡，而不是仅通过技能平衡玩家。确保对战中的每个人都在一个技能范围内，对多个数字属性（例如联赛或对战风格）进行划分，并根据诸如地图或游戏模式之类的字符串属性进行分组。您也可以随着时间的推移创建扩展。例如，可以创建扩展，让玩家等待的时间越长，进入对战的技能等级范围就越大。

  对于 40 人以下的对战，您可以使用新的简化规则表达式。

2021 年 6 月 3 日：亚马逊 GameLift 实时客户端 SDK 和服务器 SDK 更新

更新的软件开发工具包版本：实时客户端软件开发工具包 1.2.0、适用于 Unreal 的服务器软件开发工具包 3.4.0

通过最新的软件开发工具包更新，您现在可以将 IL2CPP 集成到使用 RTS 客户端软件开发工具包并遵循框架最佳实操的移动应用程序中。你现在也可以为虚幻版本4.26构建亚马逊 GameLift 服务器软件开发工具包。此更新包含与你的 Windows 或 Linux 游戏服务器集成的组件，包括 C++ 和 C# 版本的亚马逊 GameLift 服务器 SDK、Amazon L GameLift ocal 和虚幻引擎插件。

要点：

- 在 RTS 客户端软件开发工具包中增加了对 IL2CPP 的支持，也支持将原生库构建为框架，因此您可以为最新的移动设备构建 RTS 客户端。
- 可以使用 DescribePlayerSessions() 获取单个玩家会话的信息、游戏会话中所有玩家会话的信息或者与单个玩家 ID 相关联的所有玩家会话的信息。
- 可以使用 GetInstanceCertificate() 检索与实例集及其实例相关联的经过 PEM 编码的 TLS 证书的文件位置。
- 为 Unreal 版本 4.26 创建了服务器软件开发工具包支持。
- 现有 C# 软件开发工具包 4.0.2 版本已通过验证与 Unity 2020.3 兼容。无需更新软件开发工具包。

了解更多：

- Amazon GameLift 开发者指南：
  - DescribePlayerSessions()
  - GetInstanceCertificate()

2021 年 3 月 23 日：亚马逊 GameLift 为游戏会话放置添加通知

更新了 SDK 版本：S AWS DK 1.8.168

现在，您可以使用事件来监控游戏会话队列的游戏会话放置活动。创建亚马逊简单通知服务 (Amazon SNS) Service 主题以发布事件通知，或者使用事件设置事件跟踪。 CloudWatch

要点：

- 对于每个队列，您可以设置要包含在所有事件消息中的自定义文本字符串。
- 使用 Amazon SNS 主题时，您可以设置其他访问条件，将发布限制为特定队列。

了解更多：

- Amazon GameLift 开发者指南：

- 请参阅设置游戏会话置放通知。（新）
- 游戏会话放置事件（新）

- API 参考（AWS 软件开发工具包）
  - 新的游戏会话队列参数NotificationTarget和CustomEventData:GameSessionQueue, CreateGameSessionQueue, UpdateGameSessionQueue
- 亚马逊 GameLift论坛

2021 年 3 月 16 日：亚马逊 GameLift 增加了多区域机队，六个新区域

更新了 SDK 版本：S AWS DK 1.8.163

Amazon 托 GameLift 管主机现已在 21 个 AWS 地区推出。新的区域分别是开普敦（af-south-1）、巴林（me-south-1）、香港特别行政区（ap-east-1）、米兰（eu-south-1）、巴黎（eu-west-3）和斯德哥尔摩（eu-north-1）。

借助新的亚马逊 GameLift 多地点队列功能，您现在可以设置一个舰队，将游戏服务器托管在亚马逊 GameLift支持的20个区域中的任意或全部（北京地区除外）。此功能旨在显著减少在全球范围内设置和维护Amazon GameLift 托管资源所需的工作。可以在以下 AWS 地区创建多地点舰队：us-east-1（弗吉尼亚北部）、（俄勒冈）、us-west-2eu-central-1（法兰克福）、eu-west-1（爱尔兰）、ap-southeast-2（悉尼）、ap-northeast-1（东京）和ap-northeast-2（首尔）。在所有其他区域，您可以根据需要继续设置单个位置实例集。在此版本之前创建的所有实例集均为单个位置实例集。使用多位置实例集不会影响您的托管成本。Amazon 的GameLift定价取决于您使用的实例的类型、位置和数量。（有关更多信息，请参阅 Amazon GameLift 定价。）AWS CloudFormation 不久将提供对多地点舰队的支持。

> **Note**
>
> 多位置实例集在中国区域中不可用。位于中国地区的亚马逊 GameLift资源不能与其他 Amazon GameLift 地区的资源交互或供其使用。

要点：

- 对于多位置实例集，请明确添加远程位置列表。Amazon 将相同类型和配置的实例（包括构建和运行时配置）GameLift 部署到队列所在地区和所有添加的地点。
- 分别调整每个位置的容量设置和扩展。自动扩缩策略适用于整个实例集，但您可以按位置将其启用或关闭。

- 在特定的实例集位置开始新的游戏会话。使用游戏会话队列或对战来放置游戏会话时，您现在可以根据位置、托管成本和玩家延迟来确定新游戏会话的起始位置。
- 在 Amazon GameLift 控制台中获取托管指标，按队列中的所有位置汇总或按每个舰队位置细分。

了解更多：

- Amazon 游戏科技博客
- API 参考（AWS 软件开发工具包）
  - 新的舰队定位行动：CreateFleetLocations、DescribeFleetLocationAttributes、DescribeFleetLocationCapacity、Describe
  - 更新了舰队运营，增加了新的多地点支持：CreateFleet、、desc ribeEC2 UpdateFleetCapacityInstanceLimits、、、DescribeInstancesStopFleetActionsStartFleetActions
  - 更新了游戏会话放置操作，增加了新的优先级和筛选功能：CreateGameSessionQueue，DescribeGameSessionQueues，UpdateGameSessionQueue
  - 更新了游戏会话创建操作，增加了新的位置支持：CreateGameSession、DescribeGameSessions、DescribeGameSessionDetails、SearchGameSess
- Amazon GameLift 开发者指南：
  - 亚马逊 GameLift 托管地点（已更新）
  - Amazon GameLift 实例集设计指南（新）

    扩展 Amazon GameLift 托管容量（已更新）
  - 设计游戏会话队列（新）
  - 查看实例集详细信息（已更新）
- 亚马逊 GameLift论坛

2021 年 2 月 9 日：亚马逊 GameLift 延长了对 AMD 独立实例的支持 FlexMatch

更新了 SDK 版本：S AWS DK 1.8.139

此版本包含以下更新：

- 现在可以将 Amazon GameLift FleetiQ 游戏服务器组配置为管理 AMD 实例系列 c5a、m5a 和 R5a。中列出的支持的 Amazon EC2 实例类型现在包括以下内容： GameServerGroup InstanceDefinition
  - c5a.large、c5a.xlarge、c5a.2xlarge、c5a.4xlarge、c5a.8xlarge、c5a.12xlarge、c5a.16xlarge、c5a.24x

  - m5a.large、m5a.xlarge、m5a.2xlarge、m5a.4xlarge、m5a.8xlarge、m5a.12xlarge、m5a.16xlarge、m5
  - r5a.large、r5a.xlarge、r5a.2xlarge、r5a.4xlarge、r5a.8xlarge、r5a.12xlarge、r5a.16xlarge、r5a.24xlar

  注意：适用于 FleetiQ 的 AMD 实例目前无法在中国（北京）区域使用。 AWS 请参阅中国特征可用性和实施差异。
- 亚马逊托 GameLift 管游戏托管现在支持由光环新网运营的中国（北京）地区的 AMD 实例。新的 AMD 实例系列包括 M5a 和 R5a。队InstanceType列中列出的支持的 EC2 实例类型现在包括以下内容：
  - m5a.large、m5a.xlarge、m5a.2xlarge、m5a.4xlarge、m5a.8xlarge、m5a.12xlarge、m5a.16xlarge、m5
  - r5a.large、r5a.xlarge、r5a.2xlarge、r5a.4xlarge、r5a.8xlarge、r5a.12xlarge、r5a.16xlarge、r5a.24xlar
- Amazon现在 GameLift FlexMatch 可以用作中国（北京）地区的独立配对解决方案，由光环新网运营。客户可以在北京地区创建 FlexMatch 匹配器并将FlexMatchMode参数配置为STANDALINE。有关 FlexMatch使用亚马逊托管主机或非亚马逊托 GameLift GameLift管解决方案的更多信息，请参阅《亚马逊 GameLift FlexMatch 开发者指南》。
- 在为亚马逊设置事件通知时 GameLift FlexMatch，您现在可以将 Amazon SNS FIFO 主题指定为通知目标。有关更多信息，请参阅：
  - MatchmakingConfiguration NotificationTarget，亚马逊 GameLift API 参考
  - 设置 FlexMatch 事件通知，《Amazon GameLift FlexMatch 开发者指南》
  - 介绍亚马逊 SNS FIFO — F irst-in-first-out Pub/ Sub 消息，新闻博客AWS

2020 年 12 月 22 日：亚马逊 GameLift 服务器 SDK 支持虚幻引擎 4.25 和 Unity 2020

更新了 SDK 版本：Amazon S GameLift erver SDK 4.0.2、虚幻插件版本 3.3.3

最新版本的 Amazon GameLift 服务器软件开发工具包包含以下组件：

- 更新后的 Unreal 插件已更新，可与 Unreal Engine 4.25 兼容。API 未更改。
- 现有 C# 软件开发工具包 4.0.2 版本已通过验证与 Unity 2020 兼容。不需要更新软件开发工具包。

在亚马逊下载最新版本的 Amazon S GameLift erver SDK GameLift 入门指南。

2020 年 11 月 24 日：亚马逊 GameLift FlexMatch 现已推出可在任何地方托管的游戏

更新了 SDK 版本：S AWS DK 1.8.95

Amazon GameLift FlexMatch 是一项可定制的多人游戏配对服务。最初是为 Amazon 托 GameLift 管主机用户设计的，现在 FlexMatch 可以集成到使用其他托管系统的游戏中 peer-to-peer，包括专有的本地计算和云计算原始类型。使用Amazon GameLift FleetiQ在亚马逊EC2上托管游戏的游戏现在可以实现与之配对。 FlexMatch

FlexMatch 提供了强大的配对算法和规则语言，可让您有很大的自由度来自定义配对流程，以便根据关键玩家特征和报告的延迟将玩家配对在一起. 此外，还 FlexMatch 提供配对请求工作流程，该工作流程支持玩家聚会、玩家接受和匹配回填等功能。当您 FlexMatch 使用亚马逊托 GameLift 管主机或实时服务器时，匹配器会自动使用亚马逊 GameLift 来查找托管资源并为新形成的比赛开始新的游戏会话。当 FlexMatch 用作独立服务时，匹配器会将比赛结果传回您的游戏，然后游戏可以使用您的托管解决方案开始新的游戏会话。

的 API 操作 FlexMatch 是亚马逊 GameLift 服务 API 的一部分，该服务包含在 AWS 软件开发工具包和 AWS Command Line Interface (AWS CLI) 中。此版本包括以下支持独立对战的更新：

- API 资源 `MatchmakingConfiguration` 具有以下更改：
  - 新属性，`FlexMatchMode`表示匹配器是用于Amazon托 GameLift 管主机还是用作独立配对。
  - 当 `FlexMatchMode` 设置为独立时，不需要 `GameSessionQueueArns` 属性。
  - 这些属性不适用于独立对战：`AdditionalPlayerCount`、`BackfillMode`、`GameProperties`、`GameSessionData`。
- 独立对战不支持自动回填特征。

2020 年 11 月 24 日：AMD 实例现已在亚马逊上市 GameLift

更新了 SDK 版本：S AWS DK 1.8.95

亚马逊支持的 Amazon EC2 实例类型列表 GameLift 现在包括三个新的实例系列：c5a、m5a 和 R5a。这些系列由 AMD 计算优化型实例组成，这些实例由 AMD EPYC 处理器提供支持，运行频率可高达 3.3 GHz。AMD 实例兼容 x86；当前在 Amazon 上运行的游戏无需更改 GameLift 即可部署到 AMD 实例类型。新实例可在以下 AWS 地区使用：美国东部（弗吉尼亚北部和俄亥俄州）、美国西部（俄勒冈州和加利福尼亚北部）、加拿大中部（蒙特利尔）、南美洲（圣保罗）、欧洲中部（法兰克福）、欧洲西部（伦敦和爱尔兰）、亚太南部（孟买）、亚太东北部（首尔和东京）和亚太东南部（新加坡和悉尼）。

新的 AMD 实例包括：

- c5a.large、c5a.xlarge、c5a.2xlarge、c5a.4xlarge、c5a.8xlarge、c5a.12xlarge、c5a.16xlarge、c5a.24xla
- m5a.large、m5a.xlarge、m5a.2xlarge、m5a.4xlarge、m5a.8xlarge、m5a.12xlarge、m5a.16xlarge、m5a

- r5a.large、r5a.xlarge、r5a.2xlarge、r5a.4xlarge、r5a.8xlarge、r5a.12xlarge、r5a.16xlarge、r5a.24xlarge

了解更多：

- Amazon 游戏科技博客
- Amazon GameLift 实例定价
- 采用 AMD EYPC 处理器的 Amazon EC2 实例
- 亚马逊 GameLift论坛

2020 年 11 月 11 日：亚马逊 GameLift 服务器 SDK 版本更新

更新的软件开发工具包版本：亚马逊 GameLift 服务器软件开发工具包 4.0.2

新的服务器软件开发工具包版本 4.0.2 修复了 API 操作 `StartMatchBackfill()` 中的一个已知问题。现在，此操作会返回对对战回填请求的正确响应。

该问题并未影响对战回填过程，此特征的工作方式也没有变化。该问题可能影响了对战回填请求的日志消息和错误处理。

在亚马逊下载最新版本的 Amazon S GameLift erver SDK GameLift 入门指南。

2020 年 11 月 5 日：新的 FlexMatch 算法自定义

FlexMatch 用户现在可以调整配对过程的以下默认行为。这些自定义设置是在对战规则集中设置的。Amazon GameLift 软件开发工具包没有变化。

- 优先考虑回填票证：在搜索可接受的对战时，您可以选择提高或降低对战回填票证的优先级。启用自动回填特征后，对回填票证进行优先排序非常有用。使用算法属性 `backfillPriority`。
- 预排序以优化匹配一致性和效率：配置您的对战构建器，使其在批量处理票证进行评估之前对票池进行预排序。通过根据关键玩家属性对票证进行预先排序，得到的对战往往会有在这些属性上更相似的玩家。您还可以通过对对战规则中使用的相同属性进行预排序来提高评估过程的效率。使用算法属性 `sortByAttributes`，并将 `strategy` 属性设置为“已排序”。
- 调整扩展等待时间的触发方式：根据未完成对战中最新（默认）或最旧票证的时效在触发扩展版之间进行选择。在最旧的票证上触发往往会更快地完成对战，而在最新的票证上触发可以提高对战质量。使用算法属性 `expansionAgeSelection`。

2020 年 9 月 17 日：亚马逊 GameLift 更新服务器 SDK

更新的软件开发工具包版本：亚马逊 GameLift 服务器软件开发工具包 4.0.1

新服务器软件开发工具包包含以下更新：

- C# API 版本 4.0.1
  - 不再支持 API 操作 TerminateGameSession()。替换为调用 ProcessEnding() 以结束游戏会话和服务器进程。
  - 操作 GetInstanceCertificate() 的已知问题已得到修复。
  - GetTerminationTime()现在，该操作会返回数据类型的值 AwsDateTimeOutcome。
- C++ API 版本 3.4.1
  - 不再支持操作 TerminateGameSession()。替换为调用 ProcessEnding() 以结束游戏会话和服务器进程。
- Unreal Engine 插件版本 3.3.2
  - 不再支持操作 TerminateGameSession()。替换为调用 ProcessEnding() 以结束游戏会话和服务器进程。
  - 向 FProcessParameters 添加了回调操作 `OnUpdateGameSession` 以支持对战回填。

在亚马逊下载最新版本的 Amazon S GameLift erver SDK GameLift 入门指南。

2020 年 8 月 27 日：亚马逊 GameLift FleetiQ 用于使用亚马逊 EC2 托管游戏（正式上市）

更新了 SDK 版本：S AWS DK 1.8.36

用于在亚马逊 EC2 上 GameLift 托管低成本、基于云的游戏的 Amazon FleetiQ 解决方案现已正式上市。Amazon GameLift FleetiQ 通过优化游戏托管的可行性，让开发者能够直接在 Amazon EC2 竞价型实例上托管游戏服务器。游戏开发者可以将 Amazon GameLift FleetiQ 用于开发新游戏或补充现有游戏的容量。该解决方案支持使用容器或其他 AWS 服务，例如 AWS Shield 和亚马逊弹性容器服务 (Amazon ECS) Service。

此正式发布版本包括对 Amazon GameLift FleetiQ 解决方案的以下更新：

- 新的 API 操作`DescribeGameServerInstances`会返回有关 Amazon GameLift FleetiQ 游戏服务器组的所有活动实例的信息，包括状态。

- 新的平衡策略 ON_DEMAND_ONLY 将游戏服务器组配置为仅使用按需型实例。您可以随时更新游戏服务器组的平衡策略，从而可以根据需要在使用竞价型实例和按需型实例之间切换。
- 已删除以下预览元素以供正式发布：
  - 对游戏服务器资源使用自定义排序键。可以根据注册时间戳对游戏服务器进行排序。
  - 为游戏服务器资源添加标签。

2020 年 4 月 16 日：亚马逊 GameLift 更新了适用于 Unity 和虚幻引擎的服务器 SDK

更新了 SDK 版本：亚马逊 GameLift 服务器 SDK 4.0.0、Ama GameLift zon Local 1.0.5

最新版本的 Amazon S GameLift erver SDK 包含以下更新的组件：

- 针对 Unity 2019 更新了 C# 软件开发工具包版本 4.0.0
- 针对 Unreal Engine 4.22、4.23 和 4.24 版本更新了 Unreal 插件版本 3.3.1。
- Amazon L GameLift ocal 版本 1.0.5 已更新，以测试使用 C# 服务器 SDK 版本 4.0.0 的集成。

在亚马逊下载最新版本的 Amazon S GameLift erver SDK GameLift 入门指南。

2020 年 4 月 2 日：亚马逊 GameLift FleetiQ 可在 EC2 上托管游戏（公开预览版）

更新了 SDK 版本：S AWS DK 1.7.310

Amazon GameLift FleetiQ 功能优化了用于游戏托管的低成本竞价型实例的可行性。此功能现已扩展到想要直接管理托管资源的客户，而不是通过亚马逊托管 GameLift服务管理其托管资源的客户。该解决方案支持使用容器或其他 AWS 服务，例如 AWS Shield 和亚马逊弹性容器服务 (Amazon ECS) Service。

了解更多：

GameTech 亚马逊 GameLift Fle@@ etiQ 上的博客文章 FleetiQ

2019 年 12 月 19 日：改进了亚马逊资源的 AWS GameLift 资源管理

更新了 SDK 版本：S AWS DK 1.7.249

现在，您可以利用 AWS 资源管理工具和 Amazon GameLift 资源。特别是，所有关键的 Amazon GameLift 资源（构建、脚本、队列、游戏会话队列、配对配置和配对规则集）现在都被分配了 Amazon 资源名称 (ARN) 值。资源 ARN 提供一致的标识符，该标识符在所有 AWS 区域中都是唯一

的。它们可用于创建资源特定 AWS Identity and Access Management (IAM) 权限策略。现在，将为资源分配 ARN 以及预先存在的资源标识符（该标识符不是区域特定的）。

此外，Amazon GameLift 资源现在支持标记。您可以使用标签来组织资源、创建 IAM 权限策略来管理对资源组的访问权限、自定义 AWS 成本明细等。管理亚马逊 GameLift资源的标签时，请使用亚马逊 GameLift API 操作`TagResource()`、`UntagResource()`和`ListTagsForResource()`。

了解更多：

- TagResource在 Amazon GameLift API 参考中
- 《AWS 一般参考》中的标记 AWS 资源
- 《AWS 一般参考》中的 Amazon 资源名称。

2019 年 11 月 14 日：新的 AWS CloudFormation 模板，中国（北京）区域中有更新

更新了 SDK 版本：S AWS DK 1.7.210

AWS CloudFormation 适用于 Amazon 的模板 GameLift

现在可以通过创建和管理 Amazon GameLift 资源 AWS CloudFormation。现有的 AWS CloudFormation 版本和舰队模板已更新，以与当前资源保持一致，并且新的模板现在可用于脚本、队列、配对配置和配对规则集。 AWS CloudFormation 模板极大地简化了管理相关 AWS 资源组的任务，尤其是在跨多个区域部署游戏时。

了解更多：

- AWS CloudFormation 用户指南中的 Amazon GameLift 资源类型参考
- 使用 AWS CloudFormation 管理资源在《亚马逊 GameLift 开发者指南》中

AWS 术语表

有关最新的 AWS 术语，请参阅《AWS 词汇表参考》中的 AWS 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。