



托管指南

Amazon GameLift Servers



Amazon GameLift Servers: 托管指南

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

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

什么是 Amazon GameLift Servers ?	1
主要 功能	1
如何使用 Amazon GameLift Servers	3
游戏托管方案	4
托管式 EC2	4
托管式容器	5
Anywhere 托管	6
混合托管	6
工作原理	7
核心组件	7
这些组件如何协同工作	12
玩家体验	13
为全球玩家提供低延迟游戏体验	13
帮助玩家快速加入游戏	14
灵活设计玩家组队方式，打造多样化游戏会话体验	14
全程保障游戏连接稳定可靠	15
基于真实数据改善玩家体验	15
将增强的游戏功能与其他 AWS 服务集成	15
生产环境中的游戏托管	16
监控主机运行状况和性能	16
管理游戏服务器更新和补丁	17
优化性能和扩展	17
故障排除和实时操作	17
工具和集成	18
相关资源	18
关键术语	18
开始使用	25
开始之前	25
选择您的路径	25
我希望快速体验 Amazon GameLift Servers	25
我希望构建自定义托管解决方案	26
推荐学习路径	26
设置一个 AWS 账户	26
注册获取 AWS 账户	28

创建具有管理访问权限的用户	28
为 Amazon GameLift Servers 设置用户权限	29
为用户设置编程式访问权限	30
为游戏设置编程式访问权限	32
IAM 权限示例	32
设置 IAM 服务角色	37
获取 开发工具	40
对于游戏服务器	41
对于游戏客户	43
用于游戏后端服务	44
对于资源管理	44
对于 Amazon GameLift Servers Realtime	45
教程：使用 Amazon GameLift Servers 封装器快速入门	45
先决条件	46
概述	46
第 1 步：获取并构建游戏服务器封装器	47
第 2 步：准备游戏服务器生成包	47
第 3 步：为实例集配置封装器	48
第 4 步：上传游戏服务器生成包	49
步骤 5：创建托管 EC2 舰队	50
第 6 步：创建游戏会话并连接	51
第 7 步：管理和监控实例集	52
第 8 步：扩展游戏服务器	52
常见问题故障排除	53
后续步骤	54
Amazon GameLift Servers 示例	55
自定义游戏服务器示例	55
构建游戏托管解决方案	56
开发路线图	56
托管式 EC2 开发路线图	57
托管容器开发路线图	62
Anywhere 开发路线图	67
混合托管开发路线图	72
准备要托管的游戏	78
借助游戏引擎插件完成准备工作	78
集成游戏服务器	155

打包游戏服务器生成包	181
Amazon GameLift Servers 交互	193
构建后端服务	198
对您的玩家进行身份验证	199
集成游戏客户端功能	199
无服务器后端	204
WebSocket基于后端	205
部署托管实例集	207
实例集特征	207
选择托管选项	208
托管 EC2 车队	210
托管式容器实例集	219
Anywhere 实例集	236
构建混合托管解决方案	245
配置游戏会话放置	248
队列特点	248
最佳实践	249
创建队列	250
设置事件通知	253
自定义游戏托管解决方案	257
游戏服务器生成包	258
玩家会话和对战	269
游戏会话放置	270
托管资源	282
管理游戏托管解决方案	302
管理游戏托管资源	302
Amazon GameLift Servers 控制台功能	302
控制台控制面板	303
Builds	304
车队	307
Aliases	325
游戏会话队列	328
为游戏发布做准备	330
准备就绪	330
准备测试	331
准备发布	331

制定发布后计划	332
使用管理游戏托管资源 CloudFormation	332
最佳实践	332
使用 CloudFormation 堆栈	333
更新构建	337
DDoS 保护	340
主要优势	340
玩家网关的工作原理	340
流量流	340
核心概念	341
监控玩家网关	343
IPv4 和 IPv6 兼容性	343
在舰队上启用玩家网关	343
先决条件	343
玩家网关模式	344
IPv4 和 IPv6 兼容性	344
将玩家网关集成到游戏中	344
后端集成	345
客户端集成	347
测试您的集成。	348
最佳实践	348
测试和故障排除	350
为迭代开发做好准备	350
构建基于云的测试环境	352
设置本地测试	354
设置本地测试（旧版）	359
调试实例集问题	366
实例集创建问题	366
服务器进程问题	368
实例集删除问题	368
远程连接到 实例集实例	368
通过控制台进行远程访问	369
使用 AWS CLI 进行远程访问	369
连接到实例（服务器 SDK 5）	372
连接到实例（服务器 SDK 4.x 或更早版本）	373
查看远程实例上的文件	375

扩展	377
在控制台中管理实例集容量	377
设置托管容量限制	378
设置容量限制	378
手动设置实例集容量	379
暂停自动扩缩	379
手动设置实例集容量	380
自动扩缩实例集容量	381
基于目标的自动扩缩	381
基于规则的自动扩缩的提示	383
管理缩放舰队 To/From Zero	387
缩小行为	387
横向扩展行为	387
配置 To/From 零比例	388
最佳实践	389
扩缩容器实例集	389
监控	391
指标比较	391
按来源划分的指标可用性	391
主题	395
使用监视器 CloudWatch	395
指标维度	396
机群指标	397
DDoS 保护（玩家网关）指标	410
队列指标	412
FlexMatch 指标	416
FleetIQ 指标	419
使用服务器遥测指标进行监控	420
遥测指标的优势	421
开始前的准备工作	422
实施	423
可用指标	424
工作原理	427
控制面板组织结构和使用方式	428
常见监控场景	436
故障排除指南	441

记录 API 调用	444
Amazon GameLift Servers 信息在 CloudTrail	444
了解 Amazon GameLift Servers 日志文件条目	445
记录服务器消息	447
EC2 vs 容器日志	447
自定义服务器的日志记录	449
安全性	453
数据保护	453
静态加密	455
传输中加密	455
互连网络流量隐私	456
Identity and access management	456
受众	457
使用身份进行身份验证	457
使用策略管理访问	458
Amazon GameLift Servers 如何与 IAM 协同工作	459
基于身份的策略示例	465
问题排查	470
AWS 托管策略	471
使用 Amazon GameLift Servers 进行日志记录和监控	473
合规性验证	473
故障恢复能力	474
基础结构安全性	475
配置和漏洞分析	475
安全最佳实践	476
确保实例集运行时环境保持最新状态	476
保护您的端口配置	477
其他安全资源	478
定价	479
定价模型	479
生成 定价估算值	480
估算 托管式托管	480
成本优化策略	483
最佳实践	483
资源成本和利用率工具	484
性能监控	485

参考指南	486
服务 API (AWS SDK)	486
管理 Amazon GameLift Servers 托管资源	487
开始游戏会话并加入玩家行列	490
服务器 SDK 5.x	491
服务器 SDK 5.x 更新	492
迁移到服务器 SDK 5.x	493
C++ 服务器 SDK	494
C# 服务器 SDK	543
Go 服务器 SDK	584
C++ (Unreal) 服务器 SDK : 操作	612
服务器 SDK 4 及更早版本	655
适用于 C++ 的服务器 SDK : 操作	656
适用于 C# 的服务器 SDK : 操作	678
适用于 Unreal 的服务器 SDK : 操作	697
游戏会话放置事件	712
放置事件语法	712
已完成展示位置	713
放置已取消	715
放置超时	715
放置失败	716
AMI 版本	717
服务位置	719
支持 AWS 的地点	719
托管式托管的位置	724
适用于 Amazon GameLift Servers Anywhere 的位置	725
适用于 Amazon GameLift Servers FlexMatch 的位置	726
Amazon GameLift Servers在中国	726
服务端点和限额	726
亚马逊 GameLift 服务器 API 限制	726
API 限制参考	727
资源级节流	735
Amazon GameLift Servers SDK API 限制	735
UDP ping 信标	738
UDP ping 信标的工作原理	738
UDP ping 信标的常见使用案例	739

获取信标端点	739
实施延迟测量	740
代码示例	741
发行说明和软件开发工具包版本	749
SDK 版本	749
发行说明	773
.....	dccccxvi

什么是 Amazon GameLift Servers ？

使用 Amazon GameLift Servers 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。基于 AWS 全球计算基础架构，Amazon GameLift Servers 有助于提供高性能、高可靠性的游戏服务器，同时动态扩展您的资源使用量以满足全球玩家的需求。

主要 功能

覆盖全球和低延迟

- 在全球的 Region AWS s 和 Local Zones 中部署托管服务，将您的游戏带给新玩家。查看 [Amazon GameLift Servers 支持的位置](#) 的完整列表。
- 无论玩家身处何地，都能提供最低延迟和卓越的游戏体验。
- 使用多位置放置队列将玩家动态地放置到游戏会话中，从而实现最佳的玩家体验。

托管式实例集的自动扩展

- 优化实例集容量以提升成本效益，同时保持服务质量。
- 根据玩家需求，使用基于目标的自动扩缩功能扩展容量
- 维护容量缓冲区以应对玩家激增的情况。

灵活的托管选项

有关更多详情，请参阅 [Amazon GameLift Servers 游戏托管方案](#)。

- 由 Amazon GameLift Servers 管理的云端游戏托管
 - 在 Amazon Elastic Compute Cloud (Amazon EC2) 基础设施上托管游戏服务器。有多种资源类型可供选择，包括计算优化 (C 系列)、内存优化 (R 系列) 和经济实惠的基于 ARM 的 Graviton 实例。
 - 自定义 Amazon GameLift Servers 如何管理游戏托管解决方案的各个方面，从部署游戏服务器和启动游戏会话，到匹配玩家并让他们进入游戏。
 - 使用多位置实例集来简化全球游戏服务器部署。
 - 通过 [Amazon EC2 竞价型实例](#) 实现低成本托管。
 - 使用托管式 EC2 托管来部署 Windows 或 Linux 游戏服务器。

- 使用托管式容器托管在游戏服务器中部署基于 Linux 的容器。
- 由您自行管理的游戏托管 (通过 Amazon GameLift Servers Anywhere)
 - 利用 Amazon GameLift Servers 游戏会话放置和会话管理功能。
 - 在您自己的硬件、本地基础设施或其他云提供商平台上自行管理游戏服务器部署和扩展。
- 混合游戏托管
 - 创建游戏托管解决方案，结合使用 Amazon GameLift Servers 托管式实例集和 Anywhere 实例集，既可与现有解决方案协同工作，又能支持迁移。

游戏会话管理

- 让 Amazon GameLift Servers 为您管理游戏服务器进程，跟踪游戏服务器状态和可用性，并根据请求放置新的游戏会话。
- 使用游戏会话放置功能来帮助玩家找到并加入游戏会话，从而提供出色的玩家体验。依靠 Amazon GameLift Servers 决策，或根据托管成本、玩家延迟和地理位置等标准进行自定义。
- 创建游戏会话放置队列，以高效处理新的游戏会话请求。利用各种功能在放置质量与速度之间取得平衡，以满足游戏需求和玩家期望。
- 利用可选的玩家会话功能在游戏会话中保留玩家位置，验证玩家的连接状态，并监控位置的可用性。搜索或筛选现有游戏会话，以查找可用的位置。

高级配对 FlexMatch

- 添加与您的 Amazon GameLift Servers 托管解决方案完全集成的强大对战功能。
- 根据玩家属性和偏好 (例如技能等级或角色选择) 建立自定义对战规则，并根据需要使用复杂的逻辑。
- 设置基于团队的对战，支持竞技或合作玩法。在球队 and/or 比赛组成中平衡球员。
- 优化对战算法以平衡对战质量和等待时间。
- 支持 2 至 200 名玩家同时参与对战。
- 支持对战回填以填充可用玩家位置并保持游戏会话满员状态。

全面监控

- 获取有关游戏会话和玩家活动使用情况和性能，以及托管式实例集的托管性能和运行状况的实时指标。

- 与 Amazon 集成 CloudWatch ，获取警报和控制面板。
- 访问服务器日志和事件跟踪，以进行故障排除。
- 分析性能和利用率数据，以优化扩展（托管式实例集）或资源分配（自行管理式实例集）。

自定义和可扩展性

- 自定义游戏会话放置逻辑，包括根据延迟、成本和位置进行优先级排序。
- 对于托管式主机，选择您的游戏托管资源，并提供在每个实例上启动和运行游戏服务器的说明。对于托管式容器，可以微调容器间的资源分配。
- 定义自己的对战规则。
- 实施自定义容量扩展策略。
- 添加游戏服务器代码以与现有 AWS 服务（例如数据库或内容存储工具）集成。
- 创建具有自定义功能的后端服务，用于管理玩家加入请求（例如玩家聚会）、处理身份验证或添加自定义逻辑和持久性。

与 AWS 生态系统集成

- 使用 Amazon DynamoDB、Amazon Simple Storage Service 和 Amazon Aurora DSQL 等服务实现游戏状态持久性。
- 实施 Amazon Cognito 以进行玩家身份验证。
- 使用 Amazon Kinesis and Amazon S3 处理游戏分析。
- 使用 Amazon Chime SDK 添加语音聊天。
- 使用 AWS Lambda 和 Amazon API Gateway 创建自定义游戏功能。

如何使用 Amazon GameLift Servers

使用这些工具来处理 Amazon GameLift Servers。

AWS CLI

使用 AWS Command Line Interface (AWS CLI) 调用 AWS SDK，包括的服务 API Amazon GameLift Servers。查看《AWS Command Line Interface 用户指南》中的[开始使用 AWS CLI](#)。

Amazon GameLift Servers 控制台

使用 [Amazon GameLift Servers 的AWS 管理控制台](#) 来配置资源、管理游戏服务器部署，并跟踪性能和使用情况指标。Amazon GameLift Servers 控制台是一种 GUI 替代方式，能够以编程方式或通过 AWS CLI管理资源。

Amazon GameLift Servers SDKs

Amazon GameLift Servers SDKs 包含在游戏客户端、游戏服务器和游戏服务与服务之间建立通信所需的库。Amazon GameLift Servers有关更多信息，请参阅 [获取 Amazon GameLift Servers 开发工具](#)。

AWS CloudFormation

通过 AWS CloudFormation 将基础设施视为代码，用于为游戏托管解决方案建模、配置和管理 AWS 资源。创建描述资源的模板，并 CloudFormation 自动执行将资源配置和部署到您指定的位置的任务。

Amazon GameLift Servers 游戏托管方案

Amazon GameLift Servers 为基于会话的多人游戏提供多种游戏服务器托管方案。无论您是想设置由 Amazon GameLift Servers 托管的基于云的主机，还是想要整合已有的托管资源，均可借助 Amazon GameLift Servers 构建满足玩家需求的游戏托管方案。有关游戏托管解决方案的更多详细信息，请参阅 [使用 Amazon GameLift Servers 托管的工作原理](#)。

主题

- [托管式 EC2](#)
- [托管式容器](#)
- [Anywhere 托管](#)
- [混合托管](#)

托管式 EC2

将生产游戏服务器的管理工作转移到具有托管式 EC2 托管的 Amazon GameLift Servers。在充分利用多人游戏服务器优化的同时，仍可依托 Amazon Elastic Compute Cloud (Amazon EC2) 与 AWS 全球计算基础设施的高性能与可靠性。

[开始构建 Amazon GameLift Servers 托管式 EC2 托管解决方案](#)

特性

- 托管在 Amazon Linux 或 Windows Server 操作系统上运行的多人游戏服务器。
- 使用 Amazon GameLift Servers 部署和管理您的自定义游戏服务器，以托管资源 AWS Cloud。选择硬件类型及其部署位置，并仅在需要时配置额外的详细信息。
- 配置运行时进程管理器，使其根据需要自动维护游戏服务器进程，以托管游戏会话。
- 设置自定义运行状况跟踪，以帮助 Amazon GameLift Servers 检测并处置性能不佳的游戏服务器。
- 利用 Amazon GameLift Servers 性能监控。跟踪硬件性能和服务器进程生命周期等指标。您还可以下载和存储游戏会话日志。
- 使用 Amazon GameLift Servers 管理游戏服务器进程和其他 AWS 资源之间的访问权限。
- 对于生产托管，使用 AWS CloudFormation 模板和 自动部署和管理游戏托管 AWS Cloud Development Kit (AWS CDK)。利用 AWS CodePipeline 等持续集成和持续交付 (CI/CD) 工具和服务。

托管式容器

Amazon GameLift Servers 为容器化游戏服务器提供完整的云托管解决方案，因此您可以充分利用容器的核心优势，例如可移植性、敏捷性和容错能力。与托管式 EC2 托管一样，托管式容器托管在 Amazon EC2 实例上部署和运行您的容器。

[开始为您的容器化游戏服务器开发 Amazon GameLift Servers 托管式解决方案。](#)

特性

- 开发包含轻量级容器的自定义架构，用于运行游戏服务器生成包、依赖项和辅助软件。
- 使用 Docker 工具创建基于 Linux 的容器映像。在 Amazon Elastic Container Registry (Amazon ECR) 存储库中存储映像，以便进行部署。
- 使用 Amazon GameLift Servers 部署和管理您的自定义游戏服务器，以托管资源 AWS Cloud。选择硬件类型及其部署位置，并仅在需要时配置额外的详细信息。
- 借助游戏服务器版本建模工具来管理托管实例集生命周期。根据需要部署游戏服务器更新和其他配置更改。
- 使用特定于容器的性能指标跟踪游戏托管性能。使用硬件指标监控实例集资源的运行状况。
- 使用 AWS CloudFormation 模板管理容器舰队资源 Amazon GameLift Servers。

Anywhere 托管

借助 Anywhere 托管，您可以利用 Amazon GameLift Servers 游戏会话管理功能（包括对战和游戏会话指标），在任何位置托管游戏服务器。自行管理您的游戏服务器部署、游戏服务器运行状况监控和容量扩展。

Note

Anywhere 实例集特别适合作为快速迭代游戏开发的测试环境。为您自己的本地工作站或基于云的托管资源设置 Anywhere 实例集。有关使用 Anywhere 进行测试的更多信息，请参阅 [使用 Amazon GameLift Servers Anywhere 设置本地测试](#)。

[开始为您的游戏开发 Amazon GameLift Servers Anywhere 托管解决方案。](#)

特性

- 在您自己的硬件、本地基础设施或其他云提供商平台上部署游戏服务器。随时随地利用离您的玩家最近的可用硬件。
- 使用 Amazon GameLift Servers 的会话管理功能来监控游戏服务器可用性、启动新的游戏会话，并将玩家加入游戏会话位置。

混合托管

对于生产托管，您可以选择混合方法：由 Anywhere 实例集来管理您提供和管理的托管资源，并由 Amazon GameLift Servers 托管式实例集在需要时进行填充。混合解决方案使用相同的流程来启动游戏会话并连接玩家，从而提供无缝的玩家体验。无论资源类型如何，您都可以监控和收集所有托管的使用情况指标。

适合采用混合解决方案的常见场景包括：

- 将托管解决方案扩展到 AWS Cloud。通过添加 Amazon GameLift Servers 托管式托管来补充现有托管解决方案（本地硬件或其他基于云的托管）的功能。利用托管式托管，您可以增加托管容量，或者添加“爆增”容量以快速扩展并仅在需要资源时为资源付费。您还可以利用 Amazon GameLift Servers 服务的全球覆盖范围来触达世界各地的更多玩家，并提供他们期望的低延迟多人游戏体验。
- 为迁移到基于云的托管做准备。如果您正在考虑或计划迁移到 AWS Cloud，那么混合托管解决方案是一种可让您根据需求逐步过渡的可行方法。该方法可让您构建当前所需的解决方案，同时为将来的需求做好准备。

- 降低位于 Amazon GameLift Servers 服务范围之外位置的玩家的延迟。如果您已经在使用 Amazon GameLift Servers 托管式托管，则可能需要在特殊情况下为玩家提供支持。例如，您可能想更好地触达位于特别偏远位置的玩家或为有特殊访问需求的玩家提供支持。请添加自定义托管位置，并使用 Amazon GameLift Servers Anywhere 支持这些位置和您的托管式托管资源。

•

[开始为您的游戏开发 Amazon GameLift Servers 混合托管解决方案。](#)

使用 Amazon GameLift Servers 托管的工作原理

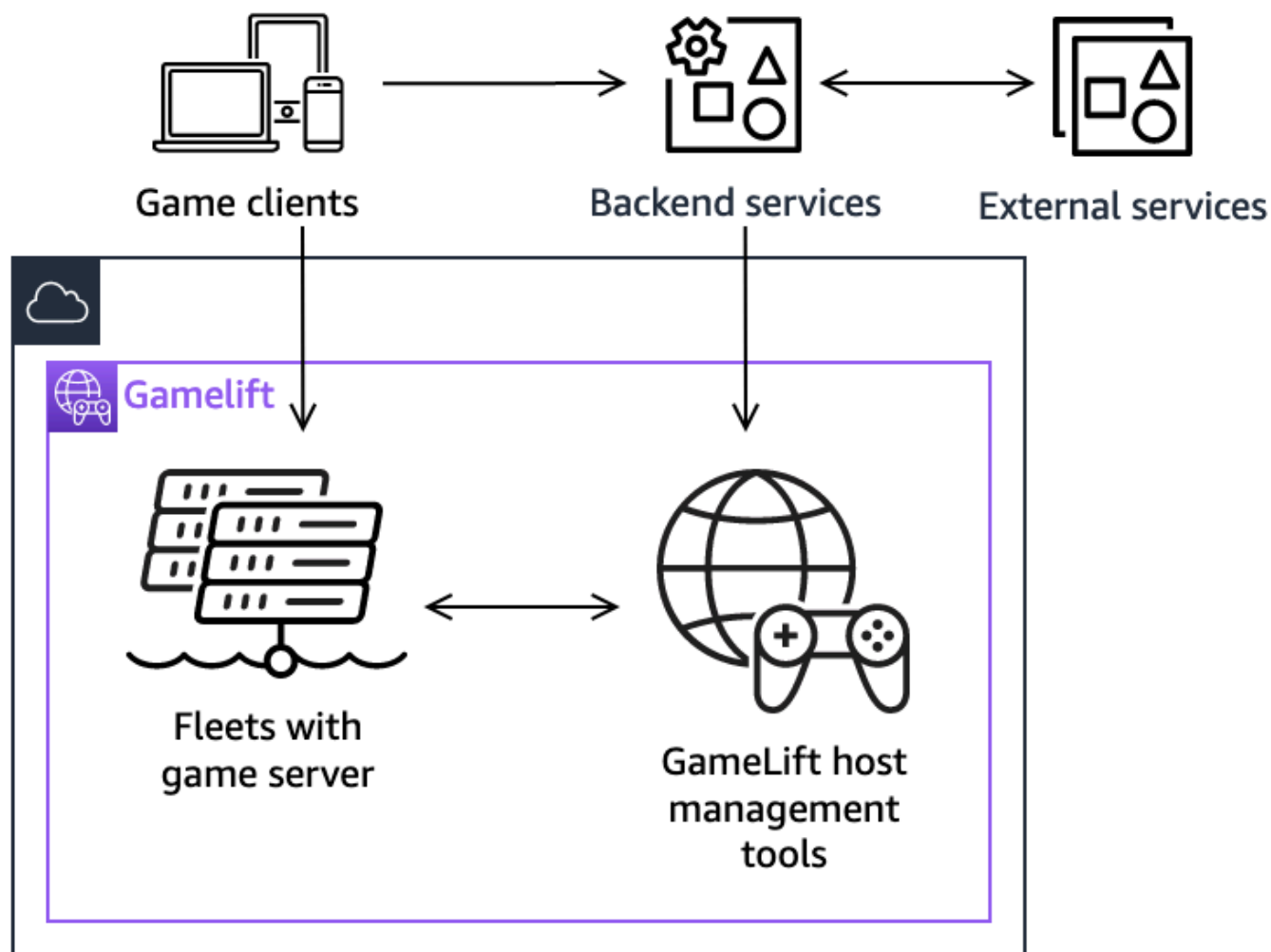
Amazon GameLift Servers 是一项服务，可为您基于会话的多人游戏提供专用的低成本服务器、基础设施预调配、扩缩和会话管理。该服务提供灵活的工具和功能，您可以根据游戏进行自定义或与自己的系统结合使用。利用 Amazon GameLift Servers 托管式主机，您可以专注于打造出色游戏体验，而不是管理服务器操作。

本主题将介绍完整的 Amazon GameLift Servers 托管解决方案的核心组件，以及它们如何协同工作为玩家提供多人游戏会话。

游戏托管解决方案的核心组件

完整的 Amazon GameLift Servers 托管解决方案由几个基本组件组成，这些组件协同工作以提供多人游戏体验。其中一些组件由您构建和管理，而其他组件则由 Amazon GameLift Servers 提供并根据您的配置选择进行管理。了解这些组件的交互方式及其各自的角色，对于开发有效的游戏托管解决方案至关重要。组件包括：

- 游戏客户端：运行于玩家设备。
- 后端服务：使游戏客户端能够与 Amazon GameLift Servers 通信，以获取游戏会话信息并加入游戏。
- 游戏服务器软件：在游戏托管资源上运行，并为的一组玩家托管游戏会话。
- 放置系统：启动游戏会话并将玩家与游戏进行匹配。
- 游戏托管实例集：在一个或多个地理位置提供游戏服务器。
- 游戏托管管理系统：监控游戏托管状态并管理容量。



游戏客户端

游戏客户端是在玩家设备上运行的游戏软件。它通过与后端服务通信来启动玩家加入游戏会话的过程，并直接连接到游戏服务器以参与游戏玩法。

游戏托管的关键功能

- 向后端服务发送加入请求。在加入请求中包含相关的玩家数据和游戏会话数据。
- 收集游戏客户端的延迟数据，并将这些信息包含在加入请求中。
- 接收来自后端服务的游戏会话连接信息，并利用这些信息来连接到游戏服务器。
- 处理会话连接场景，例如玩家验证、Amazon GameLift Servers FlexMatch 的对战接受、会话中断或断开连接。

由谁构建

您需要将游戏托管功能添加到游戏客户端软件中，并设置与后端服务的通信。

后端服务

后端服务是游戏客户端与 Amazon GameLift Servers 服务之间的协调层。它控制与服务的所有通信，以请求游戏会话放置以及检索游戏会话和玩家会话信息。最佳实践是使用后端服务，它可以代表游戏客户端与服务保持安全通信，并避免与游戏客户端共享敏感的 AWS 凭证和权限。

您的后端服务为您的游戏会话实施玩家分组策略。在玩家如何进入游戏会话方面，您拥有高度的灵活性。您可以为玩家设置游戏会话浏览器或支持玩家定义的组队。或者，您可以在收到游戏会话请求时将它们汇集起来，组成玩家组（可能经过预先排序），并为每个组启动一个游戏会话。通过 FlexMatch 对战，您可以组建玩家对战并回填现有对战。

游戏托管的关键功能

- 对来自游戏客户端的通信进行身份验证。
- 向 Amazon GameLift Servers 发送请求以启动新的游戏会话、获取游戏会话信息或加入现有游戏会话。根据需要在请求中包含相关的游戏会话和玩家数据。
- （可选）创建玩家会话请求，以更密切地监控游戏会话的可用性。使用玩家会话来预留游戏会话位置、验证玩家何时连接以及跟踪玩家断开连接情况。
- 根据要求向游戏客户端提供游戏会话连接详细信息和其他信息。

由谁构建

您需要在由您管理的环境中构建后端服务，以支持您的游戏要求并实现玩家进入游戏的方式。集成 S AWS DK 以调用 Amazon GameLift Servers 服务 API。（可选）AWS 使用亚马逊简单存储服务 (Amazon S3) Service 和 Amazon Cognito 等 AWS Lambda 服务来构建您的后端服务。

游戏服务器

游戏服务器是您的自定义服务器软件，用于管理游戏状态、处理玩家操作并在多个连接的玩家之间同步游戏。游戏服务器与 Amazon GameLift Servers 服务保持通信，以管理游戏会话托管。

游戏托管的关键功能

- 与 Amazon GameLift Servers 进行通信，以实现以下目标：

- 报告状态 (准备就绪可托管会话、准备就绪可接受玩家、运行状况) 。
- 响应服务调用 (开始或结束游戏会话) 。
- 管理游戏会话生命周期，确保每个进程一次仅托管一个游戏会话。
- 与其他 AWS 服务协调以增加功能。
- (可选) 验证新的玩家连接。

由谁构建

您负责构建您的游戏服务器软件。您可以为 Amazon GameLift Servers 集成服务器 SDK 并添加功能以建立与服务的连接，同时支持游戏会话管理。

游戏托管实例集

托管实例集是运行您的游戏服务器的计算资源集合。实例集资源可分布于多个地理位置，无论玩家位于何处，皆能提供低延迟的游戏体验。每个实例集资源都运行一个或多个游戏服务器进程，这些进程直接与 Amazon GameLift Servers 通信。每个游戏服务器进程一次只能托管一个游戏会话。

实例集的特性和功能因实例集的托管类型而异。托管舰队将资源部署到 AWS Cloud 并由管理。Amazon GameLift Servers 任何地方的实例集都是客户提供的计算资源，在 Amazon GameLift Servers 外部进行管理。

游戏托管的关键功能

- 预调配游戏服务器托管资源。
 - 托管实例集部署基于云的 Amazon EC2 实例，支持多种配置设置。实例集配置决定实例集中每个实例的计算能力、实例的物理位置和其他详细信息。托管实例集添加或移除 EC2 实例，以响应容量扩展
 - 任何地方的实例集部署都是自行管理的。实例集可以由物理硬件或其他云资源组成，并可根据需要配置。
- 安装运行时环境和游戏服务器软件。
 - 托管式 Amazon EC2 实例集实例与您的游戏服务器生成包以及包含兼容运行时环境的亚马逊机器映像 (AMI) 一同部署。
 - 托管容器实例集实例与您的游戏服务器生成包和容器优化的 AMI 一同部署，且容器优化的 AMI 中包含 Docker 工具和其他组件，可与 Amazon ECS 结合使用。
 - Anywhere 实例集将根据您选择的游戏服务器软件和操作系统进行部署。
- 管理游戏服务器进程生命周期。应用预先配置的运行时指令来启动和停止每个实例集资源的进程。

由谁构建

这取决于实例集的托管类型：

- 托管 Amazon EC2 队列：您可以将游戏服务器版本上传到 Amazon GameLift Servers，然后将其存储到队列实例中。Amazon GameLift Servers 提供舰队 AWS Cloud 的基础设施。您需要配置实例集并指导它如何在每个实例集实例上运行游戏服务器。
- 托管容器队列：您将游戏服务器编译和运行时指令打包成容器镜像，存储在 Amazon ECR 中进行部署。Amazon GameLift Servers 提供舰队 AWS Cloud 的基础设施。您需要提供容器架构，并配置实例集来托管您的容器。
- Anywhere 实例集：您负责预调配所有基础设施并管理服务器软件部署。您需要创建 Anywhere 实例集，以将您的活跃游戏托管资源与 Amazon GameLift Servers 相连接。

游戏会话放置系统

游戏会话放置系统会定位可用的游戏服务器来托管新的游戏会话。该系统使用有关游戏服务器可用性的实时信息来做出最佳的放置决策。

在 Amazon GameLift Servers 中，主要的游戏会话放置机制是队列。游戏会话队列使用您可以配置的算法来放置游戏会话，从而实现最佳结果。您可以根据最低托管成本和最低玩家延迟等因素确定放置的优先顺序，也可以配置队列以跨多个地理位置进行搜索。作为队列的替代方案，您可以指定特定的实例集来托管您的游戏会话。

游戏托管的关键功能

- 处理从后端服务收到的游戏会话放置请求。
- 根据有关托管资源可用性的实时信息进行放置。
- 使用玩家延迟数据和其他数据来确定放置选项的优先级（仅限队列）
- 提示游戏服务器进程启动新的游戏会话。
- 在游戏会话准备就绪可接受玩家之后，更新游戏会话连接信息。
- （可选）设置 FlexMatch 对战以创建玩家对战，并为对战请求游戏会话位置。

由谁构建

Amazon GameLift Servers 提供放置系统。您可以通过后端服务发出放置请求的方式，也可以通过设置游戏会话队列和 FlexMatch 对战构建器来配置放置行为。使用 Amazon GameLift Servers 控制台、AWS SDK 或为您的游戏创建和配置队列和匹配器。AWS CLI

游戏托管管理系统

游戏托管管理系统是协调和监控游戏托管解决方案各个方面的运营支柱。该系统提供了智能和自动化功能，使 Amazon GameLift Servers 的放置和缩放功能得以实现。

游戏托管的关键功能

- 跟踪所有实例集中游戏服务器进程、游戏会话和玩家会话的实时状态和可用性，以支持游戏会话放置和自动容量扩展。
- 监控实例集的运行状况和性能。
- 收集和分析游戏托管活动指标。
- 提供容量扩展工具，包括根据玩家需求和实例集利用率自动扩展。
- 管理游戏服务器软件和运行时说明的更新。

由谁构建

- 托管实例集：Amazon GameLift Servers 提供可用性跟踪、游戏托管活动和硬件性能指标以及容量扩展工具。您可以使用可用 AWS 工具来管理游戏服务器软件更新、修改运行时说明和自定义指标的使用（例如设置 CloudWatch 监控）。您可以配置自定义扩展策略并根据需要进行修改。
- Anywhere 实例集：Amazon GameLift Servers 提供可用性跟踪和游戏托管活动指标。您可以管理实例集配置更改，包括游戏服务器软件和运行时说明的更新。您可以创建系统来监控实例集性能并管理实例集容量扩展。

这些组件如何协同工作

当游戏托管计算部署完成时

- 游戏服务器软件已安装：计算资源上安装了运行时环境和您的游戏服务器生成包
- 游戏服务器启动：在计算资源上至少启动了一个带有可选启动参数的游戏服务器可执行文件实例。
- 游戏服务器连接到 Amazon GameLift Servers：作为启动操作的一部分，游戏服务器进程调用服务器 SDK 来初始化与服务的连接。
- 游戏服务器报告已准备好托管游戏会话：游戏服务器进程完成启动操作，然后调用服务器 SDK 以报告就绪状态。它开始根据其配置报告运行状态。
- Amazon GameLift Servers 跟踪可用性：该服务记录游戏服务器进程对游戏会话放置的可用性。它还会更新实例集、实例和游戏会话的指标，以跟踪使用情况和容量。
- 游戏服务器等待游戏会话分配：游戏服务器进程在等待启动游戏会话的提示时保持闲置状态。

当玩家想要加入游戏时

- 玩家启动游戏：玩家启动您的游戏客户端，使用您的后端服务进行身份验证，并要求加入游戏。
- 后端服务请求游戏会话：您的后端服务会根据其玩家分组策略调用 Amazon GameLift Servers，以查找或创建合适的游戏会话。该请求可能包含用于游戏会话的玩家或游戏数据。
- Amazon GameLift Servers 放置游戏会话：开始新的游戏会话时，放置系统会确定托管会话的最佳位置并选择可用的游戏服务器进程。该服务会提示所选进程开始新的游戏会话，并传递任何玩家或游戏数据。如果正在使用 FlexMatch 对战，则对战构建器首先创建对战，然后为对战请求放置。
- 游戏服务器启动游戏会话：游戏服务器启动相关步骤以开始游戏会话。完成后，它会向 Amazon GameLift Servers 报告自己已准备好接受玩家连接。
- 连接信息发送至游戏客户端：在游戏服务器更新其状态后，Amazon GameLift Servers 提供游戏会话连接信息。后端服务会收到这些信息并将其交付给游戏客户端。
- 玩家连接到游戏会话：游戏客户端使用连接信息直接连接到游戏服务器并开始游戏。
- Amazon GameLift Servers 监控游戏会话状态：游戏服务器进程报告运行状况、可选的玩家连接状态和游戏会话状态，以跟踪正在进行的游戏会话可用性。
- 游戏服务器进程关闭：游戏服务器进程结束游戏会话，报告状态，然后自行关闭。

Amazon GameLift Servers 和玩家体验

玩家期望获得低延迟、公平且稳定的多人游戏体验。Amazon GameLift Servers 通过为游戏开发人员提供工具，直接优化玩家在多人游戏中的连接方式和交互体验，从而为玩家实现上述体验。本主题将概述 Amazon GameLift Servers 的各项功能为玩家带来的核心优势。

为全球玩家提供低延迟游戏体验

玩家希望连接到就近的服务器，以实现最小延迟。Amazon GameLift Servers 通过智能放置和全球基础设施，确保玩家获得最优连接体验，具体包括：

- 玩家在全球范围内体验稳定的性能 — 在全球范围内可用的 AWS 区域和 Local Zones 托管游戏，以最大限度地缩短玩家与游戏服务器之间的距离（参见[Amazon GameLift Servers 服务位置](#)）。您可以从这些位置的各种 Amazon EC2 实例类型中进行选择，以找到托管游戏工作负载的最佳计算资源。
- 玩家连接至最佳可用服务器：Amazon GameLift Servers 会收集游戏客户端网络数据，并跨多个位置搜索可用服务器，最终将玩家分配到延迟可能最低的游戏会话中。您可使用 UDP ping 信标收集准确的延迟数据，同时配置延迟策略，在连接质量与玩家可接受的等待时间之间找到平衡。

- 不同区域玩家可跨区域组队：游戏会话放置功能可在维持可接受延迟的前提下，将来自不同区域玩家池的玩家匹配到同一游戏会话中。即使是玩家数量较少的区域，玩家也能在不牺牲连接质量的前提下找到匹配对手/队友。

帮助玩家快速加入游戏

无论玩家需求如何变化，玩家都希望能快速进入游戏。Amazon GameLift Servers 提供一系列容量扩展工具，可帮助您确保玩家在游戏高峰期和低峰期均能快速匹配到可用游戏，具体包括：

- 玩家可轻松找到可用服务器：借助自动扩展工具，您可以确保托管资源在需要时随时可用。目标跟踪功能会管理灵活的容量缓冲区，通过预测游戏的玩家需求模式主动进行扩展。
- 玩家在特殊活动期间享受流畅体验：将自动扩缩与手动扩展技术结合使用，以应对计划性活动（如游戏版本更新、营销推广活动、电竞赛事）后突发的玩家流量涌入。
- 玩家无需等待服务器启动：在所有游戏托管资源上，游戏服务器均会提前预热，始终处于就绪状态，玩家一旦加入，即可立即开启新游戏会话。

灵活设计玩家组队方式，打造多样化游戏会话体验

玩家希望以公平、平衡的方式与他人竞技或协作。Amazon GameLift Servers 提供多种玩家组队方案，支持将玩家灵活分组进入游戏会话，具体包括：

- 对玩家进行分组以优化玩家体验：您可以使用一系列功能来管理最适合自己游戏的玩家分组。例如，允许玩家搜索或浏览可用游戏会话；或在玩家发起请求时，根据技能水平、延迟及其他属性自动分组。您既可自主管理玩家分组逻辑，也可实施 FlexMatch 对战系统，并通过可自定义的对战规则实现分组。
- 玩家可加入正在进行的对局：支持创建玩家群组以开启新游戏会话；也可通过玩家会话功能跟踪现有会话中的空余位置，将新玩家加入正在进行的对局。借助 FlexMatch，还能自动为现有会话回填，匹配与当前对局水平适配的玩家。
- 好友可组队共同玩游戏：支持“玩家队伍”功能，允许好友群组一同进入游戏。游戏会话可选择为剩余位置匹配其他玩家，或限制仅好友群组进入。借助 FlexMatch，好友可预先组建固定队伍参与匹配。
- 玩家可携带自定义数据进入会话：在发起游戏会话和玩家会话请求时，可附带游戏相关信息及玩家信息。这些信息会传递至游戏服务器，供当前游戏会话使用。
- 玩家无需无限等待“完美匹配”：有了 FlexMatch，您可以随着等待时间增加，逐步放宽匹配要求，避免玩家因排队时间过长产生不满。制定“匹配规则渐进放宽”策略，优先保障匹配质量的核心维度。

- 玩家可选择接受或拒绝匹配结果：借助 FlexMatch，您可以自定义工作流程，允许玩家对推荐的匹配结果进行“接受”或“拒绝”操作。

全程保障游戏连接稳定可靠

玩家期望获得无中断、无断连的持续游戏体验。Amazon GameLift Servers 通过全面的会话管理与保护机制实现这一需求，具体包括：

- 玩家可以在 AWS 云端享受高质量的性能 — 托管由托管 Amazon GameLift Servers，玩家可以获得在 AWS 计算基础设施上运行的快速、可靠的游戏服务器。
- 玩家连接时自动完成身份验证：当您为游戏启用玩家会话后，Amazon GameLift Servers 可在游戏会话中为玩家预留位置，并在玩家发起连接时自动完成身份验证。
- 玩家可重新连接并保留游戏进度：Amazon GameLift Servers 支持服务器端和客户端中断时重新连接。
- 玩家对局可避免意外终止：可选的游戏会话保护功能可防止处于活跃状态的游戏会话在扩展事件或其他中断场景中被意外终止。

基于真实数据改善玩家体验

玩家期望游戏不仅能持续稳定运行，还能不断迭代改进。Amazon GameLift Servers 提供全面的监控和分析能力，助力优化玩家体验，具体包括

- 玩家遭遇的中断问题更少：通过对实例集性能、游戏会话及玩家活动的实时监控，可在问题对游戏体验造成显著影响前及时识别风险。同时，支持自定义游戏会话的正常关闭和迁移。
- 玩家受益于主动服务器运行状况监控：Amazon GameLift Servers 会对服务器进行持续运行状况监控，一旦发现游戏服务器异常，会自动替换异常的服务器，从而最大限度降低硬件或软件故障对玩家的影响。您可根据不同游戏的要求配置运行状况检查参数。
- 玩家体验可通过数据驱动持续改进：利用游戏会话日志与详细的分析数据，可挖掘玩家行为模式和服务器性能规律。同时，支持为游戏特定事件添加自定义日志。

将增强的游戏功能与其他 AWS 服务集成

玩家需要集成功能，例如语音聊天、安全身份验证和持续进度。Amazon GameLift Servers 可以与其他 AWS 服务集成，提供全面的游戏体验：

- 玩家享受便捷的身份验证流程：通过集成 Amazon Cognito，可支持多种身份提供商和身份验证方法，同时在不同游戏会话间保持玩家身份的安全性，并简化了回归玩家的登录流程。
- 玩家游戏进度可跨会话永久保存：借助 Amazon DynamoDB 集成，可存储玩家的游戏进度、道具背包及其他永久数据，并具备高性能数据访问能力，既能最大限度降低对游戏运行的影响，又能支持玩家在不同会话间无缝接续游戏。
- 玩家体验可借助分析见解持续优化：通过集成 Amazon Kinesis 和 Amazon Simple Storage Service (Amazon S3)，可处理游戏分析数据，比如收集有关玩家行为和偏好的见解，从而实现实时分析，帮助开发人员适应不断变化的玩家模式，保持游戏的新鲜感和吸引力。

借助 Amazon GameLift Servers 管理生产环境中的游戏托管

借助 Amazon GameLift Servers 启动您的游戏后，您需要对游戏托管基础设施进行管理，以确保服务具备最优性能、高可靠性及良好的玩家体验。本主题将介绍生产环境中用于管理游戏托管生命周期的关键功能和工具。

监控主机运行状况和性能

Amazon GameLift Servers 提供全面的监控工具，帮助您跟踪游戏托管基础设施的运行状况和性能，具体包括：

- 性能指标：监控托管式实例集的关键性能指标，例如实例利用率、玩家会话数和游戏会话放置。在 Amazon 中 Amazon GameLift Servers 或亚马逊中跟踪这些指标 CloudWatch，您还可以在其中设置警报，以便在指标超过阈值时收到通知。
- 实例集指标和事件：跟踪实例集特定的指标，包括活跃服务器进程数、可用游戏会话数、活跃玩家会话数，以确保实例集具备最优容量。
- 游戏会话放置指标和事件：跟踪相关指标和事件，以监控游戏会话放置系统的运行状况和性能。
- 游戏会话日志：访问并分析游戏会话日志，以识别问题并了解玩家的行为模式。
- Realtime 脚本日志：对于使用实时服务器的游戏，可通过详细日志监控脚本执行情况 and 性能。
- AWS Health Dashboard-随时了解可能影响 Amazon GameLift Servers 部署的 AWS 服务运行状况。

您可以通过 Amazon GameLift Servers 控制台、AWS CLI 访问这些监控工具，也可以使用 AWS SDK Amazon GameLift Servers 来创建自定义仪表板和监控解决方案。

管理游戏服务器更新和补丁

保持游戏服务器更新对于保障安全性、优化性能及新增功能至关重要。Amazon GameLift Servers 提供多种更新管理方案，具体如下：

- 生成包管理：上传并管理游戏服务器生成包的多个版本。每个生成包均受版本控制，可部署到不同的实例集。
- 实例集替换：使用 AMI 版本和游戏服务器生成包更新创建新实例集，并使用游戏会话放置队列和别名逐步将流量从旧实例集迁移到新实例集。
- 脚本更新：对于实时服务器，通过上传新的脚本版本来更新服务器脚本，而无需替换整个实例集。
- 容器更新：对于基于容器的部署，请更新容器映像和定义。将新版本的游戏服务器部署到现有实例集中。替换实例集以更新 AMI 版本。
- 自动部署-使用 AWS CodePipeline 和创建 AWS CodeDeploy 用于自动更新游戏服务器的 CI/CD 管道。

在计划更新时，请考虑使用 blue/green 部署策略来最大限度地减少对活跃玩家的干扰，并在检测到问题时允许快速回滚。

优化性能和扩展

随着玩家群体的发展，您需要调整托管配置，以维持服务的最优性能和成本效益，具体优化方向如下：

- 自动扩缩：根据玩家人数或游戏会话利用率等指标配置实例集扩展策略，以自动调整容量。
- 区域部署调整：根据玩家地域分布和延迟需求，在多区域部署架构中新增或移除相应区域。
- 队列管理：优化游戏会话放置队列配置，在保障玩家体验与控制托管成本之间找到最佳平衡。
- 实例类型选择：分析性能指标，确定最具成本效益的实例类型，以满足您的游戏服务器需求。
- 竞价型实例使用情况：利用竞价型实例处理非关键工作负载以降低成本，同时制定适当的回退策略以切换至按需型实例。
- FlexMatch 调优：根据实际的玩家数据和反馈完善对战规则和算法。

定期查看 CloudWatch 指标和成本报告，以确定优化机会并通过 Amazon GameLift Servers 控制台或 API 实施更改。

故障排除和实时操作

高效的故障排除和实时操作管理是保障良好玩家体验的关键，具体措施如下：

- 实例集事件通知：针对实例集事件（例如扩展活动、实例终止或游戏会话放置失败）设置 Amazon Simple Notification Service 通知。
- 游戏会话放置调试：使用详细的放置日志来识别和解决游戏会话放置的问题。
- 服务器进程运行状况检查：监控服务器进程运行状况并自动替换未通过运行状况检查的进程。
- 远程访问：使用 AWS Systems Manager Session Manager 连接到实例集实例，直接进行故障排除。
- 别名管理：使用别名快速将玩家流量从存在问题的实例集重定向至正常的实例集，无需更改客户端配置。
- 备份和恢复：定期备份关键游戏数据和配置，以便从故障中快速恢复。

针对常见问题制定明确的事件响应程序和运行手册，以最大限度地减少停机时间，降低对玩家的影响。

工具和集成

Amazon GameLift Servers 与各种 AWS 服务和第三方工具集成，以增强您的生产管理能力和：

- AWS CloudFormation-将您的 GameLift 资源定义为基础设施即代码，以实现一致的部署。
- AWS Lambda：创建无服务器函数，以自动执行日常管理任务并对事件做出响应。
- Amazon EventBridge-构建可自动响应环境变化的事件驱动架构。Amazon GameLift Servers
- AWS SDK 集成- AWS SDKs 用于构建根据您的特定需求量身定制的自定义管理工具和仪表板。
- 第三方监控-使用 CloudWatch 指标导出功能与第三方监控和分析平台集成。

利用这些集成来创建符合团队工作流程和要求的全面管理解决方案。

相关资源

- [GameLift 使用亚马逊监控亚马逊 CloudWatch](#)
- [更新您的 Amazon GameLift 车队](#)
- [借助 Amazon 扩展游戏托管容量 GameLift](#)
- [亚马逊 GameLift 服务器 SDK 参考](#)

Amazon GameLift Servers 关键术语

本主题提供了 Amazon GameLift Servers 文档和服务中使用的关键技术术语的定义。

活跃游戏会话

当前正在运行并接受或托管玩家的游戏会话。用作监控实例集利用率和容量规划的指标。请参阅[Amazon GameLift Servers使用 Amazon 进行监控 CloudWatch](#)。

Alias

实例集的友好标识符，在游戏客户端和实例集资源之间提供了抽象层。别名允许您在不更改游戏客户端代码的情况下更新实例集配置。请参阅[使用别名抽象化 Amazon GameLift Servers 实例集名称](#)。

AMI (Amazon Machine Image)

预先配置的虚拟机映像，用于为托管队列创建 Amazon EC2 实例。包含运行游戏服务器所需的操作系统、运行时库和 Amazon GameLift Servers 代理软件。请参阅[Amazon GameLift Servers托管 EC2 车队](#)。

Anywhere 实例集

一种使用客户提供的计算资源的实例集类型，在 Amazon GameLift Servers 外部进行管理。Anywhere 实例集允许您将现有基础架构或混合云解决方案与 Amazon GameLift Servers 游戏会话管理相集成。请参阅[Amazon GameLift Servers Anywhere 实例集](#)。

身份验证令牌

由 Amazon GameLift Servers 生成的安全凭证，用于对在 Anywhere 计算资源上运行的游戏服务器进程进行身份验证。服务器进程需要使用此令牌，才能与 Amazon GameLift Servers 服务通信。请参阅[Amazon GameLift Servers Anywhere 实例集](#)。

后端服务

游戏客户端与 Amazon GameLift Servers 服务之间的协调层，用于控制游戏会话放置请求的通信并检索游戏会话信息。作为安全的中间层来保护托管资源免受恶意攻击。请参阅[使用 Amazon GameLift Servers 托管的工作原理](#)。

游戏服务器生成包

游戏服务器软件的打包版本，包含在 Amazon GameLift Servers 托管资源上运行游戏服务器所需的所有文件。其中包括游戏服务器可执行文件、依赖项和资产。请参阅[打包游戏服务器生成包用于部署](#)。

计算

在容器实例集中，表示实例集实例上游戏服务器容器组的副本。每个计算资源一次可托管一个游戏会话。请参阅[容器在 Amazon GameLift Servers 中的工作原理](#)。

容器实例集

一种托管式实例集类型，使用容器映像而非传统构建版本来运行容器化游戏服务器。它为部署具有特定运行时环境和依赖项的游戏服务器提供了灵活性。请参阅[Amazon GameLift Servers 托管式容器实例集](#)。

容器组

一组协同工作以提供游戏服务器功能的容器。与容器任务或容器组 (pod) 类似，它定义了容器在容器实例集中的行为方式、依赖关系和资源共享。请参阅[容器在 Amazon GameLift Servers 中的工作原理](#)。

目标位置

在游戏会话队列中，可以接收游戏会话放置请求的实例集或别名。队列可以包含多个具有不同优先级和延迟策略的目标。请参阅[自定义游戏会话队列](#)。

实例集

运行游戏服务器的计算资源集合。实例集资源可以分布在多个地理位置，每个资源运行一个或多个游戏服务器进程。请参阅[为 Amazon GameLift Servers 部署托管实例集](#)。

实例集位置

部署舰队实例的特定 AWS 区域或本地区域。多位置实例集可以跨越多个位置，以减少延迟并提高可用性。请参阅[为 Amazon GameLift Servers 部署托管实例集](#)。

FlexMatch

Amazon GameLift Servers 对战服务，能够根据可配置的规则自动形成玩家对战，并为已完成的对战匹配请求游戏会话放置。支持基于技能的对战、团队平衡和对战回填。看看[什么是 GameLift FlexMatch ?](#)

游戏客户端

在玩家设备上运行的游戏软件。通过与后端服务通信来启动玩家加入游戏会话，并直接连接到游戏服务器以让玩家参与游戏。请参阅[使用 Amazon GameLift Servers 托管的工作原理](#)。

游戏属性

与游戏会话关联的自定义数据键值对。用于将配置信息、游戏规则或其他元数据传递给游戏服务器和客户端。请参阅[GameProperty](#) 《API 参考》。

游戏服务器

您的自定义服务器软件，用于管理游戏状态、处理玩家操作并在多个连接的玩家之间同步游戏。与 Amazon GameLift Servers 服务器 SDK 集成来与服务进行通信。请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

游戏服务器激活

在托管资源上启动和初始化游戏服务器进程的过程。包括加载游戏服务器可执行文件、与 Amazon GameLift Servers 建立通信以及准备托管游戏会话。请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

游戏服务器容器

游戏服务器容器组中运行游戏服务器应用程序的容器。被系统自动指定为容器组生命周期必不可少的组件。请参阅[容器在 Amazon GameLift Servers 中的工作原理](#)。

游戏服务器容器组

一种容器组类型，用于管理运行游戏服务器应用程序和支持软件的容器。这种容器组是容器实例集托管游戏会话的必要组件，并且可以跨实例集实例进行复制。请参阅[容器在 Amazon GameLift Servers 中的工作原理](#)。

游戏服务器进程

在托管资源上运行的游戏服务器可执行文件实例。每个进程一次可托管一个游戏会话，并直接与 Amazon GameLift Servers 通信以报告状态和接收指令。请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

游戏会话

在游戏服务器进程上运行的多人游戏实例。代表多名玩家可以一起加入并参与游戏的单次游戏体验。请参阅 API 参考[GameSession](#)中的[使用 Amazon GameLift Servers 托管的工作原理](#)和。

游戏会话数据

与游戏会话相关的信息，包括玩家数据、游戏状态、会话属性以及游戏服务器和客户端使用的其他运行时信息。请参阅[GameSession](#)《API 参考》。

游戏会话放置

查找可用游戏服务器以托管新游戏会话的过程。使用有关游戏服务器可用性的实时信息，根据成本和延迟等因素做出最佳放置决策。请参阅 API 参考[StartGameSessionPlacement](#)中的[集成 Amazon GameLift Servers 游戏客户端功能](#)和。

游戏会话保护

一种设置，可防止 Amazon GameLift Servers 在扩缩事件或实例集更新期间终止正在托管活跃游戏会话的游戏服务器进程。确保正在进行的游戏不会中断。请参阅[手动设置Amazon GameLift Servers舰队的容量](#)。

游戏会话队列

一种放置机制，使用可配置的算法来放置游戏会话以实现最佳结果。可以跨多个地理位置，根据最低托管成本和最低玩家延迟等因素确定放置的优先顺序。请参阅[配置游戏会话放置](#)。

空闲实例

正在运行但当前未托管任何活跃游戏会话的实例集实例。用作监控实例集效率和扩缩决策的指标。请参阅[Amazon GameLift Servers使用 Amazon 进行监控 CloudWatch](#)。

入站权限

网络访问规则，用于定义哪些 IP 地址和端口范围可用于连接到实例集实例。控制对游戏服务器的访问并确保安全连接。托管 EC2 和托管集装箱舰队都需要入站权限配置。

请参阅[IpPermission](#)和[配置网络连接](#)。

延迟策略

游戏会话队列中的规则，定义了玩家连接的可接受延迟阈值。用于确保将玩家放置到具有可接受网络性能的游戏会话中。请参阅[自定义游戏会话队列](#)。

托管式实例集

一种由 Amazon GameLift Servers 在 AWS Cloud中部署和管理计算资源的实例集类型。提供自动扩缩、运行状况监控和基础架构管理。请参阅[Amazon GameLift Servers托管 EC2 车队](#)。

对战

根据技能等级、延迟或其他游戏特定因素等标准，将玩家分组以进行多人游戏的过程。在Amazon GameLift Servers，主要由... 处理 FlexMatch。看看[什么是 GameLift FlexMatch ?](#)

指标组

用于监控和自动扩缩决策的相关性能指标的集合。对活跃游戏会话、可用游戏服务器和玩家利用率等指标进行分组。请参阅[通过 Amazon GameLift Servers 自动扩展队组容量](#)。

每个实例容器组

一种可选的容器组类型，可在每个实例集实例上运行其他软件。对于后台服务或监控工具非常有用，每个实例仅部署一个副本。请参阅[容器在 Amazon GameLift Servers 中的工作原理](#)。

玩家会话

游戏会话中玩家的连接槽。用于预留游戏会话位置、验证玩家何时连接以及跟踪玩家连接状态和断开连接情况。请参阅 API 参考 [PlayerSession](#) 中的 [集成 Amazon GameLift Servers 游戏客户端功能和](#)。

优先级配置

在选择游戏会话放置目标时，自定义游戏会话队列的优先顺序的设置。它定义队列的整体行为，与修改单个放置请求优先级的优先级覆盖列表截然不同。请参阅 [优先考虑游戏会话放置](#)。

优先级覆盖列表

可包含在游戏会话放置请求中的备用位置优先级列表。仅针对单次请求有效地替换队列中配置的位置优先级，而不会影响其他请求。请参阅 [优先考虑游戏会话放置](#)。

进程管理器

管理容器或实例中多个游戏服务器进程生命周期的组件。用于处理启动、停止和监控各个游戏服务器进程。请参阅 [在托管式 Amazon GameLift Servers 上优化游戏服务器运行时配置](#)。

实时脚本

JavaScript 定义 Amazon GameLift Servers 实时服务器游戏逻辑的代码。可处理玩家连接、消息路由和游戏状态管理，无需自定义游戏服务器。参见 [什么是 Amazon GameLift 实时服务器？](#)

路由策略

确定别名如何将流量定向到实例集的配置。可以根据定义的规则路由到特定实例集，或将流量分配到多个实例集。请参阅 [使用别名抽象化 Amazon GameLift Servers 实例集名称](#)。

运行时配置

定义游戏服务器进程如何在实例集实例上运行的设置，包括并发进程数量、启动参数和资源分配。请参阅 [在托管式 Amazon GameLift Servers 上优化游戏服务器运行时配置](#)。

扩展

根据玩家需求，通过添加或移除托管资源来调整实例集容量的过程。可以是自动的（基于利用率指标），也可以是手动的（由管理员控制）。请参阅 [通过 Amazon GameLift Servers 自动扩展队组容量](#)。

扩缩策略

定义何时以及如何自动调整实例集容量的规则。可以基于目标（保持特定的利用率），也可以基于规则（根据指标阈值进行扩缩）。请参阅 [通过 Amazon GameLift Servers 自动扩展队组容量](#)。

服务器软件开发工具包

提供将游戏服务器与 Amazon GameLift Servers 集成所需工具和库的软件开发工具包。在游戏服务器与服务之间建立通信，以实现会话管理、运行状况报告和状态更新。请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

基于会话的多人游戏

一种多人游戏架构，在这种架构中，游戏玩法发生在离散的、有时间限制的会话中。每个会话有明确的开始和结束节点，玩家可以根据需要加入和离开。请参阅[使用 Amazon GameLift Servers 托管的工作原理](#)。

Spot 实例

一种以较低的成本使用备用 AWS 容量的 Amazon EC2 实例。竞价型实例在 AWS 需要恢复容量时可以中断，这使得它们适合容错的游戏工作负载。请参阅[为竞价型实例构建队列](#)。

TLS 证书

一种用于在游戏客户端和服务器之间实现加密通信的数字证书。启用 TLS 的实例集必须使用此证书来确保安全的数据传输。请参阅[创建 Amazon GameLift Servers 托管 EC2 舰队](#)。

Amazon GameLift Servers 入门

利用这些入门资源详细了解 Amazon GameLift Servers 服务以及如何开始为基于会话的多人游戏开发自定义托管解决方案。

开始之前

- 创建一个 AWS 账户（或指定一个现有的）以与一起使用 Amazon GameLift Servers。
- 为用户设置访问 Amazon GameLift Servers 和相关 AWS 服务的权限。
- 选择 AWS 区域要使用的。开发阶段建议选择靠近您所在位置的区域。您可随时更改区域。

[设置 AWS 用户账户](#)

选择您的路径

我希望快速体验 Amazon GameLift Servers

适用场景：学习、创建概念验证、快速原型开发

快速启动选项：

- [适用于 Amazon GameLift Servers 的游戏服务器封装器](#)：参阅[教程：使用 Amazon GameLift Servers 封装器快速入门](#)。该工具是通过 Amazon GameLift Servers 托管游戏服务器并运行游戏会话的最快捷、最简便的方式，无需修改游戏代码。游戏服务器封装器提供基本的游戏会话管理功能，以及简化的游戏服务器部署流程。如果您希望通过自有游戏项目或示例项目，亲自评估 Amazon GameLift Servers 的功能，该工具是理想选择。准备好构建自定义游戏托管解决方案时，可切换至任一自定义开发选项，通过 Amazon GameLift Servers 的服务器 SDK 实现完全集成。如果您的游戏无需自定义托管解决方案，也可继续使用游戏服务器封装器，在生产环境中部署和托管游戏服务器。
- [适用于 Unreal Engine 或 Unity 的 Amazon GameLift Servers 插件](#)：该插件提供 GUI 工作流程和示例资源，可引导您完成初始操作步骤，并基于基本托管解决方案部署游戏服务器。使用该插件设置具有自我管理的 Anywhere 队列的托管，或者部署基于云的托管 EC2 舰队或集装箱舰队。当您准备好开发自定义托管解决方案时，可在插件构建的解决方案的基础上进行构建。
- [适用于 Amazon GameLift Servers 托管式容器的入门工具包](#)：该工具包可简化多项任务，包括集成游戏服务器、准备游戏服务器容器映像，以及部署用于托管的容器实例集。在集成环节，该工具包会为您的游戏服务器添加必要的游戏会话管理功能。此外，该工具包还通过预配置的模板搭建容器实例

集，并为游戏服务器构建自动化部署管道。准备好添加完整的游戏会话管理功能时，请遵循其中一个自定义开发路线图集成 Amazon GameLift Servers 的服务器 SDK。

我希望构建自定义托管解决方案

适用场景：为自定义游戏服务器创建生产级解决方案

查看“[Amazon GameLift Servers 游戏托管方案](#)”页面，然后为您的解决方案选择其中一个选项。遵循其中一个开发路线图，开始为您的游戏构建功能齐全的自定义托管解决方案。这些路线图会提供详细指导，涵盖托管解决方案中各组件的创建、测试和自定义配置流程。

- [借助 Amazon GameLift Servers 托管式 EC2 进行托管的开发路线图](#)
- [借助 Amazon GameLift Servers 托管容器进行托管的开发路线图](#)
- [借助 Amazon GameLift Servers Anywhere 进行托管的开发路线图](#)
- [借助 Amazon GameLift Servers 进行混合托管的开发路线图](#)

推荐学习路径

1. 从小处着手。尝试使用游戏服务器封装器或适用于 Unreal 的插件。
2. 了解相关概念。查看“[使用 Amazon GameLift Servers 托管的工作原理](#)”页面。
3. 选择架构。根据游戏的需求选择合适的托管模型。
4. 构建和测试。遵循相应的开发路线图。搭建各组件的基础版本，然后进行迭代和自定义配置。
5. 扩展和优化。针对生产级使用场景优化解决方案。添加对战等高级功能。

设置 AWS 用户账户

Tip

使用这些主题来获取有关以下任务的帮助：

- 换一个新 AWS 账户 的，一起使用 Amazon GameLift Servers。
- 创建具有 Amazon GameLift Servers 资源使用权限的用户或组。
- 设置安全证书（你需要这些证书才能使用虚幻和 Unity 的 AWS CLI 工具和 Amazon GameLift Servers 插件）

-

与所有 AWS 服务一样，AWS 账户 您需要使用 Amazon GameLift Servers 服务和工具。具有两个主要功能：(1) 它为您提供一个容器，用于存放您使用该账户创建的所有 AWS 资源；以及 (2) 它允许您管理 AWS 资源的安全，包括设置用户身份验证和控制用户访问权限。AWS 账户 创建 不收取任何费用 AWS 账户。

Amazon GameLift Servers 使用或不使用 AWS 账号进行探索

您无需 AWS 账户即可执行以下操作：

- 在 [for Games 上探索用于构建、运行和提升游戏体验AWS 的 AWS 工具](#)。阅读 [博客](#) 并浏览 [游戏解决方案库](#)。
- Amazon GameLift Servers 在 [产品概述和资源](#) 中了解更多信息。FAQs 向 AWS 提问以获取关于产品问题的解答。（试试这个：“寻找低成本方案来托管我的多人游戏”。）
- 查阅 [技术文档](#)，更深入地了解 Amazon GameLift Servers 的工作原理，包括有关托管和对战的开发人员指南以及服务 API 参考指南。
- 查看有关 [Amazon GameLift Servers 定价](#) 和成本优化技术的信息。试用 [定价计算器](#)，了解根据高峰并发玩家用量（CCU）计算托管费用的方式。
- 下载并查看插件和工具包的 Amazon GameLift Servers SDKs 代码存储库。请参阅 [Amazon GameLift Servers 入门](#)。（您需要一个 AWS 账户 才能使用它们。）

您确实需要一个 AWS 账户才能执行以下操作：

- 使用适用于 Unreal 和 Unity 的 Amazon GameLift Servers 插件来完成入门工作流程，或者使用游戏服务器封装器完成该流程。
- 使用 AWS 管理控制台创建和管理 AWS 资源。
- 使用 AWS Command Line Interface 创建和管理 AWS 资源。
- 结合使用 Amazon Q 和 Amazon GameLift Servers 技术文档来查找答案、获取指导和建议。

主题

- [注册获取 AWS 账户](#)
- [创建具有管理访问权限的用户](#)

- [为 Amazon GameLift Servers 设置用户权限](#)
- [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 Servers 的 IAM 权限示例](#)
- [为 Amazon GameLift Servers 设置 IAM 服务角色](#)

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/注册>。
2. 按照屏幕上的说明操作。

在注册时，将接到电话或收到短信，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。作为最佳安全实践，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。您可以随时前往 <https://aws.amazon.com/> 并选择“我的账户”，查看您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 管理控制台](#)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 [Signing in as the root user](#)。

2. 为您的根用户启用多重身份验证 (MFA) 。

有关说明，请参阅 [IAM 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 URL。

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 [Create a permission set](#)。

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 [Add groups](#)。

为 Amazon GameLift Servers 设置用户权限

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 Servers 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 Servers 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。
- 通过身份提供者在 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[针对第三方身份提供者创建角色 \(联合身份验证 \)](#)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 管理控制台。授予编程访问权限的方式取决于正在访问的用户类型 AWS。

要向用户授予程式访问权限，请选择以下选项之一。

哪个用户需要程式访问权限？	目的	方式
IAM	(推荐) 使用控制台凭证作为临时凭证，签署对 AWS CLI AWS SDKs、或的编程请求 AWS APIs。	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》中的“登录 AWS 本地开发”。 • 有关信息 AWS SDKs，请参阅《工具参考指南》AWS SDKs 和《工具参考指南》中的“登录进行 AWS 本地开发”。

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 SDKs、或发出的编程请求 AWS APIs。	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”。 • 有关工具和 AWS SDKs AWS APIs，请参阅《工具参考指南》中的 IAM 身份中心身份验证 AWS SDKs 和工具参考指南。
IAM	使用临时证书签署向 AWS CLI AWS SDKs、或发出的编程请求 AWS APIs。	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 SDKs、或发出的编程请求 AWS APIs。	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 SDKs 和工具，请参阅《工具参考指南》AWS SDKs 和《工具参考指南》中的使用长期凭证进行身份验证。 • 有关信息 AWS APIs，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](#)。

为游戏设置程式访问权限

大多数游戏都使用后端服务与Amazon GameLift Servers之通信 AWS SDKs。使用后端服务（代表游戏客户端）请求游戏会话、让玩家进入游戏中以及执行其他任务。这些服务需要程式访问权限和安全凭证来验证对 Amazon GameLift Servers 服务 API 的调用。

对于Amazon GameLift Servers，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 GameLift Servers 提出请求之前，后端服务必须包含担任此角色的代码。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 GameLift Servers 通信时必须存储和使用这些证书。

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 Servers 的 IAM 权限示例

使用这些示例中的语法为需要访问Amazon GameLift Servers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[为 Amazon GameLift Servers 设置用户权限](#)。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。

如果您要Amazon GameLift ServersFleetIQ用作独立解决方案，请参阅[设置您 AWS 账户 的Amazon GameLift ServersFleetIQ](#)。

管理权限示例

这些示例为托管管理员或开发人员提供了有针对性的访问权限，以便管理 Amazon GameLift Servers 游戏托管资源。

Example Amazon GameLift Servers 资源完全访问权限语法

以下示例扩展了对所有 Amazon GameLift Servers 资源的完全访问权限。

JSON

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

Example 支持默认未启用区域的 Amazon GameLift Servers 资源权限的语法

以下示例将访问权限扩展到所有默认情况下未启用的 Amazon GameLift Servers 资源和 AWS 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](#)。

JSON

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

Example Amazon GameLift Servers 资源访问 Amazon ECR 中的容器映像的语法

以下示例扩展了 Amazon GameLift Servers 用户在使用托管式容器实例集时所需的 Amazon Elastic Container Registry (Amazon ECR) 操作的访问权限。

JSON

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

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

以下示例扩展了对所有 Amazon GameLift Servers 资源的访问权限，并允许用户将 IAM 服务角色传递给 Amazon GameLift Servers。服务角色使 Amazon GameLift Servers 能够有限地代表您访问其他资源和服务，如[为 Amazon GameLift Servers 设置 IAM 服务角色](#)中所述。例如，在响应 CreateBuild 请求时，Amazon GameLift Servers 需要访问您在 Amazon S3 存储桶中的生成包文件。有关该 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](#)。

JSON

```
{
  "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 Servers 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

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

JSON

```

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

```

Example对战权限的语法

以下示例扩展了FlexMatch对管理配对活动的访问权限。Amazon GameLift Servers APIs FlexMatch为新游戏或现有游戏会话匹配玩家，并为托管的游戏启动游戏会话放置。Amazon GameLift Servers有关FlexMatch的更多信息，请参阅[什么是 Amazon GameLift Servers FlexMatch ?](#)

JSON

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

Example手动游戏会话放置权限的语法

以下示例扩展了对在Amazon GameLift Servers APIs 指定队列上手动创建游戏会话和玩家会话的访问权限。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（“list-and-pick”方法）。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "PlayerPermissionsForManualGameSessions",
    "Effect": "Allow",
    "Action": [
      "gamelift:CreateGameSession",

```



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

为 Amazon GameLift Servers 设置 IAM 服务角色

某些 Amazon GameLift Servers 功能要求您将有限的访问权限扩展到您拥有的其他 AWS 资源。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 角色与 IAM 用户类似，因为它是一个具有权限策略的 AWS 身份，该策略决定了该身份可以做什么和不能做什么 AWS。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，当您担任角色时，它会为您的角色会话提供临时安全证书。

本主题介绍如何创建可用于 Amazon GameLift Servers 托管式实例集的角色。如果您使用 Amazon GameLift Servers FleetIQ 来优化亚马逊弹性计算云 (Amazon EC2) 实例上的游戏托管，[请参阅设置 AWS 账户](#) 您的。Amazon GameLift Servers FleetIQ

在以下步骤中，创建一个拥有自定义权限策略和允许 Amazon GameLift Servers 担任角色的信任策略的角色。

为 Amazon GameLift Servers 托管式 EC2 实例集创建 IAM 服务角色

步骤 1：创建权限策略。

使用本页面上的说明和示例，为您正在使用的 Amazon GameLift Servers 实例集类型创建自定义权限策略。

使用 JSON 策略编辑器创建策略

1. 登录 AWS 管理控制台 并打开 IAM 控制台，网址为 <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，则会显示欢迎访问托管式策略页面。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》中的[调整策略结构](#)。

7. （可选）在中创建或编辑策略时 AWS 管理控制台，可以生成可在模板中使用的 JSON 或 YAML 策略 CloudFormation 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。要了解更多信息 CloudFormation，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[AWS Identity and Access Management 资源标签](#)。
11. 选择创建策略可保存您的新策略。

步骤 2：创建 Amazon GameLift Servers 可以担任的角色。

创建 IAM 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。

JSON

```
{
  "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。

为 Amazon GameLift Servers 托管式容器创建 IAM 角色

如果您使用的是 Amazon GameLift Servers 托管式容器，需要创建用于容器实例集的 IAM 服务角色。此角色授予 Amazon GameLift Servers 管理您的容器实例集资源并代表您执行操作所需的有限权限。

为容器实例集创建 IAM 角色

1. 登录 AWS 管理控制台 并打开 IAM 控制台，网址为 <https://console.aws.amazon.com/iam/>。
2. 在 IAM 控制台的导航窗格中，选择角色，然后选择创建角色。
3. 在“选择可信实体”页面上，选择 AWS 服务，然后选择用例“GameLift”。选择下一步。
4. 在添加权限页面上，选择托管策略 GameLiftContainerFleetPolicy。选择下一步。有关此策略的更多信息，请参阅 [AWS 的托管策略 Amazon GameLift Servers](#)。
5. 在命名、查看和创建页面上，输入角色名称，然后选择创建角色以保存新角色。

权限策略语法

- Amazon GameLift Servers 担任服务角色的权限

JSON

```
{
  "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

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "gamelift.amazonaws.com",
          "gamelift.ap-east-1.amazonaws.com",
          "gamelift.me-south-1.amazonaws.com",
          "gamelift.af-south-1.amazonaws.com",
          "gamelift.eu-south-1.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

获取 Amazon GameLift Servers 开发工具

Amazon GameLift Servers 提供了一组工具 SDKs 和其他工具，可帮助您为游戏构建游戏托管解决方案。为游戏服务器、游戏客户端和后端服务 SDKs 添加功能，使它们能够与 Amazon GameLift Servers 服务进行交互。有关 Amazon GameLift Servers SDK 版本及兼容性的最新信息，请参阅 [Amazon GameLift Servers 发布说明](#)。

对于游戏服务器

使用 Amazon GameLift Servers 的服务器 SDK 集成和构建 64 位游戏服务器。游戏服务器使用服务器 SDK 与 Amazon GameLift Servers 服务通信以进行游戏会话管理，包括启动、更新和停止游戏会话。有关将服务器 SDK 集成到游戏项目的帮助，请参阅[准备使用 Amazon GameLift Servers 托管游戏](#)。

开发 Linux 支持

- 开发操作系统
 - Windows
 - Linux
- 编程语言

[获取 Amazon GameLift Servers SDK](#)。有关特定于版本的信息和安装说明，请参阅每个软件包中包含的自述文件。

- [C++ 服务器 SDK](#)
 - [服务器 SDK 参考](#)
 - [如何集成](#)
- [C# 服务器 SDK](#) (对 .NET 4、.NET 6、.NET 8 的支持因版本而异，请参阅[SDK 版本](#))
 - [服务器 SDK 参考](#)
 - [如何集成](#)
- [Go 服务器 SDK](#)
 - [服务器 SDK 参考](#)
 - [如何集成](#)
- 游戏引擎支持

功能齐全的插件 Amazon GameLift Servers 包括用户界面工作流程和示例资产，以及 AWS SDK 和服务器 SDK 的内置版本。这些工作流程将指导您完成如何配置和部署游戏服务器，以便通过托管式 EC2 实例集、托管式容器实例集或自行管理的 Anywhere 实例集进行托管。如果您不需要引导式工作流程，则只需从相同的 GitHub 存储库中获取游戏引擎的服务器 SDK。

如果您使用的是该插件不支持的其他游戏引擎或开发环境，请获取适用于您的编程语言的服务器 SDK 并将其添加到您的游戏项目中。

有关特定于版本的信息和安装说明，请参阅每个软件包中包含的自述文件。

- [适用于 Unreal Engine 的插件](#)：专为适用于 Unreal Engine 5.0、5.1、5.2、5.3、5.4、5.5 和 5.6 版本而构建。请查看特定版本的自述文件，了解 Unreal 的版本支持详情。
 - [Unreal Engine 插件指南](#)
 - [适用于 Amazon GameLift Servers 的 C++ \(Unreal \) 服务器 SDK 5.x – 操作](#)
- [适用于 Unity 的插件](#)：专为适用于 Unity Editor 6.0、2022.3 或 2021.3 的 LTS 版本而构建。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。请查看特定版本的自述文件，了解 Unity 的版本支持详情。
 - [Unity 插件指南](#)
 - [C# 服务器 SDK 参考](#)
- [适用于 Unreal 的服务器 SDK](#)
 - [服务器 SDK 参考](#)
 - [将 Amazon GameLift Servers 集成到 Unreal Engine 项目中](#)
- [适用于 Unity 的服务器 SDK](#)
 - [服务器 SDK 参考](#)
 - [将 Amazon GameLift Servers 集成到 Unity 项目中](#)

运行时支持

对于托管托管解决方案，请将您的游戏服务器构建为在以下 Amazon 计算机映像之一上运行 (AMIs)。有关更多详细信息，请参阅适用于 Amazon GameLift Servers 的 [Amazon GameLift Servers AMI 版本](#)。

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

Note

亚马逊 Linux 2 (AL2) 将于 2026 年 6 月 30 日终止支持。在[亚马逊 Linux 2](#) 中查看更多详情 FAQs。对于托管在服务器 SDK 4.x 上 AL2 并使用 Amazon GameLift Servers 服务器 SDK 4.x 的游戏服务器，请先将游戏服务器版本更新为服务器 SDK 5.x，然后部署到 AL2023 实例。请参阅[迁移到 Amazon GameLift Servers 服务器 SDK 5.x](#)。

其他工具

[适用于 Amazon GameLift Servers 的游戏服务器封装器](#)

此工具可帮助您部署用于托管的游戏服务器，并提供一套基础的游戏会话管理功能。使用此工具，您无需对游戏代码进行更改或集成 Amazon GameLift Servers 的服务器 SDK。通过游戏服务器封装器，您可对游戏服务器进行打包，并基于 Amazon GameLift Servers 的三类托管解决方案（Anywhere、托管式 EC2 或托管式容器）中的任意一种，部署服务器以实现游戏托管。此工具最适用于使用自有游戏或示例游戏进行早期评估或原型开发，因它不支持游戏服务器的自定义配置。如果您的游戏不需要自定义功能，则可以使用游戏服务器封装器部署游戏服务器以进行生产托管。

[Amazon GameLift Servers 工具包](#)

Amazon GameLift Servers Toolkit 是我们开发的一系列脚本及其他工具，旨在帮助开发人员解决常见场景和问题。Toolkit 材料包含脚本、示例代码和自述文件。

- [容器入门工具包](#)：使用此工具可以简化设置游戏服务器生成包以通过 Amazon GameLift Servers 托管容器进行托管的任务。该工具包会将核心游戏会话管理功能集成至游戏服务器，并通过预配置的模板创建容器实例集，同时为游戏服务器生成包设置自动化部署管道。部署完成后，您可以使用 Amazon GameLift Servers 控制台和 API 工具监控实例集性能、管理游戏会话并分析指标。该套件与 Amazon Simple Storage Service 集成，AWS CodeBuild 用于构建自动化，AWS CloudFormation 用于存储和基础设施部署。
- [快速生成包更新工具](#)：使用此工具可修改已部署至托管式 EC2 实例集的游戏服务器生成包。该工具旨在帮助您快速更换游戏生成包文件，而无需在每次更改时重新配置并创建新的 EC2 实例集。您可选择更新实例集中的单个实例或所有实例。提供的选项允许您替换特定生成包文件或完整生成包，并可管理更新后游戏服务器的重启方式。

对于游戏客户

将您的游戏客户端与适用于的客户端 SDK 集成 Amazon GameLift Servers。客户端 SDK 为游戏客户端提供实用工具，供其使用包括玩家网关和 UDP ping 信标在内的 Amazon GameLift Servers 功能。

获取客户端 SDK

- [适用于 C++ 的客户端 SDK](#) — 包括带有内置端点选择算法的玩家网关和 UDP ping 信标实用程序，以及演示集成的示例应用程序。需要 C++17 和一个线程库。
- [适用于虚幻引擎的客户端 SDK](#) — 包括玩家网关（无论是否修改引擎）和带有内置端点选择算法的 UDP ping 信标实用程序。需要源代码构建的虚幻引擎 5.1 或更高版本。访问权限需要在 [Epic Games 组织](#) 中的成员资格 GitHub。

有关集成指南，请参阅每个 GitHub 存储库中的文档。

用于游戏后端服务

为您的游戏创建 64 位后端服务，并将其与 AWS SDK 中的功能集成，其中包括的服务 API。Amazon GameLift Servers 使用后端服务来处理与该 Amazon GameLift Servers 服务的互动，包括启动或查找游戏会话以及让玩家加入游戏。

[获取 AWS SDK](#)

有关将 AWS SDK 与配合使用的更多信息 Amazon GameLift Servers，请参阅以下资源：

- [Amazon GameLift Servers API 参考](#)
- 集成游戏客户端功能
 - [适用于带自定义游戏服务器生成包的游戏的客户端服务集成](#)
 - [适用于带 Realtime 服务器的客户端服务集成](#)
- [设计后端服务](#)

对于 Amazon GameLift Servers 资源管理

使用以下工具创建、更新和监控您的 Amazon GameLift Servers 托管式托管资源。

- [AWS 管理控制台](#)— AWS 控制台是一个基于 Web 的应用程序，可集中访问所有单独的 AWS 服务控制台，包括 Amazon GameLift Servers。使用主机创建或登录 AWS 账户，然后打开 Amazon GameLift Servers 主机使用您的游戏托管资源。配置并部署托管实例集及其他资源，查看使用情况和性能指标，在控制面板中跟踪资源，以及执行其他各类操作。[转到 Amazon GameLift Servers 控制台。](#)
- [Amazon GameLift Servers 的服务 API](#)：此 API 允许您以编程方式访问所有 Amazon GameLift Servers 资源。它是 AWS SDK 的一部分，您可以下载它以用于大多数流行的编程语言。[获取 AWS 软件开发工具包。](#)
- [AWS 命令行界面 \(CLI\)](#)— AWS CLI 允许您使用命令行外壳与 AWS 服务进行交互。这些工具允许公众 APIs 直接访问 AWS 服务以及可用于服务的自定义命令。[获取 C AWS LI。](#)
- [AWS CloudFormation](#) for Amazon GameLift Servers — 该 CloudFormation 服务可帮助您建模和设置 AWS 资源，以简化基础架构的部署和管理。创建 CloudFormation 模板来描述您的托管解决方案的 Amazon GameLift Servers 资源，然后使用该模板来构建其他资源或更新配置。查看 [Amazon GameLift Servers 资源类型参考](#)。

对于 Amazon GameLift Servers Realtime

配置和部署 Realtime 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 Realtime 服务器，请使用 Amazon GameLift Servers Realtime 客户端 SDK。首先，[下载 Realtime 客户端 SDK](#)。有关配置信息，请参阅[为 Amazon GameLift Servers Realtime 集成游戏客户端](#)。

SDK 支持

Realtime 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

您可将 Realtime 服务器部署到运行以下平台的托管资源：

- [Amazon Linux 2](#)

Note

AL2 支持已接近尾声。在[亚马逊 Linux 2](#) 中查看更多详情 FAQs。

- [Amazon Linux 2023](#)

教程：使用 Amazon GameLift Servers 封装器快速入门

欢迎使用 Amazon GameLift Servers 的入门教程。在本教程中，您将快速部署游戏服务器，使其托管于基于云的计算资源实例集上。通过本教程，您无需将 Amazon GameLift Servers 的服务器 SDK 集成到游戏代码中，而是直接部署具备基本功能的游戏，使其能够与 Amazon GameLift Servers 服务通信并运行游戏会话。您将搭建基础托管解决方案，并通过该解决方案体验自动扩展、对战系统等全套功能。此解决方案同样适用于托管游戏原型，可用于现场演示或测试场景。

这种入门方法的核心优势：

- 快速部署游戏服务器，实现高效托管。
- 无需更改游戏代码，零修改要求。
- 适用于任何游戏可执行文件，不受游戏引擎限制。
- 探索所有 Amazon GameLift Servers 管理工具，包括监控游戏会话活动和托管运行状况。

Note

该封装器适用于评估场景及基础生产环境使用。详细玩家会话管理等高级功能需完整集成服务器 SDK 方可使用。

先决条件

在开始之前，请确保您满足以下条件：

- AWS 账户 具有适当权限的
- AWS CLI 已安装
- Go 1.18 及以上版本
- 多人游戏服务器可执行文件
- Make 工具 (Linux/Mac 系统)
- 已安装 Git 且账户处于活跃状态

概述

本教程将引导您完成以下操作：

1. 获取并构建封装器
2. 准备游戏生成包
3. 配置封装器
4. 上传游戏服务器生成包
5. 创建托管 EC2 舰队
6. 创建游戏会话并连接

7. 监控和管理游戏服务器
8. 扩展游戏服务器

第 1 步：获取并构建游戏服务器封装器

使用以下命令获取游戏服务器封装器源代码并构建封装器。这些命令基于 SSH 执行，您也可直接访问 GitHub 存储库完成操作。

Windows

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

Mac 和 Linux

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

构建成功后，amazon-gamelift-servers-game-server-wrapper 目录下会新增一个“out”目录。该目录有三个文件夹，分别对应三种受支持的托管实例集选项，其中均包含一组构建构件。在本教程中，您将部署到托管 EC2 队列，因此您将使用该文件夹gamelift-servers-managed-ec2。

第 2 步：准备游戏服务器生成包

在此步骤中，您将准备要上传到 Amazon GameLift Servers 的游戏服务器生成包文件。

创建游戏目录

现在，在本地计算机上准备一个游戏目录。该目录需包含通过 Amazon GameLift Servers 运行游戏服务器所需的全部文件，包括游戏服务器封装器、游戏服务器生成包，以及使封装器与游戏服务器搭配运行的 config.yaml 配置文件。

使用以下步骤：

1. 托管 EC2 车队。在游戏服务器包装器输出文件夹中，找到用于部署到托管 EC2 队列的构建工件。该构建构件会写入如下指定的 out 目录：out\linux\amd64\gamelift-servers-managed-ec2。
2. 将游戏服务器可执行文件及其运行所需的全部关联文件复制至 gamelift-servers-managed-ec2 文件夹。您可根据需要创建嵌套目录。

示例目录结构如下所示：

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

第 3 步：为实例集配置封装器

Amazon GameLift Servers 会管理实例集的计算实例生命周期，启动已安装您的服务器生成包的新实例，并根据需要回收实例。该服务同时管理每个实例上运行的游戏服务器进程生命周期。托管 EC2 舰队可以在多个地点拥有实例，无论玩家身在何处，都能为他们提供支持。

编辑 config.yaml 文件以配置封装器的日志、端口设置及服务器初始化相关参数。

1. 配置日志设置。游戏服务器封装器会为每个游戏服务器进程生成日志信息。默认日志级别为 debug，以输出最详尽的日志内容。该级别在部署配置和故障排除阶段非常实用，其日志消息详尽程度为最高级。可选日志级别包括 debug、info、warn 和 error（日志详尽程度最低）。
2. 指定游戏服务器日志目录的路径。游戏服务器日志的默认路径是 ./game-server-logs。该目录包含游戏服务器生成的所有日志，且每个实例均会生成独立日志文件。日志会自动上传至 Amazon GameLift Servers，您可在事件选项卡中查看。有关更多详细信息，请查看“故障排除”部分。
3. 定义网络端口配置。游戏端口可按需自定义设置。本教程仅需指定一个端口，因为创建的实例集每个实例仅运行一个并发游戏服务器进程。若需同时运行多个进程，则需为每个并发进程配置足够的端口。配置文件中默认端口值为 37016，通用端口规则如下：对于使用 Linux 生成包的实例集，使

用 22 端口及 1026-60000 端口段；对于使用 Windows 生成包的实例集，使用 1026-60000 端口段。

4. 设置游戏服务器可执行文件的路径。对于 `./MyGame/my-server-executable.exe`，请根据实际游戏服务器可执行文件的名称和位置自定义其路径。该路径是启动游戏服务器的入口。
5. 配置游戏服务器参数。至少需指定 `-port` 参数，并使用之前定义的游戏端口值。“pos”值设为 0，表示这是第一个参数。可按需添加其他参数，这些参数会在游戏服务器启动时传入，用于配置其运行时行为。
 - a. 参数：`--port`
 - b. 值：`"{{.port number here}}"`
 - c. 位置：0（列表中的第一个参数）

示例配置：

```
log-config:
  wrapper-log-level: debug
game-server-logs-dir: ./game-server-logs
ports:
  gamePort: 37016
game-server-details:
  executable-file-path: ./MyGame/my-server-executable
game-server-args:
  - arg: "--port"
    val: "{{.gameport}}"
    pos: 0
```

第 4 步：上传游戏服务器生成包

您现已完成游戏服务器生成包的全部必要配置（含游戏服务器封装器、`config.yaml` 配置文件及游戏服务器文件），可将该游戏生成包上传至 Amazon GameLift Servers 进行托管。上传游戏版本的最快方法是使用 AWS CLI 命令，`upload-build` 如以下示例所示。

通过 Windows 上传游戏生成包：

```
aws gamelift upload-build \
  --name gamelift-test-2025-03-11-1 \
  --build-version gamelift-test-2025-03-11-1 \
```

```
--build-root out/windows/amd64/gamelift-servers-managed-ec2 \  
--operating-system WINDOWS_2022 \  
--server-sdk-version 5.3.0 \  
--region us-west-2
```

Note

对于 Mac 和 Linux 生成包，请使用 `--operating-system AMAZON_LINUX_2023`

创建生成包时，请记录 API 响应中的生成包 ID 以将其用于实例集创建。

步骤 5：创建托管 EC2 舰队

以下步骤描述了最小化实例集配置流程，帮助您快速启动并运行实例集。

创建实例集：

1. 登录 AWS 管理控制台 并导航至 Amazon GameLift Servers。
2. 在控制台窗口顶部的菜单栏中，确认您的生成包所在的区域 并记录，因为您的实例集必须与生成包处于同一区域，否则将无法查找或选择该生成包。
3. 在导航窗格的“托管”EC2 部分中，选择“生成”。
4. 选择您之前上传的生成包，以显示生成包详情信息页面。
5. 在“舰队”部分中，选择“创建舰队”，这将显示“定义托管 EC2 舰队”详细信息页面，您可以从中跟踪舰队的状态以及在“事件”选项卡上查看舰队创建事件。
6. 填写实例集名称和描述，然后选择下一步。
7. 在“定义实例详细信息”页面中，默认显示生成包所在的区域。如需添加任何其他区域，可自行选择。
8. 对于实例集类型，请选择按需型。
9. 在实例类型下选择 c5.large，然后选择下一步。
10. 在运行时配置下，由于您上传的游戏生成包已使用封装器，因此需指定封装器可执行文件路径而非游戏服务器本身。对于 Windows 游戏服务器，路径为 `C:\game\amazon-gamelift-servers-game-server-wrapper.exe`。对于 Linux 游戏服务器，路径为 `/local/game/amazon-gamelift-servers-game-server-wrapper`。

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

此外，配置游戏端口范围值，该范围需包含您在 `config.yaml` 中设置的端口及运行时配置的启动参数端口。`config.yaml` 中的端口无需与实例集的运行时配置中指定的端口一致，但运行时若后者端口不同，该值将覆盖 `config.yaml` 中的配置。运行时配置中输入的启动参数同样会覆盖 `config.yaml` 中的相应设置。

11. 在审核并创建页面上，仔细核对所有配置，然后选择提交以创建您的实例集。实例集会启动容量以托管游戏服务器，状态将随之更新，很快显示为活跃。激活完成且实例集部署就绪后，服务会自动启动封装器，此时封装器已准备就绪，可接收游戏会话请求。

第 6 步：创建游戏会话并连接

当实例集状态显示为活跃时，表明游戏服务器已准备就绪，可等待托管游戏会话。要启动游戏会话，需向 Amazon GameLift Servers 服务发起游戏会话请求。在这里，您将使用 AWS CLI 来提出此请求。

Note

请记住，使用创建游戏会话对于测试和熟悉该过程非常有用。AWS CLI 在某个时候，你会将编程 AWS SDK 调用添加到游戏后端服务中，作为游戏配对或游戏会话放置系统的一部分。

使用以下命令创建游戏会话：

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

您也可向服务器可执行文件传递自定义游戏属性。有关详细信息，请参阅自述文件中的“游戏服务器参数”部分。当它接到 `create-game-session` 呼叫时，会 Amazon GameLift Servers 通知包装器启动游戏服务器可执行文件并启动游戏会话。其中，`config.yaml` 中的配置影响游戏服务器的配置，而控制台中设置的启动参数则决定游戏会话本身的配置。

添加游戏属性的格式示例：

```
defaultArgs:
```

```
- arg: "--port"
  val: "{{.GamePort}}"
  pos: 0
- arg: "--ipAddress"
  val: "{{.IpAddress}}"
  pos: 1
- arg: "--gameSessionId"
  val: "{{.GameSessionId}}"
  pos: 2
```

第 7 步：管理和监控实例集

您的游戏服务器实例集已配置完成且游戏会话已启动，您可通过 Amazon GameLift Servers 控制台对其进行管理和监控。最优操作方式是访问实例集详细信息页面，您可以在其中编辑实例集详细信息或在扩展选项卡上更改实例集扩展和容量。有关如何扩展游戏服务器，请参阅以下部分。

选择指标选项卡可查看描述监控活动指标和硬件指标的图表。有关指标图表的详细信息，请选择您的实例集 ID 旁边的信息链接。此外，虽然您可以通过“指标”选项卡密切监控游戏服务器，但也可以在 CloudWatch 控制面板中为这些指标添加警报。

要从控制台导航到 CloudWatch 仪表板，请执行以下操作：

1. 在搜索栏中键入 CloudWatch “”，然后从搜索结果列表中选择它以显示 CloudWatch 概览。
2. 向下滚动并选择“查看 GameLift 控制面板”，查看基于舰队和游戏会话关键指标的图表。

第 8 步：扩展游戏服务器

下一步将设置自动扩缩。通过自动扩缩，实例集容量会根据游戏服务器的活动负载动态调整。当玩家加入并创建游戏会话时，自动扩缩会新增实例；当玩家需求减少时，自动扩缩会释放闲置实例。这是一种高效的优化方案，可在保障玩家流畅快速体验的前提下，最大程度地降低托管资源消耗和成本。

在游戏启动前，建议为实例集配置自动扩缩。自动扩缩是一种推荐的高效优化方案，可在保障玩家流畅快速体验的前提下，最大程度地降低托管资源消耗和成本。

手动设置实例集容量

手动设置实例集容量

1. 前往实例集详细信息页面中的扩展选项卡。

2. 选择一个位置并选择编辑。
3. 要突破当前配置值上限进行扩展，可更改所需实例数值，并调整最小和最大实例数设置，完成后选择确认。

Note

最大实例数设置可作为临时管控措施，防止扩展过度导致资源浪费与成本超支。

使用基于目标的自动扩缩

使用基于目标的自动扩缩

基于目标的自动扩缩（目标跟踪模式）将实例集扩展与可用游戏会话百分比关联。当玩家流量激增导致可用游戏会话减少时，系统会自动为实例集新增实例以应对需求。

1. 在基于目标的自动扩缩策略下，选择添加策略，然后将实例集的容量设置为在达到您设置的可用游戏会话百分比的阈值时自动更改。设置更大的缓冲空间可更好地应对流量峰值，保障新玩家快速进入游戏，但可能会增加托管成本。
2. 选择确认以接受更改。

基于规则的自动扩缩提供更精细的控制能力，例如能够将扩展与其他实例集指标关联起来，并设置自定义阈值和扩展响应。它提供功能强大的选项，但需使用 CLI 配置且需经过充分测试，以验证自定义规则的实际运行效果。本教程重点介绍如何首先设置基于目标的策略。

常见问题故障排除

以下是游戏服务器及游戏会话部署运行过程中可能遇到的常见问题。如果服务器或游戏会话运行异常，第一步是检查日志，日志内容可能会揭示新部署项目或生产环境游戏中出现的以下问题之一。

日志中常见问题及排查方向如下：

- 游戏服务器进程无法启动。这可能是封装器配置错误—请验证配置文件中的启动路径、启动参数及参数项是否正确。
- 游戏服务器生成包无法运行。大概率是游戏代码存在错误。
- 玩家无法连接到游戏会话。大概率是端口配置错误。
- 连接延迟或缓慢。请查看扩展策略和阈值设置。

- 完全无法建立连接。请验证实例集的端口规则和配置是否正确。

查看 Amazon GameLift Servers 实例集的事件日志

查看 Amazon GameLift Servers 实例集的事件日志

1. 打开 Amazon GameLift Servers 控制台。
2. 在实例集详细信息页面上，选择事件选项卡并下载日志。您还可以通过指标选项卡，实时监控游戏服务器运行状况、游戏会话激活情况等活动指标和硬件指标。

查看游戏会话日志

查看游戏会话日志

1. 通过控制台打开您的实例集并打开游戏会话选项卡。
2. 从列表中选择游戏会话 ID 即可显示其概述页面。
3. 选择下载日志，即可将日志文件下载至本地。

要通过 CLI 查看游戏会话日志，请使用 GetGameSessionLogURL API。Amazon GameLift Servers 会自动将日志存储 14 天。

您还可以为您的舰队配置 Amazon CloudWatch 日志。这提供了额外的日志记录功能以及与其他 AWS 监控服务的集成。

通过 CloudWatch 以下方式实时访问日志或延长保留期：

1. 在 Amazon GameLift Servers 控制台仪表板顶部的搜索栏中键入 CloudWatch “”，然后从结果下拉列表中进行选择。
2. 转到 CloudWatch 日志组并搜索特定会话。最简单的方法是单击“全部搜索”，然后使用 gameSessionId 或 clientID 进行筛选。

后续步骤

- [创建多位置实例集以在其他位置添加托管](#)
- [添加游戏会话队列，为跨多个位置的游戏会话提供最优放置位置](#)
- [通过为游戏创建 FlexMatch 匹配器和规则集来尝试配对](#)

- [开发游戏客户端和后端服务组件功能，支持玩家发起加入请求并直接连接到游戏会话](#)
- [准备就绪后，迁移至全集成式解决方案](#)

Amazon GameLift Servers 示例

如果您正在考虑使用 Amazon GameLift Servers 管理您的自定义游戏服务器，或者您有兴趣利用 Amazon GameLift Servers Realtime，我们建议您在将该服务用于自己的游戏之前尝试以下示例。自定义游戏服务器示例为您提供了在 Amazon GameLift Servers 控制台中托管游戏的体验。Amazon GameLift Servers Realtime 示例向您展示了如何使用 Realtime 服务器为托管游戏做好准备。

自定义游戏服务器示例

此示例演示了将示例游戏服务器部署到托管 Amazon GameLift Servers 管 EC2 队列进行托管的过程。请使用示例游戏客户端连接到实时游戏会话。您可以体验如何使用 Amazon GameLift Servers 工具（包括控制台和 AWS CLI）来监控实例集的托管性能和使用情况。

该示例将引导您完成以下步骤：

- 上传示例游戏服务器生成包。
- 创建实例集来运行游戏服务器生成包。
- 获取示例游戏客户端，用它来连接到游戏服务器并加入游戏会话。
- 查看实例集和游戏会话指标。

启动多个游戏客户端并玩游戏来生成托管数据。使用 Amazon GameLift Servers 控制台查看托管资源、跟踪指标并探索扩展实例集的托管容量的选项。

要开始使用，请登录 [Amazon GameLift Servers 控制台](#)。在左侧导航栏中，前往资源、试玩示例游戏。

使用 Amazon GameLift Servers 构建游戏托管解决方案

本节包含有关如何构建 Amazon GameLift Servers 游戏托管解决方案所有组件的指南，包括设计最佳实践。无论您是从零开始构建，还是迁移现有游戏，均可通过本指南全面了解如何利用 Amazon GameLift Servers 构建、实施和优化游戏托管基础设施。

托管解决方案组件

托管解决方案必须具备以下组件的某个版本。有关这些组件及其协作方式的详细信息，请参阅[使用 Amazon GameLift Servers 托管的工作原理](#)。

- 游戏客户端：运行于玩家设备。
- 后端服务：协调游戏客户端与 Amazon GameLift Servers 之间的通信。
- 游戏服务器软件：管理多人游戏流程。
- 放置系统：将玩家匹配至游戏服务器。
- 游戏托管实例集：在各地理区域提供计算资源。
- 游戏托管管理系统：监控游戏托管状态并管理容量。

主题

- [开发路线图](#)
- [准备使用 Amazon GameLift Servers 托管游戏](#)
- [为 Amazon GameLift Servers 构建后端服务](#)
- [为 Amazon GameLift Servers 部署托管实例集](#)
- [配置游戏会话放置](#)
- [自定义游戏托管解决方案](#)

开发路线图

开发路线图为构建不同类型的游戏托管解决方案提供了结构化的 step-by-step 指导。Amazon GameLift Servers 每个路线图都针对特定的托管方法量身定制，并引导您完成从初始设置到部署和测试的基本任务。

选择最符合您的托管要求和技术限制条件的路线图。您还可以遵循多个路线图，组合使用各种方法，以创建满足您特定需求的混合解决方案。

主题

- [借助 Amazon GameLift Servers 托管式 EC2 进行托管的开发路线图](#)
- [借助 Amazon GameLift Servers 托管容器进行托管的开发路线图](#)
- [借助 Amazon GameLift Servers Anywhere 进行托管的开发路线图](#)
- [借助 Amazon GameLift Servers 进行混合托管的开发路线图](#)

借助 Amazon GameLift Servers 托管式 EC2 进行托管的开发路线图

本路线图将指导您如何为多人游戏开发 Amazon GameLift Servers 托管式 EC2 托管解决方案。Amazon GameLift Servers 提供多个游戏托管选项；有关这些选项的更多信息，请参阅 [Amazon GameLift Servers 游戏托管方案](#)。

使用托管 Amazon GameLift Servers 托管，您的游戏服务器托管在 AWS Cloud 基于虚拟计算的资源上，这些资源根据您的配置 Amazon GameLift Servers 拥有和运行。您将获得 Amazon Elastic Compute Cloud (Amazon EC2) 实例的安全性、可靠性和全球可用性，这些实例针对多人游戏托管进行了进一步优化。Amazon GameLift Servers 利用自动服务器部署、生命周期处理和容量自动扩缩等工具简化了托管管理。

Amazon GameLift Servers 托管式解决方案由以下组件组成：

- 一个与 Amazon GameLift Servers 的服务器 SDK 集成的游戏服务器生成包，可在所有实例集中部署。
- 与 AWS SDK 集成的游戏客户端和后端服务，用于与 Amazon GameLift Servers 服务交互并请求游戏会话。
- 一个 Amazon GameLift Servers 队列或其他放置机制，用于放在所有实例集中都有可用游戏服务器的新游戏会话。
- (可选) 一个 FlexMatch 对战构建器，用于创建多人对战并为其设置游戏会话。
- 一个或多个 Amazon GameLift Servers 托管式实例集，使用针对多人游戏托管进行了优化的 Amazon Elastic Compute Cloud (Amazon EC2) 实例。

本路线图提供了一个简化的路径，让您的多人游戏可以通过 Amazon GameLift Servers 托管式 EC2 托管成功启动并运行。准备好必要的组件后，您就可以继续迭代游戏开发并自定义您的托管解决方案。临近发布时，请参阅 [借助 Amazon GameLift Servers 托管服务，为游戏发布做好准备](#)，帮助您为生产级使用准备托管解决方案。

i 借助适用于 Unreal Engine 和 Unity 的 Amazon GameLift Servers 插件快速上手开发为实现更快速的部署，请尝试使用适用于 Unreal Engine 和 Unity 的 [Amazon GameLift Servers 插件](#)。它提供了引导式 UI 工作流程，可通过最少的设置快速部署游戏服务器，助您即时测试游戏组件的运行效果。后续可在此基础上扩展，为游戏创建自定义托管方案。有关更多详细信息，请参阅[借助 Amazon GameLift Servers 插件完成 Unreal 或 Unity 游戏的准备工作](#)。

步骤 1：让游戏服务器准备好使用 Amazon GameLift Servers

为您的游戏服务器添加功能，使其在为托管进行部署时能与 Amazon GameLift Servers 服务进行通信。

- 为您的游戏项目获取 Amazon GameLift Servers (版本 5.x) 的服务器 SDK。服务器 SDK 支持 C++、C# 和 Go 语言。[下载 Amazon GameLift Servers 服务器 SDK](#)。
- 修改游戏服务器代码以添加服务器 SDK 功能。有关指南，请参阅[准备使用 Amazon GameLift Servers 托管游戏](#)。至少执行以下操作：
 - 添加代码以初始化 Amazon GameLift Servers SDK 并与 Amazon GameLift Servers 服务建立 WebSocket 连接。使用服务器 SDK 操作 `InitSdk()`。
 - 添加代码，以在服务器进程准备好托管游戏会话时向 Amazon GameLift Servers 服务报告。使用服务器 SDK 操作 `ProcessReady()`。
 - 实现所需的回调函数 `OnProcessTerminate()` 和 `OnStartGameSession()`。借助这些函数，游戏服务器进程可以保持与 Amazon GameLift Servers 服务的连接、在 Amazon GameLift Servers 提示时启动游戏会话，以及响应结束游戏服务器进程的提示。
 - 添加代码，以在服务器进程结束游戏会话时向 Amazon GameLift Servers 服务报告。使用服务器 SDK 操作 `ProcessEnding()`。
- 打包游戏服务器生成包。使用生成包文件、依赖项和支持软件创建安装脚本。请参阅[打包游戏生成包文件](#)。我们建议使用 Amazon Simple Storage Service (Amazon S3) 存储桶来存储游戏生成包的版本。
- 测试游戏服务器集成。对于这项任务，我们建议为本地工作站设置 Amazon GameLift Servers Anywhere 实例集，如[使用 Amazon GameLift Servers Anywhere 设置本地测试](#)中所述。在此步骤中，手动将游戏服务器生成包安装到测试设备上，然后启动服务器进程。使用 AWS CLI 请求一个新的游戏会话，并验证 Amazon GameLift Servers 服务是否成功提示您的服务器进程启动游戏会话。

步骤 2：让游戏客户端准备好加入托管的游戏会话

为您的游戏客户端创建一种方法，使其能够请求加入游戏会话、获取连接信息，然后直接连接到托管的游戏会话。最常见的方法是设置后端服务功能，作为游戏客户端和 Amazon GameLift Servers 服务之间的桥梁。此方法可以保护您的托管资源，并让您更好地控制玩家进入游戏会话的方式。

- 构建用于托管的后端服务功能。后端服务会与 Amazon GameLift Servers 服务通信并向游戏客户端提供连接信息。此功能包括启动游戏会话、将玩家放入游戏以及检索游戏会话信息。有关指南，请参阅[准备使用 Amazon GameLift Servers 托管游戏](#)。至少执行以下操作：
 - 获取 AWS 适用的 SDK Amazon GameLift Servers 并将其添加到您的后端服务项目中。请参阅[面向客户端服务的 Amazon GameLift Servers SDK 资源](#)。
 - 添加代码以初始化 Amazon GameLift Servers 客户端并存储键设置。请参阅[设置 Amazon GameLift Servers API](#)。
 - 添加调用 AWS SDK 操作 `CreateGameSession()` 并向游戏客户端提供游戏会话连接信息的功能。请参阅 [Create a game session on a specific fleet](#)。

调用 `CreateGameSession()` 是请求新游戏会话的一个方便的入手点。在部署游戏会话放置系统（请参阅“步骤 3”）后，您需要将此代码替换为对 `StartGameSessionPlacement()`（如果您使用的是 FlexMatch，则为 `StartMatchmaking()`）的调用。

有关设计后端服务的指导，请参阅[为 Amazon GameLift Servers 构建后端服务](#)。

- 向游戏客户端添加允许玩家加入托管游戏会话的功能。游戏客户端将向您的后端服务发出请求，而不是直接向 Amazon GameLift Servers 发出请求。在后端服务提供游戏会话连接信息后，游戏客户端会直接连接到游戏会话来玩游戏。
- 测试游戏客户端集成。您可将同一 Amazon GameLift Servers Anywhere 实例集与本地工作站搭配使用以进行测试。

对于快速迭代开发或与多人团队合作，我们建议您设置[基于云的测试环境](#)。此 Amazon GameLift Servers Toolkit 解决方案可模拟 Amazon GameLift Servers 托管式实例集的行为，而且让您能够以最短的周转时间更新游戏服务器生成包。

步骤 3：设置游戏会话放置

您可自定义 Amazon GameLift Servers 处理新游戏会话请求的方式，并找到可用的游戏服务器来托管这些请求。Amazon GameLift Servers 会自动跟踪所有实例集上的所有游戏服务器的可用性。当游戏客户端发送加入游戏会话的请求时，Amazon GameLift Servers 会根据一组已定义的优先级（如最低延迟、成本和可用性）寻找“最佳”放置。

- 创建游戏会话队列，用于将新的游戏会话与可用的游戏服务器一起放置。队列是游戏会话放置的主要机制。有关指南，请参阅[创建游戏会话队列](#)。
- 至少在队列中将您的 Anywhere 实例集添加为目标。所有其他设置都是可选自定义项。
- 在后端服务代码中，将 `CreateGameSession()` 调用转换为 `StartGameSessionPlacement()`。请参阅 [Create a game session in a multi-location queue](#)。
- 创建一个机制，用于在游戏会话准备好加入时通知游戏客户端。在开发过程中，你可以通过致电来轮询游戏会话状态 `DescribeGameSessionPlacement`。但是，在使用队列处理大量数据之前，您需要启用事件通知。请参阅[请参阅设置游戏会话置放通知](#)。
- （可选）添加 FlexMatch 对战组件。有关指导，请参阅《[Amazon GameLift Servers FlexMatch 开发人员指南](#)》。

步骤 4：创建基于云的托管式实例集

到目前为止，您已经使用自管理式 Anywhere 实例集来测试和迭代您的游戏组件，并且您已经微调了您的游戏会话放置。您的解决方案的最后一部分是设置生产系统所需的托管资源类型。要开始规划和配置生产环境，您需要过渡到使用 Amazon GameLift Servers 托管式实例集。

- 打包游戏服务器生成包并上传到 Amazon GameLift Servers。使用生成包文件、依赖项和支持软件创建安装脚本。请参阅[为 Amazon GameLift Servers 创建游戏服务器生成包](#)。您可以使用控制台或 AWS CLI 将您的版本上传到 Amazon GameLift Servers。

在上传您的版本之前，请确定 AWS 区域 要创建队列的内容。您必须将生成包上传到同一区域。有关选择实例集位置的更多信息，请参阅[地理位置](#)。

- 创建托管式 EC2 实例集。创建实例集时，Amazon GameLift Servers 会立即开始部署游戏服务器生成包进行托管。您可以配置托管式实例集的许多方面。有关指南，请参阅[创建 Amazon GameLift Servers 托管 EC2 舰队](#)。至少执行以下操作：
 - 为实例集命名并指定要部署的已上传游戏生成包。
 - 为实例集选择按需型实例，并选择在实例集所在位置可用的实例类型。竞价型实例集是一个很有价值的选择，但需要额外的设计和配置。
 - 为实例集创建运行时配置。至少为游戏服务器可执行文件指定启动路径。
 - 指定端口设置以允许入站流量访问游戏服务器。
- 将托管式实例集添加到队列。在游戏会话队列中，将 Anywhere 实例集替换为您的托管式实例集。
- 使用托管式实例集测试游戏托管。此时，您应该能够测试整个托管周期，即游戏客户端请求游戏会话、获取连接信息和成功连接到游戏会话。

步骤 5：自定义托管式实例集

在为游戏发布做准备时，您需要对托管式托管资源进行微调。需要考虑的决策包括：

- 考虑添加竞价型实例集以节省成本。请参阅[借助竞价型实例集降低游戏托管成本](#)。
- 如果您的游戏服务器需要通信其他 AWS 资源，请设置 IAM 角色来管理访问权限。请参阅[将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。
- 确定要将游戏服务器放置在哪个地理位置。将偏远位置添加到您的托管式实例集。请参阅[托管资源自定义](#)。
- 选择实例类型和大小，并配置运行时以运行多个服务器进程，从而优化实例集性能。请参阅[在托管式 Amazon GameLift Servers 上优化游戏服务器运行时配置](#)。
- 试验适用于托管式实例集的游戏会话放置选项，包括自定义优先级设置。请参阅[自定义游戏会话队列](#)。
- 设置自动容量扩展以满足预期的玩家需求。请参阅[利用 Amazon GameLift Servers 扩展游戏托管容量](#)。
- 在其他队列中设置备用队列，AWS 区域 并在需要时修改队列和 auto Scaling 以处理故障转移。
- 设置托管可观测性工具，包括分析和日志记录。请参阅[监控 Amazon GameLift Servers](#)。
- 使用[基础设施即代码 \(IaC\)](#) 自动执行部署。请参阅[使用管理 Amazon GameLift Servers 托管资源 CloudFormation](#)。

Amazon GameLift Servers 支持将 AWS CloudFormation 模板用于任何特定于部署的配置。您也可以使用 AWS Cloud Development Kit (AWS CDK) 来定义您的 Amazon GameLift Servers 资源。有关更多信息 AWS CDK，请参阅《[AWS Cloud Development Kit \(AWS CDK\) 开发人员指南](#)》。

要管理 CloudFormation 堆栈的部署，我们建议使用持续集成和持续交付 (CI/CD) 工具和服务，例如。AWS CodePipeline 这些工具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。使用 CI/CD 工具或服务，新游戏服务器版本的资源部署可能如下所示：

- 构建和测试游戏服务器二进制文件。
- 将二进制文件上传到 Amazon GameLift Servers。
- 部署具有新生成包的新实例集。
- 将新实例集添加到游戏会话队列，并删除具有先前生成包版本的实例集。
- 当使用先前版本的舰队不再托管活跃的游戏会话时，请删除这些舰队 CloudFormation 堆栈。

借助 Amazon GameLift Servers 托管容器进行托管的开发路线图

本路线图将指导您如何为容器化游戏服务器开发 Amazon GameLift Servers 托管式托管解决方案。托管容器只是 Amazon GameLift Servers 提供的一种托管解决方案。有关托管选项的更多信息，请参阅[Amazon GameLift Servers 游戏托管方案](#)。

基于 Amazon GameLift Servers 的托管容器解决方案包含下列组件：

- 一个包含游戏服务器生成包的容器映像，需上传至 Amazon Elastic Container Registry (Amazon ECR) 私有存储库。该游戏服务器生成包需与 Amazon GameLift Servers 的服务器 SDK 集成，且需构建为在 Linux 上运行。
- 一项后端服务，代表您的游戏客户端与 Amazon GameLift Servers 服务进行交互。该后端服务需使用 Amazon GameLift Servers 的服务 API 中的功能，它是 AWS SDK 的一部分。
- 一个 Amazon GameLift Servers 游戏会话队列或其他放置机制，负责处理新游戏会话的创建请求，在所有实例集中搜索可用的游戏服务器，并提示游戏服务器启动游戏会话。
- (可选) 一个 FlexMatch 对战构建器，用于创建多人对战并为其设置游戏会话。
- 一个或多个容器实例集，使用针对多人游戏托管进行了优化的 Amazon Elastic Compute Cloud (Amazon EC2) 实例。

本路线图提供了一个简化的路径，让容器化游戏服务器可以通过 Amazon GameLift Servers 托管容器成功启动并运行。准备好必要的组件后，您就可以继续迭代游戏开发并自定义您的托管解决方案。临近发布时，请参阅[借助 Amazon GameLift Servers 托管服务，为游戏发布做好准备](#)，帮助您为生产级使用准备托管解决方案。

借助以下托管容器工具加快入门速度：

- [容器入门工具包](#)可简化集成和实例集设置。它为游戏服务器添加核心游戏会话管理功能，并通过预配置的模板构建容器实例集和游戏服务器的自动化部署管道。部署完成后，可通过 Amazon GameLift Servers 控制台和 API 工具监控实例集性能、管理游戏会话并分析指标。
- 对于 Unreal Engine 和 Unity 开发人员，可使用 [Amazon GameLift Servers 插件](#)在游戏引擎的开发环境内集成游戏服务器并构建容器实例集。该插件提供的引导式工作流程可帮助您借助托管容器创建基于云的快速、简单的托管解决方案。后续可在此基础上扩展，为游戏创建自定义托管方案。

步骤 1：让游戏服务器准备好使用 Amazon GameLift Servers

为您的游戏服务器添加功能，使其在为托管进行部署时能与 Amazon GameLift Servers 服务进行通信。

- 为您的游戏项目获取 Amazon GameLift Servers (5.2 或更高版本) 的服务器 SDK。服务器 SDK 支持 C++、C# 和 Go 语言。[下载 Amazon GameLift Servers 的服务器 SDK](#)。服务器 SDK 支持 C++、C# 和 Go 语言。
- 修改游戏服务器代码以添加服务器 SDK 功能。有关指南，请参阅[准备使用 Amazon GameLift Servers 托管游戏](#)。至少执行以下操作：
 - 添加代码以初始化 Amazon GameLift Servers SDK 并与 Amazon GameLift Servers 服务建立 WebSocket 连接。使用服务器 SDK 操作 `InitSdk()`。
 - 添加代码，以在服务器进程准备好托管游戏会话时向 Amazon GameLift Servers 服务报告。使用服务器 SDK 操作 `ProcessReady()`。
 - 实现所需的回调函数 `OnStartGameSession()` 和 `OnProcessTerminate()`。借助这些函数，游戏服务器进程可以保持与 Amazon GameLift Servers 服务的连接、在 Amazon GameLift Servers 提示时启动游戏会话，以及响应结束游戏服务器进程的请求。
 - 添加代码，以在服务器进程结束游戏会话时向 Amazon GameLift Servers 服务报告。使用服务器 SDK 操作 `ProcessEnding()`。
- 打包游戏服务器生成包。构建您的游戏服务器以在 Linux 上运行。准备运行游戏服务器所需的生成包和其他文件。如果您要在 Windows 上进行开发，则此步骤可能涉及设置单独的 Linux WorkSpace 或使用适用于 Linux 的 Windows 子系统 (WSL) 等工具。您需要一种 Linux 环境来测试游戏服务器生成包，以及构建和测试容器映像。
- 测试游戏服务器集成。确认您的集成游戏服务器可以连接到 Amazon GameLift Servers 服务并对提示做出响应。我们建议设置一个简单的 Amazon GameLift Servers Anywhere 实例集，将本地工作站作为测试主机，如[使用 Amazon GameLift Servers Anywhere 设置本地测试](#)所述。将游戏服务器生成包安装到测试主机上，然后启动服务器进程。使用 AWS CLI 请求新的游戏会话，并验证 Amazon GameLift Servers 服务是否成功提示您的服务器进程启动游戏会话。

步骤 2：创建游戏服务器容器映像

成功集成游戏服务器后，使用游戏服务器可执行文件创建容器映像。将映像存储到 Amazon Elastic Container Registry (Amazon ECR) 存储库中以与 Amazon GameLift Servers 搭配使用。有关详细说明，请参阅[为 Amazon GameLift Servers 构建容器映像](#)。

- 获取游戏服务器容器的 Dockerfile 模板（由 Amazon GameLift Servers 提供）。修改游戏服务器生成包文件的文件。
- 构建游戏服务器容器映像。在 Linux 环境中操作时，使用 Docker 工具创建您的映像。
- 将容器映像推送到 Amazon ECR。在 Amazon ECR 中创建公共或私有存储库，使用该存储库 AWS 账户以及您计划部署容器队列 AWS 区域的位置。随后将容器映像推送至该存储库。
- 使用 Anywhere 实例集测试您的容器映像（可选）。在将容器映像部署到云托管的容器实例集之前，您可能需要在本地对它进行测试。您可将现有的 Amazon GameLift Servers Anywhere 实例集与本地工作站搭配使用以进行测试。安装并运行游戏服务器容器后，需验证以下两项内容：（1）Amazon GameLift Servers 服务是否成功提示您的服务器进程启动游戏会话；（2）游戏客户端能否连接到该游戏会话。

步骤 3：创建基于云的容器实例集

到目前为止，您已经使用自管理式 Anywhere 实例集来测试和迭代游戏组件。当您拥有可正常运行且已集成 Amazon GameLift Servers 的游戏服务器生成包后，即可开始设置基于云的 Amazon GameLift Servers 托管式容器实例集托管资源，这些资源是您搭建生产环境所需的核心托管资源。

- 创建容器组定义。容器组定义用于描述实例集的容器架构，并确定要部署哪些容器映像。请参阅[为 Amazon GameLift Servers 容器实例集创建容器组定义](#)。在存储容器镜像的相同 AWS 区域位置创建容器组定义。有关选择实例集位置的更多信息，请参阅[地理位置](#)。至少执行以下操作：
 - 创建游戏服务器容器组定义。
 - 在游戏服务器生成包中添加带有容器映像的容器定义。
 - 为容器的游戏服务器进程配置端口范围。
- 创建托管式容器实例集。创建实例集时，Amazon GameLift Servers 会立即开始部署游戏服务器生成包进行托管。您可以配置托管式实例集的许多方面。有关指南，请参阅[创建 Amazon GameLift Servers 托管式容器实例集](#)。至少执行以下操作：
 - 为容器舰队设置 AWS Identity and Access Management (IAM) 服务角色。请参阅[为 Amazon GameLift Servers 设置 IAM 服务角色](#)。
 - 指定要部署到实例集实例的游戏服务器容器组定义。
 - 对于所有其他参数，如有可用默认值，可直接使用。Amazon GameLift Servers 会计算部分参数以实现最优配置。
- 将容器实例集添加到队列。在游戏会话队列中，将 Anywhere 测试实例集替换为添加您的托管式容器实例集。
- 使用容器实例集测试游戏托管。此时您应能对整个解决方案进行测试。启动游戏客户端，并通过后端服务请求游戏会话。获取连接信息后，连接至容器实例集上的游戏会话。

- 迭代实例集部署。您可以更新容器组定义和实例集配置，然后将更新部署到现有实例集中。

第 4 步：准备好游戏客户端以加入托管的游戏会话

为您的游戏客户端创建一种方法，使其能够请求加入游戏会话、获取连接信息，然后直接连接到托管的游戏会话。最常见的方法是设置后端服务功能，作为游戏客户端和 Amazon GameLift Servers 服务之间的桥梁。此方法可以保护您的托管资源，并让您更好地控制玩家进入游戏会话的方式。

- 构建用于托管的后端服务功能。后端服务会与 Amazon GameLift Servers 服务通信并向游戏客户端提供连接信息。此功能包括启动游戏会话、将玩家放入游戏以及检索游戏会话信息。有关指南，请参阅[准备使用 Amazon GameLift Servers 托管游戏](#)。至少执行以下操作：
 - 获取 AWS 适用的 SDK Amazon GameLift Servers 并将其添加到您的后端服务项目中。请参阅[面向客户端服务的 Amazon GameLift Servers SDK 资源](#)。
 - 添加代码以初始化 Amazon GameLift Servers 客户端并存储键设置。请参阅[设置 Amazon GameLift Servers API](#)。
 - 添加调用 AWS SDK 操作 `CreateGameSession()` 并向游戏客户端提供游戏会话连接信息的功能。请参阅 [Create a game session on a specific fleet](#)。

调用 `CreateGameSession()` 是请求新游戏会话的一个方便的入手点。在部署游戏会话放置系统（请参阅“步骤 3”）后，您需要将此代码替换为对 `StartGameSessionPlacement()`（如果您使用的是 FlexMatch，则为 `StartMatchmaking()`）的调用。

有关设计后端服务的指导，请参阅[为 Amazon GameLift Servers 构建后端服务](#)。

- 向游戏客户端添加允许玩家加入托管游戏会话的功能。游戏客户端将向您的后端服务发出请求，而不是直接向 Amazon GameLift Servers 发出请求。在后端服务提供游戏会话连接信息后，游戏客户端会直接连接到游戏会话来玩游戏。
- 测试游戏客户端集成。您可将现有的 Amazon GameLift Servers Anywhere 实例集与本地工作站搭配使用以进行测试。使用新的后端服务请求一个新的游戏会话，并验证以下两项内容：（1）Amazon GameLift Servers 服务是否成功提示您的服务器进程启动游戏会话；（2）游戏客户端能否连接到该游戏会话。

第 5 步：设置游戏会话位置

您可自定义 Amazon GameLift Servers 处理新游戏会话请求的方式，并找到可用的游戏服务器来托管这些请求。Amazon GameLift Servers 会自动跟踪所有实例集上的所有游戏服务器的可用性。当游戏客户端发送加入游戏会话的请求时，Amazon GameLift Servers 会根据一组已定义的优先级（如最低延迟、成本和可用性）寻找“最佳”放置。

- 创建游戏会话队列，用于将新的游戏会话与可用的游戏服务器一起放置。队列是游戏会话放置的主要机制。有关指南，请参阅[创建游戏会话队列](#)。
- 至少在队列中将您的 Anywhere 实例集添加为目标。所有其他设置都是可选自定义项。
- 在后端服务代码中，将 `CreateGameSession()` 调用转换为 `StartGameSessionPlacement()`。请参阅 [Create a game session in a multi-location queue](#)。
- 创建一个机制，用于在游戏会话准备好加入时通知游戏客户端。在开发过程中，您可以使用对 `DescribeGameSessionPlacement` 的调用来轮询游戏会话状态。但是，在使用队列处理大量数据之前，您需要启用事件通知。请参阅[请参阅设置游戏会话置放通知](#)。
- 添加 FlexMatch 对战（可选）。构建对战规则集并创建对战配置以与您的游戏会话队列协同工作。有关设置对战系统的指导，请参阅《[Amazon GameLift Servers FlexMatch 开发人员指南](#)》。
- 测试放置系统。您可将现有的 Amazon GameLift Servers Anywhere 实例集与本地工作站搭配使用以进行测试。使用后端服务请求一个新的游戏会话，并验证 Amazon GameLift Servers 服务是否成功提示您的服务器进程启动游戏会话。

步骤 6：自定义托管容器解决方案

在为游戏发布做准备时，您需要对托管式托管资源进行微调。需要考虑的决策包括：

- 优化您的容器实例集配置。请参阅[自定义 Amazon GameLift Servers 容器实例集](#)。
- 考虑添加竞价型实例集以节省成本。请参阅[借助竞价型实例集降低游戏托管成本](#)。
- 如果您的游戏服务器需要通信其他 AWS 资源，请设置 IAM 角色来管理访问权限。请参阅[将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。
- 确定要将游戏服务器放置在哪个地理位置。将偏远位置添加到您的托管式实例集。请参阅[托管资源自定义](#)。
- 试验适用于托管式实例集的游戏会话放置选项，包括自定义优先级设置。请参阅[自定义游戏会话队列](#)。
- 设置自动容量扩展以满足预期的玩家需求。请参阅[利用 Amazon GameLift Servers 扩展游戏托管容量](#)。
- 在其他队列中创建队列，AWS 区域 并根据需要修改队列和 auto Scaling 以处理故障转移。
- 设置托管可观测性工具，包括分析和日志记录。请参阅[监控 Amazon GameLift Servers](#)。
- 使用[基础设施即代码 \(IaC \)](#) 自动执行实例集部署。请参阅[使用管理 Amazon GameLift Servers 托管资源 CloudFormation](#)。

Amazon GameLift Servers 支持将 AWS CloudFormation 模板用于任何特定于部署的配置。您也可以使用 AWS Cloud Development Kit (AWS CDK) 来定义您的 Amazon GameLift Servers 资源。有关更多信息 AWS CDK，请参阅 [《AWS Cloud Development Kit \(AWS CDK\) 开发人员指南》](#)。

要管理 CloudFormation 堆栈的部署，我们建议使用持续集成和持续交付 (CI/CD) 工具和服务，例如。AWS CodePipeline 这些工具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。使用 CI/CD 工具或服务，新游戏服务器版本的资源部署可能如下所示：

- 构建和测试游戏服务器二进制文件。
- 将二进制文件上传到 Amazon GameLift Servers。
- 部署具有新生成包的新实例集。
- 将新实例集添加到游戏会话队列，并删除具有先前生成包版本的实例集。
- 当使用先前版本的舰队不再托管活跃的游戏会话时，请删除这些舰队 CloudFormation 堆栈。

借助 Amazon GameLift Servers Anywhere 进行托管的开发路线图

本路线图将指导您如何为多人游戏开发托管解决方案，以与您自己的资源（本地硬件或虚拟机）搭配使用。Amazon GameLift Servers 提供多个游戏托管选项；有关这些选项的更多信息，请参阅 [Amazon GameLift Servers 游戏托管方案](#)。

使用 Amazon GameLift Servers Anywhere 托管时，您的游戏服务器将托管在您提供和管理的计算资源上。您可以创建一个具有所需配置且地理位置随玩家所在位置而定的 Anywhere 实例集。Amazon GameLift Servers 为 Anywhere 实例集提供以下功能：

- 根据您的配置为您处理游戏会话放置流程，包括：
 - 跟踪 Anywhere 实例集中游戏服务器的可用性。
 - 处理来自游戏客户端服务的游戏请求，并将游戏请求与可用服务器进行匹配。
 - 提示 Anywhere 实例集上的游戏服务器启动游戏会话。
 - 将连接详细信息传回游戏客户端。
- 收集会话放置流程的性能指标以及游戏会话和玩家的使用情况指标。
- 支持完整的 FlexMatch 对战功能集，使您可以构建对战构建器并将其与游戏会话放置系统集成。
- 提供 Amazon GameLift Servers 代理，用于处理 Anywhere 实例集上的关键主机管理任务。
- 支持与 Amazon GameLift Servers 托管式实例集相结合，打造灵活的混合解决方案。

Amazon GameLift Servers Anywhere 解决方案由以下组件组成：

- 一个与 Amazon GameLift Servers 的服务器 SDK 集成的游戏服务器生成包，可在所有实例集中部署。
- 与 AWS SDK 集成的游戏客户端和后端服务，用于与 Amazon GameLift Servers 服务交互并请求游戏会话。
- 一个 Amazon GameLift Servers 队列或其他放置机制，用于放在所有实例集中都有可用游戏服务器的新游戏会话。
- (可选) 一个 FlexMatch 对战构建器，用于创建多人对战并为其设置游戏会话。
- 一个或多个具有您的本地或其他托管资源的 Amazon GameLift Servers Anywhere 实例集，使用您现有的配置管理和部署工具进行管理。(您可以选择使用 AWS Systems Manager。)

本路线图提供了一个简化的路径，让您的多人游戏可以通过 Amazon GameLift Servers Anywhere 托管成功启动并运行。准备好必要的组件后，您就可以继续迭代游戏开发并自定义您的托管解决方案。临近发布时，请参阅[借助 Amazon GameLift Servers 托管服务，为游戏发布做好准备](#)，帮助您为生产级使用准备托管解决方案。

 借助适用于 Unreal Engine 和 Unity 的 Amazon GameLift Servers 插件快速上手开发
为实现更快速的部署，请尝试使用适用于 Unreal Engine 和 Unity 的 [Amazon GameLift Servers 插件](#)。它提供了引导式 UI 工作流程，可通过最少的设置快速部署游戏服务器，助您即时测试游戏组件的运行效果。后续可在此基础上扩展，为游戏创建自定义托管方案。有关更多详细信息，请参阅[借助 Amazon GameLift Servers 插件完成 Unreal 或 Unity 游戏的准备工作](#)。

步骤 1：让游戏服务器准备好使用 Amazon GameLift Servers

为您的游戏服务器添加功能，使其在为托管进行部署时能与 Amazon GameLift Servers 服务进行通信。

- 为您的游戏项目获取 Amazon GameLift Servers (版本 5.x) 的服务器 SDK。服务器 SDK 支持 C++、C# 和 Go 语言。[下载 Amazon GameLift Servers 服务器 SDK](#)。
- 修改游戏服务器代码以添加服务器 SDK 功能。有关指南，请参阅[准备使用 Amazon GameLift Servers 托管游戏](#)。至少执行以下操作：
 - 添加代码以初始化 Amazon GameLift Servers SDK 并与 Amazon GameLift Servers 服务建立 WebSocket 连接。使用服务器 SDK 操作 `InitSdk()` 并加入 Anywhere 实例集所需的服务器参数。

- 添加代码，以在服务器进程准备好托管游戏会话时向 Amazon GameLift Servers 服务报告。使用服务器 SDK 操作 `ProcessReady()`。
- 实现所需的回调函数 `OnProcessTerminate()` 和 `OnStartGameSession()`。借助这些函数，游戏服务器进程可以保持与 Amazon GameLift Servers 服务的连接、在 Amazon GameLift Servers 提示时启动游戏会话，以及响应结束游戏服务器进程的提示。
- 添加代码，以在服务器进程结束游戏会话时向 Amazon GameLift Servers 服务报告。使用服务器 SDK 操作 `ProcessEnding()`。
- 打包游戏服务器生成包。使用生成包文件、依赖项和支持软件创建安装脚本。请参阅[打包游戏生成包文件](#)。我们建议使用 Amazon Simple Storage Service (Amazon S3) 存储桶来存储游戏生成包的版本。
- 测试游戏服务器集成。对于这项任务，我们建议为本地工作站设置 Amazon GameLift Servers Anywhere 实例集，如[使用 Amazon GameLift Servers Anywhere 设置本地测试](#)中所述。在此步骤中，手动将游戏服务器生成包安装到测试设备上，然后启动服务器进程。使用 AWS CLI 请求一个新的游戏会话，并验证 Amazon GameLift Servers 服务是否成功提示您的服务器进程启动游戏会话。

步骤 2：让游戏客户端准备好加入托管的游戏会话

为您的游戏客户端创建一种方法，使其能够请求加入游戏会话、获取连接信息，然后直接连接到托管的游戏会话。最常见的方法是设置后端服务功能，作为游戏客户端和 Amazon GameLift Servers 服务之间的桥梁。此方法可以保护您的托管资源，并让您更好地控制玩家进入游戏会话的方式。

- 构建用于托管的后端服务功能。后端服务会与 Amazon GameLift Servers 服务通信并向游戏客户端提供连接信息。此功能包括启动游戏会话、将玩家放入游戏以及检索游戏会话信息。有关指南，请参阅[准备使用 Amazon GameLift Servers 托管游戏](#)。至少执行以下操作：
- 获取 AWS 适用的 SDK Amazon GameLift Servers 并将其添加到您的后端服务项目中。请参阅[面向客户端服务的 Amazon GameLift Servers SDK 资源](#)。
- 添加代码以初始化 Amazon GameLift Servers 客户端并存储键设置。请参阅[设置 Amazon GameLift Servers API](#)。
- 添加调用 AWS SDK 操作 `CreateGameSession()` 并向游戏客户端提供游戏会话连接信息的功能。请参阅 [Create a game session on a specific fleet](#)。

调用 `CreateGameSession()` 是请求新游戏会话的一个方便的入手点。在部署游戏会话放置系统（请参阅“步骤 3”）后，您需要将此代码替换为对 `StartGameSessionPlacement()`（如果您使用的是 FlexMatch，则为 `StartMatchmaking()`）的调用。

有关设计后端服务的指导，请参阅[为 Amazon GameLift Servers 构建后端服务](#)。

- 向游戏客户端添加允许玩家加入托管游戏会话的功能。游戏客户端将向您的后端服务发出请求，而不是直接向 Amazon GameLift Servers 发出请求。在后端服务提供游戏会话连接信息后，游戏客户端会直接连接到游戏会话来玩游戏。
- 测试游戏客户端集成。您可将同一 Amazon GameLift Servers Anywhere 实例集与本地工作站搭配使用以进行测试。

步骤 3：设置游戏会话放置

您可自定义 Amazon GameLift Servers 处理新游戏会话请求的方式，并找到可用的游戏服务器来托管这些请求。Amazon GameLift Servers 会自动跟踪所有实例集上的所有游戏服务器的可用性。当游戏客户端发送加入游戏会话的请求时，Amazon GameLift Servers 会根据一组已定义的优先级（如最低延迟、成本和可用性）寻找“最佳”放置。

- 创建游戏会话队列，用于将新的游戏会话与可用的游戏服务器一起放置。队列是游戏会话放置的主要机制。有关指南，请参阅[创建游戏会话队列](#)。
 - 至少在队列中将您的 Anywhere 实例集添加为目标。所有其他设置都是可选的。
- 在后端服务代码中，将 **CreateGameSession()** 调用转换为 **StartGameSessionPlacement()**。请参阅 [Create a game session in a multi-location queue](#)。
- 创建一个机制，用于在游戏会话准备好加入时通知游戏客户端。在开发过程中，你可以通过致电来轮询游戏会话状态 DescribeGameSessionPlacement。但是，在使用队列处理大量数据之前，您需要启用事件通知。请参阅[请参阅设置游戏会话置放通知](#)。
- （可选）添加 FlexMatch 对战组件。有关指导，请参阅《[Amazon GameLift Servers FlexMatch 开发人员指南](#)》。

步骤 4：设置 Anywhere 实例集和 Amazon GameLift Servers 代理

到目前为止，您一直在使用本地设备（注册为 Anywhere 实例集计算）来测试和迭代游戏组件。下一步是设置生产系统所需的实例集类型。对于这些资源，请使用 Amazon GameLift Servers 代理来管理一些关键的计算主机管理任务。有关更多详细信息，请参阅[使用 Amazon GameLift Servers 代理](#)。

- 获取 Amazon GameLift Servers 代理并将其添加到游戏服务器安装包。获取并构建代理源代码（可在 [Amazon GameLift Servers 代理 Github 存储库](#) 中找到）。将生成的 JAR 文件可执行文件放在与游戏生成包可执行文件相同的目录中。
- 根据需要修改代理的启动脚本。确保在计算开始运行后立即启动代理可执行文件。如需获取在托管计算上安装和运行代理方面的帮助，请参阅代理存储库中的自述文件。启动命令应至少包含用于指定 Anywhere 实例集 ID、AWS 区域、自定义位置和计算名称的选项。

代理会自动为您处理以下任务，因此，如果您一直使用脚本处理这些任务，则可以将脚本删除：

- 调用 `RegisterCompute()`，将计算添加到 Anywhere 实例集。
- 调用 `GetComputeAuthToken()`，在游戏服务器连接到 Amazon GameLift Servers 服务时对该服务器进行身份验证。代理负责管理身份验证令牌的获取和刷新，在计算上运行的所有游戏服务器进程都可以使用该令牌。
- 根据一组运行时指令在计算上启动新的服务器进程。
- 为 Anywhere 实例集中的计算创建运行时配置。您可以使用 Amazon GameLift Servers 控制台或 AWS CLI 来创建或修改队列的运行时指令。代理将执行这些指令并定期从 Amazon GameLift Servers 服务请求更新。
- 根据需要设置或修改游戏会话队列。创建新队列（或更新现有队列），以使用随 Amazon GameLift Servers 代理部署的 Anywhere 实例集。
- 测试代理与 Anywhere 实例集的集成。检查代理是否根据运行时配置正确启动服务器进程。

步骤 5：自定义 Anywhere 实例集

在为游戏发布做准备时，您需要对托管式托管资源进行微调。需要考虑的决策包括：

- 根据需要自动启动和关闭计算，包括安装和运行游戏服务器软件。回收计算有助于确保计算定期更新，而关闭不需要的计算则可以节约成本。
- 如果您的游戏服务器需要与其他 AWS 资源进行通信，请设置 IAM 角色来管理访问权限。请参阅[将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。
- 确定要将游戏服务器放置在哪个地理位置。将偏远位置添加到您的托管式实例集。请参阅[托管资源自定义](#)。
- 通过选择计算资源配置来优化实例集性能，然后配置运行时指令，使每个计算运行最优数量的服务器进程。
- 试验适用于托管式实例集的游戏会话放置选项，包括自定义优先级设置。请参阅[自定义游戏会话队列](#)。
- 创建处理手动或自动容量扩展的机制，以满足预期的玩家需求。考虑哪些因素会促使系统增加或减少可用于托管游戏会话的计算数量。
- 根据需要设计并实施到其他资源的失效转移。
- 设置托管可观测性工具，包括分析和日志记录。请参阅[监控 Amazon GameLift Servers](#)。

借助 Amazon GameLift Servers 进行混合托管的开发路线图

本路线图将指导您如何为多人游戏开发托管解决方案。Amazon GameLift Servers 提供多个游戏托管选项；有关这些选项的更多信息，请参阅 [Amazon GameLift Servers 游戏托管方案](#)。

混合解决方案使用托管资源的组合，包括由 Amazon GameLift Servers 管理的基于云的资源以及您自己的自管理式托管资源。有关混合托管的更详细讨论，请参阅以下文章：[借助 Amazon GameLift Servers Anywhere 实现混合游戏服务器托管](#)。借助 Amazon GameLift Servers，您可以设置一个使用通用组件和流程的混合解决方案，这样就能集中管理全球实例集，并在所有类型的资源之间轻松转移负载。

混合架构包含以下组件：

- 单个与 Amazon GameLift Servers 的服务器 SDK 集成的游戏服务器生成包，可在所有实例集中部署。
- 单个与 AWS SDK 集成的游戏客户端和后端服务，用于与 Amazon GameLift Servers 服务交互和请求游戏会话。
- 一个共享的 Amazon GameLift Servers 队列，用于放置在所有实例集中都有可用游戏服务器和均衡负载的新游戏会话。
- （可选）与 Anywhere 实例集一起部署的进程管理器代理，用于简化所有实例集中跨计算的服务器进程管理任务。
- （可选）一个 FlexMatch 对战构建器，用于创建多人对战并为其设置游戏会话。
- 一个或多个 Amazon GameLift Servers 托管式实例集，使用针对多人游戏托管进行了优化的 Amazon Elastic Compute Cloud (Amazon EC2) 实例。
- 一个或多个 Amazon GameLift Servers Anywhere 实例集，使用您现有的本地或其他托管资源，包括您的配置管理和部署工具。（您可以选择使用 AWS Systems Manager。）

本路线图提供了一个简化的路径，让您的多人游戏可以在使用 Amazon GameLift Servers 的混合托管解决方案中成功启动并运行。准备好必要的组件后，您就可以继续迭代游戏开发并自定义您的托管解决方案。临近发布时，请参阅[借助 Amazon GameLift Servers 托管服务，为游戏发布做好准备](#)，帮助您为生产级使用准备托管解决方案。

借助 Amazon GameLift Servers 插件快速上手开发

如果您正在使用 Unreal Engine 或 Unity 开发项目，请借助 Amazon GameLift Servers 插件开始设置游戏进行托管。使用该插件，您可以将 Amazon GameLift Servers SDKs 添加到您的游

戏项目中，并使用指导式工作流程来构建混合托管解决方案的简单工作版本，该解决方案包含 Anywhere 队列和托管 Amazon GameLift Servers 队列。然后，您可以在这些基础上进行构建和按需自定义。

步骤 1：让游戏服务器准备好使用 Amazon GameLift Servers

为您的游戏服务器添加功能，使其在为托管进行部署时能与 Amazon GameLift Servers 服务进行通信。如果游戏服务器在 Amazon GameLift Servers 托管式实例集或 Anywhere 实例集上运行，则也需要这项功能。

- 为您的游戏项目获取 Amazon GameLift Servers (版本 5.x) 的服务器 SDK。服务器 SDK 支持 C++、C# 和 Go 语言。[下载 Amazon GameLift Servers 的服务器 SDK](#)。
- 修改游戏服务器代码以添加服务器 SDK 功能。有关指南，请参阅[准备使用 Amazon GameLift Servers 托管游戏](#)。至少执行以下操作：
 - 添加代码以初始化 Amazon GameLift Servers SDK 并与 Amazon GameLift Servers 服务建立 WebSocket 连接。使用服务器 SDK 操作 `InitSdk()`。在 Anywhere 实例集计算上运行时，加入指定服务器参数的代码。
 - 添加代码，以在服务器进程准备好托管游戏会话时向 Amazon GameLift Servers 服务报告。使用服务器 SDK 操作 `ProcessReady()`。
 - 实现所需的回调函数 `OnProcessTerminate()` 和 `OnStartGameSession()`。借助这些函数，游戏服务器进程可以保持与 Amazon GameLift Servers 服务的连接、在 Amazon GameLift Servers 提示时启动游戏会话，以及响应结束游戏服务器进程的提示。
 - 添加代码，以在服务器进程结束游戏会话时向 Amazon GameLift Servers 服务报告。使用服务器 SDK 操作 `ProcessEnding()`。
- 打包游戏服务器生成包。使用生成包文件、依赖项和支持软件创建安装脚本。请参阅[打包游戏生成包文件](#)。我们建议使用 Amazon Simple Storage Service (Amazon S3) 存储桶来存储游戏生成包的版本。
- 测试游戏服务器集成。对于这项任务，我们建议为本地工作站设置 Amazon GameLift Servers Anywhere 实例集，如[使用 Amazon GameLift Servers Anywhere 设置本地测试](#)中所述。在此步骤中，手动将游戏服务器生成包安装到测试设备上，然后启动服务器进程。使用 AWS CLI 请求一个新的游戏会话，并验证 Amazon GameLift Servers 服务是否成功提示您的服务器进程启动游戏会话。

步骤 2：让游戏客户端准备好加入托管的游戏会话

为您的游戏客户端创建一种方法，使其能够请求加入游戏会话、获取连接信息，然后直接连接到托管的游戏会话。最常见的方法是设置后端服务功能，作为游戏客户端和 Amazon GameLift Servers 服务之间的桥梁。此方法可以保护您的托管资源，并让您更好地控制玩家进入游戏会话的方式。

- 构建用于托管的后端服务功能。后端服务会与 Amazon GameLift Servers 服务通信并向游戏客户端提供连接信息。此功能包括启动游戏会话、将玩家放入游戏以及检索游戏会话信息。有关指南，请参阅[准备使用 Amazon GameLift Servers 托管游戏](#)。至少执行以下操作：
 - 获取 AWS 适用的 SDK Amazon GameLift Servers 并将其添加到您的后端服务项目中。请参阅[面向客户端服务的 Amazon GameLift Servers SDK 资源](#)。
 - 添加代码以初始化 Amazon GameLift Servers 客户端并存储键设置。请参阅[设置 Amazon GameLift Servers API](#)。
 - 添加调用 AWS SDK 操作 `CreateGameSession()` 并向游戏客户端提供游戏会话连接信息的功能。请参阅 [Create a game session on a specific fleet](#)。

调用 `CreateGameSession()` 是请求新游戏会话的一个方便的入手点。在部署游戏会话放置系统（请参阅“步骤 3”）后，您需要将此代码替换为对 `StartGameSessionPlacement()`（如果您使用的是 FlexMatch，则为 `StartMatchmaking()`）的调用。

有关设计后端服务的指导，请参阅[为 Amazon GameLift Servers 构建后端服务](#)。

- 向游戏客户端添加允许玩家加入托管游戏会话的功能。游戏客户端将向您的后端服务发出请求，而不是直接向 Amazon GameLift Servers 发出请求。在后端服务提供游戏会话连接信息后，游戏客户端会直接连接到游戏会话来玩游戏。
- 测试游戏客户端集成。您可将同一 Amazon GameLift Servers Anywhere 实例集与本地工作站搭配使用以进行测试。

在开发阶段，如果您想测试您的游戏生成包在 Amazon GameLift Servers 托管式实例集中的行为方式，我们建议您还设置一个[基于云的测试环境](#)。此 Amazon GameLift Servers Toolkit 解决方案可模拟托管式实例集的行为，而且让您能够以最短的周转时间更新游戏服务器生成包。

步骤 3：设置游戏会话放置

您可自定义 Amazon GameLift Servers 处理新游戏会话请求的方式，并找到可用的游戏服务器来托管这些请求。Amazon GameLift Servers 会自动跟踪所有实例集上的所有游戏服务器的可用性。当游戏客户端发送加入游戏会话的请求时，Amazon GameLift Servers 会根据一组已定义的优先级（如最低延迟、成本和可用性）寻找“最佳”放置。

- 创建游戏会话队列，用于将新的游戏会话与可用的游戏服务器一起放置。队列是游戏会话放置的主要机制。有关指南，请参阅[创建游戏会话队列](#)。
- 至少在队列中将您的 Anywhere 实例集添加为目标。所有其他设置都是可选自定义项。
- 在后端服务代码中，将 `CreateGameSession()` 调用转换为 `StartGameSessionPlacement()`。请参阅 [Create a game session in a multi-location queue](#)。
- 创建一个机制，用于在游戏会话准备好加入时通知游戏客户端。在开发过程中，您可以使用对 `DescribeGameSessionPlacement` 的调用来轮询游戏会话状态。但是，在使用队列处理大量数据之前，您需要启用事件通知。请参阅[请参阅设置游戏会话置放通知](#)。
- (可选) 添加 FlexMatch 对战组件。有关指导，请参阅《[Amazon GameLift Servers FlexMatch 开发人员指南](#)》。

步骤 4：设置 Anywhere 实例集和 Amazon GameLift Servers 代理

到目前为止，您一直在使用本地设备（注册为 Anywhere 实例集计算）来测试和迭代游戏组件。下一步是设置生产系统所需的实例集类型。首先配置一个 Anywhere 实例集，然后添加 Amazon GameLift Servers 代理来管理一些关键的计算主机管理任务。有关更多详细信息，请参阅[使用 Amazon GameLift Servers 代理](#)。

- 获取 Amazon GameLift Servers 代理并将其添加到游戏服务器安装包。获取并构建代理源代码（可在 [Amazon GameLift Servers 代理 Github 存储库](#) 中找到）。将生成的 JAR 文件可执行文件放在与游戏生成包可执行文件相同的目录中。
- 根据需要修改代理的启动脚本。确保在计算开始运行后立即启动代理可执行文件。如需获取在托管计算上安装和运行代理方面的帮助，请参阅代理存储库中的自述文件。您的启动命令应包括至少指定 Anywhere 队列 ID AWS 区域、自定义位置和计算名称的选项。

代理会自动为您处理以下任务，因此，如果您一直使用脚本处理这些任务，则可以将脚本删除：

- 调用 `RegisterCompute()`，将计算添加到 Anywhere 实例集。
- 调用 `GetComputeAuthToken()`，在游戏服务器连接到 Amazon GameLift Servers 服务时对该服务器进行身份验证。代理负责管理身份验证令牌的获取和刷新，在计算上运行的所有游戏服务器进程都可以使用该令牌。
- 根据一组运行时指令在计算上启动新的服务器进程。
- 为 Anywhere 实例集中的计算创建运行时配置。至少为游戏服务器可执行文件指定启动路径。您可以使用 Amazon GameLift Servers 控制台或 AWS CLI 来创建或修改队列的运行时指令。代理将执行这些指令并定期从 Amazon GameLift Servers 服务请求更新。

- 根据需要设置或修改游戏会话队列。创建新队列（或更新现有队列），并为随 Amazon GameLift Servers 代理部署的 Anywhere 实例集指定目标。
- 测试代理与 Anywhere 实例集的集成。检查代理是否根据运行时配置正确启动服务器进程。

步骤 5：创建基于云的托管式实例集

创建 Amazon GameLift Servers 托管式 EC2 实例集，以补充您的 Anywhere 实例集。如果您在步骤 2 中设置了基于云的测试环境以加快开发速度，请计划在完成大部分游戏开发和测试后创建托管式实例集。您需要一个完全托管式实例集来配置和测试其他设置，如自动容量扩展。

- 打包游戏服务器生成包并上传到 Amazon GameLift Servers。使用生成包文件、依赖项和支持软件创建安装脚本。您可以对 Anywhere 和托管式实例集使用相同的生成包软件。请参阅[Amazon GameLift Servers 创建游戏服务器生成包](#)。您可以使用控制台或 AWS CLI 将您的版本上传到 Amazon GameLift Servers。

在上传生成包之前，请确定要在哪个 AWS 区域 中创建托管式实例集。您必须将生成包上传到同一区域。有关选择实例集位置的更多信息，请参阅[地理位置](#)。

- 创建托管式 EC2 实例集。您可以使用 Amazon GameLift Servers 控制台或 AWS CLI 创建托管队列。创建实例集时，Amazon GameLift Servers 会立即开始部署游戏服务器生成包进行托管。您可以配置托管式实例集的许多方面。有关指南，请参阅[创建 Amazon GameLift Servers 托管 EC2 舰队](#)。至少执行以下操作：
 - 为实例集命名并指定要部署的已上传游戏生成包。
 - 为实例集选择按需型实例，并选择在实例集所在位置可用的实例类型。竞价型实例集是一个很有价值的选择，但需要额外的设计和配置。
 - 使用与 Anywhere 实例集相似的设置创建运行时配置。至少为游戏服务器可执行文件指定启动路径。
 - 指定端口设置以允许入站流量访问游戏服务器。
- 将托管式实例集添加到共享游戏会话队列。更新步骤 4 中的队列，使其包含一些针对托管式实例集和随 Amazon GameLift Servers 代理部署的 Anywhere 实例集的目标。
- 使用托管式实例集测试游戏托管。此时，您应该能够测试整个托管周期，即游戏客户端请求游戏会话、获取连接信息和成功连接到游戏会话。

步骤 6：自定义实例集

在为游戏发布做准备时，您需要对托管解决方案进行微调。需要考虑的决策包括：

- 对于 Anywhere 实例集，根据需要自动启动和关闭计算，包括安装和运行游戏服务器软件。回收计算有助于确保计算定期更新，而关闭不需要的计算则可以节约成本。
- 如果您的游戏服务器需要与其他 AWS 资源通信，请设置 IAM 角色来管理访问。请参阅[将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。
- 确定要将游戏服务器放置在哪个地理位置。将偏远位置添加到您的托管式实例集。请参阅[托管资源自定义](#)。
- 对于托管式实例集，考虑使用竞价型实例集来节省成本。请参阅[借助竞价型实例集降低游戏托管成本](#)。
- 通过选择计算资源配置来优化实例集性能，然后配置运行时指令，使每个计算运行最优数量的服务器进程。请对 Anywhere 实例集和托管式实例集执行此操作。请参阅[在托管式 Amazon GameLift Servers 上优化游戏服务器运行时配置](#)。
- 试验适用于托管式实例集的游戏会话放置选项，包括自定义优先级设置。请参阅[自定义游戏会话队列](#)。
- 对于托管式实例集，设置自动容量扩展以满足预期的玩家需求。请参阅[利用 Amazon GameLift Servers 扩展游戏托管容量](#)。
- 对于 Anywhere 实例集，创建处理手动或自动容量扩展的机制，以满足预期的玩家需求。
- 根据需要设计并实施到其他资源的失效转移。在其他 AWS 区域 中设置备用实例集，并修改队列和自动扩缩，以便在需要时处理失效转移。
- 设置托管可观测性工具，包括分析和日志记录。请参阅[监控 Amazon GameLift Servers](#)。创建指标组以汇总所有托管资源的分析。
- 使用[基础设施即代码 \(IaC\)](#) 自动执行部署。请参阅[使用管理 Amazon GameLift Servers 托管资源 CloudFormation](#)。

Amazon GameLift Servers 支持将 AWS CloudFormation 模板用于任何特定于部署的配置。您也可以使用 AWS Cloud Development Kit (AWS CDK) 来定义您的 Amazon GameLift Servers 资源。有关更多信息 AWS CDK，请参阅《[AWS Cloud Development Kit \(AWS CDK\) 开发人员指南](#)》。

要管理 CloudFormation 堆栈的部署，我们建议使用持续集成和持续交付 (CI/CD) 工具和服务，例如。AWS CodePipeline 这些工具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。使用 CI/CD 工具或服务，新游戏服务器版本的资源部署可能如下所示：

- 构建和测试游戏服务器二进制文件。
- 将二进制文件上传到 Amazon GameLift Servers。
- 部署具有新生成包的新实例集。
- 将新实例集添加到游戏会话队列，并删除具有先前生成包版本的实例集。

- 当具有先前生成包的实例集不再托管活动游戏会话时，请删除这些实例集的 CloudFormation 堆栈。

准备使用 Amazon GameLift Servers 托管游戏

在将游戏部署到 Amazon GameLift Servers 进行托管之前，您需要准备游戏服务器和游戏客户端组件。这包括将游戏服务器与 Amazon GameLift Servers Server SDK 集成，以及构建用于处理玩家身份验证和游戏会话管理的后端服务。

Server SDK 集成使您的游戏服务器能够与 Amazon GameLift Servers 服务通信，报告其运行状况并管理游戏会话。后端服务充当游戏客户端和 Amazon GameLift Servers 服务之间的中介，处理玩家请求并协调游戏会话放置。

主题

- [借助 Amazon GameLift Servers 插件完成 Unreal 或 Unity 游戏的准备工作](#)
- [将游戏服务器与 Amazon GameLift Servers 集成](#)
- [打包游戏服务器生成包用于部署](#)
- [与游戏 client/server 互动 Amazon GameLift Servers](#)

借助 Amazon GameLift Servers 插件完成 Unreal 或 Unity 游戏的准备工作

Amazon GameLift Servers 插件是 Unreal 或 Unity 游戏引擎的全功能附加组件。它会指导您完成通过 Amazon GameLift Servers 部署游戏进行托管的基本步骤。利用该插件的工具集和 workflows，您可以在游戏引擎开发环境中工作，以便让游戏服务器做好托管准备、在本地计算机上设置托管以进行测试、创建简单的后端服务，以及将游戏服务器部署到基于云的托管式托管。

使用该插件可体验使用 Amazon GameLift Servers 以及快速启动并运行游戏托管解决方案。您可以使用示例游戏资产或您自己的游戏项目。该插件可自动执行多个步骤，让您可以快速构建简单高效的解决方案。完成插件的引导式 workflows 后，您将能够通过 Amazon GameLift Servers 将游戏客户端连接到托管的实时游戏会话。使用该插件创建简单的托管解决方案后，您就可以根据游戏需求自定义解决方案。

该插件适用于以下游戏引擎：

- Unreal Engine
- Unity

该插件包含每个游戏引擎的以下组件：

- 游戏引擎编辑器的插件模块。安装插件后，新的主菜单按钮可让您访问 Amazon GameLift Servers 的功能。
- 适用于具有客户端功能的 Amazon GameLift Servers 服务 API 的库。
- 适用于 Amazon GameLift Servers 服务器 SDK (版本 5) 的库。
- 用于测试服务器集成的示例资产。
- 以 CloudFormation 模板形式的可编辑配置，用于定义您的游戏服务器解决方案。

主题

- [插件工作流](#)
- [Amazon GameLift Servers 虚幻引擎插件](#)
- [Amazon GameLift Servers 适用于 Unity 的插件 \(服务器 SDK 5.x \)](#)
- [适用于 Unity 的 Amazon GameLift Servers 插件 \(服务器 SDK 4 \)](#)

插件工作流

以下步骤描述了在 Amazon GameLift Servers 上准备和部署游戏项目的典型路径。您可以通过游戏引擎编辑器和游戏代码来完成这些步骤。

1. 创建用户个人资料，链接到您的 AWS 账户用户，并向访问凭证提供使用权限 Amazon GameLift Servers。
2. 设置插件在托管解决方案中使用的相关 AWS 资源（称为“引导”）。
3. 将服务器代码添加到您的项目，以便在正在运行的游戏服务器和 Amazon GameLift Servers 服务之间建立通信。
4. 将客户端代码添加到您的项目，让游戏客户端向 Amazon GameLift Servers 发送请求，以便启动新游戏会话，然后连接到这些会话。
5. 使用 Anywhere 工作流程将本地工作站设置为 Anywhere 计算，并托管游戏服务器。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。
6. 使用托管 EC2 工作流程将游戏服务器上传到 Amazon GameLift Servers 并部署简单但完整的云托管解决方案。通过插件在本地启动游戏客户端，请求一个游戏会话并连接到该会话，然后开始玩游戏。

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，这些操作可能包含在[AWS 免费套餐](#)中。

Amazon GameLift Servers 虚幻引擎插件

此插件将 Amazon GameLift Servers C++ 服务器 SDK 和工具添加到 UE 编辑器中。使用引导式 UI 工作流程将服务器 SDK 功能集成到您的游戏项目中，并为您的游戏服务器部署 Amazon GameLift Servers 托管解决方案。

利用插件，您可以构建一个基本的托管解决方案，然后根据需要进行优化和自定义。设置以本地工作站为主机的 Amazon GameLift Servers Anywhere 实例集。对于使用托管式 EC2 或托管式容器实例集的云托管，请使用完整的解决方案部署游戏服务器，以便管理游戏会话请求和客户端连接。

主题

- [为 Unreal 游戏项目安装插件](#)
- [后续步骤：自定义游戏托管解决方案](#)
- [虚幻引擎插件：设置 AWS 用户个人资料](#)
- [适用于 Unreal 的插件：集成游戏代码](#)
- [适用于 Unreal 的插件：使用 Amazon GameLift Servers Anywhere 在本地托管游戏](#)
- [适用于 Unreal 的插件：将游戏部署到托管式 EC2 实例集](#)
- [适用于 Unreal 的插件：将游戏部署到托管式容器实例集](#)

为 Unreal 游戏项目安装插件

[从以下地址获取虚幻引擎 Amazon GameLift Servers 插件 GitHub](#)

有关如何在虚幻编辑器中为游戏项目安装插件的信息，请参阅 GitHub 存储库自述文件。

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 GameLift Servers 的功能。
- 适用于 Amazon GameLift Servers 服务 API 的 C++ 库。在客户端后端服务中使用 API 功能来帮助游戏客户端请求游戏会话和 send/retrieve 游戏会话信息。
- 适用于 Amazon GameLift Servers 服务器 SDK (版本 5) 的 Unreal 库。在游戏服务器代码中使用服务器 SDK 来管理托管游戏服务器进程与 Amazon GameLift Servers 服务之间的通信。

- 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。
- 插件在部署游戏服务器进行托管时使用的 CloudFormation 模板形式的可编辑配置。

此插件使用 AWS CloudFormation 模板为常见游戏场景部署托管解决方案。您可以使用提供的这些解决方案，也可以根据游戏需要对其进行自定义。

后续步骤：自定义游戏托管解决方案

使用该插件的指导式工作流程可以很好地快速启动并运行 Amazon GameLift Servers 托管解决方案。借助该插件，您可以为解决方案的每个组件设置基本版本。

准备就绪后，您可以通过自定义每个组件，在此基本解决方案的基础上进行构建，并可以在为游戏发布做准备时对解决方案进行微调。考虑以下选项：

- 修改实例集和实例集配置。请参阅[托管资源自定义](#)。
- 自定义游戏会话队列配置。请参阅[自定义游戏会话队列](#)。
- 向游戏服务器和游戏客户端添加功能。请参阅[将游戏服务器与 Amazon GameLift Servers 集成](#)和[集成 Amazon GameLift Servers 游戏客户端功能](#)。
- 自定义后端服务。请参阅[为 Amazon GameLift Servers 构建后端服务](#)。
- 设置自动容量扩展以满足预期的玩家需求。请参阅[利用 Amazon GameLift Servers 扩展游戏托管容量](#)。
- 设置托管可观测性工具，包括分析和日志记录。请参阅[监控 Amazon GameLift Servers](#)。
- 使用[基础设施即代码 \(IaC \)](#) 自动执行部署。该插件针对托管解决方案的指导式工作流程使用 AWS CloudFormation 模板。您可以根据需要对模板进行自定义。请参阅[使用管理 Amazon GameLift Servers 托管资源 CloudFormation](#)。

主题

- [虚幻引擎插件：设置 AWS 用户个人资料](#)
- [适用于 Unreal 的插件：集成游戏代码](#)
- [适用于 Unreal 的插件：使用 Amazon GameLift Servers Anywhere 在本地托管游戏](#)
- [适用于 Unreal 的插件：将游戏部署到托管式 EC2 实例集](#)
- [适用于 Unreal 的插件：将游戏部署到托管式容器实例集](#)

虚幻引擎插件：设置 AWS 用户个人资料

安装插件后，使用有效的 AWS 帐户设置用户个人资料。您可以在插件中维护多个配置文件，但一次只能选择一个配置文件。每当使用插件时，请选择要使用的配置文件。每个工作流程页面都会显示当前选择的配置文件。

维护多个配置文件使您能够在不同的托管部署之间切换。例如，您可以设置使用相同 AWS 账户但部署到不同 AWS 区域的配置文件。或者，您可以使用不同的 AWS 帐户、用户和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，则该 Amazon GameLift Servers 插件将对其进行检测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。

必须对所有配置文件进行引导，才能在您的账户用户下设置一些必需的 AWS 资源。

管理您的 AWS 个人资料

1. 在 Unreal 编辑器主工具栏中，选择 Amazon GameLift Servers 菜单，然后选择 AWS 访问凭证。此操作会打开 Amazon GameLift Servers 插件并转到设置您的用户配置文件页面。
2. 使用按钮创建新 AWS 帐户或为已有的 AWS 帐户设置用户个人资料。
3. 如果您还没有用户配置文件，系统会提示您输入配置文件详细信息并创建新的配置文件。提供以下信息：
 - 一个 AWS 账户。如果您创建了一个新 AWS 帐户，请使用链接 [AWS 管理控制台](#) 并按照提示进行操作。有关更多详细信息，请参阅 [创建 AWS 账户](#)。
 - 有权使用 Amazon GameLift Servers 和其他必需 AWS 服务的 AWS 用户。有关设置具有 Amazon GameLift Servers 权限的 AWS Identity and Access Management (IAM) 用户以及使用长期证书进行编程访问的说明，请参阅 [设置 AWS 用户账户](#)。
 - 您的 AWS 用户的凭证。这些证书由 AWS 访问密钥 ID 和 AWS 密钥组成。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。从 [支持的区域列表](#) 中选择一个 AWS 区域。
4. 如果插件检测到现有配置文件，则会显示可用配置文件列表。从列表中选择现有配置文件，或选择添加其他配置文件以创建新的配置文件。

引导用户配置文件

所有配置文件必须进行引导才能与 Amazon GameLift Servers 插件搭配使用。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、构建构件和其他依赖项。存储桶不会在其他配置文件之间共享。

Bootstrapp 涉及创建新 AWS 资源，并且可能会产生成本。

要引导您的配置文件，请执行以下操作：

1. 在 AWS 访问凭证页面上，检查要使用的用户配置文件的引导状态。如果配置文件的引导状态为“非活动”，并且未列出 S3 存储桶，则需要引导配置文件。
2. 选择要使用的配置文件，然后选择引导配置文件。
3. 等待引导状态变为“活动”。这可能需要几分钟的时间。

适用于 Unreal 的插件：集成游戏代码

在将游戏服务器部署到实例集之前，您需要对游戏代码进行一系列更新，并将游戏组件打包以供 Amazon GameLift Servers 服务使用。

本主题将介绍进行最小集成的步骤。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。

- [设置生成包目标和模块规则](#)
- [更新游戏服务器代码](#)
- [整合客户端游戏地图](#)
- [打包游戏组件](#)

设置生成包目标和模块规则

修改您的游戏项目文件，以正确生成可与 Amazon GameLift Servers 搭配使用的生成包组件。

要添加客户端和服务端生成包目标，请执行以下操作：

1. 打开游戏项目的代码文件并找到 `.../Games/[your application name]Source/[your application name]Target.cs` 文件。示例：`.../Source/GameLiftUnrealAppTarget.cs`。（如果您使用 Visual Studio，请打开项目的 `.sln` 文件。）
2. 复制此文件，以在 `Source/` 目录中创建两个新的目标文件。

- 客户端目标：将新文件重命名为 *[your application name]*Client.Target.cs。编辑内容以更新类名称和目标类型值，如以下示例代码所示：

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

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

- 服务器目标：将新文件重命名为 *[your application name]*Server.Target.cs。编辑内容以更新类名称和目标类型值，如以下示例代码所示：

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

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

3. 重新生成您的项目文件。如果您使用 Visual Studio，可以右键单击游戏项目的 .uproject 文件，然后选择生成 Visual Studio 项目文件。

要更新游戏项目模块规则，请执行以下操作：

更新游戏项目的模块规则，使其依赖于插件。

1. 打开游戏项目的代码文件并找到 `.../Games/[your application name]Source/[your application name].Build.cs` 文件。示例：`.../Source/GameLiftUnrealApp.Build.cs`。（如果您使用 Visual Studio，请打开项目的 `.sln` 文件。）
2. 找到 `ModuleRules` 类并进行更新，如以下示例代码所示：

```
using UnrealBuildTool;

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

3. 创建新的目标文件并修改模块规则后，重新构建您的游戏项目。

更新游戏服务器代码

更新您的游戏服务器代码，以实现游戏服务器进程与 Amazon GameLift Servers 服务之间的通信。您的游戏服务器必须能够响应来自 Amazon GameLift Servers 的请求，例如启动和停止新的游戏会话。

为 Amazon GameLift Servers 添加服务器代码

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。

gameMode.h 示例代码

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

#pragma once

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

struct FProcessParameters;

DECLARE_LOG_CATEGORY_EXTERN(GameServerLog, Log, All);

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

public:
    AGameLiftUnrealAppGameMode();

protected:
    virtual void BeginPlay() override;

private:
    void InitGameLift();

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

- 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。

以下示例代码展示了如何添加服务器与 Amazon GameLift Servers 集成所需的最少元素：

- 初始化 Amazon GameLift Servers API 客户端。Amazon GameLift Servers Anywhere 实例集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 Servers 服务的请求，包括 OnStartGameSession、OnProcessTerminate 和 onHealthCheck。
- 准备好托管游戏会话时，请使用指定端口调用 ProcessReady() 以通知 Amazon GameLift Servers 服务。

游戏服务器示例代码

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

#include "GameLiftUnrealAppGameMode.h"

#include "UObject/ConstructorHelpers.h"
#include "Kismet/GameplayStatics.h"

#if WITH_GAMELIFT
#include "GameLiftServerSDK.h"
#include "GameLiftServerSDKModels.h"
#endif

#include "GenericPlatform/GenericPlatformOutputDevices.h"

DEFINE_LOG_CATEGORY(GameServerLog);

AGameLiftUnrealAppGameMode::AGameLiftUnrealAppGameMode() :
    ProcessParameters(nullptr)
{
    // Set default pawn class to our Blueprinted character
    static ConstructorHelpers::FClassFinder<APawn> PlayerPawnBPClass(TEXT("/Game/ThirdPerson/Blueprints/BP_ThirdPersonCharacter"));
}
```

```

    if (PlayerPawnBPClass.Class != NULL)
    {
        DefaultPawnClass = PlayerPawnBPClass.Class;
    }

    UE_LOG(GameServerLog, Log, TEXT("Initializing AGameLiftUnrealAppGameMode..."));
}

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

#ifdef WITH_GAMELIFT
    InitGameLift();
#endif
}

void AGameLiftUnrealAppGameMode::InitGameLift()
{
#ifdef WITH_GAMELIFT
    UE_LOG(GameServerLog, Log, TEXT("Calling InitGameLift..."));

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

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

    bool bIsAnywhereActive = false;
    if (FParse::Param(FCommandLine::Get(), TEXT("glAnywhere")))
    {
        bIsAnywhereActive = true;
    }

    if (bIsAnywhereActive)
    {
        UE_LOG(GameServerLog, Log, TEXT("Configuring server parameters for
Anywhere..."));

        // If GameLift Anywhere is enabled, parse command line arguments and pass them
        in the ServerParameters object.

```



```

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

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

        FString glAnywhereProcessId = "";
        if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereProcessId="),
glAnywhereProcessId))
        {
            ServerParametersForAnywhere.m_processId =
TCHAR_TO_UTF8(*glAnywhereProcessId);
        }
        else
        {
            // If no ProcessId is passed as a command line argument, generate a
            randomized unique string.
            FString TimeString = FString::FromInt(std::time(nullptr));
            FString ProcessId = "ProcessId_" + TimeString;
            ServerParametersForAnywhere.m_processId = TCHAR_TO_UTF8(*ProcessId);
        }

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

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

```

```

    }

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

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

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

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

    UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_YELLOW);
    UE_LOG(GameServerLog, Log, TEXT(">>>> WebSocket URL: %s"),
*ServerParametersForAnywhere.m_webSocketUrl);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Fleet ID: %s"),
*ServerParametersForAnywhere.m_fleetId);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Process ID: %s"),
*ServerParametersForAnywhere.m_processId);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Host ID (Compute Name): %s"),
*ServerParametersForAnywhere.m_hostId);

```

```

        UE_LOG(GameServerLog, Log, TEXT(">>>> Auth Token: %s"),
*ServerParametersForAnywhere.m_authToken);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Aws Region: %s"),
*ServerParametersForAnywhere.m_awsRegion);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Access Key: %s"),
*ServerParametersForAnywhere.m_accessKey);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Secret Key: %s"),
*ServerParametersForAnywhere.m_secretKey);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Session Token: %s"),
*ServerParametersForAnywhere.m_sessionToken);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }

    UE_LOG(GameServerLog, Log, TEXT("Initializing the GameLift Server..."));

    //InitSDK will establish a local connection with GameLift's agent to enable further
communication.
    FGameLiftGenericOutcome InitSdkOutcome = GameLiftSdkModule-
>InitSDK(ServerParametersForAnywhere);
    if (InitSdkOutcome.IsSuccess())
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_GREEN);
        UE_LOG(GameServerLog, Log, TEXT("GameLift InitSDK succeeded!"));
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }
    else
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_RED);
        UE_LOG(GameServerLog, Log, TEXT("ERROR: InitSDK failed : ("));
        FGameLiftError GameLiftError = InitSdkOutcome.GetError();
        UE_LOG(GameServerLog, Log, TEXT("ERROR: %s"), *GameLiftError.m_errorMessage);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
        return;
    }

    ProcessParameters = MakeShared<FProcessParameters>();

    //When a game session is created, Amazon GameLift Servers sends an activation
request to the game server and passes along the game session object containing game
properties and other settings.
    //Here is where a game server should take action based on the game session object.
    //Once the game server is ready to receive incoming player connections, it should
invoke GameLiftServerAPI.ActivateGameSession()

```

```

    ProcessParameters->OnStartGameSession.BindLambda( [=]
(Aws::GameLift::Server::Model::GameSession InGameSession)
    {
        FString GameSessionId = FString(InGameSession.GetGameSessionId());
        UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"),
*GameSessionId);
        GameLiftSdkModule->ActivateGameSession();
    });

    //OnProcessTerminate callback. Amazon GameLift Servers will invoke this callback
before shutting down an instance hosting this game server.
    //It gives this game server a chance to save its state, communicate with services,
etc., before being shut down.
    //In this case, we simply tell Amazon GameLift Servers we are indeed going to
shutdown.
    ProcessParameters->OnTerminate.BindLambda( [=]()
    {
        UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
        // First call ProcessEnding()
        FGameLiftGenericOutcome processEndingOutcome = GameLiftSdkModule-
>ProcessEnding();
        // Then call Destroy() to free the SDK from memory
        FGameLiftGenericOutcome destroyOutcome = GameLiftSdkModule->Destroy();
        // Exit the process with success or failure
        if (processEndingOutcome.IsSuccess() && destroyOutcome.IsSuccess()) {
            UE_LOG(GameServerLog, Log, TEXT("Server process ending successfully"));
        }
        else {
            if (!processEndingOutcome.IsSuccess()) {
                const FGameLiftError& error = processEndingOutcome.GetError();
                UE_LOG(GameServerLog, Error, TEXT("ProcessEnding() failed. Error:
%s"),
                    error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
*error.m_errorMessage);
            }
            if (!destroyOutcome.IsSuccess()) {
                const FGameLiftError& error = destroyOutcome.GetError();
                UE_LOG(GameServerLog, Error, TEXT("Destroy() failed. Error: %s"),
                    error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
*error.m_errorMessage);
            }
        }
    });

```

```

//This is the HealthCheck callback.
//Amazon GameLift Servers will invoke this callback every 60 seconds or so.
//Here, a game server might want to check the health of dependencies and such.
//Simply return true if healthy, false otherwise.
//The game server has 60 seconds to respond with its health status. Amazon GameLift
Servers will default to 'false' if the game server doesn't respond in time.
//In this case, we're always healthy!
ProcessParameters->OnHealthCheck.BindLambda([]()
{
    UE_LOG(GameServerLog, Log, TEXT("Performing Health Check"));
    return true;
});

//GameServer.exe -port=7777 LOG=server.mylog
ProcessParameters->port = FURL::UrlConfig.DefaultPort;
TArray<FString> CommandLineTokens;
TArray<FString> CommandLineSwitches;

FCommandLine::Parse(FCommandLine::Get(), CommandLineTokens, CommandLineSwitches);

for (FString SwitchStr : CommandLineSwitches)
{
    FString Key;
    FString Value;

    if (SwitchStr.Split("=", &Key, &Value))
    {
        if (Key.Equals("port"))
        {
            ProcessParameters->port = FString::Atoi(*Value);
        }
    }
}

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

```

```

//The game server calls ProcessReady() to tell Amazon GameLift Servers it's ready
to host game sessions.
UE_LOG(GameServerLog, Log, TEXT("Calling Process Ready..."));
FGameLiftGenericOutcome ProcessReadyOutcome = GameLiftSdkModule-
>ProcessReady(*ProcessParameters);

if (ProcessReadyOutcome.IsSuccess())
{
    UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_GREEN);
    UE_LOG(GameServerLog, Log, TEXT("Process Ready!"));
    UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
}
else
{
    UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_RED);
    UE_LOG(GameServerLog, Log, TEXT("ERROR: Process Ready Failed!"));
    FGameLiftError ProcessReadyError = ProcessReadyOutcome.GetError();
    UE_LOG(GameServerLog, Log, TEXT("ERROR: %s"),
*ProcessReadyError.m_errorMessage);
    UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
}

UE_LOG(GameServerLog, Log, TEXT("InitGameLift completed!"));
#endif
}

```

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)。

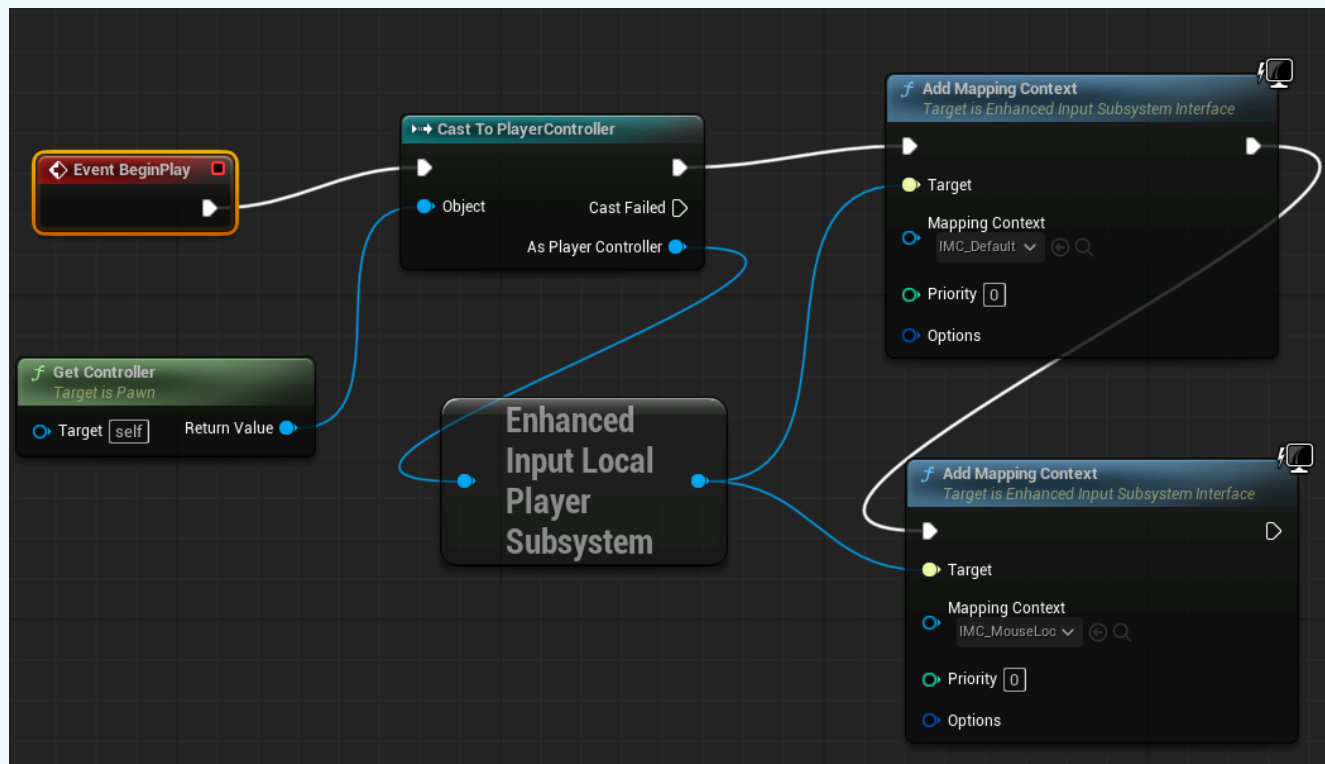
要使用启动游戏地图

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 Servers 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. 对于服务器默认地图，对于虚幻引擎 5.6 或更高版本，选择 “Lv1_ThirdPerson”，对于早期版本，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。

Note

对于虚幻引擎 5.6 或更高版本，如果在连接到游戏服务器后无法移动角色，请更新 BP_ThirdPersonCharacter 蓝图以添加 IMC_Default 和 的输入映射上下文，IMC_MouseLook 如下所示：



打包游戏组件

打包游戏服务器和游戏客户端生成包

1. 在 Unreal Engine 编辑器的源代码构建版本中打开游戏项目。
2. 如果使用虚幻引擎 5.6 或更高版本，请前往编辑、项目设置、打包。在项目内容目录中找到 Cook 所有内容并将其启用。
3. 使用编辑器打包游戏客户端和服务端生成包。
 - 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：[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端构建可执行文件的路径。

适用于 Unreal 的插件：使用 Amazon GameLift Servers Anywhere 在本地托管游戏

使用此工作流程将您的本地工作站设置为使用 Anywhere 实例集的游戏服务器主机。在部署到基于云的托管式实例集之前，您可以使用该流程来测试您的游戏服务器集成。它也可用于在迭代游戏开发期间进行本地测试。

要启动 Amazon GameLift Servers Anywhere 工作流程，请执行以下操作：

- 在 Unreal 编辑器主工具栏中，选择 Amazon GameLift Servers 菜单，然后选择通过 Anywhere 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 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。

设置用户配置文件

- 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”。
- 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”。

步骤 2：设置游戏代码

在此步骤中，准备您的游戏服务器和游戏客户端生成包，使其能够与 Amazon GameLift Servers 搭配使用。如果您尚未集成游戏代码，请参阅[适用于 Unreal 的插件：集成游戏代码](#)。在本地工作站上输入游戏可执行文件的路径。

- 游戏服务器：**将您的游戏服务器与适用于 Amazon GameLift Servers 的服务器 SDK 集成，并打包您的游戏服务器生成包。有关说明，请参阅[适用于 Unreal 的插件：集成游戏代码](#)。游戏服务器必须与服务器 SDK 集成，才能与 Amazon GameLift Servers 服务建立通信，并响应启动新游戏会话和接受游戏客户端连接的提示。
- 游戏客户端：**您至少需要一个可以使用 IP 地址和端口信息连接到您的游戏服务器的游戏客户端。如果您尚未设置游戏客户端组件 Amazon GameLift Servers，则可以使用该 AWS CLI 工具手动请求新的游戏会话、获取连接信息，然后使用该信息连接游戏客户端。

有时，您需要借助后端服务向 Amazon GameLift Servers 服务发送新游戏会话请求，并将连接信息回传给游戏客户端。您可以使用插件附带的测试地图，将客户端 Amazon GameLift Servers 功能添

加到您的游戏项目中。有关构建自定义解决方案的帮助，请参阅[集成 Amazon GameLift Servers 游戏客户端功能](#)。

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 AWS 所在地区的 Anywhere 舰队。
- 如果目前没有实例集，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 实例集后，Amazon GameLift Servers 会验证实例集状态是否处于活动状态并显示实例集 ID。您可以在 Unreal 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。

将您的工作站注册为 Anywhere 计算

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。您也可以使用 localhost (127.0.0.1)，前提是在同一台计算机上运行游戏客户端和服务端。
3. 选择注册计算。您可以在 Unreal 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 Servers 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 Amazon GameLift Servers 服务的通信时所需的控制台参数。

步骤 5：生成身份验证令牌

基于 Anywhere 计算运行的游戏服务器进程需要身份验证令牌才能调用 Amazon GameLift Servers 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。

上面提供的代码示例还可让您使用[适用于 API 请求的 AWS 签名版本 4 \(SigV4 \)](#)。Sigv4 是用于向 API 请求添加身份验证信息的 AWS 签名协议。

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 GameLift Servers 在本地工作站上启动和玩多人游戏所需的所有任务。

玩托管游戏

1. 启动游戏服务器。游戏服务器将在准备好托管游戏会话时通知 Amazon GameLift Servers。
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。此请求通过新的后端服务发送到 Amazon GameLift Servers。作为回应，Amazon GameLift Servers 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 GameLift Servers 会提供连接信息，供游戏客户端加入游戏会话。

适用于 Unreal 的插件：将游戏部署到托管式 EC2 实例集

在此工作流程中，部署您的游戏，使其托管在 Amazon GameLift Servers 托管的基于云的计算资源上。将您的集成式游戏服务器生成包上传到 Amazon GameLift Servers 服务进行部署。如果您尚未集成游戏代码，请参阅[适用于 Unreal 的插件：集成游戏代码](#)。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且正常运行的游戏客户端。

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

- 在 Unreal 编辑器主工具栏中，选择 Amazon GameLift Servers 菜单，然后选择通过托管式 EC2 进行托管。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 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。

设置用户配置文件

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“活动”，请选择引导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，准备您的游戏服务器和游戏客户端生成包，使其能够与适用于 Unreal 的 Amazon GameLift Servers C++ 服务器 SDK 搭配使用。如果您尚未集成游戏代码并构建游戏客户端和服务端可执行文件，请参阅[适用于 Unreal 的插件：集成游戏代码](#)。在本地工作站上输入游戏可执行文件的路径。

在工作流程的这个步骤中，该插件会提供指向说明和源代码的链接，用于设置 Unreal 编辑器的源代码构建版本。在构建客户端和服务端组件时，需要使用源代码构建版本。

构建与服务器 SDK 集的游戏服务器后，请完成以下任务来做好准备，以便将其上传到 Amazon GameLift Servers 进行托管。

准备好服务器生成包以进行云部署 (Windows)

在 Unreal 编辑器默认存储服务器生成包文件的 WindowsServer 文件夹中，添加以下内容：

1. 将服务器生成包安装脚本复制到 **WindowsServer** 文件夹的根目录中。安装脚本包含在插件下载内容中。查找文件 [project-name]/Plugins/Resources/CloudFormation/extra_server_resources/install.bat。Amazon GameLift Servers 使用此文件在您的托管计算资源上安装服务器生成包。
2. 将 **VC_redist.x64.exe** 文件复制到 **WindowsServer** 文件夹的根目录中。如果您使用的是 Unreal Engine 5.6 版或更高版本，可以跳过此步骤。此文件包含在您的 Visual Studio 安装中，通常位于 C:/Program Files (x86)/Microsoft Visual Studio/2019/Professional/VC/Redist/MSVC/v142。
3. 将 OpenSSL 库文件添加到游戏服务器生成包中。如果您的游戏服务器与服务器 SDK 5.3 或更高版本集成，可以跳过此步骤。此版本包含在适用于 Unreal 的 Amazon GameLift Servers 插件 3.0 版或更高版本中。

手动找到 OpenSSL 库并将其复制到您的游戏生成包目录中，位置为 <YourGame>/Binaries/Win64。您必须使用与您的 Unreal Engine 5 版本相同的 OpenSSL 版本。使用错误的 OpenSSL 库部署的游戏生成包将无法与 Amazon GameLift Servers 服务通信。

在您的游戏引擎源代码中查找 OpenSSL 库。具体位置因开发环境而异：

在 Windows 上：

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

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

在 Linux 上：

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

准备好服务器生成包以进行云部署 (Linux)

有关如何准备专为 Linux 构建的游戏服务器的更多详细说明，请参阅[在 Amazon Linux 上为适用于 Unreal Engine 5 的 Amazon GameLift Servers 构建服务器 SDK](#)。

1. 指定一个工作目录来整理生成包文件。工作目录的结构按原样部署到每个托管计算资源上。添加您的 Linux 构建的游戏服务器以及所有依赖文件。
2. 在工作目录的根目录中创建服务器生成包安装脚本。如果需要，请创建一个 install.sh 文件，并添加正确安装游戏服务器生成包所需的所有命令。Amazon GameLift Servers 使用此文件将服务器生成包安装到每个 EC2 托管资源上。
3. 将 OpenSSL 库文件添加到游戏服务器生成包中。如果您的游戏服务器与服务器 SDK 5.3 或更高版本集成，可以跳过此步骤。

手动查找并复制库。您必须使用与您的 Unreal Engine 5 版本相同的 OpenSSL 版本。使用错误的 OpenSSL 库部署的游戏生成包将无法与 Amazon GameLift Servers 服务通信。

- a. 在您的游戏引擎源代码中查找 OpenSSL 库。具体位置因开发环境而异：

在 Windows 上：

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

在 Linux 上：

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

- b. 找到 OpenSSL 库后，将其复制到您的游戏生成包目录中，位置为 <YourGame>/Binaries/Linux。

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。
 - 与 APIs 用户池关联的 API 网关授权器。
 - Web ACI 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。此方案使用低成本的竞价型托管和一个多实例集、多位置结构，以实现持久可用性。当您准备好开始为托管解决方案设计对战构建器组件时，此方法非常有用。在这种方案下，您将为此解决方案创建基本资源，并可以根据需要后续对其进行自定义。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。
 - 与 APIs 用户池关联的 API 网关授权器。
 - Web ACI 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，您将描述要上传到的游戏 AWS；

- 服务器生成包名称：为游戏服务器生成包提供一个有意义的名称。AWS 使用此名称来指代上传并用于部署的服务器生成包的副本。
- 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版本中包含您的 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 验证服务器生成包、预置托管资源、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，此过程可能需要数分钟。

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。要了解更多信息，您可以登录 AWS 管理控制台查看 AWS 和查看事件通知。请务必使用与插件中活跃用户个人资料相同的账户、用户和 AWS 地区登录。

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使用 Amazon GameLift Servers 托管的多人游戏所需的所有任务。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。

适用于 Unreal 的插件：将游戏部署到托管式容器实例集

使用此引导式插件工作流程为您的游戏服务器创建容器映像，并将其部署到基于容器的托管解决方案。如果您尚未集成游戏代码，请参阅[适用于 Unreal 的插件：集成游戏代码](#)。成功完成此工作流程后，您的容器化游戏服务器将在云中运行，您可以使用该插件启动游戏客户端、连接到游戏会话和玩游戏。

开始之前

此工作流程假定您已完成以下任务。

- 将游戏服务器代码与 Amazon GameLift Servers 服务器 SDK 集成。您的托管游戏服务器必须能够与 Amazon GameLift Servers 服务通信，以便能够响应启动新游戏会话和报告游戏会话状态的请求。如果您尚未完成此任务，建议您先按照“通过 Anywhere 托管”插件工作流程进行操作。有关准备游戏服务器代码的指导，请参阅[更新游戏服务器代码](#)。对于托管式容器实例集，必须将游戏与服务器 SDK 5.2 版或更高版本集成。

Note

如果您使用的是启动游戏地图，则此任务已经为您完成了。

- 将游戏服务器可执行文件打包，以便在 Linux 上运行。如果您在 Windows 上开发，并且集成了 C++ 服务器 SDK 5.2.x 版或更早版本，则需要使用 [Unreal 交叉编译工具包](#)。或者，您可以设置一个单独的 Linux 工作区，或使用适用于 Linux 的 Windows 子系统 (WSL) 之类的工具。
- 收集要与游戏服务器生成包一起部署的文件。在本地计算机上，创建一个工作目录来整理将内置到游戏服务器容器映像中的文件。这些文件可能包括游戏依赖项、启动游戏服务器的脚本以及启动容器时的其他进程等。
- 为游戏服务器生成包添加 OpenSSL 库文件。如果您的游戏服务器与服务器 SDK 5.3 或更高版本集成，可以跳过此步骤。

手动找到 OpenSSL 库并将其复制到您的游戏生成包目录中，位置为 <YourGame>/Binaries/Win64。您必须使用与您的 Unreal Engine 5 版本相同的 OpenSSL 版本。使用错误的 OpenSSL 库部署的游戏生成包将无法与 Amazon GameLift Servers 服务通信。

要找到 OpenSSL 库，请查找您的游戏引擎源代码。具体位置因开发环境而异：

在 Windows 上：

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

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

在 Linux 上：

- Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libssl.so.1.1
- Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libcrypto.so.1.1
- 将游戏客户端代码与 Amazon GameLift Servers 集成。要完成此任务，其中一种方法是添加已集成的示例资产（包含在插件中）。有关准备游戏客户端代码的指导，请参阅[整合客户端游戏地图](#)。
- 在本地计算机上安装 Docker。如果您希望插件为您创建容器映像并将其推送到 ECR 存储库，则需要安装此工具。或者，您可以手动执行这些任务，并指示插件使用现有的容器映像。有关手动构建映像的更多信息，请参阅[为 Amazon GameLift Servers 构建容器映像](#)。

要启动 Amazon GameLift Servers 托管式容器工作流程，请执行以下操作：

- 在 Unreal 编辑器主工具栏中，选择 Amazon GameLift Servers 菜单，然后选择托管式容器。此操作将打开插件页面“使用托管容器托管”，该页面显示了使用游戏服务器版本创建容器镜像、将其部署到容器队列以及启动游戏的 step-by-step 过程。

第 0 步：设置您的个人资料

此部分显示您当前选择的用户配置文件。请确认当前的用户配置文件就是您希望在此工作流程中使用的配置文件。您在此工作流程中创建的所有资源都与个人资料的 AWS 帐户相关联，并放置在配置文件的默认 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。

在以下情况下，您可能需要修改所选的用户配置文件：

- 当前未选择任何配置文件。
- 您想要选择其他配置文件或创建新的配置文件。
- 您需要引导所选的配置文件（如果引导状态为非活动）。

设置或更改所选用户配置文件

- 在 Amazon GameLift Servers 菜单中，选择“开放 AWS 访问凭证”。

第 1 步：评测容器就绪情况

在将游戏服务器部署到容器实例集之前，必须将其打包成容器映像并存储在 Amazon ECR 存储库中。该插件可以为您完成这些任务，或者您也可以手动完成这些任务。在此步骤中，提供有关您的容器映像和 ECR 存储库状态的信息。

使用评测问题告知插件，它需要采取哪些步骤：

- 创建新的容器映像。如果您选择此选项，下一步将提示您输入游戏服务器生成包目录和生成包可执行文件的位置。该插件使用 Dockerfile 模板（由 Amazon GameLift Servers 提供），并自动为您的游戏配置该模板。您可以参阅[为 Amazon GameLift Servers 构建容器映像](#)，以查看该模板。选择此选项后，请指明您希望插件存储新映像的位置：
 - 创建一个新的 Amazon ECR 存储库，并将容器映像推送到该存储库。该插件使用您选择的用户个人资料 AWS 区域中的 AWS 帐户和默认帐户创建私有 ECR 存储库。
 - 将容器映像推送到之前创建的 Amazon ECR 存储库。如果您选择此选项，下一步将提示您从列表中选择现有的 Amazon ECR 存储库。该列表包括该 AWS 帐户的所有 Amazon ECR 存储库，并且默认出现在您选择的用户个人资料 AWS 区域中。您可以选择公有存储库或私有存储库。
- 使用现有的容器映像。如果您手动构建了映像，我们建议您使用 Amazon GameLift Servers 提供的 Dockerfile 模板，该模板可在[为 Amazon GameLift Servers 构建容器映像](#)中获取。选择此选项后，请指明映像的位置。
 - 本地存储的 Docker 生成的映像。如果您选择此选项，该插件会创建一个新的 Amazon ECR 私有存储库，并将本地映像文件推送到该存储库。下一步将提示您输入映像 ID，插件会使用该 ID 来找到映像文件。
 - 已存储在 Amazon ECR 存储库中的容器映像。如果您选择此选项，下一步将提示您从列表中选择现有的 Amazon ECR 存储库和映像。该列表包括该 AWS 帐户的所有 Amazon ECR 存储库，并且默认出现在您选择的用户个人资料 AWS 区域中。您可以选择公有存储库或私有存储库。

第 2 步：配置映像部署

在此步骤中，提供插件将您的容器映像部署到容器实例集所需的信息。此步骤要求提供以下信息：

- 您的游戏服务器生成包、容器映像或 Amazon ECR 存储库的位置，具体取决于您在第 1 步中的选择。
- 用于托管式容器部署的方案。
- 客户端配置输出路径。在您的客户端版本中选择包含您的 AWS 配置的文件夹。在以下位置寻找：`[client-build]/[project-name]/Content/CloudFormation`。
- 可选部署设置。此部分包含插件默认使用的配置设置。您可以修改这些设置，也可以保留默认值。

- 默认情况下，游戏名称设置为游戏项目的名称。插件创建的所有 AWS 资源都引用游戏名称值。
- 端口范围、内存限制和 vCPU 限制是容器实例集的配置设置。有关自定义这些值的更多信息，请参阅[配置网络连接](#)以了解连接端口范围，并参阅[设置资源限制](#)以了解资源限制。
- 容器映像标签用于在 Amazon ECR 中对容器映像进行分类。默认值为 unreal-gamelift-plugin。
- Amazon ECR 存储库的名称。只有在插件为您创建 ECR 存储库时，您才能编辑此字段以建议自定义名称。默认值为 unreal-game lift-plugin-ecr-repository。

部署方案选项

单区域容器实例集

此方案将您的游戏服务器部署到单个容器实例集。它是测试服务器与 AWS 集成和容器配置的良好起点。它部署了以下资源：

- Amazon GameLift Servers 容器组定义，描述了如何在容器实例集上部署和运行您的容器映像。
- 已安装并运行您的游戏服务器容器的 Amazon GameLift Servers 容器实例集（按需型），附带别名。
- 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。
- 将用户池与 APIs 关联的 API Gateway 授权方。
- Web 访问控制列表（ACL），用于限制玩家对 API 网关的过多调用。
- 代表游戏客户端向 Amazon GameLift Servers 服务发出请求的后端服务，例如请求游戏会话和加入游戏：
 - API 网关 + Lambda 函数，供玩家申请游戏会话位置。如果没有可用的空闲位置，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。

包含 FlexMatch 的单区域容器实例集

此方案将您的游戏服务器部署到容器实例集、配置游戏会话放置并设置 FlexMatch 对战。当您准备开始为托管解决方案设计自定义配对器时，此方案非常有用。使用此方案时，您将为该解决方案创建基本资源，并可以根据需要后续对其进行自定义。它部署了以下资源：

- Amazon GameLift Servers 容器组定义，描述了如何在容器实例集上部署和运行您的容器映像。

- 已安装并运行您的游戏服务器容器的 Amazon GameLift Servers 容器实例集（按需型），附带别名。
- FlexMatch 对战配置和对战规则设置为接受玩家请求和组成对战。
- Amazon GameLift Servers 游戏会话队列，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。
- 将用户池与 APIs 关联的 API Gateway 授权方。
- Web 访问控制列表（ACL），用于限制玩家对 API 网关的过多调用。
- 代表游戏客户端向 Amazon GameLift Servers 服务发出请求的后端服务，例如请求游戏会话和加入游戏：
 - API 网关 + Lambda 函数，供玩家申请游戏会话位置。如果没有可用的空闲位置，则此函数调用 `StartMatchmaking()`。
 - API 网关 + Lambda 函数，供玩家获取游戏请求的连接信息。
- DynamoDB 表，用于存储玩家的对战票证和游戏会话信息。
- 亚马逊 SNS 主题 + 用于处理事件的 Lambda 函数。GameSessionQueue

部署容器实例集

实例集配置完成后，选择部署容器实例集按钮以开始部署。插件将创建容器映像并将其推送到 ECR，为容器实例集预调配托管资源，并为所选托管解决方案场景部署实例集和其他 AWS 资源，整个过程可能需要几分钟。

开始部署时，您可以跟踪每个步骤的进度。根据您的配置，步骤可能包括以下内容：

- 配置容器映像
- 创建新的 Amazon ECR 存储库
- 构建映像并推送到 Amazon ECR
- 创建容器组定义
- 创建容器实例集

有关更详细的部署信息，请选择在 AWS 管理控制台中查看。当容器实例集达到活动状态时，即表示实例集正在主动运行容器，其中的服务器进程已准备好托管游戏会话。

部署完成后，您将拥有一个正常运行的容器实例集，随时可以托管游戏会话并接受玩家连接。

您无法停止正在进行的部署。如果部署进入不良状态或失败，您可以使用重置部署选项重新开始。

启动客户端

至此，您已经完成了启动和玩使用 Amazon GameLift Servers 托管的多人游戏所需的所有任务。要玩游戏，请选择启动客户端以启动游戏客户端的本地实例。

- 如果您部署了单个实例集方案，请打开一个包含一名玩家的游戏客户端实例，然后进入服务器地图并进行移动操作。您可以打开游戏客户端的第二个实例，将第二名玩家添加到同一个服务器游戏地图中。
- 如果您部署了 FlexMatch 方案，则托管解决方案会等待至少两个游戏客户端发出对战请求。打开至少两个游戏客户端实例，每个实例各包含一名玩家。这两名玩家将进行匹配，并收到加入游戏会话以进行对战的提示。

更新容器实例集

如果您已成功部署托管式容器托管解决方案，可以使用更新部署功能。此选项可让您更新已部署容器实例集的配置设置，而无需创建新的实例集。

更新部署时，您可以部署包含不同游戏服务器生成包的容器映像，更改 Amazon ECR 存储库，选择不同的部署方案，以及自定义可选配置设置。

如果您已准备好部署更改，请选择“更新”。部署更新所需的时间与全面部署所需的时间类似。要了解详细的部署信息，请选择在 AWS 管理控制台中查看。

清理已部署的资源

作为最佳实践，请在不再需要托管容器解决方案时立即清理这些 AWS 资源。如果不删除这些资源，它们可能会继续产生费用。

请删除以下资源：

- 托管式容器资源堆栈。此堆栈中的资源取决于您选择的部署方案。要删除整个堆栈，请使用 CloudFormation 控制台。从 Amazon GameLift Servers 插件生成的堆栈使用以下命名规范：GameLiftPluginForUnreal-{GameName}-Containers。在插件中启动新的托管式容器部署之前，请等待堆栈删除过程完成。有关更多信息，请参阅[从 CloudFormation 控制台删除堆栈](#)。
- Amazon ECR 存储库。如果您使用插件为容器映像创建了存储库，可能需要删除所有不再需要的存储库。在重置托管式容器部署之前，无需删除存储库。如果您更新或重置部署，除非指示使用其他存储库，否则插件将自动使用同一个存储库。有关更多信息，请参阅[在 Amazon ECR 中删除私有存储库](#)。

Amazon GameLift Servers适用于 Unity 的插件 (服务器 SDK 5.x)

此插件将 Amazon GameLift Servers C# 服务器 SDK 和工具添加到 Unity 编辑器中。使用引导式 UI 工作流程将服务器 SDK 功能集成到您的游戏项目中，并为您的游戏服务器部署 Amazon GameLift Servers 托管解决方案。

利用插件，您可以构建一个基本的托管解决方案，然后根据需要进行优化和自定义。设置以本地工作站为主机的 Amazon GameLift Servers Anywhere 实例集。对于使用托管式 EC2 或托管式容器实例集的云托管，请使用完整的解决方案部署游戏服务器，以便管理游戏会话请求和客户端连接。

主题

- [为 Unity 游戏项目安装插件](#)
- [Unity 插件：设置 AWS 用户个人资料](#)
- [适用于 Unity 的插件：使用 Amazon GameLift Servers Anywhere 设置本地测试](#)
- [适用于 Unity 的插件：将游戏部署到托管式 EC2 实例集](#)
- [适用于 Unity 的插件：将游戏部署到托管式容器实例集](#)

为 Unity 游戏项目安装插件

[从以下地址获取 Unity Amazon GameLift Servers 插件 GitHub](#)

有关如何为游戏项目安装插件的信息，请参阅 GitHub 存储库自述文件。

该插件包括以下组件：

- Unity 编辑器的插件模块。安装插件后，新的主菜单项可让您访问 Amazon GameLift Servers 的功能。
- 适用于具有客户端功能的 Amazon GameLift Servers 服务 API 的 C# 库。
- 适用于 Amazon GameLift Servers 服务器 SDK (版本 5.x) 的 C# 库。
- 示例游戏内容 (包括资产和场景)，因此即使您没有生成包就绪的多人游戏，也可以试试 Amazon GameLift Servers。
- 解决方案配置，作为 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。

此插件使用 AWS CloudFormation 模板为常见游戏场景部署托管解决方案。您可以使用提供的这些解决方案，也可以根据游戏需要对其进行自定义。

Unity 插件：设置 AWS 用户个人资料

安装插件后，使用有效的 AWS 帐户设置用户个人资料。您可以在插件中维护多个配置文件，但一次只能选择一个配置文件。每当使用插件时，请选择要使用的配置文件。每个工作流程页面都会显示当前选择的配置文件。

维护多个配置文件使您能够在不同的托管部署之间切换。例如，您可以设置使用相同 AWS 账户但部署到不同 AWS 区域的配置文件。或者，您可以使用不同的 AWS 帐户、用户和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，则该 Amazon GameLift Servers 插件将对其进行检测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。

必须启动所有配置文件才能在您的账户用户下设置一些必需的 AWS 资源。

管理您的 AWS 个人资料

1. 在 Unity 主工具栏中，选择 Amazon GameLift Servers 菜单，然后选择 AWS 访问凭证。此操作会打开 Amazon GameLift Servers 插件并转到设置您的用户配置文件页面。
2. 使用按钮创建新 AWS 帐户或为已有的 AWS 帐户设置用户个人资料。
3. 如果您还没有用户配置文件，系统会提示您输入配置文件详细信息并创建新的配置文件。提供以下信息：
 - 一个 AWS 账户。如果您创建了一个新 AWS 帐户，请使用链接 [AWS 管理控制台](#) 并按照提示进行操作。有关更多详细信息，请参阅 [创建 AWS 账户](#)。
 - 有权使用 Amazon GameLift Servers 和其他必需 AWS 服务的 AWS 用户。有关设置具有 Amazon GameLift Servers 权限的 AWS Identity and Access Management (IAM) 用户以及使用长期证书进行编程访问的说明，请参阅 [设置 AWS 用户账户](#)。
 - 您的 AWS 用户的凭证。这些证书由 AWS 访问密钥 ID 和 AWS 密钥组成。有关更多详细信息，请参阅 [获取访问密钥](#)。
 - AWS 区域。这是您要创建托管 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。从 [支持的区域列表](#) 中选择一个 AWS 区域。
4. 如果插件检测到现有配置文件，则会显示可用配置文件列表。从列表中选择现有配置文件，或选择添加其他配置文件以创建新的配置文件。

引导用户配置文件

所有配置文件必须进行引导才能与 Amazon GameLift Servers 插件搭配使用。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、构建构件和其他依赖项。存储桶不会在其他配置文件之间共享。

Bootstrapp涉及创建新 AWS 资源，并且可能会产生成本。

要引导您的配置文件，请执行以下操作：

1. 在 AWS 访问凭证页面上，检查要使用的用户配置文件的引导状态。如果配置文件的引导状态为“非活动”，并且未列出 S3 存储桶，则需要引导配置文件。
2. 选择要使用的配置文件，然后选择引导配置文件。
3. 等待引导状态变为“活动”。这可能需要几分钟的时间。

适用于 Unity 的插件：使用 Amazon GameLift Servers Anywhere 设置本地测试

在此工作流程中，您可以为 Amazon GameLift Servers 功能添加客户端和服务端游戏代码，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端组件。

要启动 Amazon GameLift Servers Anywhere 工作流程，请执行以下操作：

- 在 Unity 编辑器主菜单中，选择 Amazon GameLift Servers，然后选择通过 Anywhere 托管。此操作将打开插件页面，用于使用 @Anywhere 实例集设置您的游戏。该页面提供了集成、构建和启动游戏组件的五步流程。

设置配置文件

选择您在遵循此工作流程时要使用的配置文件。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关联的 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 Servers 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”。

将游戏代码与 C# 服务器 SDK 集成。

 Note

如果您导入了示例游戏，则可以跳过这一步。示例游戏资产已准备好所需的服务器和客户端代码。

在 workflows 的这一步中，您需要更新游戏项目中的客户端和服务端代码。

- 游戏服务器必须能够与 Amazon GameLift Servers 服务通信才能接收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、加入或启动游戏会话以及获取连接信息才能加入游戏。

集成服务器代码

如果将您自己的游戏项目与自定义场景一起使用，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：

1. 在您的游戏项目文件中，打开 Assets/Scripts/Server 文件夹。如果该文件夹不存在，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 Server 文件夹中。构建服务器可执行文件时，请使用此文件作为构建目标。

示例代码包含以下所需的最少元素，这些元素使用了 Amazon GameLift Servers C# 服务器 SDK (版本 5)：

- 初始化 Amazon GameLift Servers API 客户端。Amazon GameLift Servers Anywhere 实例集需要使用服务器参数进行 `InitSDK()` 调用。这些设置会自动设置，以便在插件中使用。
- 实施所需的回调函数以响应来自 Amazon GameLift Servers 服务的请求，包括 `OnStartGameSession`、`OnProcessTerminate`、和 `onHealthCheck`。
- 当服务器进程准备好托管游戏会话时，请使用指定端口调用 `ProcessReady()` 以通知 Amazon GameLift Servers 服务。

如果要自定义示例服务器代码，请参阅以下资源：

- [借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)
- [适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 操作](#)

集成客户端代码

如果将您自己的游戏项目与自定义场景一起使用，则需要将基本功能集成到游戏客户端。您还需要添加 UI 元素，让玩家可以登录并加入游戏会话。使用 Amazon GameLift Servers (在 AWS SDK 中) 的服务 API 来获取游戏会话信息、创建新的游戏会话或加入现有游戏会话，

使用 Anywhere 实例集构建要进行本地测试的客户端时，您可以添加对 Amazon GameLift Servers 服务的直接调用。当您开发要进行云托管的游戏时（或者您计划使用 Anywhere 实例集进行生产托管），您需要创建一个客户端后端服务来处理游戏客户端和 Amazon GameLift Servers 服务之间的所有通信。

要将 Amazon GameLift Servers 集成到客户端代码，请使用以下资源作为指南。

- 将客户端与 GitHub repo 中的 GameLiftCoreApi 类集成 `aw amazon-gamelift-plugin-unity s/`。该类提供了用于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 GameLift Servers 添加到您的 Unity 游戏客户端”中的说明进行操作。

对于连接到 Anywhere 实例集的游戏客户端，您的游戏客户端需要以下信息。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。这是您在用户配置文件中设置的区域。
- ProfileName -本地计算机上的 AWS 凭据配置文件，允许访问的 AWS SDK Amazon GameLift Servers。游戏客户端使用这些凭证来验证对 Amazon GameLift Servers 服务的请求的身份。

 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 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活跃用户配置文件 AWS 所在地区的 Anywhere 舰队。
- 如果目前没有实例集，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 实例集后，Amazon GameLift Servers 会验证实例集状态是否处于活动状态并显示实例集 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。您可以在 Unity 编辑器的输出日志中跟踪此请求的进度。

插件会注册您的本地工作站，并将 IP 地址设置为 localhost (127.0.0.1)。此设置假定您将在同一台计算机上运行游戏客户端和服务端。

作为对这一操作的响应，Amazon GameLift Servers 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。插件将使用该资产来存储游戏客户端连接到 Amazon GameLift Servers 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，请创建一个新的 `GameLiftClientSettings` 资产。在 Unity 编辑器主菜单中，选择资产、创建、Amazon GameLift、客户端设置。如果您在项目中创建了多个 `GameLiftClientSettings` 副本，插件会自动检测到这一点，并通知您插件将更新哪个资产。
 - b. 在启动游戏中，选择配置客户端：应用 Anywhere 设置。此操作将更新游戏客户端设置，以使用您刚刚设置的 Anywhere 实例集。
2. 构建并运行游戏客户端。
 - a. 使用标准 Unity 构建流程构建客户端可执行文件。在文件、构建设置中，将平台切换为 Windows、Mac 或 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. 启动新构建的游戏客户端可执行文件的一个或多个实例。
 3. 在 Anywhere 实例集中启动游戏服务器。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. 启动或加入游戏会话。在您的游戏客户端实例中，使用 UI 将每个客户端加入游戏会话。如何执行此操作取决于您是如何向客户端添加功能的。

如果您使用的是示例游戏客户端，那么它具有以下特征：

- 一个玩家登录组件。连接到 Anywhere 实例集上的游戏服务器时，不会进行玩家验证。您可以输入任意值来加入游戏会话。
- 一个简单的加入游戏 UI。当一个客户端尝试加入游戏时，该客户端会自动查找有可用玩家位置的活跃游戏会话。如果没有可用的游戏会话，该客户端会请求新的游戏会话。如果有可用的游戏会话，该客户端会请求加入该游戏会话。使用多个并发客户端来测试游戏时，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。
- 有四个玩家位置的游戏会话。您最多可以同时启动四个游戏客户端实例，它们将加入同一游戏会话。

通过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 实例集上进行测试。

1. 使用标准 Unity 构建流程构建服务器可执行文件。在文件、构建设置中，将平台切换为专用服务器，然后构建。

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。当您创建实例集时，实例集 ID 将显示在连接到 Anywhere 实例集中。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. 通过命令行启动新构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]
```

适用于 Unity 的插件：将游戏部署到托管式 EC2 实例集

在此工作流程中，您将使用插件准备游戏，使其托管在由 Amazon GameLift Servers 管理的基于云的计算资源上。您需要为 Amazon GameLift Servers 功能添加客户端和服务端游戏代码，然后将服务器生成包上传到 Amazon GameLift Servers 服务进行托管。此工作流程完成后，您将拥有在云中运行的游戏服务器，以及可以连接到该服务器且正常运行的游戏客户端。

要启动 Amazon GameLift Servers 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 Servers，然后选择通过托管式 EC2 进行托管。此工作流程提供了集成、构建、部署和启动游戏组件的六步流程。

设置配置文件

选择您在遵循此工作流程时要使用的配置文件。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 Servers 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 GameLift Servers 集成

对于此任务，您需要更新游戏项目中的客户端和服务端代码。

- 游戏服务器必须能够与 Amazon GameLift Servers 服务通信才能接收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、加入或启动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，则可以跳过这一步。示例游戏资产已准备好所需的服务器和客户端代码。

集成服务器代码

将您自己的游戏项目与自定义场景一起使用时，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。如果您已将游戏项目与 Anywhere 实例集集成以进行测试，您就已经完成此步骤中的指令。

1. 在您的游戏项目文件中，打开 Assets/Scripts/Server 文件夹。如果该文件夹不存在，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 Server 文件夹中。构建服务器可执行文件时，请使用此文件作为构建目标。

示例代码包含以下所需的最少元素，这些元素使用了 Amazon GameLift Servers C# 服务器 SDK（版本 5）：

- 初始化 Amazon GameLift Servers API 客户端。Amazon GameLift Servers Anywhere 实例集需要使用服务器参数进行 `InitSDK()` 调用。这些设置会自动设置，以便在插件中使用。
- 实施所需的回调函数以响应来自 Amazon GameLift Servers 服务的请求，包括 `OnStartGameSession`、`OnProcessTerminate`、和 `onHealthCheck`。
- 当服务器进程准备好托管游戏会话时，请使用指定端口调用 `ProcessReady()` 以通知 Amazon GameLift Servers 服务。

如果要自定义示例服务器代码，请参阅以下资源：

- [借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)

- [适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 操作](#)

集成客户端代码

对于连接到基于云的游戏服务器的游戏客户端，最佳实践是使用客户端后端服务对 Amazon GameLift Servers 服务进行调用，而不是直接从游戏客户端进行调用。

在托管在托管式 EC2 实例集上的插件工作流程中，每个部署方案都包含一个预先构建的后端服务，其中包含以下组件：

- 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，用于生成唯一玩家 IDs 并对玩家连接进行身份验证。

要使用这些组件，游戏客户端需要能够向后端服务发送执行以下操作的请求：

- 在 Cognito 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。
- 使用对战加入游戏。

请使用以下资源作为指南。

- 将客户端与 GitHub repo 中的 [GameLiftCoreApi](#) 类集成 [aws/amazon-gamelift-plugin-unity](#)。该类提供了用于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 [aws/amazon-gamelift-plugin-unity](#)，。Samples~/SampleGame/Assets/Scripts/Client/GameLiftClient.cs
- [集成 Amazon GameLift Servers 游戏客户端功能](#)。

(仅限 Unity 6.3+) 集成 install.sh 脚本

Linux 上的 Unity 6.3 需要 glibc 2.35。亚马逊 Linux 2023 包括 glibc 2.34。提供了一个 install.sh 脚本，用于处理从源代码构建 glibc 2.35 以及将 Unity 二进制文件配置为使用 glibc 2.35。

1. 前往 [aws/GitHub](#) 仓库 [amazon-gamelift-plugin-unity](#) 并打开路径。ExtraServerResources
2. 找到该文件 install.sh 并将其复制到包含您的服务器版本的文件夹中。
3. 如果您已有脚本 install.sh，请将两个脚本的内容合并为一个 install.sh。

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。此场景是测试服务器与服务器版本配置的集成 AWS 和服务器版本配置的良好起点。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。
 - 与 APIs 用户池关联的 API 网关授权器。
 - Web API 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。此方案使用低成本的竞价型托管和一个多实例集、多位置结构，以实现持久可用性。当您准备好开始为托管解决方案设计对战构建器组件时，此方法非常有用。在这种方案下，您将为此解决方案创建基本资源，并可以根据需要后续对其进行自定义。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。
 - 与 APIs 用户池关联的 API 网关授权器。
 - Web API 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，您将描述要上传到的游戏 AWS。

- **游戏名称**：为游戏项目提供一个有意义的名称。此名称在插件中使用。
- **实例集名称**：为托管式 EC2 实例集提供一个有意义的名称。Amazon GameLift Servers 在 AWS 控制台中列出资源时会使用此名称（以及实例集 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 Servers 并用于部署的服务器版本的副本。
- **启动参数**：输入在托管式 EC2 实例集实例上启动服务器可执行文件时要运行的可选指令。最大长度为 1024 个字符。
- **游戏服务器文件夹**：提供包含服务器生成包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的名称。

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 验证服务器生成包、预置托管资源、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，此过程可能需要数分钟。

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。要了解更多信息，您可以登录 AWS 管理控制台查看 AWS 和查看事件通知。请务必使用与插件中活跃用户个人资料相同的账户、用户和 AWS 地区登录。

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

成功部署实例集后，您的游戏服务器就会开始运行，并可用于托管游戏会话。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。插件将使用该资产来存储游戏客户端连接到 Amazon GameLift Servers 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，请创建一个新的 GameLiftClientSettings 资产。在 Unity 编辑器主菜单中，选择资产、创建、Amazon GameLift、客户端设置。如果您在项目中创建了多个 GameLiftClientSettings 副本，插件会自动检测到这一点，并通知您插件将更新哪个资产。
 - b. 在启动游戏中，选择配置客户端：应用托管式 EC2 设置。此操作将更新游戏客户端设置，以使用您刚刚部署的托管式 EC2 实例集。

2. 构建游戏客户端。使用标准 Unity 构建流程构建客户端可执行文件。在“文件”、“构建设置”中，将平台切换为 Windows、Mac 或 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，并在每个实例中使用 UI 以加入游戏会话。

如果您使用的是示例游戏客户端，那么它具有以下特征：

- 一个玩家登录组件。连接到 Anywhere 实例集上的游戏服务器时，不会进行玩家验证。您可以输入任意值来加入游戏会话。
- 一个简单的加入游戏 UI。当一个客户端尝试加入游戏时，该客户端会自动查找有可用玩家位置的活跃游戏会话。如果没有可用的游戏会话，该客户端会请求新的游戏会话。如果有可用的游戏会话，该客户端会请求加入该游戏会话。使用多个并发客户端来测试游戏时，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。
- 有四个玩家位置的游戏会话。您最多可以同时启动四个游戏客户端实例，它们将加入同一游戏会话。

适用于 Unity 的插件：将游戏部署到托管式容器实例集

使用此引导式插件工作流程为您的游戏服务器创建容器映像，并将其部署到基于容器的托管解决方案。成功完成此工作流程后，您的容器化游戏服务器将在云中运行，您可以使用该插件启动游戏客户端、连接到游戏会话和玩游戏。

开始之前

此工作流程假定您已完成以下任务。

- 将游戏服务器代码与 Amazon GameLift Servers 服务器 SDK 集成。我们的托管游戏服务器必须能够与 Amazon GameLift Servers 服务通信，以便能够响应启动新游戏会话和报告游戏会话状态的请求。如果您尚未完成此任务，建议您先按照“通过 Anywhere 托管”插件工作流程进行操作。有关准备游戏服务器代码的指导，请参阅[集成服务器代码](#)。对于托管式容器实例集，必须将游戏与服务器 SDK 5.2 版或更高版本集成。

 Note

如果您导入了示例游戏，则此任务已为您完成。

- 将游戏服务器可执行文件打包，以便在 Linux 上运行。

- 收集要与游戏服务器生成包一起部署的文件。在本地计算机上，创建一个工作目录来整理将内置到游戏服务器容器映像中的文件。这些文件可能包括游戏依赖项、启动游戏服务器的脚本以及启动容器时的其他进程等。
- 将游戏客户端代码与 Amazon GameLift Servers 集成。要完成此任务，其中一种方法是添加已集成的示例资产（包含在插件中）。有关准备游戏客户端代码的指导，请参阅[集成客户端代码](#)。
- 在本地计算机上安装 Docker。如果您希望插件为您创建容器映像并将其推送到 ECR 存储库，则需要安装此工具。或者，您可以手动执行这些任务，并指示插件使用现有的容器映像。有关手动构建映像的更多信息，请参阅[为 Amazon GameLift Servers 构建容器映像](#)。

要启动 Amazon GameLift Servers 托管式容器工作流程，请执行以下操作：

- 在 Unity 编辑器主工具栏中，选择 Amazon GameLift Servers 菜单，然后选择托管式容器。此操作将打开插件页面“使用托管容器托管”，该页面显示了使用游戏服务器版本创建容器镜像、将其部署到容器队列以及启动游戏的 step-by-step 过程。

第 0 步：设置您的个人资料

此部分显示您当前选择的用户配置文件。请确认当前的用户配置文件就是您希望在此工作流程中使用的配置文件。您在此工作流程中创建的所有资源都与个人资料 AWS 帐户相关联，并放置在配置文件的默认 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。

在以下情况下，您可能需要修改所选的用户配置文件：

- 当前未选择任何配置文件。
- 您想要选择其他配置文件或创建新的配置文件。
- 您需要引导所选的配置文件（如果引导状态为非活动）。

设置或更改所选用户配置文件

- 在 Amazon GameLift Servers 菜单中，选择开放 AWS 访问凭证。

第 1 步：评测容器就绪情况

在将游戏服务器部署到容器实例集之前，必须将其打包成容器映像并存储在 Amazon ECR 存储库中。该插件可以为您完成这些任务，或者您也可以手动完成这些任务。在此步骤中，提供有关您的容器映像和 ECR 存储库状态的信息。

使用评测问题告知插件，它需要采取哪些步骤：

- 创建新的容器映像。如果您选择此选项，下一步将提示您输入游戏服务器生成包目录和生成包可执行文件的位置。该插件使用 Dockerfile 模板（由 Amazon GameLift Servers 提供），并自动为您的游戏配置该模板。您可以参阅[为 Amazon GameLift Servers 构建容器映像](#)，以查看该模板。选择此选项后，请指明您希望插件存储新映像的位置：
- 创建一个新的 Amazon ECR 存储库，并将容器映像推送到该存储库。该插件使用您选择的用户个人资料 AWS 区域 中的 AWS 帐户和默认帐户创建私有 ECR 存储库。
- 将容器映像推送到之前创建的 Amazon ECR 存储库。如果您选择此选项，下一步将提示您从列表中选择现有的 Amazon ECR 存储库。该列表包括该 AWS 账户的所有 Amazon ECR 存储库，并且默认出现在您选择的用户个人资料 AWS 区域 中。您可以选择公有存储库或私有存储库。
- 使用现有的容器映像。如果您手动构建了映像，我们建议您使用 Amazon GameLift Servers 提供的 Dockerfile 模板，该模板可在[为 Amazon GameLift Servers 构建容器映像](#)中获取。选择此选项后，请指明映像的位置。
- 本地存储的 Docker 生成的映像。如果您选择此选项，该插件会创建一个新的 Amazon ECR 私有存储库，并将本地映像文件推送到该存储库。下一步将提示您输入映像 ID，插件会使用该 ID 来找到映像文件。
- 已存储在 Amazon ECR 存储库中的容器映像。如果您选择此选项，下一步将提示您从列表中选择现有的 Amazon ECR 存储库和映像。该列表包括该 AWS 账户的所有 Amazon ECR 存储库，并且默认出现在您选择的用户个人资料 AWS 区域 中。您可以选择公有存储库或私有存储库。
- Linux 上的 Unity 6.3 需要 glibc 2.35。亚马逊 Linux 2023 包括 glibc 2.34。下面提供了一个 Dockerfile 模板，用于从源代码构建 glibc 2.35 以及将 Unity 二进制文件配置为使用 glibc 2.35。

Unity 6.3 游戏服务器容器镜像的 Dockerfile 模板

此模板包含游戏服务器容器要在 Amazon GameLift Servers 实例集中可用所需的最少指令。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.
FROM public.ecr.aws/amazonlinux/amazonlinux

# Game build directory
# -----
```

```

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

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

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

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

# Change directory to home
WORKDIR $HOME_DIR

# Build glibc 2.35 and patch Unity binaries
RUN GLIBC_VERSION="2.35" && \
    INSTALL_PREFIX="/opt/glibc-`${GLIBC_VERSION}`" && \
    ARCH=$(uname -m) && \
    if [ "$ARCH" = "x86_64" ]; then \

```

```

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

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

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

USER gamescale

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

# Check path before starting the container
RUN echo $PATH

```

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

第 2 步：配置映像部署

在此步骤中，提供插件将您的容器映像部署到容器实例集所需的信息。此步骤要求提供以下信息：

- 您的游戏服务器生成包、容器映像或 Amazon ECR 存储库的位置，具体取决于您在第 1 步中的选择。
- 用于托管式容器部署的方案。
- 可选部署设置。此部分包含插件默认使用的配置设置。您可以修改这些设置，也可以保留默认值。
 - 默认情况下，游戏名称设置为游戏项目的名称。插件创建的所有 AWS 资源都引用游戏名称值。
 - 端口范围、内存限制和 vCPU 限制是容器实例集的配置设置。有关自定义这些值的更多信息，请参阅[配置网络连接](#)以了解连接端口范围，并参阅[设置资源限制](#)以了解资源限制。
 - 容器映像标签用于在 Amazon ECR 中对容器映像进行分类。默认值为 unity-gamelift-plugin。

部署方案选项

单区域容器实例集

此方案将您的游戏服务器部署到单个容器实例集。它是测试服务器与 AWS 集成和容器配置的良好起点。它部署了以下资源：

- Amazon GameLift Servers 容器组定义，描述了如何在容器实例集上部署和运行您的容器映像。
- 已安装并运行您的游戏服务器容器的 Amazon GameLift Servers 容器实例集（按需型），附带别名。
- 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。
- 将用户池与 APIs 关联的 API Gateway 授权方。
- Web 访问控制列表（ACL），用于限制玩家对 API 网关的过多调用。
- 代表游戏客户端向 Amazon GameLift Servers 服务发出请求的后端服务，例如请求游戏会话和加入游戏：
 - API 网关 + Lambda 函数，供玩家申请游戏会话位置。如果没有可用的空闲位置，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。

包含 FlexMatch 的单区域容器实例集

此方案将您的游戏服务器部署到容器实例集、配置游戏会话放置并设置 FlexMatch 对战。当您准备开始为托管解决方案设计自定义配对器时，此方案非常有用。使用此方案时，您将为该解决方案创建基本资源，并可以根据需要后续对其进行自定义。它部署了以下资源：

- Amazon GameLift Servers 容器组定义，描述了如何在容器实例集上部署和运行您的容器映像。
- 已安装并运行您的游戏服务器容器的 Amazon GameLift Servers 容器实例集（按需型），附带别名。
- FlexMatch 对战配置和对战规则设置为接受玩家请求和组成对战。
- Amazon GameLift Servers 游戏会话队列，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。
- 将用户池与 APIs 关联的 API Gateway 授权方。
- Web 访问控制列表（ACL），用于限制玩家对 API 网关的过多调用。
- 代表游戏客户端向 Amazon GameLift Servers 服务发出请求的后端服务，例如请求游戏会话和加入游戏：
 - API 网关 + Lambda 函数，供玩家申请游戏会话位置。如果没有可用的空闲位置，则此函数调用 `StartMatchmaking()`。
 - API 网关 + Lambda 函数，供玩家获取游戏请求的连接信息。
- DynamoDB 表，用于存储玩家的对战票证和游戏会话信息。
- 亚马逊 SNS 主题 + 用于处理事件的 Lambda 函数。GameSessionQueue

部署容器实例集

实例集配置完成后，选择部署容器实例集按钮以开始部署。插件将创建容器映像并将其推送到 ECR，为容器实例集预调配托管资源，并为所选托管解决方案场景部署实例集和其他 AWS 资源，整个过程可能需要几分钟。

开始部署时，您可以跟踪每个步骤的进度。根据您的配置，步骤可能包括以下内容：

- 配置容器映像
- 创建新的 Amazon ECR 存储库
- 构建映像并推送到 Amazon ECR

- 创建容器组定义
- 创建容器实例集

有关更详细的部署信息，请选择在 AWS 管理控制台中查看。当容器实例集达到活动状态时，即表示实例集正在主动运行容器，其中的服务器进程已准备好托管游戏会话。

部署完成后，您将拥有一个正常运行的容器实例集，随时可以托管游戏会话并接受玩家连接。

您无法停止正在进行的部署。如果部署进入不良状态或失败，您可以使用重置部署选项重新开始。

启动客户端

至此，您已经完成了启动和玩使用 Amazon GameLift Servers 托管的多人游戏所需的所有任务。要玩游戏，请选择启动客户端以启动游戏客户端的本地实例。

- 如果您部署了单个实例集方案，请打开一个包含一名玩家的游戏客户端实例，然后进入服务器地图并进行移动操作。您可以打开游戏客户端的第二个实例，将第二名玩家添加到同一个服务器游戏地图中。
- 如果您部署了 FlexMatch 方案，则托管解决方案会等待至少两个游戏客户端发出对战请求。打开至少两个游戏客户端实例，每个实例各包含一名玩家。这两名玩家将进行匹配，并收到加入游戏会话以进行对战的提示。

更新容器实例集

如果您已成功部署托管式容器托管解决方案，可以使用更新部署功能。此选项可让您更新已部署容器实例集的配置设置，而无需创建新的实例集。

更新部署时，您可以部署包含不同游戏服务器生成包的容器映像，更改 Amazon ECR 存储库，选择不同的部署方案，以及自定义可选配置设置。

如果您已准备好部署更改，请选择“更新”。部署更新所需的时间与全面部署所需的时间类似。要了解详细的部署信息，请选择在 AWS 管理控制台中查看。

清理已部署的资源

作为最佳实践，请在不再需要托管容器解决方案时立即清理这些 AWS 资源。如果不删除这些资源，它们可能会继续产生费用。

请删除以下资源：

- 托管式容器资源堆栈。此堆栈中的资源取决于您选择的部署方案。要删除整个堆栈，请使用 CloudFormation 控制台。从 Amazon GameLift Servers 插件生成的堆栈使用以下命名规范：GameLiftPluginForUnity-{GameName}-Containers。在插件中启动新的托管式容器部署之前，请等待堆栈删除过程完成。有关更多信息，请参阅[从 CloudFormation 控制台删除堆栈](#)。
- Amazon ECR 存储库。如果您使用插件为容器映像创建了存储库，可能需要删除所有不再需要的存储库。在重置托管式容器部署之前，无需删除存储库。如果您更新或重置部署，除非指示使用其他存储库，否则插件将自动使用同一个存储库。有关更多信息，请参阅[在 Amazon ECR 中删除私有存储库](#)。

适用于 Unity 的 Amazon GameLift Servers 插件（服务器 SDK 4）

Note

本主题提供了适用于 Unity 的 Amazon GameLift Servers 插件的早期版本相关信息。版本 1.x 使用适用于 Amazon GameLift Servers 4.x 或更早版本的服务器 SDK。有关插件最新版本（使用服务器 SDK 5.x 并支持 Amazon GameLift Servers Anywhere 和托管式容器托管等新功能）的文档，请参阅[Amazon GameLift Servers 适用于 Unity 的插件（服务器 SDK 5.x）](#)。

Amazon GameLift Servers 提供一些工具，用于让您的多人游戏服务器准备就绪，从而在 Amazon GameLift Servers 上运行。借助适用于 Unity 的 Amazon GameLift Servers 插件，您可以更轻松地将 Amazon GameLift Servers 集成到 Unity 游戏项目中并部署 Amazon GameLift Servers 资源进行云托管。使用 Unity 插件访问 Amazon GameLift Servers APIs 和部署常见游戏场景的 AWS CloudFormation 模板。

设置插件后，您可以试用 [Amazon GameLift Servers Unity 示例](#) GitHub。

主题

- [将 Amazon GameLift Servers 与 Unity 游戏服务器项目集成](#)
- [将 Amazon GameLift Servers 与 Unity 游戏客户端项目集成](#)
- [安装并设置插件](#)
- [在本地测试游戏](#)
- [部署场景](#)
- [在 Unity 中将游戏与 Amazon GameLift Servers 集成](#)
- [导入并运行示例游戏](#)

将 Amazon GameLift Servers 与 Unity 游戏服务器项目集成

Note

本主题提供了适用于 Unity 的 Amazon GameLift Servers 插件的早期版本相关信息。版本 1.x 使用适用于 Amazon GameLift Servers 4.x 或更早版本的服务器 SDK。有关插件最新版本（使用服务器 SDK 5.x 并支持 Amazon GameLift Servers Anywhere 和托管式容器托管等新功能）的文档，请参阅[Amazon GameLift Servers 适用于 Unity 的插件（服务器 SDK 5.x）](#)。

本主题可帮助您将托管在 Amazon GameLift Servers 上的自定义游戏服务器准备就绪。游戏服务器必须能够通知 Amazon GameLift Servers 自身状态，在提示时开始和停止游戏会话，以及执行其他任务。有关更多信息，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

先决条件

在集成游戏服务器之前，请完成以下任务：

- [为 Amazon GameLift Servers 设置 IAM 服务角色](#)
- [安装并设置插件](#)

设置新的服务器进程

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0（使用服务器 SDK 4.x 或更早版本）。

设置与 Amazon GameLift Servers 的通信，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 Servers 服务调用的回调函数的名称，例如 `OnGameSession()`、`OnGameSessionUpdate()`、`OnProcessTerminate()`、`OnHealthCheck()`。GameLift Servers 可能需要几分钟来提供回调。
3. Amazon GameLift Servers 将服务器进程的状态更新为 `ACTIVE`。

4. Amazon GameLift Servers 定期调用 onHealthCheck。

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

```
//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

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (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 Servers 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[适用于 Amazon GameLift Servers 4.x 的 C# 服务器 SDK – 操作](#)。

以下代码示例说明了如何使用 Amazon GameLift Servers 启动游戏会话。

```
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

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (使用服务器 SDK 4.x 或更早版本)。

游戏会话结束时通知 Amazon GameLift Servers。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 `onProcessTerminate` 的函数以接收来自 Amazon GameLift Servers 的请求并调用 `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);
```

创建服务器生成包并上传到 Amazon GameLift Servers

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (使用服务器 SDK 4.x 或更早版本)。

将游戏服务器与 Amazon GameLift Servers 集成后，将生成包文件上传到实例集中，这样 Amazon GameLift Servers 就可以将其部署到游戏托管。有关如何将服务器上传到 Amazon GameLift Servers 的更多信息，请参阅[为 Amazon GameLift Servers 创建游戏服务器生成包](#)。

将 Amazon GameLift Servers 与 Unity 游戏客户端项目集成

Note

本主题提供了适用于 Unity 的 Amazon GameLift Servers 插件的早期版本相关信息。版本 1.x 使用适用于 Amazon GameLift Servers 4.x 或更早版本的服务器 SDK。有关插件最新版本 (使用服务器 SDK 5.x 并支持 Amazon GameLift Servers Anywhere 和托管式容器托管等新功能) 的文档，请参阅[Amazon GameLift Servers 适用于 Unity 的插件 \(服务器 SDK 5.x\)](#)。

本主题可帮助您设置游戏客户端，以通过后端服务连接到 Amazon GameLift Servers 托管的游戏会话。Amazon GameLift Servers APIs 用于发起配对、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 Servers 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。

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。
- 从 Amazon GameLift Servers 服务请求有关活动游戏会话的信息。
- 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

在使用 Amazon GameLift Servers 客户端设置游戏服务器通信前，请完成以下任务：

- [设置 AWS 用户账户](#)
- [安装并设置插件](#)
- [将 Amazon GameLift Servers 与 Unity 游戏服务器项目集成](#)
- [为 Amazon GameLift Servers 部署托管实例集](#)

初始化游戏客户端

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (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 Servers 功能的必要条件。

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[设置 Amazon GameLift Servers API](#)。
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 Servers 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 Servers service by setting up a configuration.
        //The default configuration specifies a location.
        var config = new AmazonGameLiftConfig();
        config.RegionEndpoint = Amazon.RegionEndpoint.USEast1;

        CredentialProfile profile = null;
        var nscf = new SharedCredentialsFile();
        nscf.TryGetProfile(profileName, out profile);
        AWSCredentials credentials = profile.GetAWSCredentials(null);
        //Initialize Amazon GameLift Servers Client with default client
configuration.
        aglc = new AmazonGameLiftClient(credentials, config);
    }
}
```

按照特定的实例集创建游戏会话

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (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 Amazon GameLift Servers 创建了新游戏会话并返回 `GameSession` 后，便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

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (使用服务器 SDK 4.x 或更早版本)。

在 Amazon GameLift Servers 创建了新游戏会话并返回 `GameSession` 后，便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. 连接到游戏会话。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 playerId, string playerIdSessionId)
{
    //Call ConnectPlayer with player ID and player session ID.
    return server.ConnectPlayer(playerId, playerIdSessionId);
}
```

从游戏会中移除玩家

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (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 Servers 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，Amazon GameLift Servers 将玩家位置更改为可用，Amazon GameLift Servers 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 playerId)
{
    //Receive the player session ID.
    string playerIdSessionId = playerIdSessions[playerId];
    var outcome = GameLiftServerAPI.RemovePlayerSession(playerIdSessionId);
    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

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (使用服务器 SDK 4.x 或更早版本) 。

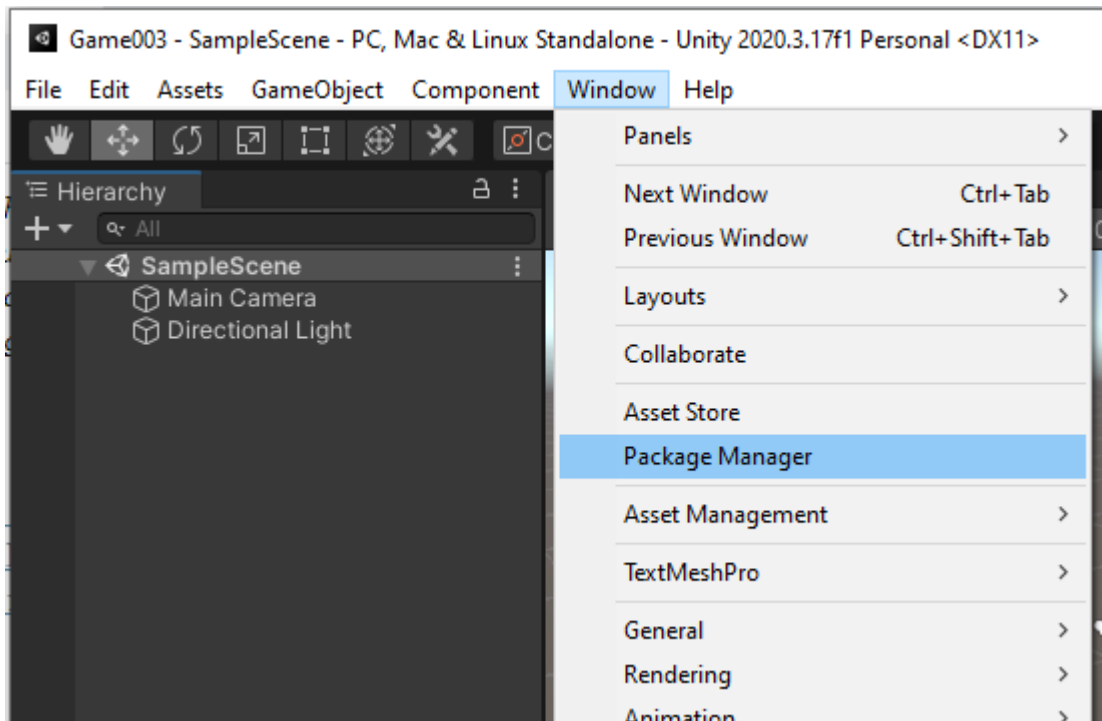
本节介绍如何下载、安装和设置适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0。

先决条件

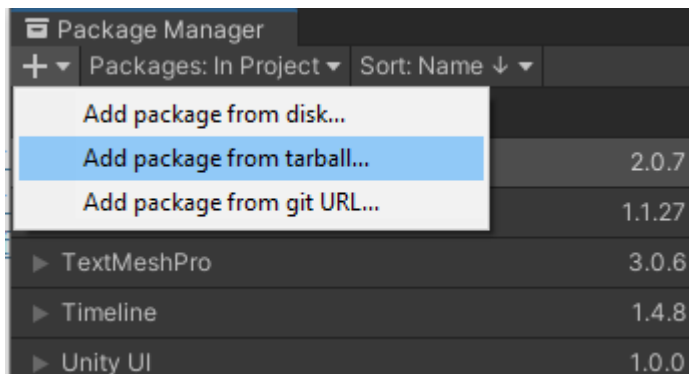
- 适用于 Windows 的 Unity 2019.4 LTS、适用于 Windows 的 2020.3 LTS 或适用于 macOS 的 Unity
- Java 的当前版本
- .NET 4.x 的当前版本

下载并安装适用于 Unity 的插件

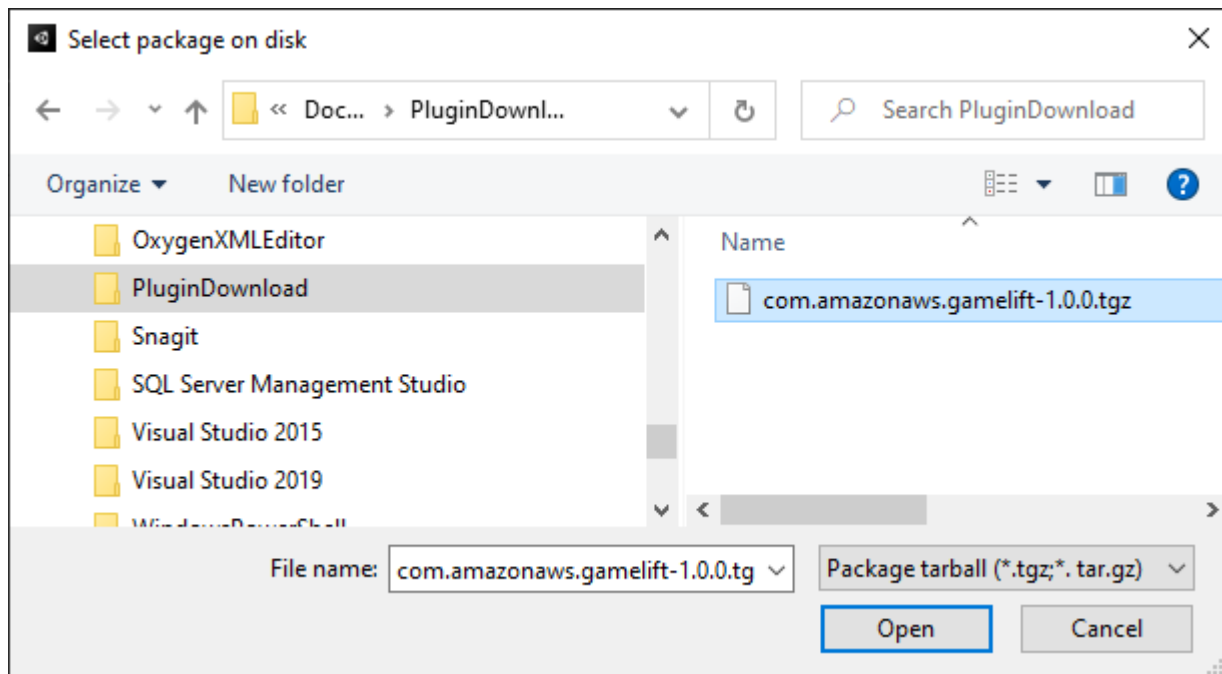
1. 下载适用于 Unity 的 Amazon GameLift Servers 插件。您可以在[适用于 Unity 存储库的 Amazon GameLift Servers 插件](#)页面上找到最新版本。在[最新版本](#)下，选择资产，然后下载 `com.amazonaws.gamelift-version.tgz` 文件。
2. 启动 Unity 并选择一个项目。
3. 在顶部导航栏的窗口下，选择 Package Manager：



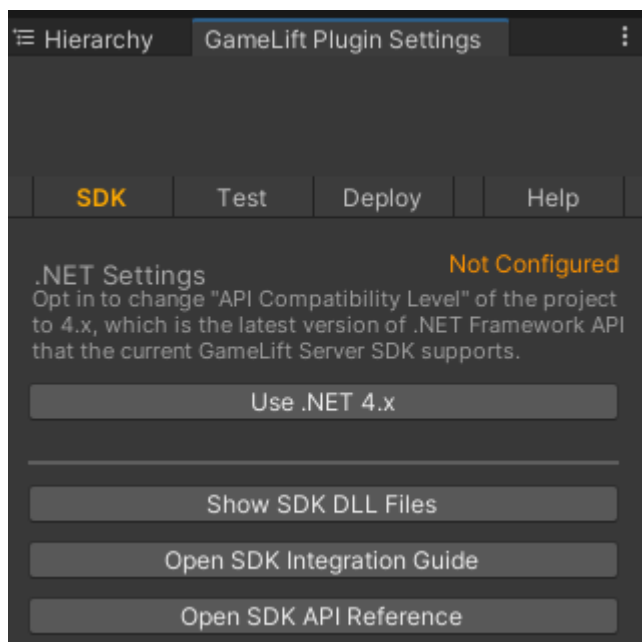
4. 在 Package Manager 选项卡下，选择 +，然后选择从 tarball 添加软件包...：



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：



- Unity 加载插件后，Amazon GameLift Servers 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。Amazon GameLift Servers 插件设置选项卡会自动打开。



- 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。

在本地测试游戏

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (使用服务器 SDK 4.x 或更早版本)。

使用 Amazon GameLift Servers Local 在本地设备上运行 Amazon GameLift Servers。无需网络连接，即可使用 Amazon GameLift Servers Local 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 GameLift Servers Local。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 Amazon GameLift Servers 插件设置窗口中，选择 Amazon GameLift Servers Local 路径，导航到解压缩的文件夹，选择 GameLiftLocal.jar 文件，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。
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

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (使用服务器 SDK 4.x 或更早版本)。

场景使用 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 Amazon GameLift Servers 提供的场景及其使用方法。

先决条件

要部署该场景，您需要为 Amazon GameLift Servers 服务创建一个 IAM 角色。有关如何为 Amazon GameLift Servers 创建角色的信息，请参阅[设置 AWS 用户账户](#)。

每种场景都需要访问以下资源的权限：

- Amazon GameLift Servers
- Amazon S3
- CloudFormation
- API Gateway
- AWS Lambda
- AWS WAFV2
- Amazon Cognito

场景

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (使用服务器 SDK 4.x 或更早版本)。

适用于 Unity 的 Amazon GameLift Servers 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

此场景创建了一个包含单个 Amazon GameLift Servers 实例集的游戏后端服务。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，用于搜索舰队上有开放玩家槽位的现有游戏会话。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

此场景使用 Amazon GameLift Servers 队列和自定义对战构建器将等候池中等待时间最长的玩家分组在一起，从而形成对战。其创建了以下资源：

- Amazon Simple Notification Service 主题，Amazon GameLift Servers 可向其发布消息。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 Servers 队列和自定义对战构建器形成对战，并配置三个实例集。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[托管资源自定义](#)。
- 用于保持服务器高可用性和低成本的 Amazon GameLift Servers 队列。有关队列的更多信息和最佳实践，请参阅[自定义游戏会话队列](#)。

FlexMatch

此场景使用 FlexMatch 托管配对服务将玩家配对在一起。有关 FlexMatch 的更多信息，请参阅[什么是 Amazon GameLift Servers 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，用于侦听 FlexMatch 匹配事件。

为避免对您产生不必要的费用 AWS 账户，请在使用完每个场景创建的资源后将其删除。删除相应的 CloudFormation 堆栈。

更新 AWS 凭证

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (使用服务器 SDK 4.x 或更早版本)。

适用于 Unity 的 Amazon GameLift Servers 插件需要安全凭证才能部署场景。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 证书](#)。

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 凭证或选择现有证书。
 - 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 凭据”窗口中，选择“更新凭据配置文件”。

更新账户引导程序

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (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。AWS 区域 您为存储桶选择的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

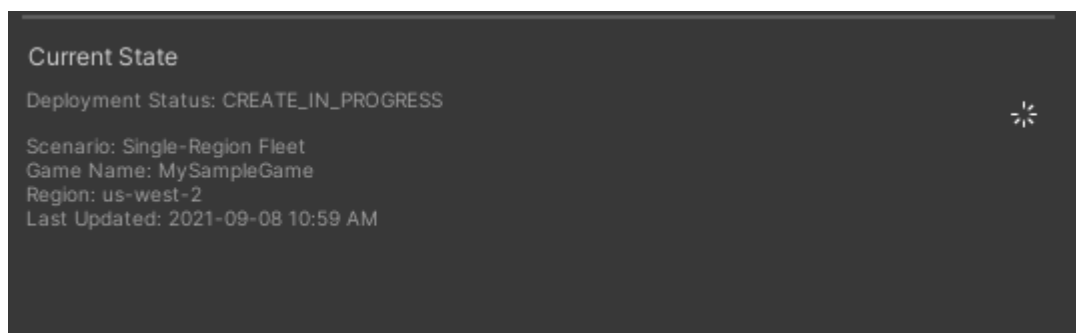
本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (使用服务器 SDK 4.x 或更早版本)。

您可以使用场景在 Amazon GameLift Servers 上测试您的游戏。每个场景都使用 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，Amazon GameLift Servers 会将游戏资产复制到引导位置，作为部署的一部分。

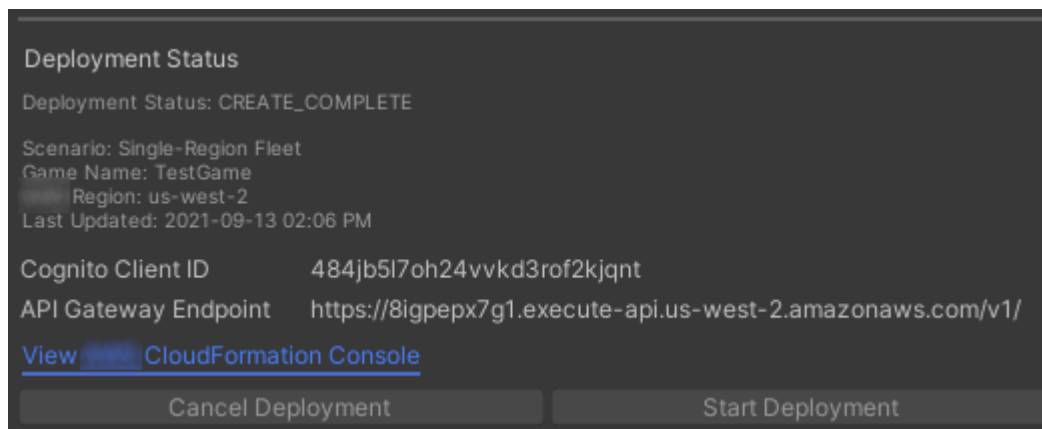
您必须配置 AWS 凭据和 AWS 账户引导才能部署方案。

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 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。部署场景可能需要数分钟。



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 Cognito 客户端 ID。选择与场景部署相同的 AWS 区域选项。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

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (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 CloudFormation 堆栈。

删除由场景创建的资源

1. 在 Unity 部署 Amazon GameLift Servers 插件窗口中，选择查看 AWS CloudFormation 控制台以打开 CloudFormation 控制台。
2. 在 CloudFormation 主机中，选择 Stacks，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。CloudFormation 删除场景使用的堆栈后，其状态将更改为 ROLLBACK_COMPLETE。

在 Unity 中将游戏与 Amazon GameLift Servers 集成

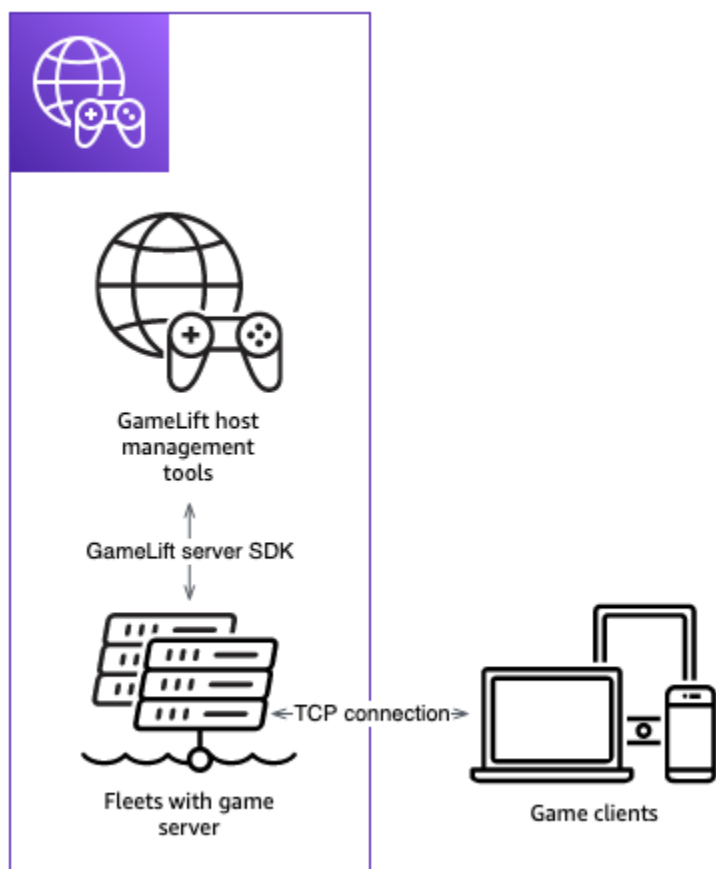
Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (使用服务器 SDK 4.x 或更早版本)。

完成以下任务，将您的 Unity 游戏与 Amazon GameLift Servers 集成：

- [将 Amazon GameLift Servers 与 Unity 游戏服务器项目集成](#)
- [将 Amazon GameLift Servers 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，带有游戏服务器的实例集部署到了 Amazon GameLift Servers。游戏客户端与游戏服务器通信，游戏服务器与 Amazon GameLift Servers 通信。



导入并运行示例游戏

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (使用服务器 SDK 4.x 或更早版本)。

适用于 Unity 的 Amazon GameLift Servers 插件包含一个示例游戏，可用于探索将游戏与 Amazon GameLift Servers 集成的基础知识。在本节中，您将构建游戏客户端和游戏服务器，然后使用 Amazon GameLift Servers Local 在本地进行测试。

先决条件

- [设置 AWS 用户账户](#)
- [安装并设置插件](#)

构建并运行示例游戏服务器

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (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 Servers，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 Amazon GameLift Servers，然后选择应用 Windows 示例服务器构建设置或应用 MacOS 示例服务器构建设置。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

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (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 Amazon GameLift Servers，然后选择应用 Windows 示例客户端构建设置或应用 MacOS 示例客户端构建设置。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 Servers 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，您将运行游戏并了解其如何与 Amazon GameLift Servers 互动。

在本地测试示例游戏

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (使用服务器 SDK 4.x 或更早版本)。

使用 Amazon GameLift Servers Local 运行导入的示例游戏。

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 Servers 本地日志窗口。窗口包含日志消息，包括游戏服务器和 Amazon GameLift Servers Local 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 Servers 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 Servers 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: playerId
0 joined
2021-09-15T19:58:0335 PID:20728 Log SERVER: Frame: 8696 Connection accepted: playerId
1 joined
2021-09-15T19:58:0338 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from playerId 0 Msg:
CONNECT: server IP localhost
2021-09-15T19:58:0338 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from player 0:CONNECT:
server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 CONNECT: player index 0
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from playerId 1 Msg:
CONNECT: server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from player 1:CONNECT:
server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 CONNECT: player index 1

```

Amazon GameLift Servers Local 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: gameId: "arn:aws:gamelift:local::gamesession/  
fleet-123/gsess-59f6cc44-4361-42f5-95b5-fdb5825c0f3d"
```

关闭服务器进程

Note

本主题指的是适用于 Unity 的 Amazon GameLift Servers 插件版本 1.0.0 (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。

将游戏服务器与 Amazon GameLift Servers 集成

您的自定义游戏服务器在托管资源上部署并运行后，必须能够与 Amazon GameLift Servers 进行交互。本节中的主题介绍如何将游戏服务器项目与 Amazon GameLift Servers 的服务器 SDK 集成，以添加必需和可选功能。本节包括可用于任何特定语言的服务器的一般说明。SDKs 还包括有关集成 Unreal Engine 和 Unity 游戏项目的详细指导。

主题

- [借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)
- [集成游戏引擎的服务器 SDK](#)

借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器

本主题提供有关需要添加到游戏服务器代码中的服务器 SDK 功能的一般指导；该功能使游戏服务器进程能够与 Amazon GameLift Servers 服务进行通信。对于计划部署到 Amazon GameLift Servers 托管式 EC2 实例集、托管式容器实例集或 Anywhere 实例集上的游戏服务器，请使用以下说明进行操作。

游戏服务器进程会与 Amazon GameLift Servers 服务通信，以接收来自该服务的指令并报告服务器进程运行状况和游戏会话状态。有关您的游戏托管解决方案各组件 (游戏服务器、后端服务、游戏客户端和 Amazon GameLift Servers) 之间交互的详细信息，请参阅[与游戏 client/server 互动 Amazon GameLift Servers](#)。

获取服务器 SDK

要为托管游戏做好准备，需将 Amazon GameLift Servers 的服务器 SDK 添加到您的游戏服务器项目中。该服务器软件开发工具包支持多种语言。有关游戏服务器工具支持（包括服务器 SDK）的更多信息，请参阅[获取 Amazon GameLift Servers 开发工具](#)。

请从 github.com/amazon-gamelift 下载适用于您的开发语言的服务器 SDK。您可在该地址找到服务器 SDK 各版本、插件及其他开发工具的存储库。

Note

如果您使用 Unreal Engine 或 Unity 进行开发，请下载适用于这些游戏引擎的 Amazon GameLift Servers 插件。服务器 SDK 已内置其中，可直接使用。有关更多信息，请参阅以下链接：

- Unreal Engine ([下载插件](#)) ([集成指南](#))
- Unity ([下载插件](#)) ([集成指南](#))

服务器软件开发工具包 API 参考：

- [适用于 Amazon GameLift Servers 的 C++ 服务器 SDK 5.x – 操作](#)
- [适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 操作](#)
- [适用于 Amazon GameLift Servers 的 C++ \(Unreal \) 服务器 SDK 5.x – 操作](#)
- [适用于 Amazon GameLift Servers 的 Go 服务器 SDK – 操作](#)

初始化服务器进程

添加代码以与 Amazon GameLift Servers 建立通信并报告游戏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 Servers 代码之前运行。

1. 通过调用 `InitSdk()` 初始化 Amazon GameLift Servers API 客户端。如果您正在准备将游戏服务器部署至 Amazon GameLift Servers 托管式 EC2 实例集，请使用默认的 `InitSDK()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#))，无需传入任何参数。API 客户端会为您处理与 Amazon GameLift Servers 服务的连接。

i 如果您正在准备要在 Amazon GameLift Servers Anywhere 实例集上使用的游戏服务器：请通过使用以下 `ServerParameters` 调用 `InitSdk()` 来初始化 Amazon GameLift Servers API 客户端：

- 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 Servers Anywhere 计算的实例集的 ID。
- Amazon GameLift Servers 操作 [GetComputeAuthToken](#) 生成的授权令牌。

2. 通知该服务，服务器进程已准备好托管游戏会话。使用以下 `ProcessParameters` 调用 `ProcessReady()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#))。每个游戏服务器进程仅需调用 `ProcessReady()` 一次。

- 服务器进程的端口号。当服务器进程启动游戏会话时，它会将该端口提供给 Amazon GameLift Servers 服务，该服务负责更新游戏会话信息。您的游戏可检索此信息并将其提供给游戏客户端，然后客户端利用该信息连接至服务器进程并加入游戏会话。
- 您希望 Amazon GameLift Servers 为您存储的文件的位置。这些文件可能包含游戏会话日志和服务器进程在游戏会话期间生成的其他文件。尽管 Amazon GameLift Servers 会在服务器进程运行的计算实例上临时保存这些文件，但这些文件仅在实例关闭前可用。您可以通过[Amazon GameLift Servers 控制台](#)或调用 Amazon GameLift Servers API 操作 [GetGameSessionLogUrl\(\)](#) 来访问存储的文件。

i 如果您正在准备要在 Amazon GameLift Servers 托管式容器实例集上使用的游戏服务器：

无需指定容器实例集的日志参数，而是将游戏会话和其他日志数据发送到标准输出。容器实例集会 自动将所有容器标准输出捕获为日志流。

- 以下回调函数允许 Amazon GameLift Servers 向游戏服务器进程发送消息或提示。您必须在游戏服务器代码中实现这些函数。有关更多信息，请参阅 `ProcessParameters` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#))。
- (可选) `onHealthCheck` : Amazon GameLift Servers 定期调用此函数向服务器请求运行状况报告。

- `onStartGameSession`— 响应客户端请求Amazon GameLift Servers调用此函数 [CreateGameSession\(\)](#)。
- `onProcessTerminate` : Amazon GameLift Servers 强制服务器进程停止，让它正常关闭。
- (可选) `onUpdateGameSession` : Amazon GameLift Servers 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。

您还可以设置游戏服务器，使其能够安全地访问您拥有或控制的其他 AWS 资源。有关更多信息，请参阅 [将您的Amazon GameLift Servers托管游戏服务器连接到其他 AWS 资源](#)。

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 Servers 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 Servers 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 Servers 会认为服务器进程正常，除非服务器不响应。

Amazon GameLift Servers 服务会依据服务器进程运行状况，终止异常进程并释放资源。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，该服务可能会关闭该进程然后启动新进程。该服务会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](#)) ([Go](#))。

启动游戏会话

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

该 `onStartGameSession` 函数将 [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 Servers 服务。调用服务器 API 操作 `ActivateGameSession()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#))。作为对成功调用的响应，该服务会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 Servers 会跟踪当前玩家及可用的游戏会话位置。

为了进行验证，尝试加入游戏会话的游戏客户端必须包含玩家会话 ID。Amazon GameLift Servers 当您的游戏通过调用 [StartGameSessionPlacement\(\)](#) 或 [StartMatchmaking\(\)](#) 开始新的游戏会话时生成此 ID。在这些请求中，游戏会话会为该玩家会话预留一个空闲位置。

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](#)) ([Go](#))。作为响应，Amazon GameLift Servers 会验证玩家会话 ID 是否与游戏会话中预留的空闲位置相对应。在 Amazon GameLift Servers 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 Servers 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 Servers。只要服务器进程检测到断开的连接，此代码就应运行。Amazon GameLift Servers 会使用此通知跟踪游戏会话中的当前玩家及可用玩家位置。

要在代码中处理断开的连接，请使用相应的玩家会话 ID 添加对服务器 API 操作 `RemovePlayerSession()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) 的调用。

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 Servers。为回收和刷新托管资源，请在游戏会话完成后关闭每个服务器进程。

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](#)) ([Go](#))。此调用会通知 Amazon GameLift Servers，该服务器进程正在关闭。Amazon GameLift Servers 会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 Servers 的通知。当服务器进程持续报告运行状况异常或服务器进程正在运行的实例被终止时，该服务会发送此通知。Amazon GameLift Servers 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。竞价型实例中断时会提供 2 分钟通知期，让服务器进程有足够时间正常断开玩家的连接、保存游戏状态数据，并执行其他清理任务。

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()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#))。此函数应调用用于关闭游戏服务器的代码。
- 从游戏服务器关闭代码中调用服务器 API 操作 `GetTerminationTime()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#))。如果 Amazon GameLift Servers 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](#)) ([Go](#))。此调用将通知 Amazon GameLift Servers 服务，服务器进程正在关闭。然后，该服务会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。

集成游戏引擎的服务器 SDK

您可以将托管的 Amazon GameLift Servers 服务与支持 C++ 或 C# 库的大多数主要游戏引擎配合使用，包括 O3DE、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 Servers SDKs、支持的开发平台和操作系统的更多信息，[获取 Amazon GameLift Servers 开发工具](#) 请参阅“游戏服务器”。

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 Amazon GameLift Servers 的更多帮助：

- [借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#) - 在游戏服务器中集成 Amazon GameLift Servers 的详细说明。
- [集成 Amazon GameLift Servers 游戏客户端功能](#) - 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。

Unreal Engine

游戏服务器

通过将[适用于 Unreal Engine 的 Amazon GameLift Servers 服务器 SDK](#) 添加到您的项目并实现所需的服务器功能，在 Amazon GameLift Servers 上准备游戏服务器的托管。有关设置 Unreal Engine 插件和添加 Amazon GameLift Servers 代码的帮助，请参阅[将 Amazon GameLift Servers 集成到 Unreal Engine 项目中](#)。

游戏客户端和服务

使您的游戏客户端 and/or 游戏服务能够与 Amazon GameLift Servers 服务进行交互，例如查找可用的游戏会话或创建新会话，以及将玩家添加到游戏中。[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 Servers 集成到 Unreal Engine 游戏项目中，请参阅[集成 Amazon GameLift Servers 游戏客户端功能](#)。

Unity

游戏服务器

通过将[适用于 C# 的 Amazon GameLift Servers 服务器 SDK](#) 添加到您的项目并实现所需的服务器功能，在 Amazon GameLift Servers 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 Servers 代码的帮助，请参阅[将 Amazon GameLift Servers 集成到 Unity 项目中](#)。

游戏客户端和服务

使您的游戏客户端 and/or 游戏服务能够与 Amazon GameLift Servers 服务进行交互，例如查找可用的游戏会话或创建新会话，以及将玩家添加到游戏中。[适用于 .NET 的 AWS SDK](#) 中提供了核心客户端功能。要将 Amazon GameLift Servers 集成到 Unity 游戏项目中，请参阅[集成 Amazon GameLift Servers 游戏客户端功能](#)。

其他引擎

有关适用于游戏服务器和客户端的完整列表，请参阅[the section called “获取 开发工具”](#)。Amazon GameLift Servers SDKs

将 Amazon GameLift Servers 集成到 Unreal Engine 项目中

了解如何将适用于 Unreal Engine 的 Amazon GameLift Servers SDK 集成到您的游戏项目中，以访问完整的服务器 SDK 功能集。

 提示

为实现更快速的部署，请尝试使用适用于 Unreal Engine 的 Amazon GameLift Servers 独立插件。它提供了引导式 UI 工作流程，可通过最少的设置快速部署游戏服务器，助您即时测试游戏组件的运行效果。请参阅[Amazon GameLift Servers 虚幻引擎插件](#)。

其他资源

- [适用于 Amazon GameLift Servers 的 C++ \(Unreal \) 服务器 SDK 5.x – 操作](#)
- [获取 Amazon GameLift Servers 开发工具](#)

安装适用于 Unreal 的服务器 SDK

从这里获取虚幻引擎的开源 Amazon GameLift Servers SDK [GitHub](#)。该存储库的自述文件包含先决条件和安装说明。

设置生成包目标和模块规则

修改您的游戏项目文件，以正确生成可与 Amazon GameLift Servers 搭配使用的生成包组件。

要添加客户端和服务端生成包目标，请执行以下操作：

1. 打开游戏项目的代码文件并找到 `.../Games/[your application name]Source/[your application name]Target.cs` 文件。示例：`.../Source/GameLiftUnrealAppTarget.cs`。（如果您使用 Visual Studio，请打开项目的 `.sln` 文件。）
2. 复制此文件，以在 `Source/` 目录中创建两个新的目标文件。
 - 客户端目标：将新文件重命名为 `[your application name]Client.Target.cs`。编辑内容以更新类名称和目标类型值，如以下示例代码所示：

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

public class GameLiftUnrealAppClientTarget : TargetRules
{
    public GameLiftUnrealAppClientTarget ( TargetInfo Target ) : base
    ( Target )
    {
        Type = TargetType.Client;
```

```

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

```

- 服务器目标：将新文件重命名为 *[your application name]*Server.Target.cs。编辑内容以更新类名称和目标类型值，如以下示例代码所示：

```

using UnrealBuildTool;
using System.Collections.Generic;

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

```

3. 重新生成您的项目文件。如果您使用 Visual Studio，可以右键单击游戏项目的 .uproject 文件，然后选择生成 Visual Studio 项目文件。

要更新游戏项目模块规则，请执行以下操作：

更新游戏项目的模块规则，使其依赖于插件。

1. 打开游戏项目的代码文件并找到 .../Games/*[your application name]*Source/*[your application name]*.Build.cs 文件。示例：.../Source/GameLiftUnrealApp.Build.cs。（如果您使用 Visual Studio，请打开项目的 .sln 文件。）
2. 找到 ModuleRules 类并进行更新，如以下示例代码所示：

```

using UnrealBuildTool;

public class GameLiftUnrealApp : ModuleRules
{
    public GameLiftUnrealApp ( ReadOnlyTargetRules Target ) : base ( Target )
    {

```

```

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

```

3. 创建新的目标文件并修改模块规则后，重新构建您的游戏项目。

为您的服务器代码添加游戏托管功能

在完成服务器 SDK 的安装和配置后，下一步是将游戏托管功能集成到服务器代码中。服务器 SDK 使您的游戏服务器能够与 Amazon GameLift Servers 服务通信，接收游戏会话指令，报告状态和运行状况，并执行其他操作。

本主题提供示例代码，用于添加使用 Amazon GameLift Servers 托管游戏所需的最低功能。

步骤 1：更新 **GameMode** 标头文件

1. 打开游戏项目的代码文件并找到 *Your-application-name*GameMode.h 文件。示例：GameLiftUnrealAppGameMode.h。如果您使用 Visual Studio，请打开游戏项目的 .sln 文件。
2. 更改标头文件以包含以下示例代码。请务必用您自己的应用程序名称替换 GameLiftUnrealApp “”。

GameMode.h 的示例代码

```

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

#pragma once

```

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

struct FProcessParameters;

DECLARE_LOG_CATEGORY_EXTERN(GameServerLog, Log, All);

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

public:
    AGameLiftUnrealAppGameMode();

protected:
    virtual void BeginPlay() override;

private:
    void InitGameLift();

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

步骤 2：将所需的服务器 SDK 调用添加到游戏服务器代码中

使用本节中的示例代码，将您的游戏服务器代码集成到 Amazon GameLift Servers 服务器中。有关代码功能的详细信息，请参阅[初始化服务器进程](#)和[适用于 Amazon GameLift Servers 的 C++ \(Unreal \) 服务器 SDK 5.x – 操作](#)。

Note

WITH_GAMELIFT 预处理器标志有两个用途：

- 将 Amazon GameLift Servers 后端 API 调用限制为仅限 Unreal 服务器生成包
- 确保不同 Unreal 生成包目标之间的兼容性

1. 打开相关的源文件 *Your-application-name*GameMode.cpp 文件。在本示例中：GameLiftUnrealAppGameMode.cpp。

2. 更改代码以使其与以下示例代码保持一致。请务必用您自己的应用程序名称替换 `GameLiftUnrealApp` 的任何实例。

提供的代码示例显示了如何添加与 Amazon GameLift Servers 集成所需的元素。这些方法包括：

- 初始化 Amazon GameLift Servers API 客户端。
- 实现回调函数以响应 Amazon GameLift Servers 服务的请求，包括 `OnStartGameSession`、`OnProcessTerminate` 和 `onHealthCheck`。
- 准备好托管游戏会话时，请调用 `ProcessReady()` 以通知 Amazon GameLift Servers 服务。

GameMode.cpp 的示例代码

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

#include "GameLiftUnrealAppGameMode.h"

#include "UObject/ConstructorHelpers.h"
#include "Kismet/GameplayStatics.h"

#if WITH_GAMELIFT
#include "GameLiftServerSDK.h"
#include "GameLiftServerSDKModels.h"
#endif

#include "GenericPlatform/GenericPlatformOutputDevices.h"

DEFINE_LOG_CATEGORY(GameServerLog);

AGameLiftUnrealAppGameMode::AGameLiftUnrealAppGameMode() :
    ProcessParameters(nullptr)
{
    // Set default pawn class to our Blueprinted character
    static ConstructorHelpers::FClassFinder<APawn> PlayerPawnBPClass(TEXT("/Game/
ThirdPerson/Blueprints/BP_ThirdPersonCharacter"));

    if (PlayerPawnBPClass.Class != NULL)
    {
        DefaultPawnClass = PlayerPawnBPClass.Class;
    }
}
```

```

    UE_LOG(GameServerLog, Log, TEXT("Initializing AGameLiftUnrealAppGameMode..."));
}

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

#ifdef WITH_GAMELIFT
    InitGameLift();
#endif
}

void AGameLiftUnrealAppGameMode::InitGameLift()
{
#ifdef WITH_GAMELIFT
    UE_LOG(GameServerLog, Log, TEXT("Calling InitGameLift..."));

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

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

    bool bIsAnywhereActive = false;
    if (FParse::Param(FCommandLine::Get(), TEXT("glAnywhere")))
    {
        bIsAnywhereActive = true;
    }

    if (bIsAnywhereActive)
    {
        UE_LOG(GameServerLog, Log, TEXT("Configuring server parameters for
Anywhere..."));

        // If GameLift Anywhere is enabled, parse command line arguments and pass them
        in the ServerParameters object.
        FString glAnywhereWebSocketUrl = "";
        if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereWebSocketUrl="),
glAnywhereWebSocketUrl))
        {
            ServerParametersForAnywhere.m_webSocketUrl =
TCHAR_TO_UTF8(*glAnywhereWebSocketUrl);

```

```

    }

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

    FString glAnywhereProcessId = "";
    if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereProcessId="),
glAnywhereProcessId))
    {
        ServerParametersForAnywhere.m_processId =
TCHAR_TO_UTF8(*glAnywhereProcessId);
    }
    else
    {
        // If no ProcessId is passed as a command line argument, generate a
randomized unique string.
        FString TimeString = FString::FromInt(std::time(nullptr));
        FString ProcessId = "ProcessId_" + TimeString;
        ServerParametersForAnywhere.m_processId = TCHAR_TO_UTF8(*ProcessId);
    }

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

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

    FString glAnywhereAwsRegion = "";
    if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereAwsRegion="),
glAnywhereAwsRegion))
    {

```

```

        ServerParametersForAnywhere.m_awsRegion =
TCHAR_TO_UTF8(*glAnywhereAwsRegion);
    }

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

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

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

    UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_YELLOW);
    UE_LOG(GameServerLog, Log, TEXT(">>>> WebSocket URL: %s"),
*ServerParametersForAnywhere.m_webSocketUrl);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Fleet ID: %s"),
*ServerParametersForAnywhere.m_fleetId);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Process ID: %s"),
*ServerParametersForAnywhere.m_processId);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Host ID (Compute Name): %s"),
*ServerParametersForAnywhere.m_hostId);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Auth Token: %s"),
*ServerParametersForAnywhere.m_authToken);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Aws Region: %s"),
*ServerParametersForAnywhere.m_awsRegion);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Access Key: %s"),
*ServerParametersForAnywhere.m_accessKey);

```

```

        UE_LOG(GameServerLog, Log, TEXT(">>>> Secret Key: %s"),
*ServerParametersForAnywhere.m_secretKey);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Session Token: %s"),
*ServerParametersForAnywhere.m_sessionToken);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }

    UE_LOG(GameServerLog, Log, TEXT("Initializing the GameLift Server..."));

    //InitSDK will establish a local connection with GameLift's agent to enable further
communication.
    FGameLiftGenericOutcome InitSdkOutcome = GameLiftSdkModule-
>InitSDK(ServerParametersForAnywhere);
    if (InitSdkOutcome.IsSuccess())
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_GREEN);
        UE_LOG(GameServerLog, Log, TEXT("GameLift InitSDK succeeded!"));
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }
    else
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_RED);
        UE_LOG(GameServerLog, Log, TEXT("ERROR: InitSDK failed : ("));
        FGameLiftError GameLiftError = InitSdkOutcome.GetError();
        UE_LOG(GameServerLog, Log, TEXT("ERROR: %s"), *GameLiftError.m_errorMessage);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
        return;
    }

    ProcessParameters = MakeShared<FProcessParameters>();

    //When a game session is created, Amazon GameLift Servers sends an activation
request to the game server and passes along the game session object containing game
properties and other settings.
    //Here is where a game server should take action based on the game session object.
    //Once the game server is ready to receive incoming player connections, it should
invoke GameLiftServerAPI.ActivateGameSession()
    ProcessParameters->OnStartGameSession.BindLambda([=]
(Aws::GameLift::Server::Model::GameSession InGameSession)
    {
        FString GameSessionId = FString(InGameSession.GetGameSessionId());
        UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"),
*GameSessionId);
        GameLiftSdkModule->ActivateGameSession();
    }

```

```

});

//OnProcessTerminate callback. Amazon GameLift Servers will invoke this callback
before shutting down an instance hosting this game server.
//It gives this game server a chance to save its state, communicate with services,
etc., before being shut down.
//In this case, we simply tell Amazon GameLift Servers we are indeed going to
shutdown.
ProcessParameters->OnTerminate.BindLambda([=]()
{
    UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
    // First call ProcessEnding()
    FGameLiftGenericOutcome processEndingOutcome = GameLiftSdkModule-
>ProcessEnding();
    // Then call Destroy() to free the SDK from memory
    FGameLiftGenericOutcome destroyOutcome = GameLiftSdkModule->Destroy();
    // Exit the process with success or failure
    if (processEndingOutcome.IsSuccess() && destroyOutcome.IsSuccess()) {
        UE_LOG(GameServerLog, Log, TEXT("Server process ending successfully"));
    }
    else {
        if (!processEndingOutcome.IsSuccess()) {
            const FGameLiftError& error = processEndingOutcome.GetError();
            UE_LOG(GameServerLog, Error, TEXT("ProcessEnding() failed. Error:
%s"),
                error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
*error.m_errorMessage);
        }
        if (!destroyOutcome.IsSuccess()) {
            const FGameLiftError& error = destroyOutcome.GetError();
            UE_LOG(GameServerLog, Error, TEXT("Destroy() failed. Error: %s"),
                error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
*error.m_errorMessage);
        }
    }
});

//This is the HealthCheck callback.
//Amazon GameLift Servers will invoke this callback every 60 seconds or so.
//Here, a game server might want to check the health of dependencies and such.
//Simply return true if healthy, false otherwise.
//The game server has 60 seconds to respond with its health status. Amazon GameLift
Servers will default to 'false' if the game server doesn't respond in time.
//In this case, we're always healthy!

```

```

ProcessParameters->OnHealthCheck.BindLambda([]()
{
    UE_LOG(GameServerLog, Log, TEXT("Performing Health Check"));
    return true;
});

//GameServer.exe -port=7777 LOG=server.mylog
ProcessParameters->port = FURL::UrlConfig.DefaultPort;
TArray<FString> CommandLineTokens;
TArray<FString> CommandLineSwitches;

FCommandLine::Parse(FCommandLine::Get(), CommandLineTokens, CommandLineSwitches);

for (FString SwitchStr : CommandLineSwitches)
{
    FString Key;
    FString Value;

    if (SwitchStr.Split("=", &Key, &Value))
    {
        if (Key.Equals("port"))
        {
            ProcessParameters->port = FString::Atoi(*Value);
        }
    }
}

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

//The game server calls ProcessReady() to tell Amazon GameLift Servers it's ready
to host game sessions.
UE_LOG(GameServerLog, Log, TEXT("Calling Process Ready..."));
FGameLiftGenericOutcome ProcessReadyOutcome = GameLiftSdkModule-
>ProcessReady(*ProcessParameters);

if (ProcessReadyOutcome.IsSuccess())
{

```



```

        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_GREEN);
        UE_LOG(GameServerLog, Log, TEXT("Process Ready!"));
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }
    else
    {
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_RED);
        UE_LOG(GameServerLog, Log, TEXT("ERROR: Process Ready Failed!"));
        FGameLiftError ProcessReadyError = ProcessReadyOutcome.GetError();
        UE_LOG(GameServerLog, Log, TEXT("ERROR: %s"),
*ProcessReadyError.m_errorMessage);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }

    UE_LOG(GameServerLog, Log, TEXT("InitGameLift completed!"));
#endif
}

```

步骤 3：重建游戏项目

- 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。只需在应用程序的 /Games/ 文件夹下构建项目即可。否则，Visual Studio 将重建整个 UE5 项目，这可能需要长达一个小时的时间。

打包游戏服务器以便托管

现在，您的游戏服务器代码已集成所需的最低服务器 SDK 功能，您已准备好使用 Unreal 编辑器打包游戏服务器生成包。

打包游戏服务器生成包

1. 在 Unreal 编辑器中打开游戏项目。
2. 按照 Unreal 编辑器的步骤来打包您的游戏服务器：
 - 选择目标平台 (Windows 或 Linux)。
 - 选择服务器构建目标 (*[your application name]*Server)。

打包过程会生成您的游戏服务器可执行文件：*[your application name]*Server.exe。

3. 准备好您的游戏服务器生成包，以便部署到托管资源。生成包应包含以下文件：

- 游戏服务器可执行文件
- 如果您使用的是 Unreal Engine 版本 5.5 或更早版本，请为 Windows 生成包包含以下文件。您可在源代码构建版本的 Unreal Engine 中找到这些文件：
 - VC_redist.x64.exe (UnrealEngine\Engine\Source\Programs\PrereqInstaller\Resources\VCRedist\)
 - UEPrereqSetup_x64.exe or UE5PrereqSetup_x64.exe (UnrealEngine\Engine\Extras\Redist\en-us\)
- 游戏服务器所需的所有其他依赖项。
- 如果需要，可以使用 OpenSSL 库。如果您的游戏服务器与 Amazon GameLift Servers 服务器 SDK 5.3 版本或更高版本集成，可以跳过此步骤。[最新的服务器 SDK 版本可在此处获取。](#)

使用适用于 Unreal 的服务器 SDK (5.2 版本及更早版本) 创建的游戏服务器所需的 OpenSSL 库

您必须包含与在 Unreal 中打包游戏服务器时相同的 OpenSSL 库版本。这些库位于您的游戏引擎源代码中。具体位置因开发环境而异：

在 Windows 上：

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

在 Linux 上：

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

将 OpenSSL 库复制到游戏生成包中，使其与游戏服务器可执行文件位于同一目录下。

后续步骤

既然您已准备好具备在 Amazon GameLift Servers 上托管所需最低功能的游戏服务器生成包，请考虑以下潜在后续步骤：

- 部署您的集成游戏服务器以进行测试和开发。借助 Anywhere 实例集，您可以将本地计算机设置为托管资源，并用其测试游戏服务器与游戏客户端连接。对于基于云的托管，请将游戏服务器部署到托管式 EC2 实例集或托管式容器实例集。请参阅以下主题获取指导：
 - [借助 Amazon GameLift Servers Anywhere 为迭代开发做好准备](#)
 - [Amazon GameLift Servers Anywhere 实例集](#)
 - [Amazon GameLift Servers 托管 EC2 车队](#)
 - [Amazon GameLift Servers 托管式容器实例集](#)
- 通过添加可选功能，自定义游戏服务器集成。例如，您可能想要添加与唯一玩家的玩家会话 IDs、设置配对回填或管理游戏服务器对其他 AWS 资源（例如数据库或内容存储服务）的访问权限。请参阅以下主题获取指导：
 - [借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)
 - [适用于 Amazon GameLift Servers 的 C++ \(Unreal \) 服务器 SDK 5.x – 操作](#)
- 自定义您的游戏客户端组件，以请求游戏会话、接收连接信息并直接连接到游戏服务器进行游戏。请参阅以下主题获取指导：
 - [集成 Amazon GameLift Servers 游戏客户端功能](#)

将 Amazon GameLift Servers 集成到 Unity 项目中

了解如何将适用于 Unity 的 Amazon GameLift Servers SDK 集成到您的游戏项目中，以访问完整的服务器 SDK 功能集。

提示

为实现更快速的部署，请尝试使用适用于 Unity 的 Amazon GameLift Servers 独立插件。它提供了引导式 UI 工作流程，可通过最少的设置快速部署游戏服务器，助您即时测试游戏组件的运行效果。请参阅[Amazon GameLift Servers 适用于 Unity 的插件 \(服务器 SDK 5.x \)](#)。

其他资源

- [适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 操作](#)

- [the section called “获取 开发工具”](#)

安装适用于 Unity 的服务器 SDK

从这里获取 Unity Amazon GameLift Servers 的开源代码[GitHub](#)。该存储库的自述文件包含先决条件和安装说明。

设置 Amazon GameLift Servers Anywhere 实例集进行测试

您可以将开发工作站设置为 Amazon GameLift Servers Anywhere 托管实例集，以迭代测试您的 Amazon GameLift Servers 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 Servers 发送玩家加入或对战请求以启动游戏会话，并将客户端连接到新游戏会话。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 Servers 集成的各个方面。

有关设置工作站的说明，请参阅[使用 Amazon GameLift Servers Anywhere 设置本地测试](#)以完成以下步骤：

1. 为您的工作站创建自定义位置。
2. 使用新的自定义位置创建 Amazon GameLift Servers 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 实例集内的计算。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 URL 的形式返回服务 SDK 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其来验证与 Amazon GameLift Servers 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 Servers 服务器代码添加到 Unity 项目中

您的游戏服务器通过与 Amazon GameLift Servers 服务通信来接收指令并报告持续状态。为此，您需要添加使用 Amazon GameLift Servers 服务器 SDK 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 MonoBehaviour 演示如何通过 Amazon GameLift Servers 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 GameLift Servers Anywhere 实例集上运行以进行测试。它包括以下代码：

- 初始化 Amazon GameLift Servers API 客户端。该示例使用带有服务器参数的 InitSDK() 版本，这些参数适用于您的 Anywhere 实例集和计算。使用上一主题中定义 WebSocket 的 URL、队列

ID、计算名称 (主机 ID) 和身份验证令牌[设置 Amazon GameLift Servers Anywhere 实例集进行测试](#)。

- 实现回调函数以响应 Amazon GameLift Servers 服务的请求，包括 OnStartGameSession、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 Servers 服务。

提供的示例代码与 Amazon GameLift Servers 服务建立了通信，并实现了一组用于响应 Amazon GameLift Servers 服务请求的回调函数。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 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 操作](#)。

示例代码展示了如何添加所需功能，如[将 Amazon GameLift Servers 添加到您的游戏服务器](#)中所述。有关服务器 SDK 操作的更多信息，请参阅[适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 操作](#)。

集成示例代码

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

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

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

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

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

```

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

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

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

//InitSDK establishes a local connection with an Amazon GameLift Servers agent
//to enable further communication.
var initSDKOutcome = GameLiftServerAPI.InitSDK(serverParameters);
if (initSDKOutcome.Success)
{
    //Implement callback functions
    ProcessParameters processParameters = new ProcessParameters(
        //Implement OnStartGameSession callback
        (gameSession) => {
            //Amazon GameLift Servers sends a game session activation request
            //to the game server
            //with game session object containing game properties and other
            //settings.
            //Here is where a game server takes action based on the game
            //session object.
            //When the game server is ready to receive incoming player
            //connections,
            //it invokes the server SDK call ActivateGameSession().
            GameLiftServerAPI.ActivateGameSession();
        },
        (updateGameSession) => {
            //Amazon GameLift Servers sends a request when a game session is
            //updated (such as for
            //FlexMatch backfill) with an updated game session object.
            //The game server can examine matchmakerData and handle new
            //incoming players.
            //updateReason explains the purpose of the update.
        },
        () => {

```

```

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

    //The game server calls ProcessReady() to tell Amazon GameLift Servers it's
ready to host game sessions.
    var processReadyOutcome =
GameLiftServerAPI.ProcessReady(processParameters);
    if (processReadyOutcome.Success)
    {
        print("ProcessReady success.");
    }

```



```

        }
        else
        {
            print("ProcessReady failure : " +
processReadyOutcome.Error.ToString());
        }
    }
    else
    {
        print("InitSDK failure : " + initSDKOutcome.Error.ToString());
    }
}

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

```

后续步骤

既然您已准备好具备在 Amazon GameLift Servers 上托管所需最低功能的游戏服务器生成包，请考虑以下潜在后续步骤：

- 部署您的集成游戏服务器以进行测试和开发。借助 Anywhere 实例集，您可以将本地计算机设置为托管资源，并用其测试游戏服务器与游戏客户端连接。对于基于云的托管，请将游戏服务器部署到托管式 EC2 实例集或托管式容器实例集。请参阅以下主题获取指导：
 - [借助 Amazon GameLift Servers Anywhere 为迭代开发做好准备](#)

- [Amazon GameLift Servers Anywhere 实例集](#)
- [Amazon GameLift Servers 托管 EC2 车队](#)
- [Amazon GameLift Servers 托管式容器实例集](#)
- 通过添加可选功能，自定义游戏服务器集成。例如，您可能想要添加与唯一玩家的玩家会话 IDs、设置配对回填或管理游戏服务器对其他 AWS 资源（例如数据库或内容存储服务）的访问权限。请参阅以下主题获取指导：
 - [借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)
 - [适用于 Amazon GameLift Servers 的 C++ \(Unreal \) 服务器 SDK 5.x – 操作](#)
- 自定义您的游戏客户端组件，以请求游戏会话、接收连接信息并直接连接到游戏服务器进行游戏。请参阅以下主题获取指导：
 - [集成 Amazon GameLift Servers 游戏客户端功能](#)

打包游戏服务器生成包用于部署

准备好游戏服务器软件，以便在托管资源上部署。部署完成后，该软件将安装到每项托管资源中，随后启动一个或多个游戏服务器进程，准备为玩家提供游戏会话托管服务。

游戏服务器生成包的部署准备流程取决于您使用的 Amazon GameLift Servers 托管选项的类型。所有游戏服务器生成包均需与适用于 Amazon GameLift Servers 的服务器软件开发工具包集成，如[将游戏服务器与 Amazon GameLift Servers 集成](#)中所述。

本节中的主题将指导您如何让软件做好在以下场景中部署的准备。

- 对于 EC2 托管主机，请打包您的服务器软件并将其上传到以 Amazon GameLift Servers 进行部署。
- 对于托管式容器托管，使用您的服务器软件构建容器映像，并将其存储在 Amazon Elastic Container Registry 中以完成部署。
- 对于使用 Amazon GameLift Servers Anywhere 进行托管，按需打包服务器软件，适配自有托管资源的安装要求。

Note

如果您要部署 Amazon GameLift Servers Realtime 配置的脚本，请参阅[上传适用于 Amazon GameLift Servers Realtime 的脚本](#)。

主题

- [为 Amazon GameLift Servers 创建游戏服务器生成包](#)
- [为 Amazon GameLift Servers 构建容器映像](#)

为 Amazon GameLift Servers 创建游戏服务器生成包

将游戏服务器与 Amazon GameLift Servers 集成后（请参阅[准备使用 Amazon GameLift Servers 托管游戏](#)），请将游戏服务器软件安装到您的计算资源进行托管。此过程因您使用的 Amazon GameLift Servers 托管类型而异。

针对托管式托管进行部署

如果您使用的是 Amazon GameLift Servers 托管式 EC2 托管，则必须打包游戏服务器软件并将其上传到 Amazon GameLift Servers。当您创建托管式实例集时，Amazon GameLift Servers 会自动将其部署到每个实例集实例。

本节中的主题介绍如何打包编译文件以供上传，如何创建可选的构建安装脚本，然后使用 [AWS Command Line Interface \(AWS CLI\)](#) 或 AWS SDK 上传文件。

针对 Anywhere 托管进行部署

如果您使用 Amazon GameLift Servers 实例集进行自管理式托管，则需自行负责将游戏服务器软件安装到实例集中的每台计算设备并保持更新。

当集成后的游戏服务器进程开始运行时，它会自动初始化并与 Amazon GameLift Servers 服务建立通信。服务器进程将根据 Amazon GameLift Servers 的提示启动游戏会话，并将活动报告给该服务。

主题

- [打包游戏生成包文件](#)
- [添加构建安装脚本](#)
- [为托管主机创建 Amazon GameLift Servers 构建资源](#)

打包游戏生成包文件

在将您配置的游戏服务器上传到 Amazon GameLift Servers 之前，请将游戏生成包文件打包到生成包目录中。此过程对于使用 EC2 托管式实例集进行托管是必需的，也是使用 Anywhere 实例集进行托管时的最佳实践。生成包目录应包含运行游戏服务器和托管游戏会话所需的所有组件。这可能包括：

- 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[获取 Amazon GameLift Servers 开发工具](#)。
- 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 Servers 服务器 SDK 创建的游戏生成包（包括使用 Unreal 插件创建的生成包），需包含与构建服务器 SDK 时所用 OpenSSL 版本一致的 OpenSSL DLL 文件。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 Servers 托管服务器上安装游戏生成包的任务的脚本文件。将此文件放置到生成包目录的根目录中。Amazon GameLift Servers 会在创建实例集的过程中运行该安装脚本。

您可以设置版本中的任何应用程序（包括安装脚本），以便在其他 AWS 服务上安全地访问您的资源。有关如何执行此操作的信息，请参阅[将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。

在您打包生成包文件后，请确保您的游戏服务器可以在目标操作系统的干净安装上运行，以验证是否包含所有必需的依赖项以及安装脚本是否准确。

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 Servers 在托管服务器上的以下位置安装生成包并重新创建生成包中的文件目录：
 - 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 Servers 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 Servers 实例集实例](#)。
- 安装脚本不能依赖于 VPC 对等连接。直到 Amazon GameLift Servers 在实例集实例上安装生成包后，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 Servers 在服务 VPC 中运行，因此您必须向 Amazon GameLift Servers 授予代表您担任 AWS Identity and Access Management (IAM) 角色的权限。Amazon GameLift Servers 要允许代入角色，请创建一个包含 AWS 托管策略的角色 `CloudWatchAgentAdminPolicy`，并在创建队列时使用该角色。

```
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

```

为托管主机创建Amazon GameLift Servers构建资源

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 Servers 都会使用唯一的生成包 ID 和其他元数据创建新的生成包资源。生成以已初始化状态开始。Amazon GameLift Servers 获取游戏服务器文件后，生成包将变为就绪状态。

生成包准备就绪后，您可以将其部署到新的 Amazon GameLift Servers 实例集中。有关更多信息，请参阅[创建 Amazon GameLift Servers 托管 EC2 舰队](#)。当 Amazon GameLift Servers 设置新实例集时，它会将生成包文件下载到每个实例集实例，并安装生成包文件。

从文件目录创建构建

要创建存储在任何位置（包括本地目录）中的游戏版本，请使用[upload-build](#) AWS CLI 命令。此命令在 Amazon GameLift Servers 中创建一个新的生成包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 server SDK for Amazon GameLift Servers version \
  --build-root build path \
  --build-version user-defined build number \
  --region region name
```

- operating-system – 游戏服务器生成包的运行时环境。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 Servers 服务器 SDK 版本。如果您没有提供值，则 Amazon GameLift Servers 会使用默认值 4.0.2。如果您指定的服务器 SDK 版本不正确，则在调用 InitSdk 建立与 Amazon GameLift Servers 服务的连接时，游戏服务器生成包可能会失败。
- build-root – 您的构建文件的目录路径。
- name – 新生成包的描述性名称。
- build-version – 生成包文件的版本详细信息。
- region— 您要在其中创建版本的 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_2022 \  
  --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 Servers 会提供上传进度。成功上传后，Amazon GameLift Servers 会返回新的生成包记录 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

您可以将生成包文件存储在 Amazon S3 中，然后从那里将其上传到 Amazon GameLift Servers。在创建生成包时，指定 S3 存储桶的位置，然后 Amazon GameLift Servers 会从 Amazon S3 直接检索生成包文件。

创建构建资源

1. 将构建文件存储在 Amazon S3 中。创建包含已打包构建文件的.zip 文件，然后将其上传到您 AWS 账户的 S3 存储桶。记下存储桶标签和文件名，创建 Amazon GameLift Servers 版本时需要这些标签和文件名。
2. 向 Amazon GameLift Servers 提供对您的生成包文件的访问权限。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 Servers 控制台或创建新的生成记录。AWS CLI 您必须拥有 PassRole 权限，如[Amazon GameLift Servers 的 IAM 权限示例](#)中所述。

Console

1. 在 [Amazon GameLift Servers 控制台](#) 的导航窗格中，选择托管、生成包。
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 Servers 访问您的 S3 存储桶和生成包对象的角色。
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 Servers 为新生成包分配一个 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 server SDK for Amazon GameLift Servers version \  
  --operating-system supported OS \  
  --build-version user-defined build number \  
  --storage-location "Bucket"=S3 bucket label,"Key"=Build .zip file \  
  name,"RoleArn"=Access role ARN} \  
  --region region name
```

- name – 新构建的描述性名称。

- `server-sdk-version` – 您用于集成游戏服务器和 Amazon GameLift Servers 的 Amazon GameLift Servers 服务器 SDK 版本。如果您没有提供值，则 Amazon GameLift Servers 会使用默认值 `4.0.2`。
- `operating-system` – 游戏服务器生成包的运行时环境。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。例如，“`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` – 在计划部署舰队的 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_2022 \  
  --storage-location  
  "Bucket"="my_game_build_files", "Key"="mygame_build_101.zip", "RoleArn"="arn:aws:iam::111122223333:role/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。

为 Amazon GameLift Servers 构建容器映像

本主题介绍如何创建包含游戏服务器软件的容器映像，以便与 Amazon GameLift Servers 配合使用。游戏服务器容器映像包括游戏服务器可执行文件及其运行所需的任何依赖项。游戏服务器容器映像与 Amazon GameLift Servers 托管式容器托管解决方案一起使用。有关构建完整解决方案的详细信息，请参阅：

- [借助 Amazon GameLift Servers 托管容器进行托管的开发路线图](#)
- [容器在 Amazon GameLift Servers 中的工作原理](#)

完成以下任务，准备好将游戏服务器容器映像部署到 Amazon GameLift Servers 容器实例集中。在开始这些任务之前，请完成游戏服务器代码与 Amazon GameLift Servers 服务器 SDK 的集成。

主题

- [创建游戏服务器容器映像](#)
- [将容器映像推送到 Amazon ECR](#)

创建游戏服务器容器映像

在基于 Linux 的平台上或使用已安装 Docker 的适用于 Linux 的 Windows 子系统（WSL），按以下说明操作。

创建游戏服务器容器映像

1. 使用游戏服务器软件准备工作目录。在本地计算机上，创建一个工作目录来整理游戏服务器容器的文件。将容器部署到用于托管的 Amazon GameLift Servers 资源时，您的容器映像使用此文件结构。例如：

```
[~/]$ mkdir -p work/glc/gamebuild && cd work && find .  
.  
./glc  
./glc/gamebuild
```

Note

如果您正在尝试此功能，但游戏服务器版本还没有正常运行，请尝试我们的示例游戏服务器 [SimpleServer](#)，该服务器可在上使用 GitHub。

2. 使用提供的模板创建一个新的 Dockerfile。
3. 按照 Dockerfile 模板中的说明，对其进行更新以供自己使用。
 - 根据需要更新基础映像。
 - 为游戏服务器生成包设置环境变量。
4. 构建容器映像。运行 `docker build`，指定您自己的存储库名称。例如：

```
[~/work/glc]$ docker build -t <local repository name>:<optional tag> .
```

您可以使用 `docker images` 命令查看存储库和镜像 IDs，如以下示例所示：

游戏服务器容器映像的 Dockerfile 模板

此模板包含游戏服务器容器要在 Amazon GameLift Servers 实例集中可用所需的最少指令。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,
# Make sure to use an image with the same architecture as the
# Instance type you are planning to use on your fleets.
FROM public.ecr.aws/amazonlinux/amazonlinux
#
# Game build directory
# -----
# Add your gamebuild directory to the env variable below.
# The game build provided here needs to be integrated with server sdk for Amazon
GameLift Servers.
ENV GAME_BUILD_DIRECTORY="<ADD_GAME_BUILD_DIRECTORY>" \
#
# Game executable and launch parameters
# -----
# Add the relative path to your executable in the 'GAME_EXECUTABLE' env variable
below.
# The game build provided over here needs to be integrated with server sdk for Amazon
GameLift Servers.
# This template assumes that the executable path is relative to the game build
directory.
# Add any launch parameters to pass into your executable in the 'LAUNCH_PARAMS' env
variable below.
# Add 'HOME_DIR' to identify where the game executable and logs exist.
```

```
GAME_EXECUTABLE="<ADD NAME OF EXECUTABLE WITHIN THE GAME DIRECTORY>" \
LAUNCH_PARAMS=<ADD LAUNCH PARAMETERS> \
HOME_DIR="/local/game" \

# Install dependencies as necessary
RUN yum install -y shadow-utils

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

# Change directory to home
WORKDIR $HOME_DIR

# Set up for 'gamelift' user
RUN useradd -m gamelift && \
    echo "gamelift ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers && \
    chown -R gamelift:gamelift $HOME_DIR

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

USER gamelift

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

# Check path before starting the container
RUN echo $PATH

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

将容器映像推送到 Amazon ECR

创建要部署到 Amazon GameLift Servers 的容器映像后，将该映像存储在 Amazon ECR 的公共或私有存储库中。该存储库会分配有一个 URI 值，Amazon GameLift Servers 会使用此值来获取该映像的快照，以便部署到容器实例集中。

Note

如果您还没有 Amazon ECR 私有存储库，请[创建一个](#)。

将容器映像推送到 Amazon ECR

1. 获取 Amazon ECR 凭证。在将容器映像推送到 Amazon ECR 之前，需先以临时形式获取您的 AWS 凭证，并将其提供给 Docker。Docker 需要这些凭证才能登录。

```
[~/work/glc]$ aws ecr get-login-password --region us-west-2 | docker login --  
username AWS --password-stdin aws_account_id.dkr.ecr.us-west-2.amazonaws.com  
WARNING! Your password will be stored unencrypted in  
/home/user-name/.docker/config.json.  
Configure a credential helper to remove this warning.  
See https://docs.docker.com/engine/reference/commandline/login/#credentials-store  
  
Login Succeeded
```

2. 复制您要使用的 [Amazon ECR 私有存储库](#) 的 URI。
3. 向容器映像应用 Amazon ECR 标签。

Example

```
[~/work/glc]$ docker tag <IMAGE ID from above> <Amazon ECR private repository  
URI>:<optional tag>
```

4. 将容器映像推送到 Amazon ECR

Example

```
[~/work/glc]$ docker image push <Amazon ECR private repository URI>
```

与游戏 client/server 互动 Amazon GameLift Servers

Amazon GameLift Servers 托管解决方案中的组件以特定的方式相互交互，从而根据玩家需求运行游戏会话。本主题介绍当游戏服务器托管在 Amazon GameLift Servers 托管式 EC2 实例集、自管理式 Amazon GameLift Servers Anywhere 实例集或混合解决方案上时，各组件之间如何相互通信。

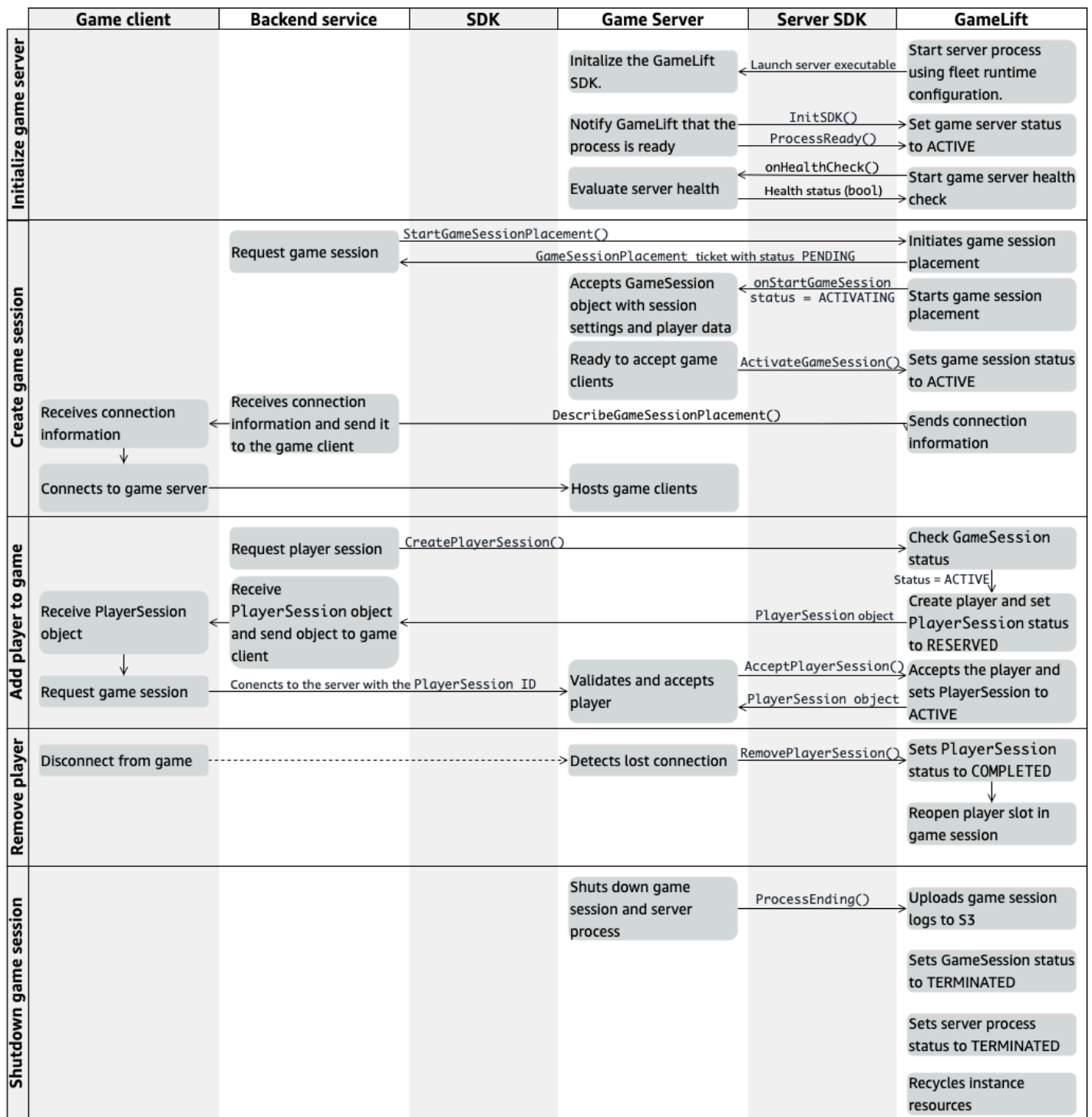
托管解决方案组件包括游戏服务器、Amazon GameLift Servers 服务、客户端后端服务和游戏客户端。游戏服务器使用 [Amazon GameLift Servers 服务器 SDK](#) 与 Amazon GameLift Servers 服务交互。后端服务使用 [Amazon GameLift Servers 服务 API](#) (AWS SDK 的一部分) 代表游戏客户端与服务进行交互。加入游戏会话时，游戏客户端使用游戏会话的唯一 IP 地址和端口号直接连接到游戏会话。

主题

- [交互示意图](#)
- [交互行为](#)

交互示意图

下图说明了各游戏托管组件如何交互，让 Amazon GameLift Servers 服务可以跟踪游戏服务器可用性状态并根据玩家需求启动游戏会话。



交互行为

以下各节介绍了各个关键交互中的事件顺序。

初始化游戏服务器进程

启动时，一个游戏服务器进程会与 Amazon GameLift Servers 服务建立通信，并将其状态报告为已准备好托管游戏会话。

1. 游戏服务器可执行文件的新进程开始在托管资源上运行。
2. 游戏服务器进程按顺序调用以下服务器 SDK 操作：
 - a. `InitSDK()`，用于初始化服务器 SDK、对服务器进程进行身份验证以及与 Amazon GameLift Servers 服务建立通信。
 - b. `ProcessReady()`，用于传达已准备好托管游戏会话。此调用还会报告进程的连接信息（游戏客户端用来连接到游戏会话）以及其他信息。

然后，服务器进程会等待 Amazon GameLift Servers 服务的提示。

3. Amazon GameLift Servers 将服务器进程的状态更新为 ACTIVE，可用于托管新游戏会话。
4. Amazon GameLift Servers 开始定期调用 `onHealthCheck` 回调，从服务器进程请求运行状况。这些调用在服务器进程保持活动状态期间会持续。服务器进程必须在一分钟内响应运行状况是否正常。如果服务器进程响应运行状况不正常或没有响应，Amazon GameLift Servers 服务就会在某个时间点更改服务器进程的活动状态，并停止发送启动游戏会话的请求。

创建游戏会话

当玩家发起玩游戏请求后，Amazon GameLift Servers 服务会响应该请求并启动一个新的游戏会话。

1. 一位使用游戏客户端的玩家请求加入游戏会话。根据您的游戏处理玩家加入流程的方式，游戏客户端会向后端服务发送请求。
2. 如果玩家加入流程要求启动新游戏会话，后端服务会向 Amazon GameLift Servers 服务发送新建游戏会话的请求。此请求将调用服务 API 操作 `StartGameSessionPlacement()`。（作为替代方案，后端服务可能会调用 `StartMatchmaking()` 或 `CreateGameSession()`。）
3. Amazon GameLift Servers 服务以创建一个状态为 PENDING 的新 `GameSessionPlacement` 票证作为响应。它会将票证信息返回到后端服务，让后端服务可以跟踪放置票证状态和确定游戏会话何时可供玩家加入。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 Servers 服务启动游戏会话放置流程。它会确定要查看的实例集，并在这些实例集中搜索未托管游戏会话的活动服务器进程。在找到可用的服务器进程后，Amazon GameLift Servers 服务会执行以下操作：

- a. 使用游戏会话设置和来自放置请求的玩家数据创建 `GameSession` 对象，并将状态设置为 `ACTIVATING`。
 - b. 提示服务器进程启动游戏会话。该服务将调用服务器进程的 `onStartGameSession` 回调并传递 `GameSession` 对象。
 - c. 将服务器进程的游戏会话数更改为 1。
5. 服务器进程运行其 `onStartGameSession` 回调函数。当准备好接受玩家连接时，服务器进程将调用服务器 SDK 操作 `ActivateGameSession()` 并等待玩家连接。
 6. Amazon GameLift Servers 服务使用服务器进程的连接信息更新 `GameSession` 对象（如对 `ProcessReady()` 的调用中所报告的那样），并将游戏会话状态设置为 `ACTIVE`。它还会将 `GameSessionPlacement` 票证状态更新为 `FULFILLED`。
 7. 后端服务调用 `DescribeGameSessionPlacement()` 来检查票证状态并获取游戏会话信息。当游戏会话处于活动状态时，后端服务会通知游戏客户端并传递游戏会话连接信息。
 8. 游戏客户端使用连接信息直接连接到游戏服务器进程并加入游戏会话。

向游戏中添加玩家

游戏可以选择使用玩家会话来跟踪玩家与游戏会话的连接。玩家会话可以单独创建，也可以作为游戏会话放置请求的一部分创建。

1. 后端服务使用游戏会话 ID 调用服务 API 操作 `CreatePlayerSession()`。
2. Amazon GameLift Servers 服务检查游戏会话状态（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，该服务将执行以下操作：
 - a. 创建新的 `PlayerSession` 对象并将其状态设置为 `RESERVED`。
 - b. 使用玩家会话信息响应后端服务请求。
3. 后端服务将玩家会话信息和游戏会话连接信息一起传递给游戏客户端。
4. 游戏客户端使用连接信息和玩家会话 ID 直接连接到游戏服务器进程并请求加入游戏会话。
5. 为了响应游戏客户端的加入尝试，游戏服务器进程调用服务 API 操作 `AcceptPlayerSession()` 来验证玩家会话 ID。服务器进程接受或拒绝连接。
6. Amazon GameLift Servers 服务执行以下操作之一：
 - a. 如果连接被接受，Amazon GameLift Servers 会将 `PlayerSession` 状态设置为 `ACTIVE` 并将 `PlayerSession` 传递到游戏服务器进程。

- b. 如果游戏服务器进程在最初的 `CreatePlayerSession()` 请求发出后的一段时间内没有为玩家会话 ID 调用 `AcceptPlayerSession()`，Amazon GameLift Servers 服务就会将 `PlayerSession` 状态更改为 `TIMEDOUT`，并在游戏会话中重新开放玩家位置。

删除玩家

对于使用玩家会话的游戏，当有玩家断开连接时，游戏服务器进程会通知 Amazon GameLift Servers 服务。该服务使用此信息来跟踪游戏会话中玩家位置的状态，并可以允许新玩家使用开放的位置。

1. 玩家断开与游戏会话的连接。
2. 游戏服务器进程检测到丢失的连接并调用服务器 SDK 操作 `RemovePlayerSession()`。
3. Amazon GameLift Servers 服务将玩家会话状态更改为 `COMPLETED`，并在游戏会话中重新开放玩家位置。

关闭游戏会话

在游戏会话结束或关闭游戏会话时，服务器进程会向 Amazon GameLift Servers 服务通知游戏会话状态。

1. 游戏服务器进程通过调用服务器 SDK 操作 `ProcessEnding()` 结束游戏会话并启动进程关闭。
2. 游戏服务器进程调用 `Destroy()` 以从内存中释放服务器 SDK。[遥测指标](#)需要执行此步骤，以防止其将正常进程退出报告为崩溃。
3. Amazon GameLift Servers 服务将执行以下操作：
 - a. 将游戏会话日志上传到 Amazon Simple Storage Service (Amazon S3)。
 - b. 将游戏会话状态更改为 `TERMINATED`。
 - c. 更改服务器进程状态为 `TERMINATED`。
 - d. 根据托管解决方案的设计，分配新的可用托管资源来运行新的游戏服务器进程。

为 Amazon GameLift Servers 构建后端服务

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 Servers 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。
- 管控 Amazon GameLift Servers 将玩家加入新建游戏会话或接入既有游戏会话的分组规则。

- 从自有资源采集玩家技能等级等属性数据用于对站匹配，避免直接采信客户端数据。

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 Servers 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 IDs 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](#)（AWS 适用于游戏博客）

集成 Amazon GameLift Servers 游戏客户端功能

将 Amazon GameLift Servers 游戏托管功能集成到需要与 Amazon GameLift Servers 服务通信的任何解决方案组件中，例如获取游戏会话信息或创建新的游戏会话。在大多数情况下，此功能内置在后端服务组件中。为核心任务添加功能，包括：

- 请求活动游戏会话的信息和状态。
- 在现有游戏会话中为新玩家预留位置。
- 为一组玩家创建新的游戏会话。
- 请求一个或多个玩家的对战。
- 为现有游戏会话提供更新数据。

[与游戏 client/server 互动 Amazon GameLift Servers](#) 有关游戏托管组件如何通过进行交互的更多详细信息，请参阅 Amazon GameLift Servers SDKs。

设置 Amazon GameLift Servers API

将 w 适用于 C++ 的 AWS SDK it Amazon GameLift Servers h 添加到项目中。

添加代码以初始化 Amazon GameLift Servers 客户端并存储键设置。此代码必须在依赖于 Amazon GameLift Servers 的任何代码之前运行。

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 Amazon GameLift Servers 时要使用的目标区域和端点。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 Servers 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. 如果您使用玩家会话功能，需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：
 - 目标实例集：如果您在特定实例集上手动创建游戏会话，请提供实例集 ID 或指向目标实例集的别名 ID。如果实例集属于多位置，请指定一个实例集位置。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。
 - 目标实例集：对于使用多实例集队列放置新游戏会话的游戏，请指定用于发送放置请求的队列的名称。
 - AWS 凭证 — 所有对的调用都 Amazon GameLift Servers 必须提供托管游戏的凭据。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。根据您的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 Servers 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《工具参考指南》AWS SDKs 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。
 - 如果您有临时安全证书，请使用 AWS Security Token Service (AWS STS) 添加代码以定期刷新证书，如 IAM 用户指南 AWS SDKs [中的使用临时安全证书](#) 中所述。该代码必须在旧凭证过期之前请求新的凭证。

获取活动的游戏会话

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。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。

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"`

管理游戏会话数据

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。创建游戏会话有两种选择，具体取决于游戏托管解决方案管理游戏会话放置的方式。

通过多位置队列创建游戏会话

[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 GameLift Servers 将新游戏会话放置在哪里。有关队列及其使用方法的更多信息，请参阅 [配置游戏会话放置](#)。

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。如果您为相关区域包括玩家延迟数据，Amazon GameLift Servers 会使用此信息将新游戏会话放置到能够为玩家提供尽可能最佳玩游戏体验的实例集上。

要获得准确的延迟测量值，请使用 Amazon GameLift Servers 的 UDP ping 信标。这些端点使您能够测量玩家设备与潜在托管位置之间的实际 UDP 网络延迟，从而比使用 ICMP ping 更准确地做出放置决策。有关使用 UDP ping 信标测量延迟的更多信息，请参阅 [UDP ping 信标](#)。

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 Servers 创建新的游戏会话并返回 [GameSession](#) 对象后，你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 Servers 的资源保护功能来限制一个玩家可创建的游戏会话数量，则需指定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。如果您的游戏使用与唯一玩家的玩家会话，则可以使用此操作 IDs。有关玩家会话的更多信息，请参阅 [Amazon GameLift Servers 和玩家体验](#)。

1. 在游戏会话中预留玩家位置。

要预留玩家位置，请在游戏会话中创建一个玩家会话。有两种方式可执行此操作：

- 如果您使用 [StartGameSessionPlacement](#) 或 [StartMatchmaking](#) 创建游戏会话，则可以在请求中包含一个或多个玩家，以便在新游戏会话中为他们创建玩家会话。
- 要将玩家添加到现有游戏会话中，请致电 [CreatePlayerSession](#) 或 [CreatePlayerSessions](#) 提供游戏会话 ID。

玩家会话请求必须包括唯一的玩家 ID。有关更多信息，请参阅 [生成玩家 ID](#)。Amazon GameLift Servers 接收到该请求后，会先验证该游戏会话是否接受新玩家，以及是否有可用的玩家位置。如果成功，则为玩家 Amazon GameLift Servers 保留一个位置，创建新的玩家会话并返回一个 [PlayerSession](#) 对象。

玩家会话可能包括一组自定义玩家数据。此数据存储在新建的玩家会话对象中。当玩家直接连接到游戏会话时，Amazon GameLift Servers 会将此对象传递到游戏服务器。如果请求创建多个玩家会话，需将每段玩家数据字符串映射至相应的玩家 ID。

2. 连接游戏会话

向后端服务添加代码以检索 `PlayerSession` 对象（例如通过调用 [DescribePlayerSessions\(\)](#)）并将其传递回游戏客户端。此对象包含 DNS 名称、IP 地址和端口。游戏客户端可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可直接通信，不需要经过 Amazon GameLift Servers。服务器与 Amazon GameLift Servers 保持通信，以报告玩家连接状态、运行状况等。

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，然后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。

将自定义游戏数据传入游戏会话

游戏客户端可以将数据传入游戏会话。创建游戏会话时，您可以在请求中包含一组游戏属性（键值对）或游戏会话数据（字符串值）。您还可以使用新的或更新的游戏数据更新现有游戏会话。这些数据将传入托管游戏会话的游戏服务器进程，供游戏服务器代码使用。您无法删除游戏属性。

例如，假设您的游戏定义了四个难度等级：Novice、Easy、Intermediate 和 Expert。当玩家选择加入 Easy 难度的游戏时，您的游戏客户端会通过后端服务请求创建新游戏会话，并附带以下游戏属性：`{"Key": "Difficulty", "Value": "Easy"}`。作为响应，Amazon GameLift Servers 会提示可用的游戏服务器启动新的游戏会话并传递 `GameSession` 对象。随后，游戏服务器进程会使用提供的游戏属性来设置游戏会话的难度等级。

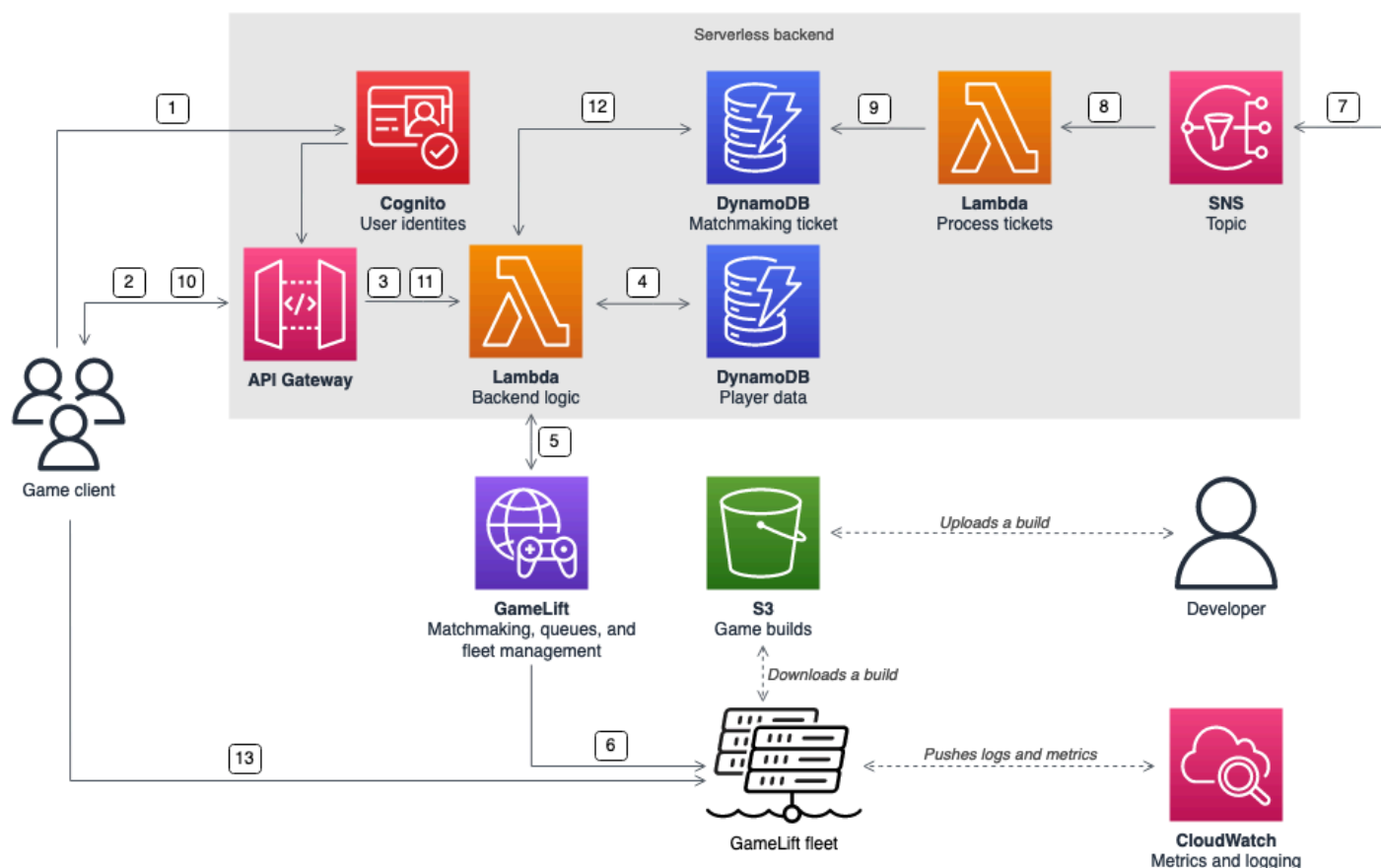
了解详情

- [GameProperty 数据类型](#)
- [SearchGameSessions\(\) 示例](#)
- [UpdateGameSession\(\) GameProperties 参数](#)

带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 Servers API。

下图显示了构建的无服务器后端 AWS 服务，该后端可以将玩家匹配到在 Amazon GameLift Servers 舰队上运行的游戏。下表提供了对每个带编号的注解的说明。要尝试此示例，请参阅开启的[基于多人会话的游戏托管](#)。AWS GitHub



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 调用一个函数。AWS Lambda
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 Servers 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 Servers 队列请求游戏会话放置。队列中有包含一个或多个 AWS 区域地点的舰队。

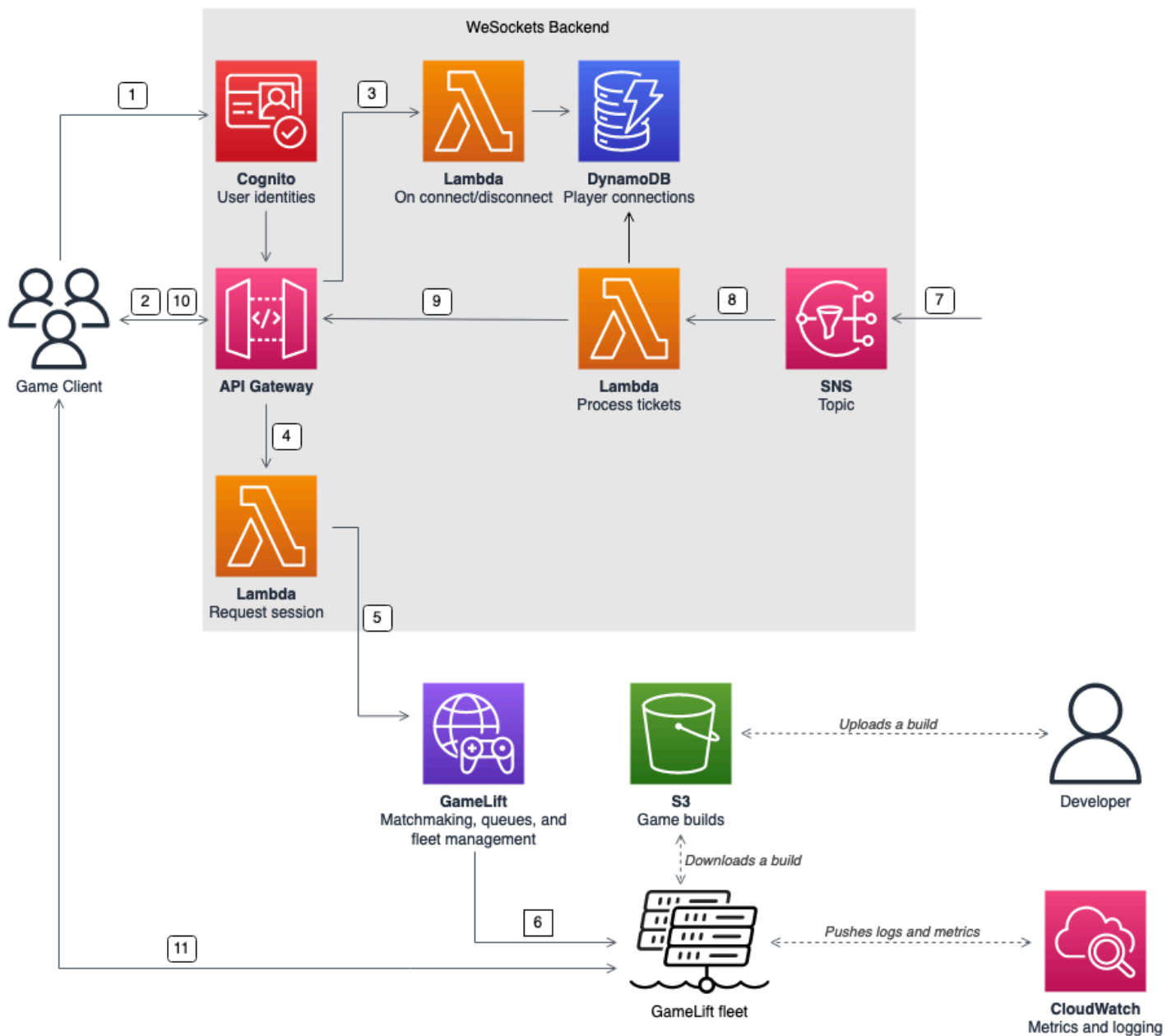
7. Amazon GameLift Servers 将会话放到其中一个实例集的位置后，Amazon GameLift Servers 向 Amazon Simple Notification Service (Amazon SNS) 主题发送一个事件通知。
8. Lambda 函数接收并处理 Amazon SNS 事件。
9. 如果对战票证是一个 MatchmakingSucceeded 事件，那么 Lambda 函数会将结果以及游戏服务器的端口和 IP 地址写入 DynamoDB 表。
10. 游戏客户端向 API Gateway 发出签名请求，以查看特定时间间隔内的对战票证状态。
11. API Gateway 使用 Lambda 函数来检查对战票证状态。
12. Lambda 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 Servers 服务器软件开发工具包验证该 ID。

带有 WebSocket 基于后端的独立游戏会话服务器

使用 WebSocket 基于 Amazon API Gateway 的架构，您可以使用服务器启动的消息向发出配对请求 WebSockets 并发送推送通知以完成配对。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 的更多信息 WebSock APIs，[请参阅使用 WebSocket APIs](#)。

下图显示了一种 WebSocket 基于后端的架构，该架构使用 API Gateway 等将玩家匹配 AWS 服务 到在 Amazon GameLift Servers 舰队上运行的游戏。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. 游戏客户端使用亚马逊 Cognito 凭证签署与 API Gateway API 的 WebSocket 连接。
3. API Gateway 在连接上调 AWS Lambda 用一个函数。该函数将连接信息存储在 Amazon DynamoDB 表中。
4. 游戏客户端通过 API Gateway API 通过 WebSocket 连接向 Lambda 函数发送一条消息，请求会话。
5. Lambda 函数通过 Amazon GameLift Servers FlexMatch 对战接收消息并请求匹配。

6. FlexMatch匹配一组玩家后，通过Amazon GameLift Servers队列FlexMatch请求游戏会话放置。
7. Amazon GameLift Servers 将会话放到其中一个实例集的位置后，Amazon GameLift Servers 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。然后，该函数通过 WebSocket 连接通过 API Gateway API 向游戏客户端发送一条消息。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 Servers 服务器软件开发工具包验证该 ID。

为 Amazon GameLift Servers 部署托管实例集

部署托管资源涉及创建和配置运行游戏服务器的计算基础架构。Amazon GameLift Servers提供多种机队类型以满足不同的托管需求，从完全托管的 AWS Cloud 资源到结合云和本地基础设施的混合解决方案。

请根据您的成本、控制权、可扩展性和地理分布需求，选择最合适的实例集类型。您还可以在单个托管解决方案中组合多种实例集类型，以针对不同的场景或玩家群体进行优化。

实例集特征

Amazon GameLift Servers 实例集是运行游戏服务器并为玩家托管游戏会话的计算资源集合。实例集可能因您使用的计算资源类型以及实例集的管理方式而异。实例集的大小（即它可以支持的游戏会话和玩家的数量）取决于您为其提供的计算资源的数量。所有 Amazon GameLift Servers 实例集都具有以下特征：

- 在所有实例集上运行的游戏服务器进程都与 Amazon GameLift Servers 服务器 SDK 集成，并以相同的方式与 Amazon GameLift Servers 服务进行通信。游戏服务器会报告自己是否可用于托管游戏会话和玩家、响应启动或停止游戏会话的提示以及其他互动。
- Amazon GameLift Servers 以相同的方式处理所有实例集的游戏会话放置。Amazon GameLift Servers 会跟踪实例集的游戏服务器状态，并从可用的游戏服务器中选择服务器来托管新的游戏会话。无论您的游戏将游戏会话放在单个实例集上，还是使用[游戏会话队列](#)来平衡多个实例集之间的托管，都会使用此流程。使用队列时，您还可以自定义放置决策，将资源成本和延迟等因素纳入考量。

- 所有实例集都支持将 FlexMatch 对战构建器与游戏会话放置队列结合使用。Amazon GameLift Servers 服务会接收玩家对战请求，形成对战，然后将其传递到游戏会话队列以查找可用的游戏服务器。
- Amazon GameLift Servers 会收集各种实例集指标。这些指标包括计算和服务器进程的状态指标，以及游戏会话和玩家活动的使用情况指标。有关可用指标的完整列表，请参阅[Amazon GameLift Servers使用 Amazon 进行监控 CloudWatch](#)。

主题

- [决策指南：选择托管选项](#)
- [Amazon GameLift Servers托管 EC2 车队](#)
- [Amazon GameLift Servers 托管式容器实例集](#)
- [Amazon GameLift Servers Anywhere 实例集](#)
- [构建混合托管解决方案](#)

决策指南：选择托管选项

Amazon GameLift Servers 提供多种托管选项，每种选项均针对不同的托管场景和需求而设计。本指南可帮助您根据游戏的技术需求、运营偏好和业务限制来选择实例集类型。有关简要说明，请参阅[Amazon GameLift Servers 游戏托管方案](#)。

实例集对比

下表比较了各种托管选项的关键特征：

托管选项	适用场景	管理级别	主要优势
托管 EC2	完全 AWS 管理的制作托管	完全托管	自动扩缩、全局覆盖、最大限度地减少运营开销
托管式容器	需要编排的容器化应用程序	完全托管	容器编排、资源效率、现代部署
Anywhere	本地、混合或开发环境	自行管理	硬件控制、满足合规性要求、成本优化

关键决策因素

在选择托管选项时，请考虑以下因素：

操作复杂性

您希望承担多少基础设施管理工作？托管式实例集的运营开销极低，而 Anywhere 实例集可提供完全控制权，但需要投入更多管理精力。

可扩展性要求

是否需要根据玩家需求实现自动扩缩？托管式实例集提供内置的自动扩缩功能，而 Anywhere 实例集则需要手动管理容量。

地理分布

您的玩家分布在哪些区域？托管队列可以在全球多个 AWS 区域部署，而 Anywhere 队列则取决于您自己的基础设施位置。

合规性与安全性

您是否有特定的合规要求，或者需要将数据保存在本地？Anywhere 实例集可让您完全掌控基础设施及数据存储位置。

成本考虑因素

您的预算和成本优化策略是什么？比较选项时，需同时考虑基础设施成本和运营开销。

混合解决方案

您不必只为游戏托管解决方案选择一种类型的实例集。许多游戏使用混合方法，结合多种实例集类型以针对不同的场景进行优化：

- 开发和测试：使用 Anywhere 实例集进行开发和测试，然后部署到托管式实例集进行生产。
- 区域优化：在玩家密度高的托管队伍中 AWS 区域使用 Anywhere 舰队，在现有基础设施的地区使用任何地方的舰队。
- 成本优化：使用托管式实例集满足基础容量需求，使用 Anywhere 实例集处理流量峰值或特定工作负载。

Amazon GameLift Servers托管 EC2 车队

Amazon GameLift Servers托管 EC2 队列为生产托管提供基于云的资源。借助托管队伍，您可以获得针对多人游戏托管而优化的 AWS Cloud 资源的灵活性、安全性和可靠性。Amazon GameLift Servers 提供了强大的主机管理工具。

托管 EC2 队列是一组根据您的配置Amazon GameLift Servers拥有和运行的亚马逊弹性计算云 (Amazon EC2) 实例。这些实例物理部署在受支持的 AWS 区域 或 Local Zones。创建队列时，您可以选择满足游戏服务器对计算能力、内存、存储和网络功能要求的 EC2 实例类型。

启动实例集中的每个实例时，Amazon GameLift Servers 将使用所需的运行时环境部署您的游戏服务器生成包。运行时环境使用创建实例集时可用的最新亚马逊机器映像 (AMI) 版本。实例集中的所有实例都使用同一 AMI 版本。

Note

作为最佳实践，我们建议每 30 天更换一次舰队，以维护托管游戏服务器的安全 up-to-date 运行环境。这需要创建一个新实例集并将玩家流量迁移到该实例集。有关更多指导，请参阅[Amazon GameLift Servers 的安全最佳实践](#)。

安装运行时环境并在实例上构建游戏服务器后，Amazon GameLift Servers 开始启动游戏服务器进程。每个游戏服务器进程都会与 Amazon GameLift Servers 服务建立连接，报告可托管游戏会话的就绪状态，并开始持续上报运行状况。此后，Amazon GameLift Servers 即可提示服务器进程启动游戏会话。

除了实例集部署外，Amazon GameLift Servers 还会帮您处理以下主机管理任务：

- 跟踪实例集中所有计算的状态，并替换掉过时或不正常的计算。
- 处理服务器进程和 Amazon GameLift Servers 服务之间的通信的身份验证。
- 根据您的运行时配置，在每个计算上自动启动和停止游戏服务器进程。
- 提供可根据玩家需求动态调整实例集容量的自动扩缩工具。
- 报告队列 EC2 实例的性能指标。

请参阅以下有关如何设置和维护托管 EC2 车队的主题：

- [借助 Amazon GameLift Servers 托管式 EC2 进行托管的开发路线图](#)
- [创建Amazon GameLift Servers托管 EC2 舰队](#)
- [利用 Amazon GameLift Servers 扩展游戏托管容量](#)

- [托管资源自定义](#)
- [更新 Amazon GameLift Servers 实例集配置](#)

托管 EC2 舰队创建工作流程

对于托管式实例集，Amazon GameLift Servers 会设置实例集资源，还会在安装并运行游戏服务器软件的情况下部署一组计算资源。创建工作流程完成并成功后，队列在舰队所在区域有一个活动 EC2 实例，在舰队的远程位置各有一个活动实例。所有实例都有游戏，随时可以托管游戏会话。

1. Amazon GameLift Servers 将在实例集的主区域创建实例集资源，并将每个位置的所需容量设置为一（1）个实例。实例集和位置状态设置为新建。
2. Amazon GameLift Servers 开始将事件写入实例集事件日志。
3. Amazon GameLift Servers 将实例集状态设置为正在下载，并开始准备游戏服务器软件以进行部署。
 - a. 获取上传的游戏服务器生成包并提取压缩文件。
 - b. 运行安装脚本（如果已提供）。
 - c. 将实例集状态设置为正在验证，并开始验证下载和安装生成包文件时未发生错误。
4. Amazon GameLift Servers 将队列状态设置为 Building，配置队列硬件，并为每个队列 EC2 实例分配一个实例。
5. Amazon GameLift Servers 将实例集状态设置为正在激活。在每个实例上启动游戏服务器进程（基于实例集的运行时指令），并测试生成包和 Amazon GameLift Servers 服务之间的连接。
6. 当每个实例上的游戏服务器进程建立连接并报告已准备好托管游戏会话时，Amazon GameLift Servers 会将实例集和位置状态设置为活动。此时，实例集被视为已准备好托管游戏会话。

创建 Amazon GameLift Servers 托管 EC2 舰队

本主题介绍如何创建 Amazon GameLift Servers 托管 EC2 舰队。托管队伍使用针对多人游戏托管进行了优化的亚马逊弹性计算云 (Amazon EC2) 计算实例。您可以创建托管队列，将计算部署到全局 AWS 区域 和支持的 Local Zones。Amazon GameLift Servers

创建新的托管 EC2 舰队资源时，会立即启动队列创建的第一阶段。在 Amazon GameLift Servers 部署 EC2 实例、安装运行时环境和在实例上构建游戏服务器以及开始启动游戏服务器时，托管队列的创建会经历几个阶段。根据您的游戏服务器构建所需的运行时环境，在创建队列时 Amazon GameLift Servers 部署最新版本的 Amazon 系统映像 (AMI)（队列中所有未来的实例都将使用相同的版本）。您可以在控制台中或使用 AWS Command Line Interface (AWS CLI) 监控队列的状态。当实例集准备就

绪并可托管游戏会话时，其状态会变更为 ACTIVE。有关创建托管式实例集的更多信息，请参阅以下主题：

Note

作为最佳实践，我们建议每 30 天更换一次舰队，以维护托管游戏服务器的安全 up-to-date 运行环境。这需要创建一个新实例集并将玩家流量迁移到该实例集。有关更多指导，请参阅[Amazon GameLift Servers 的安全最佳实践](#)。

创建托管 EC2 车队

使用 Amazon GameLift Servers 控制台或 AWS Command Line Interface (AWS CLI) 创建托管 EC2 队列。

Console

在 [Amazon GameLift Servers 控制台](#) 中，使用导航窗格打开“托管 EC2，舰队”页面。选择创建托管 EC2 队列以启动队列创建工作流程。

步骤 1 定义托管 EC2 车队的详细信息

对于最低实例集配置：

- 提供实例集名称。
- 选择二进制类型：生成包。从您之前上传到 Amazon GameLift Servers 的服务器生成包列表选择一个生成包。
- 跳过“其他详细信息”和“标签”部分以使用默认设置。

1. 填写实例集详细信息部分：

- a. 输入实例集名称。实例集名称仅为描述性，可用于对实例集列表进行排序或筛选。它们不必唯一。
- b. 提供简短的实例集描述，包括您要捕获的有关实例集的任何自定义信息。
- c. 选择二进制类型：生成包，然后使用下拉列表选择您之前上传到 Amazon GameLift Servers 的游戏服务器生成包。

2. 根据需要设置其他详细信息。

- a. 如果您的游戏服务器可执行文件需要访问您账户中的其他 AWS 资源，请指定具有必要权限的 IAM 实例角色。有关更多信息，包括如何授权其他服务器端应用程序（例如 CloudWatch 代理），请参阅[将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。此设置在您创建实例集后无法更改。

您必须先创建角色，然后才能创建使用它的实例集。此外，要创建具有实例角色的实例集，您的 AWS 用户必须具有 IAM PassRole 权限（请参阅[Amazon GameLift Servers 的 IAM 权限示例](#)）。
 - b. 启用生成 TLS 证书选项，为您的游戏设置身份验证和加密，游戏客户端会使用此证书在连接时对游戏服务器进行身份验证，以及加密所有客户端/服务器通信。对于启用 TLS 的实例集中的每个实例，Amazon GameLift Servers 还使用证书创建一个新的 DNS 条目。此设置在您创建实例集后无法更改。
 - c. Amazon GameLift Servers 会为每个单独的实例集生成指标数据。如果要合并多个实例集的指标数据，请指定一个指标组名称。对于所有需要合并指标数据的实例集，使用相同的指标组名称。CloudWatch 用于查看聚合指标组数据。
3. 向实例集资源添加标签。每个标签都包含定义的一个键和一个可选值。为要分类的 AWS 资源分配标签，例如按用途、所有者或环境进行分类。为每个要添加的标签选择添加新标签。
 4. 选择下一步以继续工作流程。

步骤 2 定义实例详细信息

在此步骤中，选择要用于游戏服务器的硬件资源类型，并指定部署位置。通过选择多个位置，您可以将游戏服务器部署到更广的地理位置，使它们更接近您的玩家并最大限度地降低延迟。队列中的所有 EC2 实例必须是相同的类型。并非所有实例类型都在所有位置可用。

对于最低实例集配置：

- 初始部署时，仅选择主位置进行部署。在初期开发与测试阶段，无需添加远程位置。
- 将实例集类型设置为“按需型”。竞价型实例集需要额外的设置工作。
- 将实例类型设置为“c5.large”。这种常用的实例类型全部可用 AWS 区域。您可以在后续阶段优化硬件使用情况。

1. 在实例部署中，指定实例集的位置和类型。

- a. 选择您要部署实例集实例的一个或多个其他位置。这些远程位置将添加到舰队的主位置（已预先选择），这是您创建此舰队 AWS 区域 的地方。您可以从所有 Amazon GameLift Servers 支持的 Local Zones AWS 区域 和 Local Zones 中选择远程位置。

要详细了解受支持的地点，包括如何使用默认未启用的位置，[Amazon GameLift Servers 服务位置](#) 请参阅托管主机。AWS 区域 另请查看 Amazon GameLift Servers 的实例集位置 [配额](#)。

- b. 选择对此实例集使用按需型实例或竞价型实例。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](#)。
2. 选择符合您需求且可在所有选定位置使用的 Amazon EC2 实例配置。此列表是根据您当前选择的位置和实例集类型进行筛选的。您可以按其他因素（如实例类型系列和架构）对其进行进一步筛选。创建实例集后无法更改实例类型。

某些位置的实例类型选择有限。如果您首选的实例类型并非在所有位置都可用，请选择位置可用性值以查看完整详细信息。为了纳入所有位置，您可能需要创建具有不同实例类型的单独实例集。

有关选择实例类型的更多信息，请参阅[实例类型](#)。要了解有关 Amazon A EC2 m 架构的更多信息，请参阅 [AWS Graviton 处理器和亚马逊 EC2实例类型](#)。有关支持的实例类型的完整列表 Amazon GameLift Servers，请参阅 [EC2InstanceType\(CreateFleet\(\)\)](#) 的 API 参考。

Note

Graviton Arm 实例需要一个针对 Linux AMI 的服务器生成包。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。

3. 选择下一步以继续工作流程。

步骤 4 配置运行时

在此步骤中，描述您希望实例集中的每个实例如何运行您的游戏服务器软件。为要在实例上运行的每个可执行文件定义一个单独的服务器进程行项目，并确定要并发运行的服务器进程的数量。在每个实例上打开端口，以允许玩家直接连接到游戏服务器。您可以随时更新这些实例集设置。

i 对于最低实例集配置：

- 为游戏服务器可执行文件定义一个服务器进程行项目。如果您的游戏服务器需要运行其他进程，请同时为每个进程创建一个定义。
- 对每个行项目使用默认的并发进程数（1）。
- 跳过游戏会话激活设置。
- 指定单个端口号。
- 跳过游戏会话资源设置。

1. 创建运行时配置，指示 Amazon GameLift Servers 如何在实例集中的每个实例上运行服务器进程。部署后，您可以随时更改实例集的运行时配置。
 - a. 在生成包中输入可执行文件的启动路径。在 Windows 实例上，游戏服务器可执行文件被构建到路径 `C:\game`。在 Linux 实例上，游戏服务器被构建到 `/local/game`。示例：`C:\game\MyGame\server.exe` 或 `/local/game/MyGame/server.exe`。
 - b. 输入可选的启动参数以传递给游戏可执行文件。示例：`+sv_port 33435`
`+start_lobby`。
 - c. 指定要在每个实例上运行的并发进程的数量。对于游戏服务器可执行文件，每个进程可以托管一个游戏会话，因此并发进程决定了实例可以同时托管的游戏会话数量。

查看每个实例的服务器进程的 Amazon GameLift Servers [配额](#)。这些配额适用于所有配置的并发进程总数。如果配置的实例集超出这些配额，实例集将无法激活。

2. 使用游戏会话激活默认值或为您的游戏自定义该值。如果运行时配置要求每个实例有多个并发游戏服务器进程，这些设置将决定新游戏会话的启动速度。
 - a. 设置最大并发游戏会话激活，以限制实例上正在准备新游戏会话的游戏服务器的数量。当启动多个新游戏会话需要耗费大量资源并可能影响其他正在运行的游戏会话的性能时，此设置非常有用。
 - b. 设置新激活超时，以反映新游戏会话完成激活和报告已准备好托管玩家所需的最长时间。如果游戏会话激活超过此值，Amazon GameLift Servers 将终止激活。
3. 打开 EC2 端口设置以允许入站流量访问队列中的服务器进程。创建实例集不需要这些设置，但您需要先完成这些设置，然后玩家才能连接到实例集上的游戏会话。

对于每个端口设置，请选择要用于游戏客户端和游戏服务器之间通信的数据传输协议类型。使用 CIDR 表示法提供端口范围（格式为 `nnnnn[-nnnnn]`）和 IP 地址范围（例如 `0.0.0.0/0`，它允许任何人访问）。

如果您需要设置多个不连续的范围，请创建多个端口设置。

4. 指定可选的游戏会话资源设置。部署后，您可以随时更新这些设置。
 - a. 为实例集中的所有实例开启或关闭游戏扩展保护策略。在缩减事件期间，如果受保护的实例集实例正在托管活动游戏会话，Amazon GameLift Servers 不会终止这些实例。
 - b. 如果要限制一名玩家在指定时间范围内可以创建的游戏会话的数量，请设置最大资源创建限制。
5. 选择下一步以继续工作流程。

步骤 5 审核并创建

在创建实例集之前，请审查您的设置。尽管有些设置可以在以后更新（请参阅[更新 Amazon GameLift Servers 实例集配置](#)），但创建实例集后不允许更改以下设置：

- 计算类型：您无法将托管 EC2 队列转换为 Anywhere 队列。
- 游戏服务器二进制文件：若要部署游戏服务器生成包的更新，必须创建一个新的实例集。
- 其他选项，包括实例角色和 TLS 证书生成。
- 实例详细信息，包括队列类型（竞价型或按需型）和 EC2 实例类型。

当您准备好部署新实例集时，请选择创建。Amazon GameLift Servers 会立即开始实例集激活流程，分配一个唯一 ID 并将实例集状态设为 NEW。在实例集页面中跟踪实例集的进度。查看实例集的详细信息页面，然后转到事件选项卡。

您可以在实例集进入 ACTIVE 状态后调整实例集的托管容量。Amazon GameLift Servers 最初在每个实例集位置部署一个包含单个实例的实例集，您可以通过向每个位置添加实例来调整容量。有关更多信息，请参阅[利用 Amazon GameLift Servers 扩展游戏托管容量](#)。

AWS CLI

使用 `create-fleet` 命令创建计算类型 EC2 的实例集。Amazon GameLift Servers 会在当前的默认 AWS 区域中创建实例集资源（您也可以添加 `--region` 标签来指定其他 AWS 区域）。

创建最小托管式实例集

以下示例请求创建了一个新实例集，该实例集具有执行以下操作所需的最低设置：部署一个实例集，其中包含游戏客户端可以连接到的正在运行的游戏服务器。新实例集具有以下特征：

- 它指定了已上传到 Amazon GameLift Servers 并处于 READY 状态的游戏服务器生成包。
- 它使用 c5.large 按需型实例，这种实例的操作系统与所选游戏生成包匹配。
- 它将舰队的总部设置 AWS 区域 为该区域，us-west-2 并将实例部署到该区域。
- 根据运行时配置，实例集中的每个计算运行一个游戏服务器进程，这意味着每个计算一次只能托管一个游戏会话。游戏会话激活超时设置为默认值 300 秒，并且对并发激活的数量没有限制。
- 玩家可以使用单端口设置 33435 连接到游戏服务器。
- 所有其他功能要么已关闭，要么使用默认设置。

```
aws gamelift create-fleet \  
  --name MinimalFleet123 \  
  --description "A basic test fleet" \  
  --region us-west-2 \  
  --ec2-instance-type c5.large \  
  --fleet-type ON_DEMAND \  
  --build-id build-1111aaaa-22bb-33cc-44dd-5555eeee66ff \  
  --runtime-configuration "ServerProcesses=[{LaunchPath=C:\game\Bin64.dedicated\  
  \MultiplayerSampleProjectLauncher_Server.exe, ConcurrentExecutions=10}]" \  
  --ec2-inbound-permissions  
  "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"
```

创建完全配置的托管式实例集

以下示例请求创建了一个生产实例集，该实例集具有所有可选功能的设置。新实例集具有以下特征：

- 它指定了已上传到 Amazon GameLift Servers 并处于 READY 状态的游戏服务器生成包。
- 它使用 c5.large 按需型实例，这种实例的操作系统与所选游戏生成包匹配。
- 它将舰队的总部设置 AWS 区域 为所在地区 us-west-2 和一个远程位置，并将实例部署到主区域和一个远程位置 sa-east-1。
- 根据运行时配置：
 - 实例集中的每个计算运行 10 个具有相同启动参数的游戏服务器进程，这意味着每个计算最多可以同时托管 10 个游戏会话。
 - 在每个计算上，仅允许同时激活两个游戏会话。激活的游戏会话必须在 300 秒（5 分钟）内准备好托管玩家，否则会被终止。

- 玩家可以使用范围为 33435 到 33535 的端口连接到游戏服务器。
- 它为游戏客户端和服务器之间的加密通信生成 TLS 证书。
- 实例集中的所有游戏会话都会开启游戏会话保护。
- 单个玩家在 15 分钟内只能创建三个新的游戏会话。
- 此实例集的指标包含在指标组 AMERfleets 中，该指标组（在本例中）汇总了北美、中美和南美的一组实例集的指标。

```
aws gamelift create-fleet \  
  --name ProdFleet123 \  
  --description "A fully configured prod fleet" \  
  --ec2-instance-type c5.large \  
  --region us-west-2 \  
  --locations "Location=sa-east-1" \  
  --fleet-type ON_DEMAND \  
  --build-id build-1111aaaa-22bb-33cc-44dd-5555eeee66ff \  
  --certificate-configuration "CertificateType=GENERATED" \  
  --runtime-configuration "GameSessionActivationTimeoutSeconds=300,  
MaxConcurrentGameSessionActivations=2, ServerProcesses=[{LaunchPath=C:\game  
\Bin64.dedicated\MultiplayerSampleProjectLauncher_Server.exe, Parameters=+sv_port  
33435 +start_lobby, ConcurrentExecutions=10}]" \  
  --new-game-session-protection-policy "FullProtection" \  
  --resource-creation-limit-policy "NewGameSessionsPerCreator=3,  
PolicyPeriodInMinutes=15" \  
  --ec2-inbound-permissions  
  "FromPort=33435,ToPort=33535,IpRange=0.0.0.0/0,Protocol=UDP" \  
  --metric-groups "AMERfleets"
```

如果创建实例集请求成功，Amazon GameLift Servers 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 Servers 会启动实例集激活进程，并将实例集状态和位置状态设置为 New。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](#)

Amazon GameLift Servers 托管式容器实例集

Amazon GameLift Servers 托管式容器实例集提供基于云的资源，用于托管您的容器化游戏服务器软件。借助托管的容器舰队，您可以获得 AWS Cloud 资源的灵活性、安全性和可靠性，并针对多人游戏托管进行了优化。Amazon GameLift Servers 提供了强大的主机管理工具。

 借助以下托管容器工具加快入门速度：

- [容器入门工具包](#)可简化集成和实例集设置。它为游戏服务器添加核心游戏会话管理功能，并通过预配置的模板构建容器实例集和游戏服务器的自动化部署管道。部署完成后，可通过 Amazon GameLift Servers 控制台和 API 工具监控实例集性能、管理游戏会话并分析指标。
- 对于虚幻引擎或Unity开发者，请使用游戏引擎[Amazon GameLift Servers插件和服务 SDKs](#)来集成游戏服务器，并从游戏引擎的开发环境中构建容器舰队。该插件提供的引导式工作流程可帮助您借助托管容器创建基于云的快速、简单的托管解决方案。后续可在此基础上扩展，为游戏创建自定义托管方案。

托管容器队列是一组运行 Linux 的亚马逊弹性计算云 (Amazon EC2) 实例，这些实例根据您的配置 Amazon GameLift Servers 拥有和运行。这些实例实际位于支持的区域 AWS 区域 或 Local Zones 中。创建容器队列时，您可以选择满足游戏服务器对计算能力、内存、存储和网络功能要求的 EC2 实例类型。

对于托管式容器实例集，您可以将基于 Linux 的容器映像存储在 Amazon Elastic Container Registry (Amazon ECR) 存储库中，然后创建容器组定义来描述容器架构。创建实例集时，Amazon GameLift Servers 会使用最新版本的 Linux 亚马逊机器映像 (AMI) 预调配实例集实例，并使用容器组定义来部署您的容器映像。即使您更新容器组定义或更改容器映像，容器实例集中的所有实例都将使用同一 AMI 版本。

Note

作为最佳实践，我们建议每 30 天更换一次舰队，以维护托管游戏服务器的安全 up-to-date 运行环境。这需要创建一个新实例集并将玩家流量迁移到该实例集。有关更多指导，请参阅[Amazon GameLift Servers 的安全最佳实践](#)。

部署容器化实例后，容器开始启动游戏服务器进程。每个游戏服务器进程都会与 Amazon GameLift Servers 服务建立连接，报告可托管游戏会话的就绪状态，并开始持续上报运行状况。此后，Amazon GameLift Servers 即可提示服务器进程启动游戏会话。

除了实例集部署外，Amazon GameLift Servers 还会帮您处理以下主机管理任务：

- 跟踪实例集中所有容器的状态，并替换掉过时或运行状况不佳的容器。
- 处理服务器进程和 Amazon GameLift Servers 服务之间的通信的身份验证。
- 提供可根据玩家需求动态调整实例集容量的自动扩缩工具。
- 报告队列 EC2 实例、容器和服务器进程的性能指标。

请参阅以下主题，了解如何设置和维护托管式容器实例集：

- [借助 Amazon GameLift Servers 托管容器进行托管的开发路线图](#)
- [创建 Amazon GameLift Servers 托管式容器实例集](#)
- [自定义 Amazon GameLift Servers 容器实例集](#)
- [利用 Amazon GameLift Servers 扩展游戏托管容量](#)
- [更新 Amazon GameLift Servers 托管式容器实例集](#)

容器在 Amazon GameLift Servers 中的工作原理

Amazon GameLift Servers 容器实例集旨在让您能够灵活地部署和扩展容器化应用程序。它使用 Amazon Elastic Container Service (Amazon ECS) 来管理 Amazon GameLift Servers 实例集的任务

部署和执行。本主题介绍了在 Amazon GameLift Servers 托管式实例集上运行容器的基本结构元素，说明了常见的架构，并概述了一些核心概念。

 借助以下托管容器工具加快入门速度：

- [容器入门工具包](#)可简化集成和实例集设置。它为游戏服务器添加核心游戏会话管理功能，并通过预配置的模板构建容器实例集和游戏服务器的自动化部署管道。部署完成后，可通过 Amazon GameLift Servers 控制台和 API 工具监控实例集性能、管理游戏会话并分析指标。
- 对于 Unreal Engine 或 Unity 开发者，可使用 [Amazon GameLift Servers 插件](#)在游戏引擎的开发环境内集成游戏服务器并构建容器实例集。该插件提供的引导式工作流程可帮助您借助托管容器创建基于云的快速、简单的托管解决方案。然后，在此基础上扩展，为游戏创建自定义托管方案。

容器实例集组件

Fleet

容器实例集是 Amazon EC2 实例的集合，用于托管容器化游戏服务器。这些实例由 Amazon GameLift Servers 代为管理。创建实例集时，您需要配置如何将包含游戏服务器软件的容器架构部署到每个实例集实例。您可以创建包含单个或多个地理位置实例的容器实例集，并借助 Amazon GameLift Servers 扩展工具，根据所托管的游戏会话和玩家数量自动调整容器实例集的容量。

实例

Amazon EC2 实例是为游戏托管提供计算容量的虚拟服务器。使用 Amazon GameLift Servers，您可以从一系列实例类型中进行选择。每种实例类型在 CPU、内存、存储和网络容量方面提供不同的组合配置。

创建容器实例集时，Amazon GameLift Servers 会根据您选择的实例类型和实例集配置来部署容器。每个已部署的实例集实例均完全相同，并且以相同的方式运行您的容器化游戏服务器软件。实例集中的实例数决定了实例集的规模和游戏托管容量。

容器组

Amazon GameLift Servers 使用容器组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，您可以定义容器的行为方式、设置依赖关系以及共享可用的 CPU 和内存资源。

每个实例集实例可以包含以下类型的容器组：

- 游戏服务器容器组管理运行游戏服务器应用程序和支持软件的容器。容器实例集必须包含一个此类容器组，才能托管游戏会话和玩家。游戏服务器容器组可以在实例集实例之间进行复制。每个实例集实例的游戏服务器组副本数量取决于您的软件的计算要求和实例上可用的计算资源。
- 每个实例的容器组是可选的，支持您在每个实例集实例上运行其他软件。它们对于运行后台服务或实用程序非常有用，例如用于监控。您的游戏服务器软件不直接依赖于每个实例组中的进程。每个实例集实例仅部署一个每实例容器组副本。

容器实例集中的每个容器组都有一个被指定为“必备”的容器。必备容器主导着容器组的生命周期。如果必备容器出现故障，整个容器组将会重启。

Container

容器是基于容器的架构中最基本的元素。它包含一个具有软件可执行文件和依赖文件的容器映像。通过定义容器，您可以配置软件的运行方式及其与 Amazon GameLift Servers 的交互方式。

Amazon GameLift Servers 定义了两种类型的容器：

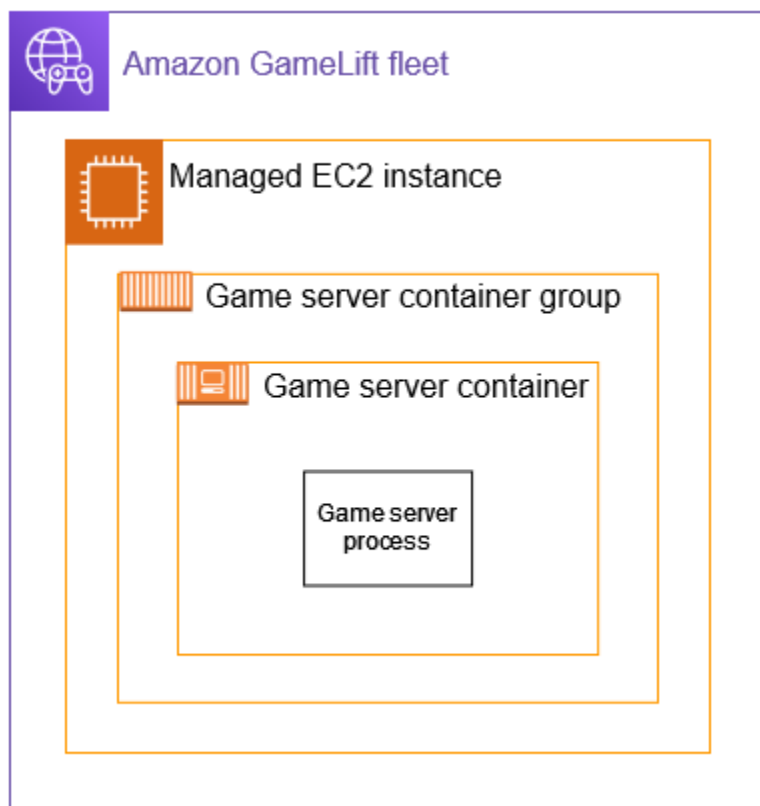
- 游戏服务器容器包含运行游戏服务器进程和为玩家托管游戏会话所需的一切，包括游戏服务器生成包和依赖软件。需为实例集的游戏服务器容器组定义一个游戏服务器容器。该容器会被自动认定为容器组的必备容器。
- 支持容器用于运行其他软件来支持游戏服务器，其概念类似于“sidecar”容器。您可以选择与游戏服务器并行运行和扩展支持软件，同时将其作为单独的容器进行管理。在游戏服务器容器组中，您可以定义零个或多个支持容器。在每个实例容器组中，所有容器均为支持容器。任何支持容器都可指定为必备容器。

计算

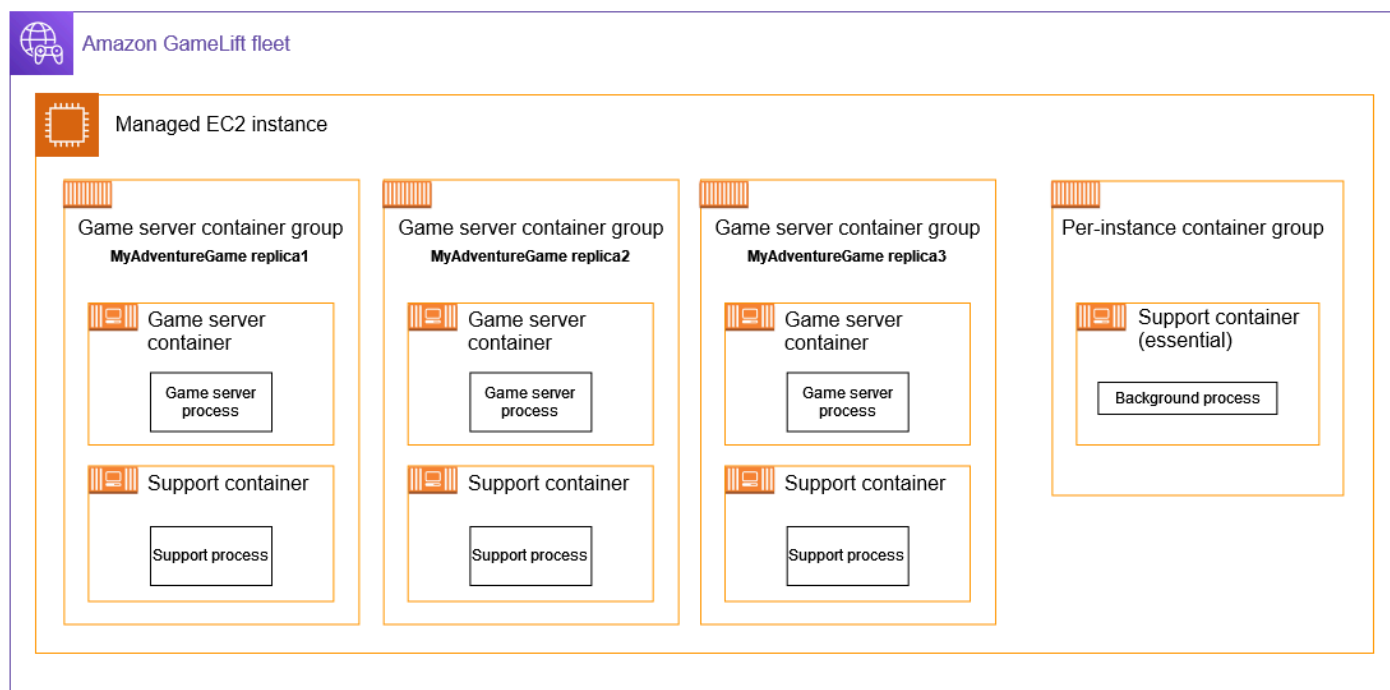
计算表示实例集实例上游戏服务器容器组的副本。

常见架构

下图展示了最简单的容器实例集结构。在此结构中，实例集中的每个实例均维护一个游戏服务器容器组的副本。该容器组包含一个游戏服务器容器，用于运行一个游戏服务器进程。在此示例中，容器实例集配置为每个实例放置一个游戏服务器容器组副本。采用此架构时，每个实例运行一个游戏服务器进程。



第二张图展示了一种更为复杂的容器实例集架构。在此结构中，实例集同时包含游戏服务器容器组和每个实例容器组。游戏服务器容器组包含单独的容器，分别用于游戏服务器进程和支持进程。实例集配置为在每个实例集实例上放置三个游戏服务器容器组副本。每个实例容器组则不会进行复制。在此示例中，容器实例集配置为每个实例放置三个游戏服务器容器组副本。采用此架构时，每个实例运行三个游戏服务器进程。



核心功能

本节总结了 Amazon GameLift Servers 如何实现一些基本的容器概念。有关如何使用容器实例集的说明，请参阅本指南中的相关主题。

活动实例集更新

托管容器提供高级支持，帮助您管理托管软件和容器架构的生命周期。您可以更新容器定义（包括容器映像），并将更改部署到现有实例集。借助此功能，可以更快速、更轻松地在开发过程中对容器进行迭代更改。它还提供一些功能，可帮助您持续构建、部署并跟踪软件版本更新。这些功能包括：

- 管理容器组定义更新和版本控制。您可以更新容器组定义的几乎所有属性，包括容器映像和配置设置。每当您更新容器时，Amazon GameLift Servers 都会自动为该更新分配版本号，且默认保留所有版本。您可以访问任何特定版本，也可以根据需要删除版本。创建容器实例集时，需要指定要部署的容器组定义和版本。
- 使用新的容器组定义和配置设置更新现有容器实例集。您可以将容器更新部署到已部署至实例集实例的实例集。您可以使用 AWS 管理控制台 或 AWS SDK 和 CLI 跟踪每个队列位置的更新部署状态。
- 配置实例集更新在活跃实例集中的部署方式。
 - 游戏会话保护。选择保护包含活跃游戏会话的实例集实例，直至游戏会话结束（安全部署）。或者，也可以选择无论游戏会话活动如何，都替换实例集实例（非安全部署）。在开发和测试阶段使用非安全部署可以缩短部署时间。

- 最低运行正常百分比。指定要在部署期间维护的正常运行任务的百分比。此功能允许您决定部署期间受影响的实例集实例数量。较低的数值会优先保证部署速度，而较高的数值则可确保游戏服务器在整个部署过程中保持较高的可用性。
- 部署失败策略。决定部署失败时要采取的措施。部署失败指部分更新后的容器未通过状态检查，被判定为异常状态。您可以将部署设置为自动将所有实例集实例回滚到先前部署的状态。或者，您也可以选择保留部分异常的实例集实例以用于调试。

当您想为游戏服务器软件部署更新时，更新活跃实例集的功能非常有用。为游戏服务器构建新的容器映像后，部署流程分为两步：第一步，使用新映像更新容器组定义；第二步，更新容器实例集。Amazon GameLift Servers 会根据需要处理所有其他任务。

容器打包

在开发用于部署到容器实例集中的容器结构时，核心目标之一是优化可用计算资源的利用率。为实现这一目标，需要将尽可能多的游戏服务器容器组打包到每个实例集实例上。

Amazon GameLift Servers 通过根据以下信息计算每个实例的最大游戏服务器容器组数来帮助实现此目的：

- 实例集的实例类型及其 vCPU 和内存资源。
- 游戏服务器容器组中所有容器的 vCPU 和内存要求。

每个实例容器组（如果有）中所有容器的 vCPU 和内存要求。

创建容器实例集时，您可以使用计算得出的最大值，也可以指定所需的数量。作为最佳实践，应计划对容器化游戏服务器软件进行试验，以确定资源需求，实现最佳游戏服务器性能。

容量扩展

实例集容量衡量实例集可以同时托管的游戏会话数量。您也可以根据实例集可以支持的并发玩家数量来衡量容量。要增加或减少实例集的托管容量，您可以添加或删除实例集实例。

容器实例集配置为在每个实例集实例上运行特定数量的并发游戏服务器进程。（您可以根据（1）每个实例的游戏服务器容器组和（2）每个容器组中运行的游戏服务器进程数来计算此值。）每个实例的并发游戏服务器数量可表明增减单个实例集实例所产生的影响。例如，如果您的容器实例集在每个游戏服务器容器组中运行 1 个游戏服务器进程，并且每个实例集实例包含 100 个游戏服务器容器组，则以 100 为增量增加或减少实例集托管并发游戏会话的容量。如果每个游戏会话包含 10 个玩家席位，则实例集的玩家承载容量将以 1000 为单位增减。

对于容器实例集，您可以使用 Amazon GameLift Servers 提供的任何容量扩展方法。这些方法包括：

- 通过设置所需的实例集实例数来手动设置实例集容量。
- 通过设定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定数量的闲置托管资源，以便新玩家可以快速进入游戏。随着玩家需求的增加或减少，该缓冲区的大小会不断调整。
- 使用自定义扩缩规则设置自动扩缩（高级功能）。此方法允许您根据自己选择的实例集指标进行扩展。

游戏 client/server 连接

借助 Amazon GameLift Servers 托管实例集，游戏客户端可以直接连接到云托管游戏服务器。当游戏客户端要求加入游戏时，Amazon GameLift Servers 会找到游戏会话，并向游戏客户端提供连接信息（IP 和端口）。您可以通过为实例集开放特定端口范围（入站权限）来控制对实例集实例的外部访问。入站权限决定了哪些端口对传入流量开放。您可以快速关闭所有端口、仅开放少数端口，或者开放所有端口。

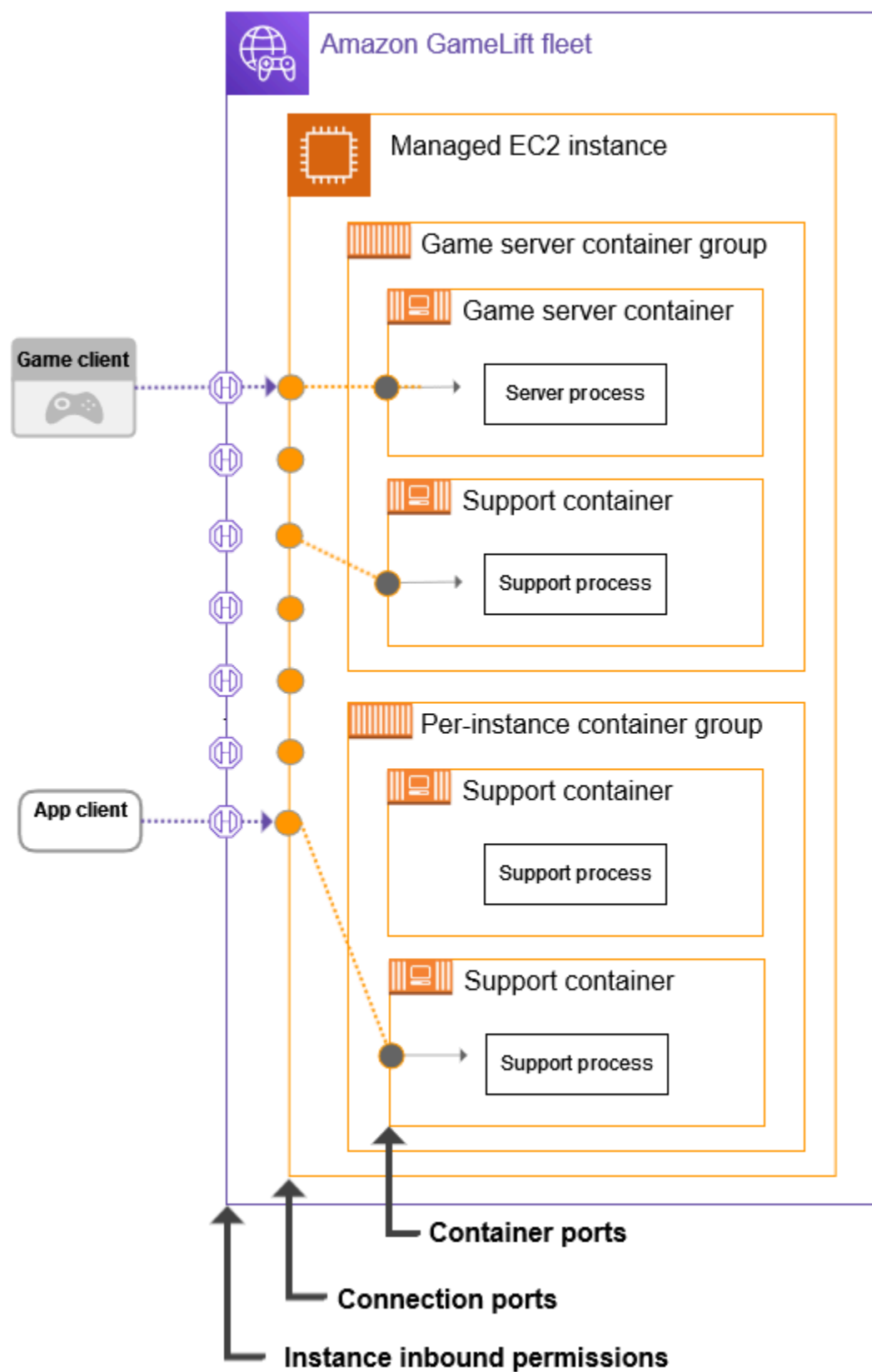
托管容器实例集需额外配置一项设置，以允许访问容器中运行的进程。创建容器定义时，需要指定一组端口，每个连接的进程对应一个端口。这包括：

- 将在游戏服务器容器中同时运行的所有游戏服务器进程。所有游戏服务器进程都必须允许游戏客户端连接，以便其能够加入游戏会话。
- 支持容器中需要外部源连接的任何进程。例如，您可以远程连接到测试应用程序。

设置面向内部的容器端口设置时，Amazon GameLift Servers 使用它们来计算游戏客户端和其他应用程序可以连接的面向外部的入站权限。Amazon GameLift Servers 还管理入站权限和各个容器端口之间的映射，允许玩家访问容器中的游戏会话。这种内部映射通过防止直接访问容器端口，为游戏服务器提供了一层安全保护。您可以根据需要选择自定义实例集的面向外部的端口设置。有关手动设置容器实例集端口的更多信息，请参阅[配置网络连接](#)。

您可以随时修改容器实例集的端口设置。此更改需要部署实例集更新。

下图展示了容器实例集中端口连接的作用。如图所示，您在各个容器上设置端口后，Amazon GameLift Servers 使用此信息在实例集实例上配置足够的端口，以映射到每个容器端口。除非您选择手动设置，否则面向外部的实例入站权限和面向内部的连接端口均由 Amazon GameLift Servers 为实例集计算。



容器日志记录

在托管容器实例集中，会捕获所有容器的标准输出（和标准错误）流。这包括游戏服务器的游戏会话日志。您可以将容器实例集配置为使用以下几个选项之一来处理输出流：

- 将容器输出保存为 Amazon CloudWatch 日志流。每个日志流都引用实例集 ID 和容器。如果您为队列选择此日志记录选项，则可以指定一个 CloudWatch 日志组，该组将组织队列中的所有日志流。然后，您可以根据需要使用 CloudWatch 功能来搜索和分析日志数据。
- 将容器输出保存到 Amazon Simple Storage Service（Amazon S3）存储桶中。您可以根据需要查看、共享或下载内容。
- 禁用日志记录。在此情况下，不会保存容器输出。

Amazon GameLift Servers 将托管集装箱船队的日志数据发送到您 AWS 账户中的 CloudWatch 或 Amazon S3 服务。要查看您的数据，请登录您的 AWS 账户并使用各项服务，使用 AWS 管理控制台或其他工具。通过为容器实例集创建服务角色，您可以将有限的访问权限扩展到 Amazon GameLift Servers 以执行这些操作。

您可以随时修改容器实例集的日志记录配置。此更改需要部署实例集更新。

容器实例集和 Amazon GameLift Servers 代理

常用的容器架构会在每个容器中运行一个进程。在 Amazon GameLift Servers 容器实例集中，游戏服务器容器组有一个游戏服务器容器，其中运行一个游戏服务器进程。采用此架构时，Amazon GameLift Servers 会管理实例集实例上每个游戏服务器容器组中单个游戏服务器进程的生命周期。

如果您选择构建在每个游戏服务器容器组中运行多个游戏服务器进程的容器架构，则需要一种方法来管理所有进程的生命周期。这包括根据需要启动、关闭和替换进程、管理要同时运行的所需数量的进程以及处理故障状态等任务。

您可以选择使用 Amazon GameLift Servers 代理来执行这些任务。对于容器实例集，该代理会执行运行时指令，指定要运行的可执行文件（以及运行数量），提供启动参数并设置游戏服务器激活规则。例如，运行时指令可能会指示代理维护 10 个用于生产环境的游戏服务器进程，以及一个带有特殊启动参数、用于测试的游戏服务器进程。

要在容器实例集中使用代理，请将代理添加到您的容器映像中，并附上一组运行时说明。有关代理的更多信息，请参阅[使用 Amazon GameLift Servers 代理](#)。

创建 Amazon GameLift Servers 托管式容器实例集

创建 Amazon GameLift Servers 托管式容器实例集，以便在 AWS Cloud 中部署和托管容器化游戏服务器。创建容器实例集时，需指定容器组定义，其中还要指定一个或多个容器映像（至少一个需包含游戏服务器生成包）和配置设置。

创建新的托管式容器实例集资源时，会立即开始实例集创建过程的第一阶段。托管式实例集的创建需经历多个阶段：Amazon GameLift Servers 首先预调配一个 EC2 实例，然后安装运行时环境，将容器组部署到实例上，最后开始启动游戏服务器进程。根据游戏服务器生成包所需的运行时环境，Amazon GameLift Servers 会在创建实例集的过程中部署最新版本的亚马逊机器映像（AMI），且实例集内的所有后续实例都将使用同一版本。您可以在控制台中或使用 AWS Command Line Interface（AWS CLI）监控实例集的状态。当实例集准备就绪并可托管游戏会话时，其状态会变更为 ACTIVE。有关实例集创建问题的帮助，请参阅[调试 Amazon GameLift Servers 队组问题](#)。

您可以选择创建一个空的容器实例集，稍后再添加或更新该实例集的容器组定义。如果您创建的实例集没有容器组定义，该实例集将无法进入活动状态。

Note

作为最佳实践，我们建议每 30 天更换一次实例集，以便为托管游戏服务器维护安全且最新的运行时环境。这需要创建一个新实例集并将玩家流量迁移到该实例集。有关更多指导，请参阅[Amazon GameLift Servers 的安全最佳实践](#)。

使用 [Amazon GameLift Servers 控制台](#) 或 AWS Command Line Interface（AWS CLI）创建容器实例集。

Console

在 [Amazon GameLift Servers 控制台](#) 中，选择要创建实例集的 AWS 区域。容器组定义必须与您要创建实例集的区域保持一致。

打开控制台的左侧导航栏，并选择托管容器：实例集。在实例集页面上，选择创建容器实例集。

步骤 1：定义托管式容器实例集详细信息

1. 在容器实例集详细信息部分，输入实例集描述。
2. 为实例集指定 IAM 角色。此角色具有 Amazon GameLift Servers 代表您管理容器实例集所需的权限。要获取创建所需服务角色的帮助，请参阅[为 Amazon GameLift Servers 设置 IAM 服务角色](#)。

3. 选择日志配置选项。CloudWatch 选项默认处于选中状态。根据所选选项提供所需信息。
4. 向实例集添加容器组。此为可选步骤。您可以选择创建不带容器组的实例集，并计划后续再添加。没有任何容器组的实例集不会部署任何实例集实例，且尚无法托管任何游戏，但会创建实例集资源。
 - 选择游戏服务器容器组定义。（可选）指定要部署的定义版本。如果不指定版本号，Amazon GameLift Servers 会自动使用最新版本。
 - （可选）添加每个实例的容器组定义和版本。如果不指定版本号，Amazon GameLift Servers 会自动使用最新版本。
5. 在其他详细信息中，您可以设置一些可选的自定义项。创建容器实例集时，不需要这些设置。

步骤 2：定义实例详细信息

1. 在实例部署中，选择一个或多个远程位置来部署实例。系统会自动选择主区域（即创建实例集的所在区域）。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。

- 对于在 2022 年 2 月 28 日之前创建的、包含未启用区域的实例集，此要求不会对其产生影响。
- 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或 Local Zones。

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](#)。请参阅《AWS Local Zones 用户指南》中的[Local Zones 入门](#)。

2. 为实例集选择一个实例配置。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筛选条件。

有关选择实例类型的更多信息，请参阅[配置容器实例集](#)。您选择的实例类型的大小将影响游戏服务器容器组打包到每个实例集实例上的方式。根据您的选择，可以考虑查看每个实例所需的游戏服务器容器组的设置。

步骤 4：审核并创建

- 检查实例集配置设置。

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 [更新 Amazon GameLift Servers 实例集配置](#)。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 [利用 Amazon GameLift Servers 扩展游戏托管容量](#)。您还可以添加或删除远程位置。

检查完成后，选择创建。

如果请求成功，控制台将显示新实例集资源的详细信息页面。初始状态为 NEW，此时 Amazon GameLift Servers 已启动实例集创建流程。您可以在实例集页面上跟踪新实例集的状态。当实例集进入 ACTIVE 状态时，即表示已准备好托管游戏会话。

AWS CLI

要使用 AWS CLI 创建容器实例集，请打开命令行窗口，然后使用 `create-container-fleet` 命令。有关此命令的更多信息，请参阅《AWS CLI 命令参考》中的 [create-container-fleet](#)。

下面显示的示例 `create-container-fleet` 请求将创建一个具有以下特征的新容器实例集：

- ContainerGroupsConfiguration 仅指定游戏服务器容器组定义：MyAdventureGameContainerGroup。将部署到每个实例集实例的游戏服务器容器组的数量由 Amazon GameLift Servers 计算。
- 该实例集默认使用 c5.large 按需型实例。
- 默认情况下，实例集会按照 Amazon GameLift Servers 的计算结果，开放一组连接端口和入站权限端口。它会将容器组部署到以下位置：

```
aws gamelift create-container-fleet \  
  --fleet-role-arn arn:aws:iam::MyAccount:role/MyContainersRole \  
  --game-server-container-group-definition-name "rn:aws:gamelift:us-  
west-2:111122223333:containergroupdefinition/MyAdventureGameContainerGroup:2" \  
  --game-server-container-group-definition-version 1
```

如果创建实例集请求成功，Amazon GameLift Servers 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容器实例集 ID。然后，Amazon GameLift Servers 会将实例集状态和位置状态设置为 New，并启动实例集激活流程。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-container-fleet](#)
- [describe-fleet-capacity](#)
- [describe-fleet-port-settings](#)
- [describe-fleet-utilization](#)
- [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-container-fleet](#)
- [update-fleet-capacity](#)
- [update-fleet-port-settings](#)
- [create-fleet-locations](#)
- [delete-fleet-locations](#)

为 Amazon GameLift Servers 容器实例集创建容器组定义

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实例集。它是一个蓝图，告知 Amazon GameLift Servers 应将哪些容器映像部署到实例集，以及如何运行这些映像。创建容器实例集时，需要指定要部署到实例集的容器组定义。有关容器组的更多信息，请参阅[容器实例集组件](#)。

开始之前

开始创建容器组定义之前的操作提示：

- 完成您的容器镜像并将其推送到亚马逊弹性容器注册表 (Amazon ECR) Container Registry 存储库，该存储库与您计划创建容器组的位置 AWS 区域 相同。Amazon GameLift Servers 在创建容器组定义时捕获每个映像的快照，并在部署到容器队列时使用该快照。请参阅[为 Amazon GameLift Servers 构建容器映像](#)。
- 将您的容器定义创建为 JSON 文件。容器组定义包含一个或多个容器定义。如果您使用创建容器组定义，则可以使用 JSON 文件 AWS CLI for Amazon GameLift Servers。
- 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[Amazon GameLift Servers 的 IAM 权限示例](#)。

创建游戏服务器容器组定义

游戏服务器容器组用于运行您的游戏服务器软件。一个游戏服务器容器组有一个游戏服务器容器，该容器运行游戏服务器可执行文件。它还可以包含一个或多个支持容器，用于运行其他软件来支持您的游戏服务器。（这类容器有时称为“Sidecar”容器。）

本主题介绍如何使用Amazon GameLift Servers控制台或 AWS CLI 工具创建简单的游戏服务器容器组定义。有关可选功能的更多详细信息，请参阅[自定义 Amazon GameLift Servers 容器实例集](#)。

Note

创建容器组定义和容器定义后，您可以修改其中的大部分设置。如果对容器定义进行更改，Amazon GameLift Servers 会捕获更新后的容器映像的新快照。

要创建简单的游戏服务器容器组定义，请执行以下操作：

以下说明介绍如何使用所需的最少参数和 Amazon GameLift Servers 默认值来创建容器组定义。

Console

在[Amazon GameLift Servers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，并选择托管容器：组定义。在容器组定义页面上，选择创建组定义。

步骤 1：定义容器组定义详情

1. 输入容器组定义名称。该名称在 AWS 账户 和区域中必须是唯一的。
2. 选择游戏服务器容器组类型。
3. 在总内存限制中，输入要为容器组中的所有容器提供的最大内存资源量。如需计算此值的帮助，请参阅[设置资源限制](#)。
4. 在 vCPU 总限制中，输入要为容器组中的所有容器提供的最大计算能力。如需计算此值的帮助，请参阅[设置资源限制](#)。

步骤 2：添加容器定义

一个游戏服务器容器组至少需包含一个游戏服务器容器。在控制台中，您创建的第一个容器定义即为游戏服务器容器。此步骤介绍如何定义游戏服务器容器定义所需的最低设置。

1. 输入容器定义名称。为该组定义的每个容器必须有唯一的名称值。
2. 关联到包含游戏服务器生成包的容器映像。输入公共或私有存储库中容器映像的 Amazon ECR 映像 URI。可使用以下任意一种格式：
 - 仅映像 URI : [AWS ##].dkr.ecr.[AWS ##].amazonaws.com/[repository ID]
 - 映像 URI + 摘要 : [AWS ##].dkr.ecr.[AWS ##].amazonaws.com/[repository ID]@[digest]
 - 映像 URI + 标签 : [AWS ##].dkr.ecr.[AWS ##].amazonaws.com/[repository ID]:[tag]
3. 指定游戏服务器生成包使用的 Amazon GameLift Servers 服务器 SDK 版本。对于容器实例集，该版本必须为 5.2.0 或更高。
4. 在内部容器端口范围中，设置协议并定义端口范围。范围大小必须大于将在此容器中运行的并发游戏服务器进程数。如果每个游戏服务器容器仅运行一个服务器进程，则此端口范围只需设置少量端口即可。有关更多详细信息，请参阅[配置网络连接](#)。
5. 根据需要添加更多容器以运行其他支持软件。新增容器将自动指定为支持容器。一个游戏服务器容器组只能有一个游戏服务器容器和最多八个支持容器。至少需提供以下必要设置：
 - 容器定义名称
 - ECR 映像 URI。
 - 容器内部端口（仅当容器存在需要访问网络的进程时才设置此端口。）

步骤 3：配置依赖关系

- 如果您的容器组定义包含多个容器，则可以选择在容器之间设置依赖关系。有关更多信息，请参阅[设置容器依赖关系](#)。

第 3 步：审核并创建

1. 查看所有容器组定义设置。使用编辑功能可对任何部分进行更改，包括容器组的每个容器定义。
2. 检查完成后，选择创建。

如果请求成功，控制台将显示新容器组定义资源的详细信息页面。初始状态为 `COPYING`，此时 Amazon GameLift Servers 开始为该组的所有容器映像拍摄快照。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，才能使用它创建容器实例集。

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。有关架构示例，请参阅[创建容器定义 JSON 文件](#)。

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。

此示例展示了创建游戏服务器容器组定义的请求。假设您已经创建了一个包含该组容器定义的 JSON 文件。

```
aws gamelift create-container-group-definition \
  --name MyAdventureGameContainerGroup \
  --operating-system AMAZON_LINUX_2023 \
  --container-group-type GAME_SERVER \
  --total-memory-limit-mebibytes 4096 \
  --total-vcpu-limit 1 \
  --game-server-container-definition file://MyAdventureGameContainers.json
```

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。容器定义需指定存储容器映像的 Amazon ECR 存储库，并可选择配置网络端口、CPU 与内存使用限制及其他设置。我们建议创建单个 JSON 文件，集中存储容器组中所有容器的配置。通过维护文件有助于存储、共享这些关键配置并进行版本跟踪。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. 创建并打开一个新的 .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 Servers 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：游戏服务器容器定义

Example

此示例描述了游戏服务器容器组的必备容器。该必备副本容器包括您的游戏服务器应用程序、Amazon GameLift Servers 代理，也可以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": "MyAdventureGameServer",
  "ImageUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/gl-
containers:myadventuregame-server",
  "PortConfiguration": {
    "ContainerPortRanges": [
      {
        "FromPort": 2000,
        "Protocol": "TCP",
        "ToPort": 2010
      }
    ]
  },
  "ServerSdkVersion": "5.2.0"
}
```

Amazon GameLift Servers Anywhere 实例集

如果您想通过自有托管资源利用 Amazon GameLift Servers 功能，可以使用 Anywhere 实例集。Anywhere 实例集通常用作迭代开发的测试环境，或者在混合托管解决方案中与托管式实例集一起使用。

Anywhere 实例集由您提供和管理的一组计算资源（虚拟或物理）组成。从本地笔记本电脑到偏远哨站，计算可以驻留在任何具有连接的地理位置。在设置 Anywhere 实例集时，您可以向实例集添加计算资源，方法是通过 Amazon GameLift Servers 注册计算资源。每项计算资源都是使用其 IP 地址（或 DNS 名称）注册的，这样 Amazon GameLift Servers 就可以与之建立连接。

您可以将游戏服务器软件部署到 Anywhere 实例集，方法是将其安装在每个计算上并启动游戏服务器进程。每个启动的游戏服务器进程都会与 Amazon GameLift Servers 服务建立连接，并报告已准备好托管游戏会话。您可以使用现有的配置管理和部署工具来处理初始的部署和主机管理任务。与 Amazon GameLift Servers 结合使用还需要执行其他几项任务，包括：

- 注册和取消注册计算以在实例集中添加或删除它们。

- 在所有计算机上保留 up-to-date 身份验证令牌。计算资源上的服务器进程在连接到 Amazon GameLift Servers 服务时会使用它们。

Note

可选择使用 Amazon GameLift Servers 代理部署 Anywhere 实例集，以自动执行这些密钥管理任务。请参阅[使用 Amazon GameLift Servers 代理](#)。

请参阅以下主题，了解如何设置和维护 Anywhere 实例集：

- [借助 Amazon GameLift Servers Anywhere 进行托管的开发路线图](#)
- [借助 Amazon GameLift Servers 进行混合托管的开发路线图](#)
- [借助 Amazon GameLift Servers Anywhere 为迭代开发做好准备](#)
- [创建 Amazon GameLift Servers Anywhere 实例集](#)
- [更新 Amazon GameLift Servers 实例集配置](#)

Anywhere 实例集创建工作流程

对于 Anywhere 实例集，Amazon GameLift Servers 只会设置实例集资源。您需要设置计算并将其注册到实例集，还需要安装游戏服务器软件并启动游戏服务器进程以托管游戏会话。

1. Amazon GameLift Servers 在实例集的主区域中创建实例集资源。实例集状态和自定义位置状态设置为新建。
2. Amazon GameLift Servers 开始将事件写入实例集事件日志。
3. 创建实例集资源之后，Amazon GameLift Servers 会将实例集状态设置为活跃。此时，您可以将新计算注册到实例集。

创建 Amazon GameLift Servers Anywhere 实例集

本主题介绍如何创建 Amazon GameLift Servers Anywhere 实例集。借助 Anywhere 实例集，您可以使用核心的 Amazon GameLift Servers 游戏会话管理功能，同时使用您自己的计算资源来托管游戏会话。可以为您的本地硬件或其他基于云的资源创建 Anywhere 队列。

Anywhere 实例集通常在混合托管解决方案中与 Amazon GameLift Servers 托管式实例集一起使用。在开发使用 Amazon GameLift Servers 托管的游戏时，它们还提供有用的测试环境。请参阅以下主题，详细了解何时以及如何将 Amazon GameLift Servers Anywhere 实例集纳入游戏托管解决方案：

- [混合托管](#)
- [为 Amazon GameLift Servers 部署托管实例集](#)
- [借助 Amazon GameLift Servers Anywhere 为迭代开发做好准备](#)

由于 Anywhere 实例集是自管理式的，因此设置实例集需要完成一些额外的工作。要让 Anywhere 实例集做好托管游戏会话和玩家的准备，您需要完成以下任务：

主题

- [开始之前](#)
- [创建自定义位置](#)
- [创建 Anywhere 实例集](#)
- [向实例集添加计算](#)
- [启动游戏服务器](#)

开始之前

在创建 Anywhere 实例集之前，请执行以下任务。有关更详细的指导，请参阅[借助 Amazon GameLift Servers Anywhere 进行托管的开发路线图](#)或[借助 Amazon GameLift Servers 进行混合托管的开发路线图](#)。

- 将您的游戏服务器代码与 Amazon GameLift Servers 服务器 SDK 版本 5.x (或更高版本) 集成。您无需完成所有游戏集成任务，只需完成游戏服务器生成包所需的任务即可。常见的做法是将本地计算机设置为 Anywhere 实例集，并使用命令行界面测试游戏服务器集成 (请参阅[使用 Amazon GameLift Servers Anywhere 设置本地测试](#))。您可以在开发时加入其他组件 (例如支持 Amazon GameLift Servers 的游戏客户端)。
- 将您的游戏服务器软件打包，以便安装到您的 Anywhere 实例集计算。该软件包应包含您的集成式游戏服务器生成包以及运行您的游戏服务器所需的所有支持软件。
- 决定是否对您的 Anywhere 实例集使用 Amazon GameLift Servers 代理。该代理是一种计算进程管理工具，可自动执行与管理服务器进程和计算资源相关的一些关键任务，以便与 Amazon GameLift Servers 配合使用。有关更多信息，请参阅[使用 Amazon GameLift Servers 代理](#)。

创建自定义位置

创建一个自定义位置来表示您的计算资源的物理位置。创建 Anywhere 实例集时，必须至少已定义一个自定义位置。您可以随时创建更多自定义位置并将其添加到现有实例集。

创建自定义位置

使用 Amazon GameLift Servers 控制台或 AWS Command Line Interface (AWS CLI) 创建自定义位置。

Console

在 [Amazon GameLift Servers 控制台](#) 中，使用导航窗格打开位置页面。选择创建位置以打开“创建”对话框。

1. 在对话框中，输入位置名称。作为最佳实践，请使用可描述一组计算资源的有意义位置的名称。这可以是地理位置、数据中心名称或其他位置标识符。Amazon GameLift Servers 会在您的自定义位置名称后面加上 custom-。
2. （可选）向自定义位置添加标签。每个标签都包含定义的一个键和一个可选值。为您希望以有用的方式（如按用途、所有者或环境）分类的 AWS 资源分配标签。为每个要添加的标签选择添加新标签。
3. 选择创建。

AWS CLI

使用 [create-location](#) 命令创建自定义位置。提供一个 location-name 值，该值必须以 custom- 开头。作为最佳实践，请使用可描述一组计算资源的有意义位置的名称。这可以是地理位置、数据中心名称或其他位置标识符。

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

Output

```
{  
  "Location": {  
    "LocationName": "custom-location-1",  
    "LocationArn": "arn:aws:gamelift:us-east-1:111122223333:location/custom-  
location-1"  
  }  
}
```

```
}
```

创建 Anywhere 实例集

为您拥有的一组计算资源创建一个 Anywhere 实例集。新的 Anywhere 实例集一开始是空的；您可以通过注册计算向实例集添加计算。

创建后，新 Anywhere 实例集的状态会从 NEW 快速切换到 ACTIVE。您可以在实例集进入 ACTIVE 状态后向其添加计算。

创建 Anywhere 实例集

使用 Amazon GameLift Servers 控制台或 AWS Command Line Interface (AWS CLI) 创建 Anywhere 实例集。

Console

在 [Amazon GameLift Servers 控制台](#) 中，使用导航窗格打开实例集页面。选择创建实例集以启动实例集创建工作流程。

步骤 1 选择计算类型

选择 Anywhere 选项，然后选择下一步。

步骤 2 定义实例集详细信息

在此步骤中，指定一些关键的实例集范围的设置。

1. 填写实例集详细信息部分：

- a. 输入实例集名称。我们建议使用实例集命名模式，以便于在查看实例集列表时识别实例集类型。
- b. 提供对实例集的简短描述。

2. 根据需要设置这些可选的其他详细信息。您可以稍后更新这些实例集设置。

- a. 创建用于生产或生产前测试的实例集时，可使用此设置指定实例集计算资源的每小时成本值。Amazon GameLift Servers 可以在游戏会话放置过程中使用此信息，以根据成本选择托管资源。
- b. 如果要合并此实例集和其他实例集的指标数据，请指定指标组名称。对要合并在一起的所有实例集使用相同的指标组名称。查看指标组的指标以查看聚合数据。

3. 向自定义位置添加可选标签。每个标签都包含定义的一个键和一个可选值。为您希望以有用的方式（如按用途、所有者或环境）分类的 AWS 资源分配标签。为每个要添加的标签选择添加新标签。
4. 选择下一步以继续工作流程。

步骤 3 选择自定义位置

在此步骤中，确定您计划添加到此实例集的计算的物理位置。您可以现在指定一个或多个位置，也可以稍后根据需要添加或删除位置。

1. 在自定义位置中，为实例集的计算选择一个或多个位置。该列表包含在您当前选择的 AWS 区域中已定义的所有自定义位置。要定义希望添加到实例集的新自定义位置，请选择创建位置。
2. 选择下一步以继续工作流程。

步骤 4 审核并创建

在创建实例集之前，请审查您的设置。

当您准备好部署新实例集时，请选择创建。Amazon GameLift Servers 会立即开始实例集激活流程，分配一个唯一 ID 并将实例集状态设为 NEW。您可以在实例集页面上跟踪实例集的进度。

AWS CLI

使用 [create-fleet](#) 命令创建计算类型 ANYWHERE 的实例集。提供一个名称和至少一个自定义位置。Amazon GameLift Servers 会在当前的默认 AWS 区域中创建 Anywhere 实例集资源（您也可以添加 `--region` 标签来指定其他 AWS 区域）。

以下示例请求创建了一个具有最少必要设置的新实例集。将 *FleetName* 和 *custom-location* 替换为您自己的信息。

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

响应示例

```
{
  "FleetAttributes": {
    "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
    "FleetArn": "arn:aws:gamelift:us-west-2:111122223333:fleet/fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
    "Name": "HardwareAnywhere",
    "CreationTime": "2023-02-23T17:57:42.293000+00:00",
    "Status": "ACTIVE",
    "MetricGroups": [
      "default"
    ],
    "CertificateConfiguration": {
      "CertificateType": "DISABLED"
    },
    "ComputeType": "ANYWHERE"
  }
}
```

创建后，新 Anywhere 实例集的状态会快速切换到 ACTIVE。您可以在实例集进入 ACTIVE 状态后向其添加计算。

请注意，响应不包含实例集位置。您可以通过调用 [describe-fleet-attributes](#) 和 [describe-fleet-location-attributes](#) 来检索完整的实例集详细信息。

向实例集添加计算

要向一个实例集添加计算资源并让该实例集做好托管游戏会话的准备，请执行以下任务：

- 将计算注册到实例集。注册会告知 Amazon GameLift Servers 哪些物理托管资源是实例集的一部分。
- 为计算请求身份验证令牌。在计算资源上运行的每台游戏服务器都需要此令牌才能连接到 Amazon GameLift Servers 服务。身份验证令牌是临时的，必须定期刷新。

Note

如果您使用 Amazon GameLift Servers 代理来部署游戏服务器软件，则可跳过此步骤。代理会自动注册每个计算并为计算保持有效的身份验证令牌。请参阅[使用 Amazon GameLift Servers 代理](#)。

您可以通过使用 AWS CLI 或以编程方式调用适用于 Amazon GameLift Servers 的 AWS SDK 来注册计算资源和请求身份验证令牌。这些操作无法通过 Amazon GameLift Servers 控制台进行。

作为最佳实践，建议通过向每个计算添加启动脚本来自动执行这两项任务。启动脚本会自动调用 `register-compute` 和 `get-compute-auth-token` 命令。您还可以自动执行任务，以便在计算的整个生命周期内定期刷新身份验证令牌，并在关机时取消注册计算。

每个启动操作都会返回特定于计算的值，您需要将这些值存储在计算上。当游戏服务器进程在计算上启动时，它必须在初始化与 Amazon GameLift Servers 服务的连接时将这些值作为服务器参数传递（请参阅服务器 SDK 参考中的 [ServerParameters](#)）。我们建议您将这些特定于计算的值（或其存储位置）设置为环境变量。如果您使用的是 Amazon GameLift Servers 代理，它将为您处理此任务。特定于计算的值如下：

- `register-compute` 会为 `GameLiftServiceSdkEndpoint` 返回一个值。请将此值设置为 `websocketUrl` 服务器参数。
- `compute-auth-token` 会返回身份验证令牌。请将此值设置为 `authToken` 服务器参数。

AWS CLI

以下说明介绍了如何使用 AWS CLI 手动提交每个请求。

注册计算

调用 [register-compute](#) 以注册计算。确定要向其添加计算的实例集 ID。提供以下计算信息：有意义的名称、IP 地址和位置。计算的位置必须是已与实例集关联的自定义位置。如果要使用其他自定义位置，请使用 Amazon GameLift Servers 控制台更新实例集，或调用 AWS CLI 命令 [create-fleet-locations](#) 向实例集添加自定义位置。

在以下示例中，替换计算和实例集的占位符值。`fleet-id` 值将在您创建 Anywhere 实例集时返回。您可以通过调用 [describe-fleet-attributes](#) 和 [describe-fleet-location-attributes](#) 来检索完整的实例集详细信息。

```
aws gamelift register-compute \  
  --compute-name HardwareAnywhere \  
  --fleet-id arn:aws:gamelift:us-east-1:111122223333:fleet/  
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \  
  --ip-address 10.1.2.3 \  
  --location custom-location-1
```

示例输出

```
{
  "Compute": {
    "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
    "FleetArn": "arn:aws:gamelift:us-west-2:111122223333:fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
    "ComputeName": "HardwareAnywhere",
    "ComputeArn": "arn:aws:gamelift:us-west-2:111122223333:compute/
HardwareAnywhere",
    "IpAddress": "10.1.2.3",
    "ComputeStatus": "Active",
    "Location": "custom-location-1",
    "CreationTime": "2023-02-23T18:09:26.727000+00:00",
    "GameLiftServiceSdkEndpoint": "wss://us-west-2.api.amazongamelift.com"
  }
}
```

请求身份验证令牌

调用 [get-compute-auth-token](#) 以请求有效的身份验证令牌。注册计算。确定实例集 ID 和计算名称。

在以下示例中，替换计算和实例集的占位符值。fleet-id 值将在您创建 Anywhere 实例集时返回。您可以通过调用 [describe-fleet-attributes](#) 来检索完整的实例集详细信息。要查找计算信息，请使用实例集 ID 调用 [list-compute](#)，以查看注册到实例集的所有计算。

```
aws gamelift get-compute-auth-token \
  --fleet-id arn:aws:gamelift:us-east-1:111122223333:fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
  --compute-name HardwareAnywhere
```

示例输出

```
{
  "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
  "FleetArn": "arn:aws:gamelift:us-east-1:111122223333:fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
  "ComputeName": "HardwareAnywhere",
  "ComputeArn": "arn:aws:gamelift:us-east-1:111122223333:compute/
HardwareAnywhere",
  "AuthToken": "0c728041-3e84-4aaa-b927-a0fb202684c0",
  "ExpirationTimestamp": "2023-02-23T18:47:54+00:00"
```

```
}
```

启动游戏服务器

创建 Anywhere 实例集并向该实例集添加一个或多个计算后，就可以开始运行游戏服务器了。

步骤 1 安装游戏服务器软件

将游戏服务器生成包和所有相关软件安装到 Anywhere 实例集中的每个计算。游戏服务器生成包必须与 Amazon GameLift Servers 服务器 SDK 版本 5.x (或更高版本) 集成，并具备与 Amazon GameLift Servers 服务通信所需的最低功能。

步骤 2 让计算做好运行游戏服务器的准备

确保每个计算都已注册并具有有效的身份验证令牌。如果您要使用脚本来管理这些任务，请确保在启动任何游戏服务器进程之前，脚本已在每个计算上运行。

如果您已将 Amazon GameLift Servers 代理与游戏服务器软件一起部署，请确保代理可执行文件已启动。

步骤 3 启动游戏服务器进程

在计算上运行游戏服务器可执行文件的实例。如果游戏服务器生成包已正确集成，游戏服务器进程会调用服务器 SDK 操作 `InitSDK()`，其中包含一组有效的服务器参数。当服务器进程准备好托管游戏会话后，它会调用 `ProcessReady()`。

Note

如果您已使用 Amazon GameLift Servers 代理部署游戏服务器软件，则可跳过此步骤。代理会根据您提供的运行时说明自动启动游戏服务器进程。

您可以通过查看激活中和已激活的服务器进程的服务器进程指标来监控进度。请参阅[Amazon GameLift Servers 队组指标](#)。如果游戏服务器进程初始化失败，请验证该进程是否正在为运行它的计算检索正确的服务器参数值。

构建混合托管解决方案

混合托管解决方案结合了多种游戏托管资源来源来托管您的游戏，包括在中运行的 Amazon GameLift Servers 托管队列 AWS Cloud 以及您自己提供和管理的资源。本主题将介绍构建混合解决方案的几种

常见模式，并提供实操建议，助力您成功整合自管理式游戏托管与由 Amazon GameLift Servers 管理的基于云的游戏托管。

常见的混合模式

- **成本优化**：针对每种场景使用最具成本效益的实例集类型，例如 Anywhere 实例集用于托管基准容量，托管式实例集用于应对峰值需求。
- **地理灵活性**：在高流量 AWS 区域 中部署托管车队，在有现有基础设施或有特定合规要求的任意位置部署受管车队。
- **风险缓解**：通过在多种实例集类型及资源提供商之间分配负载，降低对单一托管方式的依赖。
- **逐步迁移**：从本地或其他托管解决方案过渡到 AWS 逐步迁移，同时保持服务连续性。

开发与生产环境分离

为开发环境和生产环境分别选用不同类型的实例集：

- **开发**：使用 Anywhere 实例集，实现经济高效的开发和测试
- **生产**：托管 Amazon EC2 或容器舰队，用于可扩展、可靠的生产托管

区域优化

根据 AWS 区域 特征优化舰队类型：

- **高流量 AWS 区域**：托管车队具有自动缩放功能，可满足不断变化的需求
- **专业化 AWS 区域**：任何地方都有用于合规性、数据主权或现有基础架构的舰队

容量分层

为不同容量层级选用不同类型的实例集：

- **基准容量**：针对可预测负载，选用 Anywhere 实例集或预留实例
- **容量爆增**：针对峰值需求，选用具备自动扩缩能力的托管式实例集
- **溢出容量**：Spot 实例或 AWS 区域 针对极端峰值的额外实例

实施注意事项

构建混合解决方案时，需考虑以下关键因素：

游戏会话队列配置

配置游戏会话队列时，需纳入所有实例集类型，并设置合理的优先级和延迟首选项，确保在混合基础设施中实现最优放置。

监控和可观测性

对所有实例集类型实施全面监控，以实时掌握混合解决方案的性能、容量及成本情况，保障全局可见性。

操作复杂性

需应对管理多种实例集类型带来的操作复杂性提升，包括不同的部署流程、监控工具及故障排除流程。

网络连接

确保不同托管环境之间的网络连接可靠，尤其针对可能部署在本地或其他云提供商处的 Anywhere 实例集。

混合托管入门

要实施混合托管解决方案，请执行以下操作：

1. 从简入手：先从单一实例集类型开始，待需求逐渐明确后，再逐步新增其他类型的实例集。
2. 规划架构：根据您在成本、性能、合规性及操作复杂性方面的特定需求，设计混合架构。
3. 配置队列：搭建覆盖多种实例集类型的游戏会话队列，并设置合理的优先级和放置策略。
4. 全面测试：在正式投入生产前，针对混合基础设施中的游戏会话放置及失效转移场景进行全面测试。
5. 监控和优化：持续监控所有实例集类型的性能和成本，并根据需要调整配置。

提示

- 将相同的游戏客户端和服务端组件用于托管式和自管理式托管资源。在所有托管资源上提供统一的玩家体验。
- 使用相同的 FlexMatch 对战构建器跨所有托管资源放置匹配项。
- 集中管理混合托管资源，同时在全球范围内部署这些资源。
- 随着玩家需求的波动，跨托管式和自管理式资源无缝管理游戏会话负载。

- 借助 Amazon GameLift Servers 代理，您可以使用相同的工具来管理所有类型的托管资源上的游戏服务器生命周期。
- 跨所有托管资源收集游戏和玩家指标以及日志。利用 Amazon GameLift Servers 功能和其他 AWS 服务来合并数据，开发有凝聚力的可观测性解决方案。

配置游戏会话放置

游戏会话放置是寻找可用游戏服务器来托管新游戏会话的过程。Amazon GameLift Servers 通过游戏会话队列，智能地将游戏会话放置到各个实例集内，同时考虑玩家延迟、实例集容量和成本优化等因素。

配置良好的游戏会话队列可确保玩家连接到最佳可用游戏服务器，在提供卓越性能的同时高效利用您的托管资源。队列可以跨越多个舰队和 AWS 区域，以提供全球覆盖和冗余。

游戏会话队列是 Amazon GameLift Servers 用于搜索可用游戏服务器并选择其托管新游戏会话的主要机制。队列提供了一种更高效的方式来处理大量游戏会话请求，并在多个托管资源实例集内为其寻找放置位置。如果您的托管解决方案使用了多个实例集且需要处理高流量请求，则很可能需要使用队列。

当您的游戏需要为玩家启动新游戏会话时，其会向 Amazon GameLift Servers 服务发送一个放置请求，该服务会将请求导入队列。队列的配置决定了处理请求的时间和方式。处理放置请求时，Amazon GameLift Servers 会在一组实例集内搜索游戏服务器来托管游戏会话。当 Amazon GameLift Servers 到可用游戏服务器并提示其启动会话时，即为放置成功。

主题

- [队列特点](#)
- [Amazon GameLift Servers 游戏会话队列的最佳实践](#)
- [创建游戏会话队列](#)
- [请参阅设置游戏会话置放通知。](#)

队列特点

Amazon GameLift Servers 游戏会话队列是一种 AWS 云资源。您可以在 Amazon GameLift Servers 支持的任何 AWS 区域 中创建队列（请参阅 [Amazon GameLift Servers 服务位置](#)）。游戏会话放置请求将发送到该位置并在该位置进行处理。

通过队列实现游戏会话自动放置，为游戏开发人员和玩家均带来显著优势。这些方法包括：

- 队列可提供“最佳”放置。在处理游戏会话放置请求时，队列会使用 Amazon GameLift Servers FleetIQ 算法，根据一组已定义的首选项（包括成本、位置和玩家延迟）确定队列放置的优先级。
- 队列支持竞价型实例集，有助于降低游戏托管成本。您可以使用 AWS Spot 队列和按需队列配置队列，Spot 队列通常可以显著降低托管成本。由于低成本是放置的关键标准之一，因此队列始终能充分利用成本差异带来的优势。
- 队列能在高需求时期更快地放置新游戏。通过配置包含多个实例集的队列，您可以为游戏会话放置提供更灵活的选项。但是，当需求增长时，其他实例集也可以提供所需的备份容量。对于任何放置请求，如果 Amazon GameLift Servers 无法将游戏会话放置在最首选的位置，它将自动转而评估其他位置。
- 队列能增强游戏服务器的可用性弹性。可能会发生中断。在使用多实例集队列时，速度减慢或停机不一定会影响到玩家访问您的游戏。通过在队列中配置具有不同 AWS 区域 可用区域容量的舰队，您可以帮助确保玩家随时可以找到要加入的游戏会话。
- 获取有关游戏会话放置和队列性能的指标。Amazon GameLift Servers 会发出队列指标，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等统计数据。您可以在 Amazon GameLift Servers 控制台或 CloudWatch 中查看这些指标。

要开始创建基本的入门队列，请参阅[创建游戏会话队列](#)。

主题

- [Amazon GameLift Servers 游戏会话队列的最佳实践](#)
- [创建游戏会话队列](#)
- [请参阅设置游戏会话置放通知。](#)

Amazon GameLift Servers 游戏会话队列的最佳实践

游戏会话队列包含 Amazon GameLift Servers 可以放置新游戏会话的实例集的列表。每个实例集都可以在多个地理位置部署托管资源。选择放置时，队列会根据您为实例集设置的一组优先级选择实例集和实例集位置。

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。

创建游戏会话队列

队列用于在多个实例集和位置放置新游戏会话。您的游戏通过向队列提交放置请求来启动新游戏会话。队列通过配置处理请求的指令进行设置。有关启动游戏会话放置请求的更多信息，请参阅[创建游戏会话](#)。

创建游戏会话队列

这些说明演示如何使用最小配置设置和默认设置创建一个简单可用的队列。自定义队列配置有多种选项可供选择。这些选项可根据游戏需求，助您实现最优放置位置。要了解有关为游戏自定义队列的更多信息，请参阅[自定义游戏会话队列](#)。您可以随时更新大多数队列配置设置。

您可以使用 Amazon GameLift Servers 控制台或 AWS CLI 创建游戏会话队列。

Console

在[Amazon GameLift Servers控制台](#)中，选择要工作的 AWS 区域。打开控制台左侧导航栏并选择队列。

1. 在队列页面中，选择创建队列以启动工作流。
2. 在队列设置下输入以下设置：
 - a. 输入队列名称。此名称必须 AWS 区域 是您在其中创建队列的唯一名称。
 - b. 保留默认超时设置，即 600 秒（或 10 分钟）。此值控制 Amazon GameLift Servers 尝试创建新游戏会话的时间长度，超过该时间后将停止尝试。Amazon GameLift Servers 会持续搜索可用资源，直至请求超时。您可以随时更新队列超时设置。

- c. 跳过玩家延迟策略部分。队列仅在接收包含玩家延迟数据的放置请求时才会使用延迟策略。您可随时向队列添加延迟策略。有关创建延迟策略的更多信息，请参阅[创建玩家延迟政策](#)。
3. 跳过游戏会话放置位置部分，即可使用所有位置的默认设置。此设置允许您创建队列可放置位置的允许列表（也称为筛选配置）。有关按位置和筛选配置划分优先级的更多信息，请参阅[按位置确定放置优先级](#)。
4. 在目标顺序下，向队列添加一个或多个实例集。您可以使用舰队 IDs 或 ARNs，或者使用舰队别名来识别舰队。添加多个实例集时，请注意它们应运行相似的游戏版本，并且与使用此队列的任何游戏客户端兼容。此外，队列中的所有实例集必须具有相同的证书配置。
 - a. 选择创建实例集或别名的区域。对于多位置实例集，这是指“主”区域。
 - b. 对于目标类型，选择实例集或别名。
 - c. 您选择的区域和类型将填充现有实例集或别名的下拉列表。选择其中一项作为队列目标。
 - d. 要为队列指定其他实例集或别名，请选择添加目标并重复上述步骤。
 - e. 添加目的地列表后，使用该 drag-and-drop 功能对目的地进行重新排序。Amazon GameLift Servers 按目的地对展示位置进行优先排序时使用此顺序。
5. 跳过游戏会话放置优先级部分以保持默认优先级顺序。此设置允许您自定义 Amazon GameLift Servers 为新游戏会话放置选择可用托管资源的位置。有关确定放置优先级的更多信息，请参阅[优先考虑游戏会话放置](#)。您可以随时更新队列的放置优先级。
6. 在位置顺序下，保留默认值。当按实例集位置确定优先级时，可使用此设置。它提供了可供使用的位置顺序。使用默认优先级设置时，如果首选目标是拥有多个位置的实例集，则位置将用作决胜因素。
7. 跳过可选的事件通知设置部分。对于处理大量放置请求的队列，事件通知是必需的。对于处理少量队列（例如用于开发或测试目的），您可以通过轮询来跟踪放置请求的状态[DescribeGameSessionPlacement](#)。有关更多详细信息，请参阅[请参阅设置游戏会话置放通知](#)。您可以随时更新队列的事件通知设置。
8. 选择创建以生成一个自定义程度最低的新队列。

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。

- 两个实例集目标。
- 筛选后仅允许在以下位置放置：us-east-1、us-east-2、us-west-2 和 ca-central-1。
- 根据成本确定优先级，然后按指定顺序排列位置。

```
aws gamelift create-game-session-queue \  
  --name "sample-test-queue" \  
  --timeout-in-seconds 300 \  
  --destinations DestinationArn="arn:aws:gamelift:us-east-1:111122223333:fleet/  
fleet-772266ba-8c82-4a6e-b620-a74a62a93ff8" DestinationArn="arn:aws:gamelift:us-  
east-1:111122223333:fleet/fleet-33f28fb6-aa8b-4867-85b4-ceb217bf5994" \  
  --filter-configuration "AllowedLocations=us-east-1, ca-central-1, us-east-2, us-  
west-2" \  
  --priority-configuration PriorityOrder="COST","LOCATION",LocationOrder="us-  
east-1","us-east-2","ca-central-1","us-west-2" \  
  --notification-target "arn:aws:sns:us-east-1:111122223333:gamelift-test.fifo"
```

Note

您可以通过使用队列[describe-fleet-attributes](#)或别名 ID 调用 desc [ribe-alias](#) 来获取舰队和别名 ARN 值。

如果create-game-session-queue请求成功，则Amazon GameLift Servers返回具有新队列配置的[GameSessionQueue](#)对象。现在，您可以使用向队列提交请求[StartGameSessionPlacement](#)。

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 \  
  --filter-configuration "AllowedLocations=us-east-1, us-east-2, us-west-2" \  
  --priority-configuration PriorityOrder="COST","LOCATION",LocationOrder="us-  
east-1","us-east-2","us-west-2" \  
  --notification-target "arn:aws:sns:us-east-1:111122223333:gamelift-test.fifo"
```

```
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 `create-game-session-queue` 请求成功，则 Amazon GameLift Servers 返回具有新队列配置的 [GameSessionQueue](#) 对象。

请参阅设置游戏会话置放通知。

您可以使用 Events 来监控各个置放请求的状态。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。

- 让 Amazon GameLift Servers 使用队列向 Amazon Simple Notification Service (Amazon SNS) 主题发布事件通知。
- 使用自动发布的 Amazon EventBridge 事件及其工具套件来管理事件。

有关 Amazon GameLift Servers 发出的游戏会话置放事件的列表，请参阅 [游戏会话置放事件](#)。

Important

对于高吞吐量置放系统，我们建议使用标准 (非 FIFO) Amazon SNS 主题，而不是 FIFO 主题。FIFO 主题的发布限制低于标准主题，这可能会导致在高负载期间出现节流异常。如果您在使用 FIFO 主题时遇到节流，则可能会丢失队列置放通知。

设置 SNS 主题。

要让 Amazon GameLift Servers 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 Amazon GameLift Servers 事件通知设置 SNS 主题

1. [登录 AWS 管理控制台 并在 v3/home 上打开亚马逊 SNS 控制台。https://console.aws.amazon.com/sns/](https://console.aws.amazon.com/sns/)
2. 在 SNS 主题页面，选择创建主题，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。

JSON

```
{
  "Version": "2012-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__default_statement_ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SNS:GetTopicAttributes",
        "SNS:SetTopicAttributes",
        "SNS:AddPermission",
        "SNS:RemovePermission",
        "SNS:DeleteTopic",
        "SNS:Subscribe",
        "SNS:ListSubscriptionsByTopic",
        "SNS:Publish"
      ],
      "Resource": "arn:aws:sns:us-east-1:111122223333:your_topic_name",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "your_account"
        }
      }
    }
  ],
  {
    "Sid": "__console_pub_0",
```

```

    "Effect": "Allow",
    "Principal": {
      "Service": "gamelift.amazonaws.com"
    },
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-1:111122223333:your_topic_name",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:gamelift:us-east-1:111122223333:gamesessionqueue/your_queue_name"
      }
    }
  }
]
}

```

- c. (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 Simple 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 Servers 自动将所有游戏会话放置事件发布到 EventBridge。使用 EventBridge 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，将事件路由由 PlacementFulfilled 到一个处理连接到游戏会话之前的任务的 AWS Lambda 函数。有关的更多信息 EventBridge，请参阅 [Amazon 是什么 EventBridge？](#) 在《亚马逊 EventBridge 用户指南》中。

以下是一些用于 Amazon GameLift Servers 队列的 EventBridge 规则示例：

匹配来自所有 Amazon GameLift Servers 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"
  ]
}
```

参考实现 AWS CDK

有关使用 Amazon SNS 和 Amazon DynamoDB 实现基于事件的游戏会话放置的完整参考实现 AWS Lambda，请参阅亚马逊工具包中基于[事件的游戏会话](#)放置指南。GameLift 本指南包括用于部署具有处理游戏会话放置事件 AWS Lambda 功能的完整事件驱动型游戏会话放置系统的 AWS CDK 模板、用于跟踪游戏会话放置状态的 Amazon DynamoDB 表以及队列配置的最佳实践。Amazon GameLift Servers

自定义游戏托管解决方案

在搭建好基本的游戏托管解决方案后，您可以通过以下主题对其进行自定义与增强配置，以优化玩家体验、提升成本效益，并添加高级功能。本节按主要影响的组件，分类介绍各种自定义选项。请根据您的游戏需求与玩家群体特征，选择最适配的自定义设置。

主题

- [游戏服务器生成包自定义](#)
 - [将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)
 - [让您的游戏服务器访问 Amazon GameLift Servers 实例集数据](#)
 - [为 Amazon GameLift Servers 设置 VPC 对等连接](#)
- [玩家会话和对战自定义](#)
 - [生成玩家 ID](#)
 - [为 Amazon GameLift Servers 添加 FlexMatch 对战](#)
- [游戏会话放置自定义](#)
 - [自定义游戏会话队列](#)
 - [优先考虑游戏会话放置](#)
 - [为竞价型实例构建队列](#)
- [托管资源自定义](#)
 - [为托管式实例集选择计算资源](#)

- [自定义 Amazon GameLift Servers 容器实例集](#)
- [借助竞价型实例集降低游戏托管成本](#)
- [在托管式 Amazon GameLift Servers 上优化游戏服务器运行时配置](#)
- [使用 Amazon GameLift Servers 代理](#)
- [使用别名抽象化 Amazon GameLift Servers 实例集名称](#)

游戏服务器生成包自定义

游戏服务器版本自定义提供了增强游戏服务器功能的机会，包括利用其他 AWS 服务。这些自定义设置可将游戏服务器功能从基础托管层面进一步向外拓展，以支持高级功能和集成。

将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源

当您创建要在 Amazon GameLift Servers 实例集上部署的游戏服务器生成包时，您可能希望游戏生成包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 Servers 管理您的游戏托管实例集，因此您必须向 Amazon GameLift Servers 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 队列和 Anywhere 队列收集指标、日志和跟踪。
- 将实例日志数据发送到 Amazon CloudWatch 日志。
- 将消息存储在 Amazon Simple Storage Service (Amazon S3) 存储桶中。
- 读写在 Amazon DynamoDB 数据库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 SQS) 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 Servers 支持以下方法来建立访问权限：

- [使用 IAM 角色访问 AWS 资源](#)
- [通过 VPC 对等互连访问 AWS 资源](#)

使用 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. [将实例集与 IAM 角色关联](#)

创建 IAM 角色。

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

有关如何设置 IAM 角色的说明，请参阅[为 Amazon GameLift Servers 设置 IAM 服务角色](#)。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

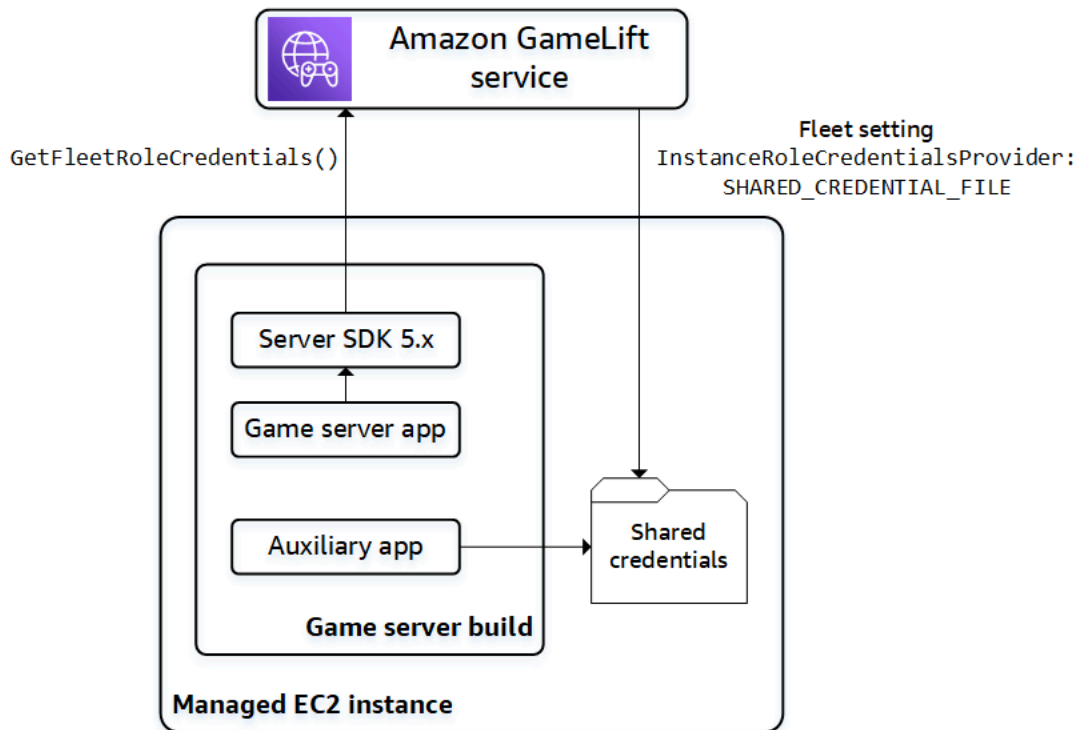
在此步骤中，您将应用程序配置为获取 IAM 角色的安全证书，并在与您的 AWS 资源交互时使用这些证书。参见下表，根据（1）应用程序的类型以及（2）您的游戏用于与 Amazon GameLift Servers 通信的服务器 SDK 版本来确定如何修改您的应用程序。

	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 <code>GetFleetRoleCredentials()</code> 。	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)。

Note

对于集装箱舰队，FleetRoleArn证书将注入每个容器。您的应用程序可以使用默认 AWS 凭证提供程序来访问这些证书。您仍然可以调用GetFleetRoleCredentials()，这会返回相同的凭据。这些舰队角色证书只能在容器内访问。

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 Servers 服务器 SDK 5.x 集成）中，调用 `GetFleetRoleCredentials()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) 以检索一组临时凭证。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 SDK 5.x 与游戏服务器生成包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证。Amazon GameLift Servers 会为每个实例集实例生成一个凭证配置文件。这些凭证可供实例上的所有应用程序使用。Amazon GameLift Servers 会不断刷新临时凭证。

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 Servers 队列实例的指标

如果您想使用 Amazon CloudWatch 代理从您的 Amazon GameLift Servers 车队收集指标、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 角色并获取与您的 AWS 资源交互的证书。任何在带有服务器 SDK 4 或更早版本的 Amazon GameLift Servers 实例集实例上运行的应用程序都可以代入 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。此操作会返回一组临时证书，用于授权应用程序访问 AWS 资源。有关更多信息，请参阅 IAM 用户指南中的[将临时证书与 AWS 资源配合使用](#)。

将实例集与 IAM 角色关联

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。

对于集装箱船队：

- [FleetRoleArn](#)— 将此参数设置为 IAM 角色的 ARN。

对于其他舰队类型：

- [InstanceRoleArn](#)— 将此参数设置为 IAM 角色的 ARN。
- [InstanceRoleCredentialsProvider](#)— Amazon GameLift Servers 要提示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) 对等互连在 Amazon GameLift Servers 实例上运行的应用程序与其他 AWS 资源之间进行通信。VPC 是您定义的虚拟专用网络，其中包括通过您管理的一组资源 AWS 账户。每个 Amazon GameLift Servers 队组都有它自己的 VPC。通过 VPC 对等互连，您可以在队列和其他 AWS 资源的 VPC 之间建立直接的网络连接。

Amazon GameLift Servers 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[为 Amazon GameLift Servers 设置 VPC 对等连接](#)。

 Note

容器队列不支持 VPC 对等互连。

让您的游戏服务器访问 Amazon GameLift Servers 实例集数据

在某些情况下，您的自定义游戏生成包或 Amazon GameLift Servers Realtime 脚本可能需要有关 Amazon GameLift Servers 实例集的信息。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。（许多游戏都使用这些数据进行 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 Servers舰队资源的属性](#)。

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",
  "name": "ReallyFunGameTestFleet08",
  "fleetType": "ON_DEMAND",
  "instanceRoleArn": "arn:aws:iam::123456789012:role/S3AccessForGameLift",
  "instanceType": "c5.large",
  "serverLaunchPath": "/local/game/reallyfungame.exe"
}
```

为 Amazon GameLift Servers 设置 VPC 对等连接

本主题提供有关如何在 Amazon GameLift Servers 托管游戏服务器与其他非 Amazon GameLift Servers 资源之间建立 VPC 对等连接的指导。使用亚马逊虚拟私有云 (VPC) Virtual Private Cloud 对等连接使您的游戏服务器能够与您的 AWS 其他资源（例如 Web 服务或存储库）进行直接私密通信。您可以与任何在您有权访问的 AWS 账户上运行 AWS 并由其管理的资源建立 VPC 对等关系。

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其他 AWS 资源进行直接和私密通信的首选选项，请参阅[将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。

如果您已经熟悉 Amazon VPCs 和 VPC 对等互连，请理解，设置与 Amazon GameLift Servers 游戏服务器的对等互连有些不同。您无权访问包含游戏服务器的 VPC（该 VPC 由 Amazon GameLift Servers 服务控制），因此无法直接为其请求 VPC 对等连接。而应首先对非 Amazon GameLift Servers 资源预授权 VPC，以接受来自 Amazon GameLift Servers 服务的对等连接请求。然后，触发 Amazon

GameLift Servers 以请求刚刚授权的 VPC 对等连接。Amazon GameLift Servers 处理创建对等连接、设置路由表和配置连接的任务。

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 IDs 通过登录[AWS 管理控制台](#)并查看您的账户设置来找到 AWS 账户。要获取凭证，请转到 IAM 控制台。

- AWS 用于管理 Amazon GameLift Servers 游戏服务器的帐户。
- AWS 用于管理非 Amazon GameLift Servers 资源的帐户。

如果您对 Amazon GameLift Servers 和非 Amazon GameLift Servers 资源使用同一账户，则仅需该账户的 ID 和凭证。

2. 为每个 VPC 获取标识符。

获取以下信息，以便两者 VPCs 相互对等：

- Amazon GameLift Servers 游戏服务器的 VPC：这是您的 Amazon GameLift Servers 实例集 ID。您的游戏服务器部署在 EC2 实例的队组上的 Amazon GameLift Servers 中。队组都自动放置在自己的 VPC 中，后者由 Amazon GameLift Servers 服务进行管理。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 Servers AWS 资源的 VPC — 您可以与您有权访问的 AWS 账户上运行 AWS 并由其管理的任何资源建立 VPC 对等关系。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[AWS 管理控制台](#)适用于 Amazon 的 VPC 并查看您的 VPC 来找到 VPC ID VPCs。

Note

设置对等互连时，两者 VPCs 必须位于同一区域。您的 Amazon GameLift Servers 队组游戏服务器的 VPC 与队组在相同区域中。

3. 授权 VPC 对等连接。

在此步骤中，您预授权将来从 Amazon GameLift Servers 将用于游戏服务器的 VPN 与用于非 Amazon GameLift Servers 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 对等互连，请调用服务 API [CreateVpcPeeringAuthorization\(\)](#) 或使用 AWS CLI 命令 `create-vpc-peering-authorization`。使用管理您的非 Amazon GameLift Servers 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 Servers 资源的 VPC。
- Amazon GameLift Servers AWS 账户 ID — 这是您用来管理 Amazon GameLift Servers 车队的账户。

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 Servers 建立对等连接。

要请求 VPC 对等互连，请调用服务 API [CreateVpcPeeringConnection\(\)](#) 或使用 AWS CLI 命令 `create-vpc-peering-connection`。使用管理您的 Amazon GameLift Servers 游戏服务器的账户发出此调用。使用以下信息来确定要 VPCs 对等的两个：

- 对等 VPC ID 和 AWS 账户 ID — 这是您的非 Amazon GameLift Servers 资源的 VPC，也是您用来管理这些资源的账户。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 Servers 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 对等连接相关的队列事件，包括成功和失败事件。

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 Servers 队组，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 管理控制台](#)并查看您的账户设置来找到 AWS 账户。要获取凭证，请转到 IAM 控制台。

- AWS 用于管理 Amazon GameLift Servers 游戏服务器的帐户。
- AWS 用于管理非 Amazon GameLift Servers 资源的帐户。

如果您对 Amazon GameLift Servers 和非 Amazon GameLift Servers 资源使用同一账户，则仅需该账户的 ID 和凭证。

2. 获取您的非 Amazon GameLift Servers 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区域中创建新的 VPC。如果您的非 Amazon GameLift Servers 资源在与您使用的 AWS 账户或 user/user 组不同的账户或组下管理 Amazon GameLift Servers，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，您可以通过查看自己的，在 Amazon VPC 控制台中找到 VPC ID VPCs。

3. 使用非 Amazon GameLift Servers 资源授权 VPC 对等连接。

当 Amazon GameLift Servers 创建新队组和对应的 VPC 时，它还会发送请求，以便与您的非 Amazon GameLift Servers 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。

使用管理非 Amazon GameLift Servers 资源的账户凭证，调用服务 API [CreateVpcPeeringAuthorization\(\)](#) 或使用 AWS CLI 命令 `create-vpc-peering-authorization`。确定以下信息：

- 对等 VPC ID：包含您的非 Amazon GameLift Servers 资源的 VPC 的 ID。
- Amazon GameLift Servers AWS 账户 ID — 用于管理 Amazon GameLift Servers 车队的账户 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. 按照说明[使用 AWS CLI 创建新队列](#)。包括以下其他参数：

- `peer-vpc-aws-account-id` — 您用于使用非Amazon GameLift Servers资源管理 VPC 的账户的 ID。
- `peer-vpc-id` — 使用您的非Amazon GameLift Servers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`。您可以通过调用[describe-vpc-peering-connections](#)用来跟踪对等互连请求的成功或失败。

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 Servers AWS 账户 ID 和 VPC ID 的组合进行唯一标识。

```
$ 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 Servers 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 Servers VPC 的 VPC 授权状态。它可能不存在或可能已过期。
 - 检查 VPCs 您要对等的两个区域的区域。如果它们不在同一个区域中，则无法建立对等连接。
- 您的两 VPCs 个 CIDR 块（请参阅[无效的 VPC 对等连接配置](#)）重叠。分配给对等设备的 IPv4 CIDR 块 VPCs 不能重叠。系统会自动为 Amazon GameLift Servers 实例集的 VPC 分配 CIDR 数据块且您

无法更改此数据块，因此，您需要为非 Amazon GameLift Servers 资源更改 VPC 的 CIDR 数据块。要解决此问题，请执行以下操作：

- 通过调用 `DescribeVpcPeeringConnections()` 查找 Amazon GameLift Servers 队组的此 CIDR 块。
- 转到 Amazon VPC 控制台，查找非 Amazon GameLift Servers 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。

玩家会话和对战自定义

借助玩家会话和对战自定义设置，您可以开发复杂的玩家管理工作流，包括精细的对战系统，助力打造公平且引人入胜的多人游戏体验。

生成玩家 ID

Amazon GameLift Servers 使用玩家会话来表示已连接到游戏会话的玩家。当玩家通过与 Amazon GameLift Servers 集成的游戏客户端连接到游戏会话时，Amazon GameLift Servers 会创建一个玩家会话。当玩家离开游戏时，玩家会话随之结束。Amazon GameLift Servers 不会重复使用玩家会话。

Important

使用 FlexMatch 对战时，如果您创建的新对战请求包含已存在于现有有效对战请求中的玩家 ID，则现有请求将被自动取消。但是，系统不会为被取消的请求发送 `MatchmakingCancelled` 事件。要监控现有对战请求的状态，请使用 [DescribeMatchmaking](#) 以不频繁的间隔（30-60 秒）轮询请求状态。被取消的请求将显示状态为 `CANCELLED`，原因标记为 `Cancelled due to duplicate player`。

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[Amazon GameLift Servers 控制台中的游戏和玩家会话](#)。

为 Amazon GameLift Servers 添加 FlexMatch 对战

使用 Amazon GameLift Servers FlexMatch 为 Amazon GameLift Servers 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 Amazon GameLift Servers Realtime 搭配使用。

FlexMatch 可将对战服务与自定义规则引擎搭配使用。您可以根据玩家属性以及适合您游戏的模式来设计玩家匹配方式。FlexMatch 则管理实际的匹配流程，包括评估正在寻找游戏的玩家、组建单队或多队对战组合，以及启动游戏会话以托管这些对战。

要使用完整 FlexMatch 服务，您必须将托管资源设置为队列。Amazon GameLift Servers 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 Servers 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 Servers FlexMatch 开发人员指南](#) 主题：

- [Amazon GameLift Servers FlexMatch 的工作原理](#)
- [FlexMatch 集成步骤](#)

游戏会话放置自定义

借助游戏会话放置自定义设置，您可以对放置系统进行微调，从而为玩家群体提供尽可能最佳的游戏体验。您还可以根据玩家的兼容性和偏好来优化放置，同时兼顾运营层面的考量因素，例如成本效益、地理分布以及从服务中断恢复的能力。

自定义游戏会话队列

本主题介绍如何自定义游戏会话队列，以实现最优的游戏会话放置决策。有关游戏会话队列及其工作原理的更多信息，请参阅[配置游戏会话放置](#)。

这些 Amazon GameLift Servers 功能需要队列支持：

- [使用 FlexMatch 对战](#)
- [为竞价型实例构建队列](#)

主题

- [定义队列的范围](#)
- [建立多位置队列](#)
- [评估队列指标](#)

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。

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 Servers 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 Servers 队列指标](#)。

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。[借助竞价型实例集降低游戏托管成本](#)

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。每个舰队都有以下 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 Servers 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 Servers 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

Game session placement priority

The values that GameLift uses to prioritize game session placement. The default order is latency, cost, destination, and location.

⋮

Latency

Prioritize locations with the lowest average player latency.

⋮

Cost

Prioritize destinations with the lowest current hosting cost.

⋮

Destination

Prioritize based on the defined destination order.

⋮

Location

Prioritize based on the defined location order.

▼ Location order

An ordered list of locations that the queue can use for game session placement.

Location

ca-central-1

▼

Remove

us-east-1

▼

Remove

us-east-2

▼

Remove

us-west-2

▼

Remove

Add locations

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 Servers 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 Servers 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放置位置的频率。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。此外，还有错误指标可识别导致 Amazon GameLift Servers 无法找到理想放置位置的常见原因。有关指标的更多信息，请参阅 [Amazon GameLift Servers使用 Amazon 进行监控 CloudWatch](#)。
- 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。对于采用 FleetIQ 算法的队列，此数据提供对玩家活动出现位置的有用见解。

评估 FleetIQ 性能的指标时，请考虑以下提示：

- 要跟踪队列寻找理想放置位置的速度，请结合使用 PlacementsSucceeded 指标与 FleetIQ 指标，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。

优先考虑游戏会话放置

Amazon GameLift Servers 采用算法来确定如何对队列的目标进行优先级排序，并决定新游戏会话的放置位置。该算法基于一组有序的标准。您可以使用默认的优先顺序，也可以自定义顺序。您可以随时编辑队列的优先级顺序。

默认优先顺序

1. 延迟 – 如果游戏会话放置请求包含玩家特定位置的延迟数据，则 Amazon GameLift Servers 会计算每个位置的玩家平均延迟，并尝试将游戏会话放置在平均值最低的实例集位置。
2. 成本 – 如果请求未包含延迟数据，或者多个实例集的延迟时间相等，则 Amazon GameLift Servers 会评估每个实例集的托管成本。实例集托管成本因实例集类型（竞价型或按需型）、实例类型及位置而异。
3. 目标 – 如果多个实例集具有相同的延迟和成本，则 Amazon GameLift Servers 会根据队列配置中列出的目标顺序对实例集进行优先级排序。

4. 位置 – 对于拥有多个位置实例集的队列，如果所有其他标准都相同，则 Amazon GameLift Servers 会根据字母顺序对实例集的位置进行优先级排序。

自定义队列对游戏会话放置的优先级排序方式

您可以选择自定义队列对放置标准的优先级排序方式。该队列将对接收到的所有游戏会话放置请求应用自定义优先级排序。

Note

如果您创建自定义优先级配置时未包含全部四个标准，则 Amazon GameLift Servers 会自动按默认顺序追加所有缺失的标准。

自定义队列的优先级配置

使用 [Amazon GameLift Servers 控制台](#) 或 AWS Command Line Interface (AWS CLI) 创建自定义优先级配置。

Console

在 [Amazon GameLift Servers 控制台](#) 中，您可以在创建新队列或更新现有队列时自定义队列的优先级。选择要工作的 AWS 区域。

打开控制台左侧导航栏并选择队列。在“队列”页面中，选择现有队列并选编辑。

1. 转到游戏会话放置优先级部分。拖放每个优先级标准以创建所需顺序。
2. 转到位置顺序部分。添加您想要确定其优先级的任何位置。当队列中有多个位置的实例集时，此列表非常有用。您至少必须指定一个位置。在此处指定的位置将优先处理，随后处理队列目标中的所有其他位置。
3. 选择保存更改。

AWS CLI

使用带有 `--priority-configuration` 选项的 [update-game-session-queue](#) 命令来自定义队列的优先级顺序。Amazon GameLift Servers 更新当前默认 AWS 区域中的队列，或者您可以添加 `--region` 标签以指定其他 AWS 区域。

以下示例请求将为指定队列添加或更新优先级配置

```
aws gamelift update-game-session-queue \  
  --name "example-queue-with-priority" \  
  --priority-configuration \  
    PriorityOrder="COST", 'LOCATION', "DESTINATION", LocationOrder="us-east-1", "us-  
east-2", "ca-central-1", "us-west-2" \  
  \
```

按玩家延迟确定放置优先级

如果您想为玩家提供最佳的玩家体验并确保将延迟降至最低，请在设置游戏会话放置系统时采取以下步骤：

- 在选择放置游戏会话的位置时，将您的队列设置为优先考虑延迟。默认情况下，延迟位于优先级列表的顶端。您还可以自定义队列的优先级配置，并选择按优先级顺序排列延迟的位置。
- 为您的队列设置玩家延迟策略。延迟策略允许您对游戏会话放置中允许的延迟量设置硬性限制。如果 Amazon GameLift Servers 无法在不超过限制的情况下放置游戏会话，则放置请求将超时并失败。您可以设置单个延迟策略，也可以创建一系列随着时间的推移逐渐放宽延迟限制的策略。通过一系列策略，您可以指定极低的初始延迟限制，并且在短暂延迟后仍然可以容纳延迟较高的玩家。有关创建延迟策略的详细信息，请参阅[创建玩家延迟政策](#)。
- 在提出游戏会话放置请求时（参见 [StartGameSessionPlacement](#)），请包括每位玩家的延迟数据。玩家延迟数据包含游戏会话可能放置的每个可能位置的对应数值。例如，对于将游戏会话置于 us-east-2 和 ca-central-1 中的队列，延迟数据可能如下所示：

```
"PlayerLatencies": [  
  { "LatencyInMilliseconds": 100, "PlayerId": "player1", "RegionIdentifier": "us-  
east-2" },  
  { "LatencyInMilliseconds": 100, "PlayerId": "player1", "RegionIdentifier": "ca-  
central-1" },  
  { "LatencyInMilliseconds": 150, "PlayerId": "player2", "RegionIdentifier": "us-  
east-2" },  
  { "LatencyInMilliseconds": 150, "PlayerId": "player2", "RegionIdentifier": "ca-  
central-1" }  
]
```

要获得准确的延迟测量值，请使用 Amazon GameLift Servers 的 UDP ping 信标。这些端点使您能够测量玩家设备与每个潜在托管位置之间的实际 UDP 网络延迟，从而比使用 ICMP ping 更准确地做出放置决策。有关使用 UDP ping 信标测量延迟的更多信息，请参阅[UDP ping 信标](#)。

按位置确定放置优先级

您可以配置队列，使其根据地理位置优先级列表放置游戏会话。位置是队列决定新游戏会话放置位置的标准之一。默认情况下，位置的优先级排在延迟、成本和目标之后，位列第四。

对于游戏会话放置，目标和位置具有略微不同的含义：

- 目标是指特定的实例集，包含该实例集的所有托管资源（无论部署于何处）。按目标确定优先级时，Amazon GameLift Servers 可能会将会话放置到实例集内的任何位置。多位置托管式实例集和 Anywhere 实例集的托管资源可部署于一个或多个位置。
- 位置是指实例集托管资源部署的具体地理位置。一个队列可以有多个位置，其中可能包括 AWS 区域 Local Zones 或自定义位置（对于 Anywhere 队列）。单位置托管式实例集仅有一个位置，且该位置始终为 AWS 区域。多位置实例集有一个主区域，并可包含远程位置。Anywhere 实例集有一个或多个自定义位置。

按位置确定放置优先级时，Amazon GameLift Servers 会查找包含优先级位置的所有队列目标，然后在其中搜索可用的托管资源。如果存在多个包含优先级位置的目标，则 Amazon GameLift Servers 会转而采用下一个优先级标准（成本、延迟、目标）。

您可以通过多种方式影响队列位置的优先级排序

- 配置队列处理所有游戏会话放置请求的方式：
 - 向队列添加优先级配置。队列的优先级配置包括位置的有序列表。您可以指定一个或多个位置来确定其优先级。此列表不会排除任何位置，仅告知 Amazon GameLift Servers 会优先在哪些位置查找可用的托管资源。有序位置列表的一个常见用途是：当您希望将大部分流量导入到一个或多个特定的地理位置，同时将其他位置作为备份容量时。通过调用添加优先级配置 [UpdateGameSessionQueue](#)。
 - 向队列添加筛选条件配置。筛选条件配置是队列的允许列表。它指示 Amazon GameLift Servers 在寻找可用的托管资源时忽略列表之外的所有位置。筛选条件配置有两种常见用途。首先，对于具有多个位置的实例集，您可以使用筛选条件来排实例集内的某些位置。其次，在某个位置出现暂时性问题时，您可能希望暂时禁止在该位置上放置。由于您可以随时更新队列的筛选条件配置，因此可以根据需要轻松添加和移除位置。通过调用添加过滤器配置 [UpdateGameSessionQueue](#)。
- 使用特殊指令处理个别放置请求：
 - 在游戏会话放置请求中包含优先级覆盖列表。您可以根据任何 [StartGameSessionPlacement](#) 请求提供备用优先位置列表。此列表仅对本次请求生效，将覆盖队列中配置的位置优先级，且不会影响任何其他请求。此覆盖功能需满足以下要求：

- 仅当队列已设置优先级配置且将 LOCATION 设为第一个优先级时，才使用覆盖列表。
- 请勿在同一放置请求中包含玩家延迟数据。包含延迟数据会在对位置进行优先级排序时引发冲突，而 Amazon GameLift Servers 无法解决此类冲突。
- 设定当 Amazon GameLift Servers 在优先级覆盖列表中找不到可用的资源时的处理方式。您可以选择回退到队列的其他位置，也可以将放置范围限制在覆盖列表内。默认情况下，Amazon GameLift Servers 会回退以尝试放置在队列的其他位置。
- 根据需要更新队列的筛选条件配置，例如在覆盖列表中添加位置。覆盖列表不会使筛选条件列表失效。

创建玩家延迟政策

如果您的放置请求包含玩家延迟数据，则 Amazon GameLift Servers 会为所有玩家寻找平均延迟最低的位置中的游戏会话。基于玩家平均延迟放置游戏会话时，Amazon GameLift Servers 会避免将大多数玩家放置到高延迟的游戏中。但 Amazon GameLift Servers 仍会为延迟极高的玩家进行放置。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 Servers 在任意位置放置请求的游戏会话，以免请求中的玩家体验到的延迟超过设定最大值。玩家延迟策略还可以阻止 Amazon GameLift Servers 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（例如要求组内所有玩家保持相似的延迟），您可以使用 [Amazon GameLift Servers 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。

Create queue

Queue settings

Name

TestQueue

The name must be unique and have 1-128 characters. Valid characters: A-Z, a-z, 0-9, and - (hyphen).

Timeout

Specify how long GameLift tries to place a game session before stopping.

600

seconds

Must be 10-600 seconds.

We recommend setting player latency policies, unless you're using GameLift FlexMatch.

Player latency policies - optional

Add policies to help place players into games with lower latency. Use multiple policies to reduce latency requirements per policy so that each player eventually finds a match.



Period start	Period end	Max player latency	
0 Seconds	120 Seconds	50 Milliseconds	Remove
120 Seconds	240 Seconds	100 Milliseconds	Remove
240 Seconds	600 Seconds	200 Milliseconds	Remove
Add policy			

为竞价型实例构建队列

通过使用竞价型实例集，您可以大幅节省托管成本。有关竞价型实例集及其使用方法的更多详细信息，请参阅[按需型实例和竞价型实例](#)。

如果您的游戏托管解决方案包括竞价型实例集，则必须使用游戏会话放置队列。Amazon GameLift Servers 使用队列在多个游戏托管资源中进行搜索，并选择最佳资源来托管新游戏会话。对于竞价型实例集，队列对于最大限度地降低托管成本和避免竞价型实例可能出现的中断尤为重要。本主题可帮助您设置弹性队列，即使出现中断、减速和故障，该队列也可以继续为玩家托管游戏。您可以根据多个因素（包括托管成本），自定义队列如何确定可用托管资源的优先级。

您是否在使用 FlexMatch 进行对战？您可以将队列与竞价型实例集配合使用，以便进行对战游戏会话放置。

实施竞价型实例集任务

在创建或更新您的游戏托管解决方案以便使用竞价型实例集时，请完成以下任务。有关如何构建可优化竞价型实例可用性和弹性的队列的更多详细指南，请参阅[借助竞价型实例集降低游戏托管成本](#)。

1. 为您的游戏会话队列选择并创建一组实例集目标。

首先决定您想让队列放置游戏会话的位置。队列可以在多个实例集中搜索以找到最佳位置。每个实例集仅包含一种实例类型，但可覆盖多个地理位置。如果队列关联的实例集在位置和实例类型上均具备多样性，则更有可能实现成功的游戏会话放置。请参阅这些最佳实践，了解如何设计有效且具有弹性的竞价型实例优化队列。

2. 创建针对竞价型实例优化的游戏会话队列。

创建队列并针对您的竞价型实例集进行配置。请参阅[创建游戏会话队列](#)以帮助您创建和配置新队列。您可以使用 Amazon GameLift Servers 控制台或 AWS CLI 来创建或编辑队列。

- 添加步骤 1 中的实例集目标位置。
- 根据需要确定目标位置的优先顺序。默认情况下，Amazon GameLift Servers 会优先按成本排序，其次才考虑目标区域，因此仅当不同区域间的最低成本相同时，才会根据目标区域顺序进行分配。
- 如果您希望将游戏托管成本的优先级置于玩家延迟之上，请提供自定义放置优先级。请参阅[优先考虑游戏会话放置](#)。

3. 更新解决方案中的其他组件以使用新队列。

当您的解决方案使用针对竞价型实例优化的队列来启动新的游戏会话时，队列会自动避免将游戏会话置于中断可能性很高的实例集中。相反，它会在所有可行的实例集中搜索与您定义的优先级相匹配的资源，包括玩家延迟、托管成本和目的地顺序。

- 如果您未使用 FlexMatch：更新您的后端服务，以在游戏会话请求中指定新的竞价型实例优化队列。后端服务代表您的游戏客户端（使用 `StartGameSessionPlacement()`）向 Amazon

GameLift Servers 发出 API 请求，并且每个请求都必须指定队列名称。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](#)。

- 如果您使用了 FlexMatch：更新您的对战配置，将游戏会话请求发送到新的竞价型实例优化队列。当对战系统形成玩家匹配时，它会向指定队列发送游戏会话放置请求，以开始该对战的新游戏会话。只有 FlexMatch 模式设置为“托管”的对战配置才能指定放置队列。您可以使用 AWS CLI 或 Amazon GameLift Servers 控制台更新配对配置（参见[编辑配对配置](#)）。

4. 评估 竞价型实例集和队列的性能。

在 Amazon GameLift Servers 控制台中或通过 Amazon 查看 Amazon GameLift Servers 指标 CloudWatch 以查看绩效。有关 Amazon GameLift Servers 指标的更多信息，请参阅 [Amazon GameLift Servers 使用 Amazon 进行监控 CloudWatch](#)。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。回收实例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。

带有竞价型实例集的队列的最佳实操

在为竞价型实例创建实例集和队列时，请使用以下最佳实践。

- 扩大队列的地理覆盖范围。即使你的玩家聚集在一个单一的队伍中 AWS 区域，也要将相邻的位置添加到你的竞技队伍中。这种方法提高了队列在区域减速、故障和竞价型实例中断期间保持容量的能力。多位置实例集适用于竞价型实例和按需型实例。
- 使队列的实例类型覆盖范围多样化。Amazon GameLift Servers 会根据实例类型评估竞价型实例可行性，因此拥有具有各种实例类型的竞价型实例集可以减少多个竞价型实例集同时不可用的可能性。至少包括两个竞价型实例集，每个位置的实例类型各不相同。

Note

定价基于您使用的实例，而不是实例集的数量。运行五个队列，每个队列有 10 个实例，与运行 50 个成本相似的实例的实例集相同。定价因实例类型、大小和位置而异。

对竞价型实例类型进行分组的提示：

- 使用同一系列中的实例类型，例如 `m6g.medium`、`m6g.large` 和 `m6g.xlarge`。实例类型越大，成本越高，但也可以同时托管更多的游戏会话。
- 选择广泛可用的实例类型。通常，旧一代系列（例如 C5、M5 和 R5）和常见规模（例如 `.large`、`.xlarge` 和 `.2xlarge`）具有更好的可用性。
- 在 Amazon GameLift Servers 控制台中查看 30-90 天的定价历史记录。寻找可用性模式一致的实例类型。
- 使用 Amazon GameLift Servers 控制台（实例集创建工具）来探索实例类型的位置覆盖范围。
- 添加按需型实例集以获得备份容量。每当竞价型实例集不可用时，游戏托管可以切换到按需型实例集。在每个位置至少放置一个按需型实例集以保持低玩家延迟。为您的备用按需型实例集添加自动伸缩功能，以便在不需要按需型实例集时缩减其规模。
- 为所有实例集目的地分配别名。为队列的每个目的地创建别名。别名可以让您在需要更换实例集时更轻松、更高效地操作。
- 应用队列优先级策略。您可以自定义队列如何确定游戏会话放置位置的优先顺序（有关更多详细信息，请参阅[优先考虑游戏会话放置](#)）。对于针对竞价型实例优化的队列，按成本划分优先级可确保尽可能使用低成本的竞价型实例集。

您还可以通过指定目的地顺序来确定某些实例集的优先级。例如，一些用户指定一组主实例集供常规使用，并将一组辅助实例集指定为备用。在此情况下，将实例集的目标顺序设置为首先列出主实例集。然后，配置队列的优先顺序，先是目的地，然后是成本。

托管资源自定义

本节将介绍用于配置和管理 Amazon GameLift Servers 基础设施的高级选项，可协助您满足特定的性能、成本和运营要求。具体而言，本节主题将阐述如何自定义 Amazon GameLift Servers 托管式托管资源，以使其充分契合您的游戏及玩家需求。

需要考虑的决策包括：

- 在何处为玩家部署托管资源？游戏延迟是选择实例集地理位置时的主要考量因素，但不同位置还存在其他差异因素，包括资源类型的可用性和成本。
- 哪些 EC2 实例类型最能支持你的游戏？从可用的实例类型中进行选择，以使用计算架构、内存、存储和网络容量的最佳组合。

- 您需要多大的实例类型？根据游戏服务器软件的资源需求（内存和 CPU）和其他因素选择实例类型大小。
- 您的实例集应使用按需型实例还是竞价型实例？考虑您能否利用较低的竞价型实例定价，以及 Amazon GameLift Servers 能否充分降低竞价型实例中断对游戏会话的影响。
- 您希望游戏服务器软件如何在每个实例集实例上运行？运行时配置会告知 Amazon GameLift Servers 要运行的服务器软件及其运行方式。
- 对于容器实例集，默认配置是否适用于您的游戏？Amazon GameLift Servers 会为您完成容器实例集配置方面的大量优化工作，但您仍可自定义大多数配置设置。

为托管式实例集选择计算资源

对于 Amazon GameLift Servers 托管式托管（包括托管式 EC2 和托管容器），该服务会将您的游戏服务器部署到 AWS Cloud 中的计算资源实例集。创建托管式实例集时，您需要配置最适合您游戏的托管资源。本主题讨论了选择和配置游戏托管式实例集时的关键决策点。

Note

如果您要同时使用 Anywhere 和 Amazon GameLift Servers 托管式实例集构建混合解决方案，请使用这些主题来设计托管式实例集，以补充您自己的自管理资源。请参阅 [为 Amazon GameLift Servers 部署托管实例集](#)。

主题

- [地理位置](#)
- [操作系统](#)
- [实例类型](#)
- [按需型实例和竞价型实例](#)
- [服务配额](#)

地理位置

考虑一下您计划部署游戏服务器的位置。通常，为提供最佳玩家体验，应将游戏服务器部署在尽可能靠近玩家的位置。对于托管 Amazon GameLift Servers 主机，您可以选择将游戏服务器放置在任何支持的区域 AWS 区域 和 Local Zones 中。如果您正在构建混合解决方案，请考虑托管式实例集部署如何能够补充您自行管理的 Amazon GameLift Servers Anywhere 实例集的部署位置。

对于大多数开发和测试场景，部署到单一位置即可满足需求。当您为游戏发布及后续运营做准备时，选择跨多个地理位置部署则具有多重优势，包括为广泛的玩家群体提供支持，以及提高游戏托管的整体弹性和可靠性。此外，多位置部署还能加快游戏会话放置速度，并在针对延迟和成本优化放置时提供更多选择，从而提升玩家体验。

有关 Amazon GameLift Servers 支持的位置列表以及关于所有实例集类型的位置的更多信息，请参阅 [Amazon GameLift Servers 服务位置](#)。

多位置实例集

单个托管式实例集可以将资源部署到多个位置。您可以针对多位置实例集中的每个独立位置，手动设置容量。

使用多位置实例集的优势：

- 简化实例集部署和管理：您只需提供游戏服务器软件和实例集配置，然后 Amazon GameLift Servers 会将其部署到多个位置的实例集实例（一次构建，随处部署）。在生产实例集中，您可以查看和管理实例集中的所有位置，而不必管理位于不同区域的多个实例集。
- 本地区域可用性-如果要使用本地区域，则必须创建多地点舰队，将 AWS 区域 总部位置和本地区域作为远程位置。Local Zones 是它的扩展 AWS 区域，可以为有需要的区域和客户提供更低的延迟。您可以将 Local Zones 添加到任何多位置实例集中；无需包括 Local Zones 的父 AWS 区域。
- 与游戏会话队列兼容：您可以构建包含一个或多个多位置实例集的游戏会话放置队列。此方法使队列在优先选择和确定用于托管新游戏会话的位置时更具灵活性。
- 高效的资源利用率：开启自动扩缩功能后，Amazon GameLift Servers 可以更好地优化实例集中所有位置的容量扩展。

使用多位置实例集的提示：

- 查看每个 AWS 区域 或车队的位置数量的配额。请参阅 [Amazon GameLift Servers 服务限额](#)。
- 并非所有实例类型都在所有位置可用。实例类型选择可能有限，具体取决于您选择的位置。Amazon GameLift Servers 控制台提供了有用的工具，可帮助您在位置和实例类型之间找到适当的平衡。
- 考虑使用 [UDP ping 信标](#)来收集所有实例集位置的玩家延迟数据。Amazon GameLift Servers 可以使用这些数据来优化游戏会话以实现低延迟，并防止玩家加入延迟过高的会话。这些特殊端点接受 UDP 消息，而不是传统的 ICMP ping 消息，同时提供准确的延迟测量，有助于您选择最佳实例集位置。

操作系统

托管式实例集中的所有实例均使用亚马逊机器映像 (AMI) 进行部署，该映像可为游戏服务器软件提供完整的运行时环境。对于托管式 EC2 实例集，您可以在将游戏服务器生成包上传到 Amazon GameLift Servers 时指定生成包的操作系统。对于托管式容器实例集，您可以在容器组定义中指定操作系统。有关最新 AMI 版本的更多信息，请参阅[Amazon GameLift Servers AMI 版本](#)。

AMI 版本会定期更新。创建新实例集时，Amazon GameLift Servers 会分配您为游戏生成包选择的 AMI 的最新可用版本。部署在该实例集中的所有实例都使用相同的版本。要使您的 AMI 版本与最新的安全和软件更新保持同步，您需要定期更换实例集。作为最佳实践，我们建议每 30 天更换一次托管式实例集，以便为游戏服务器维护运行时环境。有关指南，请参阅[Amazon GameLift Servers 的安全最佳实践](#)。

实例类型

托管式实例集的实例类型决定了为所有实例集部署的硬件类型，并且实例类型通常有多种规格可供选择。所有 Amazon GameLift Servers 托管式实例集均使用 Amazon EC2 实例，并支持多种实例类型，这些类型提供了计算能力、内存、存储和网络性能的不同组合。实例类型的可用性因您选择的位置而异。

Amazon GameLift Servers 控制台提供了有用的工具，可帮助您找到适合游戏生成包和部署位置的正确实例类型。对于托管式容器实例集，控制台还提供有关游戏的 CPU 功率和内存要求的指导。

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 实例需要一个针对 Linux AMI 的服务器生成包。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。
- 您的实例类型的大小。除了满足游戏服务器软件可执行文件的要求外，更大的实例类型还可以在每個实例上运行多个游戏服务器进程 and/or 容器。需要考虑的因素包括成本（运行少量大型实例与运行多个小型实例相比，哪种方案更经济）。还要考虑在实例集扩展事件期间，或关闭运行状况不佳的实例时，实例的增减可能会对游戏会话容量产生何种影响。如果每个实例同时运行多个游戏服务器进程，则添加或移除实例可能会严重影响游戏托管容量。

有关实例类型的更多信息，请参阅 [Amazon EC2 实例类型](#)。

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，意味着您只需为使用这些实例的时间量付费。无长期承诺。

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 只要竞价型实例需要恢复容量，就可以在两分钟通知的情况下回收竞价型实例，并且在回收的实例上正在运行的游戏会话会被中断。

Amazon GameLift Servers 提供了多种工具，可帮助降低竞价型实例中断对您游戏会话造成影响的可能性。竞价可行性算法会跟踪实例类型的历史数据，以预测中断风险何时达到临界点，并避免在该类型的竞价型实例上放置新的游戏会话。如果确实发生了中断，您的游戏服务器可以使用通知正常结束玩家的游戏会话。

使用竞价型实例集托管游戏时，必须通过队列来放置游戏会话。队列可以根据竞价型实例集的可行性、成本和其他因素来确定游戏会话放置的优先级。有关如何利用竞价型实例托管游戏服务器的更多信息，请参阅以下主题：

- [借助竞价型实例集降低游戏托管成本](#)
- [为竞价型实例构建队列](#)

服务配额

您可以使用以下工具查看 Amazon GameLift Servers 的默认服务配额和 AWS 账户 的当前配额状态：

- 有关 Amazon GameLift Servers 的通用服务配额信息，请参阅《AWS 一般参考》中的 [Amazon GameLift Servers 端点和配额](#)。
- 要查看账户每个位置的可用实例类型列表，请打开 Amazon GameLift Servers 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。

在准备发布游戏时，请在 [Amazon GameLift Servers 控制台](#) 中填写发布问卷。Amazon GameLift Servers 团队使用发布问卷来确定您的游戏的正确配额和限制。

自定义 Amazon GameLift Servers 容器实例集

本节主题介绍了 Amazon GameLift Servers 托管式容器的一些可选功能。您可以选择使用其中任意一项或全部功能。

主题

- [设置资源限制](#)
- [配置 NVMe 云端硬盘访问权限](#)
- [指定必备容器](#)
- [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](#)
- [配置容器实例集](#)

设置资源限制

对于每个容器组，您可以确定容器组运行其软件所需的内存和计算能力。Amazon GameLift Servers 依靠这些信息来管理整个容器组的资源。它还使用这些信息来计算实例集映像可以容纳的游戏服务器容器组数量。此外，您也可以为各个容器设置限制。

您可以为容器组设置内存和计算能力的最大限制。默认情况下，这些资源由组内的所有容器共享。您可以通过为单个容器设置限制来进一步自定义资源管理。

为单个容器设置可选限制

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组内资源。如果未设置特定于容器的限制，则组内所有容器将共享组资源。这种共享方式能够更灵活地按需分配资源，但也可能导致进程间资源竞争，进而引发容器故障。

可为任何容器设置以下任一 `ContainerDefinition` 属性。

- `MemoryHardLimitMebibytes`：为容器设置最大内存限制。如果容器超过此限制，则会重启。
- `Vcpu limit`：保留最低数量的 vCPU 资源供容器专用。容器始终拥有该预留资源。如果有其他资源可用，其使用量可随时超过此最低限额。（1024 个 CPU 单位相当于 1 个 vCPU。）

为容器组设置总资源限制

如果您为单个容器设置了限制，则可能需要修改容器组所需的内存和 vCPU 资源。目标是分配足够的资源来优化游戏服务器性能。Amazon GameLift Servers 使用这些限制来计算如何在实例集实例上打包游戏服务器容器组。此外，在为容器实例集选择实例类型时，也需要参考这些限制。

计算容器组所需的总内存和 vCPU。请考虑以下事项：

- 容器组内所有容器运行的全部进程有哪些？将这些过程所需的资源累加起来。注意任何特定于容器的限制。
- 您计划在每个容器组中运行多少个并发游戏服务器进程？您可以在游戏服务器容器映像中确定这一数量。

根据您对容器组需求的估计，设置以下 `ContainerGroupDefinition` 属性：

- `TotalMemoryLimitMebibytes`：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。如果已设置单个容器限制，则总内存限制必须等于或大于容器特定的最高内存限制。
- `TotalVcpuLimit`：为容器组设置最大 vCPU 限制。组中的所有容器共享分配的 CPU 资源。如果已设置单个容器限制，则总 CPU 限制必须等于或大于所有容器特定 CPU 限制的总和。作为最佳实践，建议将此值设置为容器 CPU 限制之和的两倍。

示例方案

假设我们要定义一个包含以下三个容器的游戏服务器容器组：

- 容器 A 为游戏服务器容器。一台游戏服务器的资源需求预估为 512 MiB 内存和 1024 个 CPU 单位。我们计划让容器运行 1 个服务器进程。由于此容器运行最关键的软件，我们没有设置内存限制或 vCPU 预留限制。
- 容器 B 是一个支持容器，其资源需求估计为 1024 MiB 内存和 1536 个 CPU 单位。我们将内存限制设置为 2048 MiB，CPU 预留限制为 1024 个 CPU 单位。
- 容器 C 是另一个支持容器。我们将硬性内存限制设置为 512 MiB，将 CPU 预留限制设置为 512 个 CPU 单位。

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。此值超过了最高内存限制（1024 MiB）。
- CPU 总限制：13312 个 CPU 单位。此值超过了 CPU 限制的总和（1024+512 CPU）。

配置 NVMe 云端硬盘访问权限

在 d 型实例上，NVMe 驱动器会在主机启动期间自动装载到 `/data` 目录中。要允许容器访问 SSD 存储，请设置以下 `ContainerGroupDefinition` 属性 `MountPoints`：

- `InstancePath`— 设置为 `/data` 以引用主机实例上自动装载的 NVMe 驱动器。
- `AccessLevel`— 根据容器的需求选择适当的访问级别（例如 `READ_ONLY` 或 `READ_WRITE`）。
- `ContainerPath`—（可选）指定将实例路径挂载到容器内的路径。如果未指定，则默认为实例路径。

有关挂载点的更多信息，请参阅 Amazon GameLift 服务器 API 参考 [ContainerMountPoint](#) 中的。

指定必备容器

对于每个实例容器组，需将每个容器指定为必备容器或非必备容器。每个实例容器组必须具有至少一个必备支持容器。必备容器负责容器组的关键工作，需始终保持运行状态。若其发生故障，整个容器组将重启。

将每个容器的 `ContainerDefinition` 属性 `Essential` 设置为 `true` 或 `false`。

配置网络连接

您可以自定义网络访问权限以允许外部流量连接到容器实例集中的任何容器。例如，您必须与运行游戏服务器进程的容器建立网络连接，以便游戏客户端能够接入并参与游戏。游戏客户端使用端口和 IP 地址连接到游戏服务器。

在容器实例集中，客户端与服务器之间并非直接连接。在内部，容器中的进程会监听容器端口。在外部，传入流量使用连接端口连接到实例集实例。Amazon GameLift Servers 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 GameLift Servers 为您的网络连接提供了额外一层控制。每个容器实例集都有入站权限设置，允许您控制对每个面向外部的连接端口的访问。例如，您可以移除所有连接端口的权限，以关闭对实例集容器的所有访问权限。

您可以更新实例集的入站权限、连接端口和容器端口。

Warning

如果您提供自定义 `InstanceConnectionPortRange` 或 `InstanceInboundPermissions`，则 Amazon GameLift Servers 将不再为您的队列管理任一值。您必须同时设置这两个字段，以避免出现未定义行为。

设置容器端口范围

容器端口范围需作为每个容器定义的一部分进行配置。这是容器组定义的必填参数。您需要配置足够的端口，以容纳所有需要外部访问的并发运行的进程。有些容器不需要任何端口。

运行游戏服务器的游戏服务器容器需要一个端口，用于每个并发运行的游戏服务器进程。游戏服务器进程会监听分配的端口并将其报告给 Amazon GameLift Servers。

设置连接端口范围

为容器实例集配置一组连接端口。连接端口提供对运行容器的实例集实例的外部访问权限。Amazon GameLift Servers 会根据需要分配连接端口，并将其映射到容器端口。

默认情况下，Amazon GameLift Servers 会计算所有容器组所需的端口数，并设置相应的端口范围以满足需求。我们强烈建议您使用 Amazon GameLift Servers 的计算值，该值会在您部署容器组定义更新时自动更新。如果您确实需要自定义连接端口范围，请使用以下指南。

创建容器队列时，请定义连接端口范围（请参阅 [ContainerFleet:InstanceConnectionPortRange](#)）。确保该范围有足够的端口，可以映射到在实例集中两个容器组内所有容器中定义的每个容器端口。要计算所需的最小连接端口数，请使用以下公式：

```
[Total number of container ports defined for containers in the game  
server container group] * [Number of game server container groups per  
instance] + [Total number of container ports defined for containers in  
the per-instance container group]
```

最佳实践是，将最小连接端口数量翻倍。

Note

连接端口的数量可能会限制每个实例的游戏服务器容器组的数量。如果实例集的连接端口仅足以满足每个实例部署一个游戏服务器容器组的需求，则即使实例的计算资源足以承载多个游戏服务器容器组，Amazon GameLift Servers 仍只会部署一个游戏服务器容器组。

设置入站权限

入站权限通过指定允许传入流量访问的连接端口，来控制对容器实例集的外部访问。您可以使用此设置，根据需要开启和关闭实例集的网络访问权限。

默认情况下，Amazon GameLift Servers 会计算所有容器组所需的端口数，并设置相应的端口范围以满足需求。我们强烈建议您使用 Amazon GameLift Servers 的计算值，该值会在您部署容器组定义更新时自动更新。如果您确实需要自定义连接端口范围，请使用以下指南。

创建容器队列时，请定义一组入站权限（请参阅 [ContainerFleet:InstanceInboundPermissions](#)）。入站许可端口应与实例集的连接端口范围相匹配。

Note

由于容器端口是从中随机选择的 InstanceConnectionPortRange，因此为了保证可以建立会话连接，因此中的所有端口都 InstanceConnectionPortRange 应被中的端口覆盖 InstanceInboundPermissions

示例方案

此示例说明了如何设置所有三个网络连接属性。

- 我们实例集的游戏服务器容器组有 1 个容器，其中运行 1 个游戏服务器进程。

在游戏服务器容器组定义中，我们按如下方式设置该容器的 PortConfiguration 参数：

```
"PortConfiguration": {
  "ContainerPortRanges": [ { "FromPort": 10, "ToPort": 20, "Protocol": "TCP"} ]
}
```

- 我们的实例集还有一个每个实例容器组，其中包含 1 个容器。它有 1 个需要网络访问的进程。在每个实例的容器定义中，我们按如下方式设置此容器的 PortConfiguration 参数：

```
"PortConfiguration": {
  "ContainerPortRanges": [ { "FromPort": 25, "ToPort": 25, "Protocol": "TCP"} ] }
```

- 我们的实例集为每个实例集实例配置 20 个游戏服务器容器组。有了这些信息，我们可以使用以下公式来计算需要的连接端口数：
 - 最少：21 个端口 [1 个游戏服务器容器端口 * 每个实例 20 个游戏服务器容器组 + 1 个每个实例容器端口]
 - 最佳实践：42 个端口 [最少端口 * 2]

在创建容器实例集时，我们按如下方式设置 InstanceConnectionPortRange 参数：


```
"InstanceConnectionPortRange": { "FromPort": 1010, "ToPort": 1071 }
```

- 我们希望允许访问所有可用的连接端口。在创建容器实例集时，我们按如下方式设置 `InstanceInboundPermissions` 参数：

```
"InstanceInboundPermissions": [  
  { "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。如果容器标记为必备容器，它会提示整个容器组重新启动。

所有游戏服务器容器都自动被视为是必备容器。可以将支持容器指定为必备容器，但它们需要有报告运行状况的机制。您也可以为非必备支持容器设置运行状况检查。

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，可以在 Docker 容器映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，它将覆盖容器映像中的所有设置。

为容器运行状况检查设置以下 `SupportContainerDefinition` 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运行状况。该命令的退出值必须为 1（不正常）或 0（正常）。
- `StartPeriod`：指定在开始统计运行状况检查失败次数前的初始延迟时间。此延迟为容器提供启动内部进程的时间。
- `Interval`：决定执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`：在判定为失败前，运行状况检查命令应重试多少次？

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容器的状态，影响依赖容器的启动或关闭时机。

依赖关系的一个关键使用案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。

配置容器实例集

在创建容器实例集时，请考虑以下决策点。这些要点大多取决于容器架构和配置。

确定要部署实例集的位置

通常，您应将实例集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舰队部署到任何 AWS 区域 Amazon GameLift Servers 支持的集装箱舰队。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。对于多位置实例集，您可以单独调整每个实例集位置的容量。有关支持的实例集位置的更多信息，请参阅 [Amazon GameLift Servers 服务位置](#)。

建议使用 [UDP ping 信标](#) 来收集不同地理位置的网络延迟数据，以预测玩家设备和潜在实例集位置之间的延迟。这些特殊端点接受 UDP 消息，而不是传统的 ICMP ping 消息，同时提供准确的延迟测量，有助于您选择最佳实例集位置。

为实例集选择实例类型和规格

Amazon GameLift Servers 支持多种 Amazon EC2 实例类型，所有这些类型都可用于容器实例集。实例类型的可用性和价格因位置而异。您可以在 Amazon GameLift Servers 控制台（在“资源”、“实例”和“服务配额”下）中查看按位置筛选的支持实例类型列表。

选择实例类型时，首先要考虑实例系列。不同实例系列提供不同的 CPU、内存、存储和网络功能组合。了解有关 [EC2 实例系列](#) 的更多信息。每个实例系列下均提供多种实例规格供您选择。选择实例规格时，需考虑以下问题：

- 支持您的工作负载所需的最小实例规格是什么？利用该信息排除所有规格过小的实例类型。
- 哪些实例规格适合您的容器架构？理想情况下，应选择能够容纳多个游戏服务器容器组副本且资源浪费最少的规格。
- 哪种扩缩粒度适合您的游戏？扩缩实例集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。需考虑每次添加或移除实例时希望调整的容量大小。如果玩家需求每分钟波动有数千之多，则使用可承载数百或数千个游戏会话的超大型实例可能更为合理；相反，若需要更精细的扩缩控制，则可选择小型实例类型。
- 是否可通过规格选择节省成本？部分实例类型的价格可能因地区可用性不同而存在差异，您可根据此寻找成本优化空间。

设置其他可选实例集设置

在配置容器实例集时，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[将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。
- 保护存在活跃玩家的游戏会话，防止在缩减事件期间过早终止。
- 限制单个玩家在有限的时间范围内可在实例集上创建的游戏会话数。

借助竞价型实例集降低游戏托管成本

使用 Amazon GameLift Servers 托管式托管服务托管多人游戏服务器时，竞价型实例可作为按需型实例的高性价比替代方案。竞价定价模式提供与按需型定价模式相同的硬件配置和性能表现，但可节省大量成本（高达 70-90%）。但是，它们有一个限制：当 AWS 需要恢复容量时，它可以通过两分钟的中断通知回收这些实例。

Amazon GameLift Servers 降低了游戏服务器托管中断的风险。Amazon GameLift Servers 会预测各类竞价型实例的中断概率，避免在高风险实例上部署游戏会话。如果发生罕见的中断情况，您可借助中断通知为玩家平稳结束当前游戏会话。

Amazon GameLift Servers 如何与竞价型实例集协同工作

为游戏托管设置竞价型实例集后，Amazon GameLift Servers 会持续评估竞价型实例集中实例类型和位置，以确定游戏托管的可行性。

- 竞价型实例可行性算法会按位置分析竞价型实例类型的近期可用性模式和历史中断率。
- 根据此分析，Amazon GameLift Servers 会识别出游戏会话中断风险不可接受的竞价型实例类型和位置。它将执行以下操作：
 - 将该实例类型和位置的组合标记为暂时不可行。
 - 新增游戏会话时不再考虑这些不可行的竞价型实例集位置，确保游戏会话仅部署在中断风险低、托管稳定性高的竞价型实例集位置。
 - 即使 AWS 未主动回收资源，也会逐步下线该竞价型实例集位置的既有实例，避免为无法用于游戏托管的实例付费。如果开启游戏会话保护，实例仅在当前活跃游戏会话结束后才会关闭。
- Amazon GameLift Servers 会持续重新评估您的竞价型实例集中实例类型和位置，以确定游戏托管的可行性。当历史数据更新显示某类曾不可行的实例类型恢复可行时，您可重新纵向扩展该竞价型实例集，Amazon GameLift Servers 也会恢复在该类型实例上部署游戏会话。

设计注意事项

在设计使用竞价型实例集的解决方案时，需考量以下问题：

- 评估游戏会话时长：游戏会话的平均时长会影响竞价型实例在游戏中的表现情况。游戏会话时长较短时，快速周转机制可确保游戏会话始终运行在基于最新历史数据判定的可行实例类型上。而时长较长的游戏会话会持续运行在初始实例类型上，不再评估近期可行性数据，随着时间推移面临更高的中断风险。
- 评估实例类型可用性：并非所有实例集位置都提供全类型的竞价型实例。为竞价型实例集选择实例类型时，可使用 Amazon GameLift Servers 控制台实例集创建工具来帮助您在所需位置找到竞价型实例类型。使用此工具，您可以选择实例集位置，然后查看这些位置的实例类型可用性。
- 创建多位置竞价型实例集：您可以创建具有多个位置的竞价型实例集。单个多地点 Spot 队列将具有相同实例类型的实例部署到多个 AWS 区域 或 Local Zones。竞价型实例可行性算法根据实例类型和位置综合评估可行性。如果某个竞价型实例集位置被评估为不可行，则它不会影响实例集中的其他位置，这些位置仍可正常托管游戏会话。
- 创建具有竞价型实例集多样性的队列：如果您使用竞价型实例集托管游戏，则需设置游戏会话放置队列。队列会针对每个新游戏会话请求查找可用游戏托管资源并选择最优方案。针对竞价型实例集，建议队列支持跨多个实例集搜索（覆盖不同位置和实例类型），且至少包含一个按需型实例集作为备份容量。设计完善的多实例集队列可提供多样化放置选项，能有效抵御中断、性能下降及服务中断等风险。有关为竞价型实例设计队列的更多指导，请参阅[为竞价型实例构建队列](#)。
- 平移处理中断：设置游戏服务器，以最大限度地降低竞价型实例中断对玩家的影响。AWS 回收竞价型实例时，使用服务器 SDK 回调函数 `onProcessTerminate()` 将终止通知 Amazon GameLift Servers 传递给所有受影响的服务器进程。游戏需实现该回调函数，以平稳结束游戏会话。有关更多信息，请参阅 [回应服务器进程关闭通知](#)。

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。您还需配置游戏服务器以应对突发中断情况。

- 为备份实例集配置自动扩缩，保障竞价型实例中断期间的服务连续性。目标跟踪型自动扩缩可保持容量缓冲并根据需求自动扩展。借助自动扩缩，备份实例集（竞价型或按需型）会在收到的游戏会话请求量增加时，自动开始提升容量。

当竞价型实例集判定为不可行时，可借助自定义扩展机制，利用队列及实例集的可用指标触发备份实例集的快速横向扩展，以迅速补充丢失的容量。可通过 `FirstChoiceOutOfCapacity`、`FirstChoiceNotViable` 和

PercentAvailableGameSessions 等指标检测竞价型实例集的不可行状态。通过分析近期 PlacementsStarted 指标数据预估替换容量需求。待备份实例集扩展至满足即时需求后，即可切换回常规自动扩缩模式。

- 与 FlexMatch 集成：如果您的解决方案使用 FlexMatch 对战构建器，则对竞价型实例集无特殊要求。您可将对战构建器配置为使用含竞价型实例集的队列。Amazon GameLift Servers 会自动优先处理竞价型实例集和按需型实例集间的对战部署，包括新游戏会话部署及现有游戏会话中空玩家位置的回填操作。

在托管式 Amazon GameLift Servers 上优化游戏服务器运行时配置

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 Servers 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 Servers 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 Servers 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。以下介绍 Amazon GameLift Servers 如何采纳运行时配置更改：

1. 该实例向 Amazon GameLift Servers 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 Servers 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：

- [创建生成包](#)，其中包含您要部署到实例集的游戏服务器可执行文件，然后将生成包上传到 Amazon GameLift Servers。生成包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 Servers 的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 SDK 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，以确保 Amazon GameLift Servers 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 Servers 的终止请求。
- 确保 Amazon GameLift Servers 已关闭并重新启动运行状况不正常的服务器进程。您可以通过在游戏服务器代码中实现 `OnHealthCheck()` 回调函数来将该运行状况回报给 Amazon GameLift Servers。Amazon GameLift Servers 自动关闭连续三次均报告为运行状况不正常的服务器进程。如果不实现 `OnHealthCheck()`，Amazon GameLift Servers 会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 Servers 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 Servers 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。

使用 Amazon GameLift Servers 代理

Amazon GameLift Servers 代理负责监督 Amazon GameLift Servers 实例集上游戏服务器进程的运行。该代理被部署到实例集中的每个计算上，为计算提供自动化进程管理、托管管理和日志记录。要使用该代理，必须将游戏服务器生成包与 Amazon GameLift Servers 服务器 SDK 5.x 或更高版本集成。

Amazon GameLift Servers 代理可从外部提供给不是托管式 EC2 实例集的 Amazon GameLift Servers 实例集使用。（托管式 EC2 实例集会自动处理该代理的任务。）您可以选择在有或没有该代理的情况下运行 Amazon GameLift Servers 实例集，包括 Anywhere 实例集。如果没有该代理，您必须提供完成所需任务的替代解决方案。

部署到计算时，应在启动任何游戏服务器进程之前启动 Amazon GameLift Servers 代理。启动时，该代理将完成以下任务：

- 使用 [RegisterCompute](#) API 将该计算资源注册到 Amazon GameLift Servers Anywhere 实例集。
- 调用 [GetComputeAuthToken](#) API 以获取授权令牌，并将其存储起来供在计算上运行的服务器进程使用。
- 为计算设置 WebSocket URL 环境变量，并与 Amazon GameLift Servers 服务建立 WebSocket 连接。
- 从 Amazon GameLift Servers 服务请求最新版本的实例集的运行时配置。
- 根据运行时配置说明启动和停止服务器进程。

Amazon GameLift Servers 代理的源代码和生成包说明可在 [Amazon GameLift Servers 代理](#) GitHub 上找到。

关于该代理

Amazon GameLift Servers 代理专为您的实例集处理以下任务：

进程管理

- 启动运行时指令中定义的新服务器进程。该代理可能使用与其一起部署的自定义运行时配置。或者，您也可以提供 RuntimeConfiguration 作为实例集定义的一部分。此方法的优势在于，您可以随时修改实例集的运行时配置。该代理会定期从 Amazon GameLift Servers 服务请求更新的运行时配置。
- 监控服务器进程的激活情况，并在进程未及时激活时将其终止。
- 向 Amazon GameLift Servers 发送心跳。如果该代理未发送心跳，计算可能会被标记为过时。

- 在服务器进程终止时，向 Amazon GameLift Servers 报告。Amazon GameLift Servers 会使用此信息监控游戏服务器可用性，以便进行游戏会话放置。
- 为服务器进程发出实例集事件，包括：
 - `SERVER_PROCESS_INVALID_PATH`：游戏服务器进程启动参数配置不正确。
 - `SERVER_PROCESS_TERMINATED_UNHEALTHY`：游戏服务器进程在激活后 3 分钟内未报告有效的运行状况检查，因此被终止。
 - `SERVER_PROCESS_FORCE_TERMINATED`：游戏服务器进程在发送 `OnProcessTerminate()` 后 30 秒内未完全退出。
 - `SERVER_PROCESS_CRASHED`：游戏服务器进程因某种原因而崩溃。

计算管理

- 接收 Amazon GameLift Servers 服务发出的消息以关闭计算。
- 提示由 Amazon GameLift Servers 终止计算。

日志记录

- 将日志上传到 AWS 账户中的 Amazon S3 存储桶。

使用别名抽象化 Amazon GameLift Servers 实例集名称

Amazon GameLift Servers 别名用于抽象化托管目标。托管目标告知 Amazon GameLift Servers 为玩家托管新游戏会话时可从中查找可用资源的位置。别名在以下情况下很有用：

- 如果您的游戏不使用多舰队队列来放置游戏会话，则它会通过指定 Amazon GameLift Servers 舰队 ID 来请求新的游戏会话。在一个游戏的生命周期内，您会多次更换实例集，以更新服务器生成包、更新托管硬件和操作系统或者解决性能问题。使用别名来抽象化实例集 ID，以便将玩家流量从现有实例集无缝切换到新实例集。
- 除了在游戏客户端请求创建新游戏会话时创建该会话之外，您还想执行其他操作。例如，您可能想将使用 out-of-date 客户端的玩家引导到升级网站。

别名必须指定路由策略。该策略有两种类型。简单路由策略会将玩家流量路由到指定的实例集 ID，您可以更新该 ID 以重定向流量。终端路由策略会将消息传回客户端，而不是创建新游戏会话。您可以随时更改别名的路由策略。

如果您使用队列进行游戏会话放置，那么您在替换实例集时不需要别名来重定向流量。有了队列，您只需添加新实例集并删除旧实例集即可。此操作对玩家不可见，因为系统会自动使用新实例集来执行新游戏会话请求。它不会影响现有游戏会话。您可以使用实例集 ID 或别名来识别队列目标。

主题

- [创建 Amazon GameLift Servers 别名](#)

创建 Amazon GameLift Servers 别名

本主题介绍如何创建用于游戏会话放置的 Amazon GameLift Servers 别名。

创建别名

使用 Amazon GameLift Servers 控制台或 AWS Command Line Interface (AWS CLI) 创建别名。

Console

在 [Amazon GameLift Servers 控制台](#) 中，使用导航窗格打开别名页面。

1. 选择创建别名。
2. 输入别名名称。我们建议在别名名称中包含有意义的特征，以便在查看别名列表时提供帮助。
3. 根据需要输入别名描述。
4. 为别名选择路由策略。
 - a. 如果您选择了简单路由策略，请从列表中选择要与此别名关联的实例集 ID。该列表包含当前选择的 AWS 区域中的所有实例集。您必须在实例集所在的区域中创建别名。
 - b. 如果您选择了终端路由策略，请输入一个您希望 Amazon GameLift Servers 在响应游戏会话请求时返回给游戏客户端的字符串值。带有终端别名的请求会引发嵌入的消息的异常。
5. (可选) 向别名资源添加标签。每个标签都包含定义的一个键和一个可选值。为您希望以有用的方式 (如按用途、所有者或环境) 分类的 AWS 资源分配标签。为每个要添加的标签选择添加新标签。
6. 当您准备好部署新实例集时，请选择创建。

AWS CLI

使用 [create-alias](#) 命令创建别名。Amazon GameLift Servers 会在当前的默认 AWS 区域中创建别名资源 (您也可以添加 --region 标签来指定其他 AWS 区域) 。

至少要包含别名名称和路由策略。对于简单路由策略，请指定与别名位于同一区域的实例集的 ID。对于终端路由策略，请提供消息字符串。

管理 Amazon GameLift Servers 托管解决方案 (发布及后续运营)

此部分提供有关如何长期管理 Amazon GameLift Servers 资源的指导。成功发布多人游戏仅仅是个开始。管理生产级托管解决方案需要持续关注性能、成本优化、安全更新和玩家体验。此部分提供了在生产环境中运行 Amazon GameLift Servers 的运营方面的指导，涵盖从第一天的发布活动到长期维护和优化等各种事宜。

无论您是在为发布做准备、管理正在运营的游戏还是优化已建立的托管解决方案，这些主题都将帮助您保持可靠且经济实惠的游戏托管，并可根据您的玩家规模灵活扩缩。

使用 Amazon GameLift Servers 管理游戏托管资源

管理 Amazon GameLift Servers 资源，包括生成包、实例集、游戏会话队列和对战构建器。您可以使用 Amazon GameLift Servers 控制台浏览资源列表，或查看资源详细信息（包括指标）。您也可以使用 AWS SDK 或 AWS CLI 检索资源信息 Amazon GameLift Servers

Amazon GameLift Servers 控制台功能

使用 Amazon GameLift Servers 控制台，可以近乎实时地查看和管理您的游戏托管资源和正在进行的托管活动。该控制台为 Amazon GameLift Servers 服务 API 的大部分功能提供了图形界面。您可以使用控制台执行以下操作：

- 使用控制面板获取概览信息。您可以查看所有 Amazon GameLift Servers 托管资源的数量和当前状态，并点击链接获取有关各个资源的详细信息。
- 管理单个托管资源。您可以创建、查看和删除所有 Amazon GameLift Servers 资源，并更新其可变属性。您还可以查看某些类型的托管活动，例如活动和绩效指标。
- 与游戏和玩家会话活动交互。您可以按实例集跟踪游戏会话和玩家会话活动，并使用这些信息来解决游戏会话问题。查看游戏会话的详细信息，每个游戏会话的玩家会话，以及多个游戏会话中的玩家活动。您还可以根据需要关闭单个游戏会话。

主题

- [Amazon GameLift Servers 控制台中的托管控制面板](#)
- [游戏服务器生成包](#)

- [Amazon GameLift Servers 控制台中的实例集](#)
- [Amazon GameLift Servers 控制台中的别名](#)
- [Amazon GameLift Servers 控制台中的游戏会话队列](#)

Amazon GameLift Servers 控制台中的托管控制面板

使用Amazon GameLift Servers控制台仪表板获取有关您 AWS 账户中Amazon GameLift Servers托管资源的当前状态的高级快照。Amazon GameLift Servers 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 Servers 控制面板

- 在 [Amazon GameLift Servers 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。

GameLift > Dashboard

Dashboard

Build status overview

Viewing data for all builds in N. Virginia region.

View builds

Ready

1

Initialized

0

Failed

0

Fleet status overview

Viewing data for all fleets in N. Virginia region.

View fleets

Active

0

New

0

Deleting

0

Error

0

In progress

0

Terminated

0

Resources (1)

View service quotas

Resource type	Count
Builds	1
Scripts	0
Fleets	0
Aliases	0
Queues	0
Matchmaking rule sets	0
Matchmaking configurations	0

Prepare for your game launch

[Learn more](#)

Fill out a launch questionnaire

Fill out our game launch questionnaire and email it to the GameLift launch team to ensure a smooth launch. The GameLift launch team will verify your GameLift setup and service limits, preparing you for launch.

[Prepare to launch](#)

Features spotlight

Updates on features available in N. Virginia region

March 22, 2022

Updates to Amazon GameLift FlexMatch for greater flexibility

October 28, 2021

New Asia Pacific (Osaka) region and Graviton2 support for Amazon GameLift

游戏服务器生成包

生成包资源代表您的游戏服务器软件。您可以将构建包上传到，Amazon GameLift Servers以便部署到托管 EC2 队列。

在Amazon GameLift Servers控制台中或使用或 AWS SDK 查看有关游戏服务器版本的信息Amazon GameLift Servers。

Builds

304

Console

在[Amazon GameLift Servers](#)主机的“构建”页面上，您可以查看有关您上传到托管 EC2 队列上部署的所有游戏服务器版本的信息并对其Amazon GameLift Servers进行管理。在导航窗格中，选择“托管”、“托管” EC2、“构建”。

生成包页面显示了每个生成包的以下信息。您可以根据需要使用首选项工具调整表内容（请参阅表右上角的



图标)。自定义首选项将保存到您的 AWS 账户用户，并在您查看此页面时自动应用。

Note

Builds 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 – 在 Amazon GameLift Servers 收到二进制文件之前，生成包已过期。
- 创建时间 – 将生成包上传到 Amazon GameLift Servers 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。生成包操作系统决定了 Amazon GameLift Servers 在实例集实例上安装的操作系统。
- 大小：上传到 Amazon GameLift Servers 的生成包文件的大小（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。实例集部分显示运行该生成包的实例集列表，包括与[实例集页面](#)相同的摘要信息。

AWS SDK

使用以下 AWS CLI 命令检索有关此资源的信息：

- [ListBuild](#)
- [DescribeBuild](#)

更新游戏服务器生成包

为 Amazon GameLift Servers 托管式 EC2 托管部署游戏服务器生成包时，您需要上传游戏服务器软件并创建 Amazon GameLift Servers 生成包资源。创建 Amazon GameLift Servers 版本后，您可以更新版本的元数据，但不能自行更新构建文件。要将更新部署到游戏服务器版本中，必须使用 AWS CLI [upload-build](#) 命令上传一组新文件并创建新 Amazon GameLift Servers 版本。或者，您也可以使用 [create-build](#) 命令从您控制的 Amazon S3 存储桶上传新生成包。然后通过为新生成包创建新实例集来部署新生成包。

您可以更新生成包的元数据，包括名称和描述。要执行这些任务，请使用 Amazon GameLift Servers 控制台或 [update-build](#) AWS CLI 命令。

自动执行游戏生成包更新

请按照以下提示帮助自动执行和简化为 Amazon GameLift Servers 托管式实例集更新游戏服务器生成包的流程：

- 根据需要使用游戏会话队列和换出实例集。向 Amazon GameLift Servers 发送游戏会话请求时，请指定游戏会话队列，而不是特定的实例集。借助队列，您可以添加包含新生成包的实例集，以及根据需要删除旧实例集。有关更多信息，请参阅[配置游戏会话放置](#)。
- 使用别名将玩家传输到新游戏构建。向 Amazon GameLift Servers 发送游戏会话请求时，请指定实例集别名，而不是实例集 ID。有关更多信息，请参阅[创建 Amazon GameLift Servers 别名](#)。
- 为迭代开发做好准备。在游戏开发过程中，探索设置支持快速迭代开发的托管测试环境的选项。请参阅[借助 Amazon GameLift Servers Anywhere 为迭代开发做好准备](#)。

试试 Github 上 [Amazon GameLift Servers Toolkit](#) 中的以下资源：

Fast Build Update Tool (仅用于开发)

此工具可帮助您修改已部署在托管式 EC2 实例集中的计算上的游戏服务器生成包，从而在快速开发迭代期间节省时间。此工具有多个选项；您可以替换整个游戏生成包或更改特定文件，也可以管理更新后重新启动游戏服务器进程的方式。您还可以使用它来更新实例集中的所有计算或针对单个计算进行更新。

请访问 Github 中的 Amazon GameLift Servers Toolkit 存储库，以获取 Github 中的 [Fast Build Update Tool](#) 并详细了解其使用方法。

生产部署示例脚本

此脚本说明了如何自动执行以下过程：更新部署在生产环境中的托管式 EC2 实例集上的游戏服务器生成包。要使用此脚本，您的 Amazon GameLift Servers 托管解决方案必须使用别名来抽象舰队 IDs。此示例脚本处理以下基本步骤：上传更新后的生成包，创建新生成包并部署到新实例集，将玩家流量从现有实例集重定向到新实例集，以及删除旧实例集。请自定义示例脚本以满足您的特定部署要求。

请访问 Github 中的 Amazon GameLift Servers Toolkit 存储库，以获取 Github 中的 [生产部署示例脚本](#) 并详细了解其使用方法。

Amazon GameLift Servers 控制台中的实例集

实例集资源表示一组已部署并用于运行您的游戏服务器的计算设备。根据您的托管解决方案，您可能拥有托管舰队、托管集装箱 EC2 舰队、Anywhere 舰队或两者兼而有之。

在 Amazon GameLift Servers 控制台中或使用 AWS SDK 查看有关舰队的信息。Amazon GameLift Servers

Console

您可以查看您的 AWS 账户 Amazon GameLift Servers 下为托管游戏而创建的所有舰队的信息。在控制台的左侧导航中，找到所需的托管选项（Anywhere、托管 EC2、托管容器），然后选择 Fleets。

对于每个托管选项，“车队”页面都会列出位于您当前所选 AWS 地区的车队。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。您也可以编辑舰队记录或删除舰队。

默认情况下，实例集页面显示以下摘要信息。您可以根据需要使用首选项工具调整表内容（请参阅表右上角的



图标)。自定义首选项将保存到您的 AWS 账户用户，并在您查看此页面时自动应用。

- ID – 分配给实例集的标识符。这个 ID 在创建队列 AWS 区域 的地方是唯一的。
- 名称 – 为实例集指定的易记名称。
- 状态 – 实例集的状态：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。

Note

对于在 90 天前创建的实例集，实例集会显示警告图标。作为最佳实践，我们建议每 30 天更换一次舰队，以维护托管游戏服务器的安全 up-to-date 运行环境。有关指南，请参阅 [Amazon GameLift Servers 的安全最佳实践](#)。

- 实例集类型：用于托管游戏的实例的可用性，这可能会影响托管成本。托管式实例集可以使用按需型（始终可用）或竞价型（可用性动态变化）实例。
- 活动实例-队列正在使用的 EC2 实例数量。
- 所需实例-要保持活动状态的 EC2 实例数量。
- 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。
- 玩家会话：实例集中活跃玩家会话的数量。该数据有五分钟的延迟。

AWS SDK

使用以下 AWS CLI 命令检索有关此资源的信息：

- 托管 EC2 车队
 - [ListFleets](#)
 - [ListCompute](#)
 - [DescribeFleetAttributes](#)
 - [DescribeCompute](#)
 - [DescribeFleetCapacity](#)
 - [DescribeFleetUtilization](#)
- 托管式容器实例集
 - [ListContainerFleets](#)

- [DescribeContainerFleet](#)
- [ListContainerGroupDefinitions](#)
- [DescribeContainerGroupDefinition](#)
- [ListFleets](#)
- [ListCompute](#)
- [DescribeFleetAttributes](#)
- [DescribeCompute](#)

更新 Amazon GameLift Servers 实例集配置

使用 Amazon GameLift Servers 控制台或 AWS CLI 更新您的舰队设置、更改远程位置或删除舰队。对于托管式实例集，您无法更改实例集的游戏服务器生成包或实例类型，而是必须更换实例集。

Fast Build Update Tool (仅用于开发)

对于托管 EC2 舰队，要部署游戏服务器版本更新，您需要将每个新版本上传到该 Amazon GameLift Servers 队列并为其创建新的舰队。

Fast Build Update Tool 可让您在开发过程中绕过这些步骤，从而节省时间并加快开发迭代速度。利用此工具，您可以在现有实例集的所有计算中快速更新游戏生成包文件。此工具有多个选项；您可以替换整个游戏生成包或更改 6 个特定文件，也可以管理更新后重新启动游戏服务器进程的方式。您还可以使用它来更新实例集中的单个计算。

要获取 Fast Build Update Tool 并详细了解其使用方法，请访问 Github 中的 Amazon GameLift Servers Toolkit 存储库，查看关于 [Fast Build Update Tool](#) 的页面。

您可以使用 Amazon GameLift Servers 控制台或 AWS CLI 更新可变队列属性、端口设置和运行时配置。要更改扩展限制，请参阅[通过 Amazon GameLift Servers 自动扩展队组容量](#)。

Console

1. 在 [Amazon GameLift Servers 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，亚马逊 CloudWatch 使用这些指标组来跟踪多个队列的汇总 Amazon GameLift Servers 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-更新允许对队列进行入站访问的 IP 地址和端口范围。
4. 选择确认以保存所做的更改。

AWS CLI

使用以下 AWS CLI 命令更新舰队：

- [update-fleet-attributes](#)
- [update-fleet-port-settings](#)
- [update-runtime-configuration](#)

更新实例集位置

您可以使用 Amazon GameLift Servers 控制台或 AWS CLI 来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。

Console

1. 在 [Amazon GameLift Servers 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 Servers 会将新位置添加到列表中，并将状态设置为 NEW。然后，Amazon GameLift Servers 开始在每个添加的位置预调配一个实例，并使其准备就绪以托管游戏会话。
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 Servers 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 Servers 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

使用以下 AWS CLI 命令更新舰队位置：

- [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

如果队列具有 VPC 对等连接，请先调用[CreateVpcPeeringAuthorization](#)请求授权。Amazon GameLift Servers 在删除队列期间删除 VPC 对等连接。

您可以使用 Amazon GameLift Servers 控制台或 AWS CLI 工具删除舰队。

Console

1. 在 [Amazon GameLift Servers 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 CLI 命令删除舰队：

- [delete-fleet](#)

Amazon GameLift Servers 控制台中的实例集详细信息

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。

Details

实例集设置

- 实例集 ID – 分配给实例集的标识符。这个 ID 在创建队列 AWS 区域 的地方是唯一的。
- 实例集名称：为实例集指定的易记名称。
- ARN – 分配给此实例集的唯一标识符。实例集的 ARN 将其标识为 Amazon GameLift Servers 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。

 Note

对于在 90 天前创建的实例集，实例集会显示警告图标。作为最佳实践，我们建议每 30 天更换一次舰队，以维护托管游戏服务器的安全 up-to-date 运行环境。有关指南，请参阅[Amazon GameLift Servers 的安全最佳实践](#)。

- 计算类型 – 用于托管游戏的计算类型。队列可以是托管集装箱 EC2、托管集装箱或 Anywhere 队列。
- 操作系统：部署到实例集中每个实例的亚马逊机器映像 (AMI)。AMI 版本是指创建实例集时可用的最新版本。
- 证书类型-舰队使用 TLS 证书对游戏服务器进行身份验证和加密所有 client/server 通信的能力状态。
- 实例集类型：用于托管游戏的实例的可用性，这可能会影响托管成本。实例集可以使用按需型 (始终可用) 或竞价型 (可用性动态变化) 实例。
- EC2 实例类型-创建队列时为队列选择的 Amazon EC2 [实例类型](#)。
- 实例角色 ARN — 一个 AWS IAM 角色，用于管理对您的其他 AWS 资源的访问权限 (如果在创建队列时提供了其他资源)。
- 实例角色证书提供商 — 一个 AWS IAM 角色，用于管理对您的其他 AWS 资源的访问权限 (如果在创建队列时提供了其他资源)。
- 指标组名称 – 用于聚合多个实例集的指标的组。
- 游戏扩缩保护策略：指示是否启用了实例集的游戏会话保护，该功能可以防止活动游戏会话在缩减事件期间过早结束。
- 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。
- 策略周期：用于限制玩家可创建会话数量的时间长度。

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。有关在 Amazon GameLift Servers 中使用指标的更多信息，请参阅[Amazon GameLift Servers 使用 Amazon 进行监控 CloudWatch](#)。

Events

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。请参阅《Amazon GameLift Servers 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[利用 Amazon GameLift Servers 扩展游戏托管容量](#)。

- 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，将启动扩缩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。此处显示了舰队基于规则的策略，这些策略必须使用 AWS SDK 或 CLI 进行定义。有关扩展的更多信息，请参阅[通过 Amazon GameLift Servers 自动扩展队组容量](#)。

扩展历史记录

查看容量随时间变化的图表。

位置

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[Amazon GameLift Servers 控制台中的游戏和玩家会话](#)。

Amazon GameLift Servers 控制台中的游戏和玩家会话

您可以使用 Amazon GameLift Servers 控制台来处理游戏会话和玩家会话。有关游戏会话和玩家会话的更多信息，请参阅[Amazon GameLift Servers 和玩家体验](#)。Amazon GameLift Servers 控制台提供相关信息和工具，可帮助您调查游戏会话中的问题。

支持的操作：

- 浏览托管在特定实例集上的游戏会话和玩家会话活动。
- 在多个实例集中查找特定玩家的游戏会话活动。
- 关闭特定游戏会话。

查看游戏会话详情

游戏会话和玩家会话数据按托管游戏会话的实例集组织。

访问游戏会话和玩家会话信息

1. 在 [Amazon GameLift Servers 控制台](#) 中，打开左侧导航窗格。选择托管解决方案类型，并打开实例集页面。例如：
 - 托管、Anywhere、实例集
 - 托管、托管式 EC2、实例集
 - 托管、托管容器、实例集
2. 每个 Fleets 页面都会显示您当前选择的舰队列表。AWS 区域请选择要查看其游戏会话数据的实例集。
3. 在实例集的详情页面中，打开游戏会话选项卡。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。您可以根据需要使用首选项工具调整表内容（请参阅表右上角的  图标）。自定义首选项将保存到您的 AWS 账户用户，并在您查看此页面时自动应用。
4. 从列表中选择一個游戏会话以查看更多信息。
5. 如果该游戏会话包含玩家会话数据，请选择查看玩家会话以打开玩家会话查询工具，其中会自动填充游戏会话 ID。

游戏会话详情包括以下信息：

- 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 Servers 托管游戏会话的位置。
- 创建时间 – Amazon GameLift Servers 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。
- 游戏扩缩保护策略 – 要在 Amazon GameLift Servers 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

游戏属性数据，格式为字符串，在启动时发送到游戏会话。

游戏属性

游戏属性数据，格式成 key/value 对，在启动时发送到您的游戏会话。

对战数据

如果游戏会话是使用 FlexMatch 创建的，对战数据将描述有关对战配置和规则集的信息。这包括每个匹配项的玩家属性和队伍分配。数据采用 JSON 格式。有关 FlexMatch 对战的更多信息，请参阅[构建对战构建器](#)。

查找玩家会话数据

如果您的游戏托管解决方案使用玩家会话并提供独特的玩家 IDs，则您可以浏览多个队列中过去或现在的游戏会话中特定于玩家的活动。可使用以下任一方法打开玩家会话查询工具：

- 在 Amazon GameLift Servers 控制台中，打开左侧导航窗格并选择玩家会话查找，然后选择要使用的过滤器类型。
- 要查看实例集的游戏会话详细信息，请选择查看玩家会话。查找工具将打开游戏会话，其中已预先选择了游戏会话 ID 过滤器并填写了游戏会话值。

使用查找工具时，您可以提供以下任何信息：

- 玩家会话 ID，用于获取有关特定玩家会话的信息。
- 游戏会话 ID，用于获取有关所请求游戏会话的所有玩家会话的信息。结果代表已预留位置或连接到游戏会话的所有玩家。您可以选择按玩家会话状态筛选结果。
- 玩家 ID，用于获取所请求玩家的所有玩家会话信息。结果代表玩家参与的所有游戏会话。

 Note

查找工具会搜索当前选定区域内的所有玩家会话活动 AWS 区域。如果您在该区域内有多个实例集，结果将包含所有实例集中的玩家会话活动。对于多位置实例集，结果还将包含实例集远程位置的玩家会话活动。

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。
- 游戏会话 ID – 分配给游戏会话的标识符。
- 实例集 ID – 分配给托管游戏会话的实例集的标识符。
- 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。
- 连接数据 – 玩家用于连接到游戏会话的 IP 地址、DNS 名称和端口。
- 玩家数据 – 玩家会话创建期间提供的有关玩家的信息。

关闭游戏会话

使用 Amazon GameLift Servers 控制台关闭特定的游戏会话。此功能为您提供了一种简单快捷的方法来定位游戏会话并发送终止信号。另一种终止方法则要求您找到运行游戏会话的实例集实例，远程访问该实例，然后手动关闭游戏会话。

您可以出于任何原因关闭游戏会话。最常见的原因是为了解决无法自然关闭的游戏会话问题。此类问题会导致无法腾出游戏会话的托管资源来托管新的游戏会话，而且实例集的托管容量也随之降低。

 Note

此功能依赖于您的托管解决方案的某些配置设置，并且具有以下限制：

- 游戏会话必须托管在运行游戏服务器生成包（使用适用于 Amazon GameLift Servers v5 或更高版本的服务器 SDK）的实例集上。如果您的游戏服务器部署的是旧版本，则需要通过远程访问来删除游戏会话。
- 如果游戏会话托管在 Anywhere 实例集上，则该实例集必须使用 Amazon GameLift Servers 代理来管理游戏服务器进程。

终止游戏会话

1. 在 [Amazon GameLift Servers 控制台](#) 中，打开左侧导航窗格。选择托管解决方案类型，并打开实例集页面。例如：
 - 托管、Anywhere、实例集
 - 托管、托管式 EC2、实例集
 - 托管、托管容器、实例集
2. 每个 Fleets 页面都会显示您当前选择的舰队列表。AWS 区域选择托管您要终止的游戏会话的实例集。
3. 在实例集的详情页面中，打开游戏会话选项卡。在游戏会话列表中，选择要终止的会话，然后选择终止按钮。
4. 在终止游戏会话中？窗口中，确认您正在关闭正确的游戏会话并选择终止方法。
 - 正常关闭游戏会话：此选项会向托管游戏会话的服务器进程发送关闭信号。如果您的游戏服务器生成包已正确集成 Amazon GameLift Servers，则服务器进程会启动其游戏会话关闭序列，通知 Amazon GameLift Servers 该会话即将结束，然后停止。根据您的游戏设计，关闭序列可能包括正常完成游戏会话的步骤，例如保存数据和通知活跃玩家。此方法可能需要短暂延迟来完成游戏会话关闭序列。
 - 立即关闭游戏会话：此选项会向进程管理器发送信号，要求其关闭托管游戏会话的服务器进程。此选项会绕过正常的游戏会话关闭流程。即使服务器进程无法响应，它也可以终止游戏会话。
5. 确认游戏会话终止。您可以在游戏会话控制台页面上跟踪关闭进度。游戏会话状态将更改为“正在终止”，然后在关闭完成后更改为“已终止”。

相关主题

- 您也可以使用 AWS SDK 和 关闭游戏会话 AWS CLI。有关更多详细信息和示例，请参阅 Amazon GameLift Servers API 参考主题 [TerminateGameSession](#)。

- 有关游戏服务器集成以及服务器进程如何响应来自 Amazon GameLift Servers 服务的信号的更多信息，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

更新 Amazon GameLift Servers 托管式容器实例集

您可以更新托管式容器实例集的大部分属性，包括容器组定义。根据所更新的设置，实例集更新可能会启动新的实例集部署。在实例集部署过程中，实例集中的所有实例都将被移除，并替换为采用新配置的实例。需要重新部署的设置包括：

- 容器组定义，包括对容器映像的更新
- 连接端口范围和入站权限
- 日志配置

您可以在用于创建容器实例集的 [Amazon GameLift Servers 控制台](#) 或 AWS Command Line Interface (AWS CLI) 中跟踪实例集部署状态。

Note

您无法更新实例集的运行时环境。创建实例集时，实例集的亚马逊机器映像 (AMI) 会设置为最新可用的 Linux AMI 版本。部署到该实例集的所有容器映像都必须与此版本兼容。要更改实例集的 AMI 或升级到新版本，您必须创建新的实例集。作为最佳实践，我们建议每 30 天更换一次实例集，以便为托管游戏服务器维护安全且最新的运行时环境。有关更多指导，请参阅[Amazon GameLift Servers 的安全最佳实践](#)。

Console

在 [Amazon GameLift Servers 控制台](#) 中，选择要创建实例集的 AWS 区域。容器组定义必须与您要创建实例集的区域保持一致。

打开控制台的左侧导航栏，并选择托管容器：实例集。在托管式容器实例集页面上，从列表选择一个实例集，然后选择编辑。

1. 根据需要更新容器实例集设置。完成后，选择 Create (创建)。
2. 如果您的更新需要重新部署实例集，系统会要求您指定以下部署选项：
 - 游戏会话保护。您可以选择保护具有处于活动状态的游戏会话的实例集实例 (安全部署)。使用此设置，直到游戏会话结束后才会替换实例集实例。或者，您也可以选择无论游戏会话

活动如何，都替换实例集实例（非安全部署）。非安全部署适用于开发和测试阶段，可缩短部署时间。

- 最低运行正常百分比。您可以管理替换实例集实例的速度。使用此设置，可在部署期间保持最低数量的正常任务。较低的数值会优先保证部署速度，而较高的数值则可确保游戏服务器在整个部署过程中保持较高的可用性。
- 部署失败策略。决定部署失败时要采取的措施。部署失败指部分更新后的容器未通过状态检查，被判定为异常状态。您可以将部署设置为自动将所有实例集实例回滚到先前部署的状态。或者，您也可以选择保留部分异常的实例集实例以用于调试。

如果请求成功，控制台将显示托管容器实例集的部署选项卡。您可以使用此选项卡跟踪每个部署的状态。如果您开始对实例集进行新的部署，则此操作会自动取消实例集当前正在进行的任何部署。

AWS CLI

要使用 AWS CLI 创建容器实例集，请打开命令行窗口，然后使用 `update-container-fleet` 命令。有关此命令的更多信息，请参阅《AWS CLI 命令参考》中的 [update-container-fleet](#)。

以下示例更新了一个具有以下特征的现有容器实例集：

- 它将游戏服务器容器组定义更新为使用版本 2。
- 它指定了安全部署选项。

```
{
  "DeploymentConfiguration": {
    "ImpairmentStrategy": "ROLLBACK",
    "MinimumHealthyPercentage": 75,
    "ProtectionStrategy": "WITH_PROTECTION"
  },
  "FleetId": "containerfleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
  "GameServerContainerGroupDefinitionName": "arn:aws:gamelift:us-west-2:111122223333:containergroupdefinition/MyAdventureGameContainerGroup:2"
}
```

更新 Amazon GameLift Servers 容器实例集的容器组定义

您可以更新容器实例集的大部分属性，包括各个独立的容器定义。容器组定义包含版本号。更新容器组定义时，Amazon GameLift Servers 会保存更新并递增该定义版本号。配置容器实例集时，可以指定要部署的容器组定义版本。

更新容器组定义后，您可以将新版本部署到新的或现有的容器实例集中。

更新游戏服务器容器组定义

本主题介绍如何使用 Amazon GameLift Servers 控制台或 AWS CLI 工具更新游戏服务器容器组定义。有关可选功能的更多详细信息，请参阅[自定义 Amazon GameLift Servers 容器实例集](#)。

要更新容器组定义，请执行以下操作：

Console

在[Amazon GameLift Servers 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，并选择托管容器：组定义。在容器组定义页面上，选择要更新的容器组定义和版本。

保存更新后，您可以使用新版本创建新的容器实例集，也可以将更新部署到现有的容器实例集。

步骤 1：定义容器组定义详情

- 您可以更新总内存和 vCPU 限制设置。

步骤 2：添加容器定义

容器定义可以进行以下更新：

- 更新现有容器定义。
 - 添加新的支持容器定义。
 - 移除支持容器定义。
1. 您可以更新 ECR 映像 URI。请务必更新服务器 SDK 版本设置以匹配新映像。
 2. 您可以根据需要更新内部容器端口范围。当您对这些设置所做的更改被部署到容器实例集时，可能会影响该实例集的连接端口设置。有关更多详细信息，请参阅[配置网络连接](#)。

步骤 3：配置依赖关系

- 您可以按需更改依赖关系。有关更多信息，请参阅 [设置容器依赖关系](#)。

第 3 步：审核并创建

- 查看您的容器组定义更新。使用编辑，对任何部分进行其他任何更改。完成后，选择创建以生成新版本的容器组定义。

如果请求成功，控制台将显示新容器组定义资源的详细信息页面。初始状态为 `COPYING`，此时 Amazon GameLift Servers 开始为该组的所有容器映像拍摄快照。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，才能使用它创建容器实例集。

AWS CLI

使用 AWS CLI 创建或更新容器组定义时，请在单独JSON的文件中维护您的容器定义配置。您可以在 CLI 命令中引用该文件。有关架构示例，请参阅 [创建容器定义 JSON 文件](#)。

更新定义时，您只需指定要更新的值即可。Amazon GameLift Servers 会保留您未包含在更新请求中的任何值。不过，如果要更改容器定义，则需提供一套完整的定义。

更新容器组定义

要更新新的容器组定义，请使用 `update-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》[update-container-group-definition](#) 中的。

Example：游戏服务器容器组

在检索、更新或删除容器组定义时，或者在创建或更新容器实例集时，您可以指定容器组定义版本。每个容器组定义都有一个版本属性。此外，定义的 ARN 值也会指定版本号。

此示例展示了更改游戏服务器容器组定义的请求。假设您已经创建了一个包含该组容器定义的 JSON 文件。此示例使用 ARN 值作为定义名称，并指定此次更新针对的是版本 1。

```
aws gamelift update-container-group-definition \
  --name arn:aws:gamelift:us-west-2:111122223333:containergroupdefinition/
MyAdventureGameContainerGroup:1 \
  --operating-system AMAZON_LINUX_2023 \
  --container-group-type GAME_SERVER \
  --total-memory-limit-mebibytes 4096 \
```

```
--total-vcpu-limit 1 \  
--container-definitions file://SimpleServer.json
```

克隆容器组定义

您可以使用 Amazon GameLift Servers 控制台来克隆现有的容器组定义。

克隆容器组

1. 在 [Amazon GameLift Servers 控制台](#) 中，转到左侧导航窗格，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，克隆按钮变为活动状态。
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. 依次浏览容器组和容器定义页面，进行审核，然后创建新的容器组。

删除 Amazon GameLift Servers 容器实例集的容器组定义

您可以通过多种方式来删除容器组定义。删除容器组定义时，此操作还会删除该容器组中的所有容器定义。

容器组定义可能有多个版本。容器组版本名称相同，但版本号不同。容器组定义同时 ARNs 指定名称和版本。

在检索、更新或删除容器组定义时，或者在创建或更新容器实例集时，您可以指定容器组定义版本。每个容器组定义都有一个版本属性。此外，定义的 ARN 值也会指定版本号。

删除容器组定义有以下几种方式：

- 您可以删除特定定义的所有版本。
- 您可以删除特定定义的某个特定版本。
- 您可以针对特定定义保留一定数量的最新版本，并删除其旧版本。例如，您可以删除所有版本号低于 5 的版本。

仅当容器组定义版本未在容器实例集中使用时，您才能删除该版本。

删除容器组定义

Console

在[Amazon GameLift Servers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，并选择托管容器：组定义。在容器组定义页面上，选择要修改的定义，然后选择删除。

系统会提示您选择要执行的删除类型，并根据所选删除类型指定额外设置。

AWS CLI

- 要删除容器组定义，请使用 `delete-container-group-definition` CLI 命令并提供要执行的删除类型的值。有关此命令的更多信息，请参见《AWS CLI 命令参考》[delete-container-group-definition](#)中的。

以下示例展示了删除版本号早于 5 的所有游戏服务器容器组定义的请求。

Example

```
aws gamelift delete-container-group-definition \
  --name MyAdventureGameContainerGroup \
  --version-count-to-retain 5 \
```

Amazon GameLift Servers 控制台中的别名

别名资源代表指向实例集或终端目标的指针。它们用于将玩家流量引导到特定的托管目标位置。通过别名对目标位置进行抽象化处理，您在更新托管实例集时可以更便捷地重定向流量。

在Amazon GameLift Servers控制台中或使用或 AWS SDK 查看有关别名的信息。Amazon GameLift Servers

Console

别名页面显示有关 Amazon GameLift Servers 别名的信息，这些别名用于将流量引导至特定的托管目标位置。要查看别名，可在导航窗格中依次选择托管和别名。

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。您可以根据需要使用首选项工具调整表内容（请参阅表右上角的



图标)。自定义首选项将保存到您的 AWS 账户用户，并在您查看此页面时自动应用。

- 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，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。
- 标签 - 用于标识别名的键和值对。

AWS SDK

使用以下 AWS CLI 命令检索有关此资源的信息：

- [ListAliases](#)
- [DescribeAlias](#)

编辑别名

您可以使用 Amazon GameLift Servers 控制台或 AWS CLI 命令 [update-alias](#) 来编辑别名。

本主题介绍如何编辑用于游戏会话放置的 Amazon GameLift Servers 别名。您可以进行以下编辑：

编辑别名

使用 Amazon GameLift Servers 控制台或 AWS Command Line Interface (AWS CLI) 编辑别名。

Console

在 [Amazon GameLift Servers 控制台](#) 中，使用导航窗格打开别名页面。

1. 选择要编辑的别名并选择编辑。如果没有看到要编辑的别名，请检查当前选择的 AWS 区域。
2. 在编辑别名页面上，可以进行以下编辑：
 - 更改别名名称。
 - 更改别名描述。
 - 将路由策略从“简单”更改为“终端”，或从“终端”更改为“简单”。
 - 对于包含简单路由策略的别名，请更改与该别名关联的实例集 ID。
 - 对于包含终端路由策略的别名，请更改消息文本。
3. 选择保存更改。更新包含简单路由策略的别名的实例集 ID 时，可能需要 2 分钟才能完成转换。在此期间，可能会在旧实例集上进行新的游戏会话放置。

AWS CLI

使用 [update-alias](#) 命令对别名资源进行更改。您可以更新当前默认 AWS 区域中的别名资源，也可以添加 `--region` 标签来指定其他 AWS 区域。

您可以更改以下属性：

- 别名名称。
- 别名描述。
- 路由策略类型。请务必为新路由策略提供实例集 ID 或消息字符串。
- 现有简单路由策略的实例集 ID。实例集 ID 必须与别名位于同一区域中。
- 现有终端路由策略的消息字符串。

Amazon GameLift Servers 控制台中的游戏会话队列

队列资源表示您对游戏会话放置组件的配置。它决定了 Amazon GameLift Servers 如何根据您的配置选项搜索并选择最佳计算资源来托管新游戏会话。

在 Amazon GameLift Servers 控制台中或使用 AWS SDK 查看有关游戏会话队列的信息 Amazon GameLift Servers。

Console

您可以查看所有队列的信息，这些队列用于处理新游戏会话的请求。队列页面显示当前所选队列中的游戏会话队列 AWS 区域。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 [配置游戏会话放置](#)。

“队列”页面将显示每个队列的以下摘要信息：您可以根据需要使用首选项工具调整表内容（请参阅表右上角的



图标)。自定义首选项将保存到您的 AWS 账户用户，并在您查看此页面时自动应用。

- 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数量。Amazon GameLift Servers 会在队列中的任何实例集上放置新游戏会话。

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，请转到队列页面并选择队列名称。

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。

Details

概述部分显示队列的 Amazon 资源名称 (ARN) 和超时时间。在 Amazon GameLift Servers 的其它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 Servers 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，请参阅为资源[添加标签](#)。AWS

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。

Amazon 中也提供队列指标 CloudWatch。有关这些指标的说明，请参阅 [Amazon GameLift Servers 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 Amazon GameLift Servers 搜索可用资源的目标以托管新游戏会话时，会按此处列出的默认顺序进行搜索。只要列出的第一个目标上有容量，Amazon GameLift Servers 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 Amazon GameLift Servers 搜索平均玩家延迟最小的可用目标。有关更多信息，请参阅设计您的架构。

会话放置

用于自定义游戏放置流程的配置设置。

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。

位置

“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。

AWS SDK

使用以下 AWS CLI 命令检索有关此资源的信息：

- [DescribeGameSessionQueues](#)

借助 Amazon GameLift Servers 托管服务，为游戏发布做好准备

使用以下清单来验证游戏的每个部署阶段。标有 [关键] 的项目对于您的产品发布至关重要。

请下载并填写 Amazon GameLift Servers 发布问卷，该问卷可在 [Amazon GameLift Servers 控制台](#) 中找到。我们希望每位使用 Amazon GameLift Servers 的游戏开发者都能顺利完成发布，所要求的信息有助于我们帮助您为即将到来的负载测试、软发布或公开发布做好准备。请计划在首次负载测试前的至少三（3）个月提交填写完毕的问卷。

主题

- [让您的游戏准备就绪](#)
- [准备测试](#)
- [准备发布](#)
- [制定发布后更新计划](#)

让您的游戏准备就绪

- [关键] 验证您是否已完成托管解决方案的所有[开发路线图步骤](#)，并且已准备好所有必需的组件，包括集成游戏服务器、游戏客户端的后端服务、托管实例集和游戏会话放置方法（如队列）。

- [关键] [创建 AWS Identity and Access Management \(IAM\) 角色](#)以允许您的游戏服务器在运行时访问其他 AWS 资源。
- [关键] 根据需要设计并实施到其他托管资源的失效转移。
- 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 CloudFormation 和的基础设施即 AWS Cloud Development Kit (AWS CDK)代码 (IaC) 实现 [@@ 部署自动化](#)。
- 使用亚马逊@@ [CloudWatch 和亚马逊简单存储服务 \(Amazon S3\) Service 收集日志和分析](#)。

准备测试

- [关键] [请求增加Amazon GameLift Servers服务配额](#)和其他 AWS 服务 配额，以便您的实际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。
- 制定游戏 DevOps 管理计划。如果您使用的是 Amazon CloudWatch Logs 或 Amazon CloudWatch 自定义指标，请为服务器队列中的严重或关键问题定义警报。模拟故障并测试运行手册。
- 验证您使用的计算资源是否可以支持您希望在每个计算上并发运行的服务器进程的数量。
- [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%。
- 对于 FlexMatch，使用[延迟规则](#)来匹配地理位置相近的玩家。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 支持 计划级别提高到企业版或企业版，以便在出现问题或停机时 AWS 能够对您做出响应。

准备发布

- [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。

制定发布后更新计划

- 根据玩家使用情况[调整扩展策略](#)，最大限度地减少空闲容量。
- 根据玩家延迟数据和修订后的要求，[修改 FlexMatch 规则](#)或[添加托管位置](#)。
- 优化运行时配置，以便在每个计算资源上运行尽可能多的游戏会话。以这种方式最大限度地提高性能效率会直接影响您的实例集成本，因为您可以使用相同的计算资源运行更多的服务器进程。
- [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。

使用管理Amazon GameLift Servers托管资源 CloudFormation

您可以使用 CloudFormation 来管理您的Amazon GameLift Servers资源。在中 CloudFormation，您可以创建用于对每个资源进行建模的模板，然后使用该模板来创建您的资源。要更新资源，您需要对模板进行更改并使用 CloudFormation 来实现更新。您可以将资源按照称为堆栈和堆栈集的逻辑组排列。

使用 CloudFormation 来维护您的Amazon GameLift Servers托管资源可以更有效地管理 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。

有关的更多信息 CloudFormation，请参阅《[AWS CloudFormation 用户指南](#)》。要查看有关 Amazon GameLift Servers 资源的模板信息，请参阅 [Amazon GameLift Servers 资源类型参考](#)。

最佳实践

有关使用的详细指导 CloudFormation，请参阅《AWS CloudFormation 用户指南》中的[CloudFormation 最佳实践](#)。此外，这些最佳实践与 Amazon GameLift Servers 有特殊关联。

- 仅通过 CloudFormation 一致地管理您的资源。如果您更改资源以外的资源，资源将与您的资源模板不同步。CloudFormation
- 使用 CloudFormation 堆栈和堆栈集来高效管理多个资源。
 - 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，则 CloudFormation 会替换与该版本连接的舰队。CloudFormation 然后更新现有别名以指向新的舰队。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的[使用堆栈](#)。
 - 如果您要在多个地区或 AWS 账户中部署相同的堆栈，请使用 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 CloudFormation 堆栈

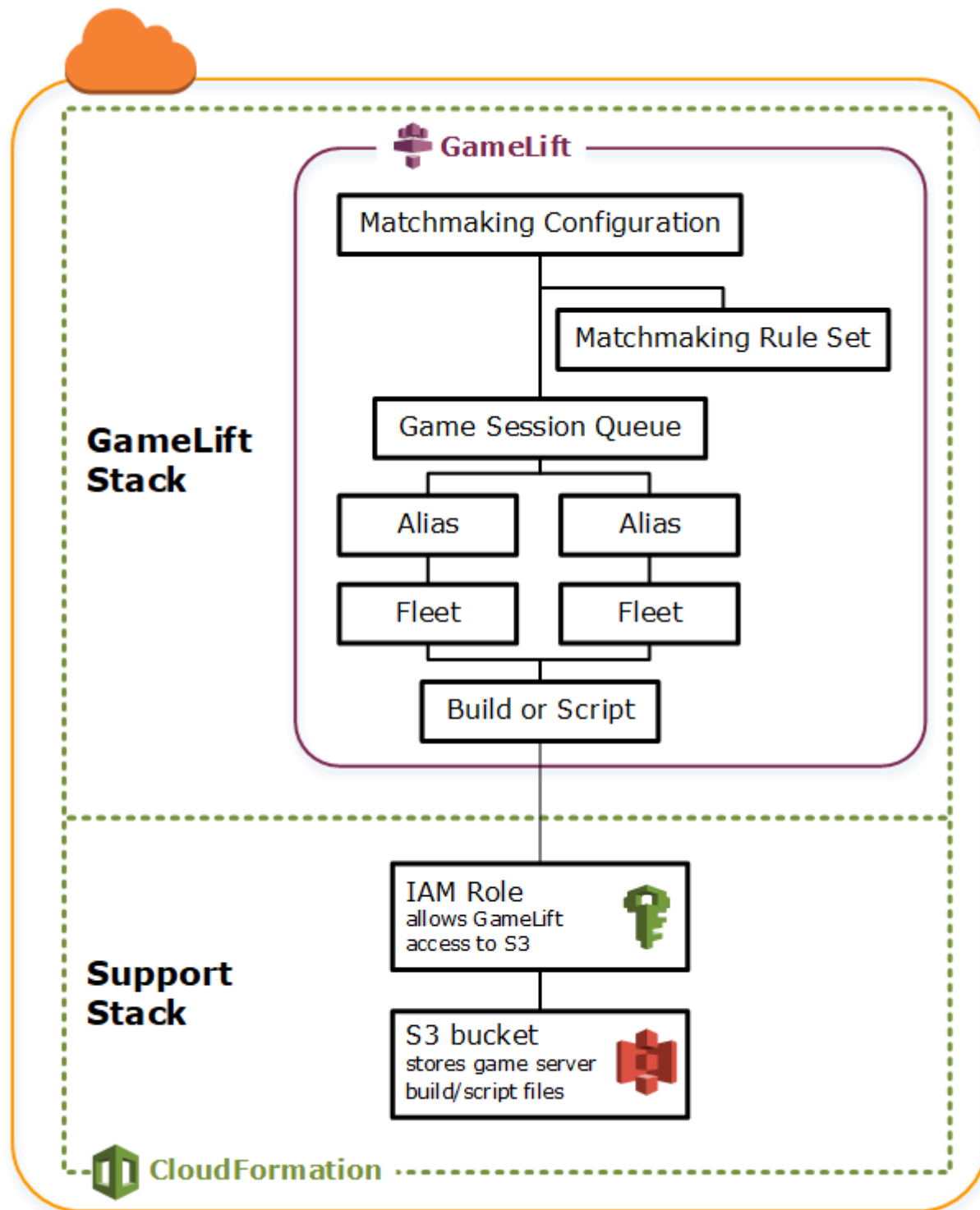
我们建议在为 Amazon GameLift Servers 资源设置 CloudFormation 堆栈时使用以下结构。您的最佳堆栈结构根据您在一个还是多个位置部署游戏而异。

用于单个位置的堆栈

要在单个位置管理 Amazon GameLift Servers 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 Servers 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。堆栈还应包括一个 IAM 角色，该角色 Amazon GameLift Servers 允许在创建 Amazon GameLift Servers 构建或脚本资源时从 S3 存储桶检索文件。该堆栈还可能包含用于游戏的其他 AWS 资源，例如 DynamoDB 表、Amazon Redshift 集群和 Lambda 函数。
- Amazon GameLift Servers stack — 此堆栈包含您的所有 Amazon GameLift Servers 资源，包括构建或脚本、一组舰队、别名和游戏会话队列。CloudFormation 使用存储在 S3 存储桶位置的文件创建生成或脚本资源，并将构建或脚本部署到一个或多个队列资源。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 配对，则此堆栈还包含配对配置和规则集。

下图说明了在单个 AWS 区域中部署资源的双栈结构。

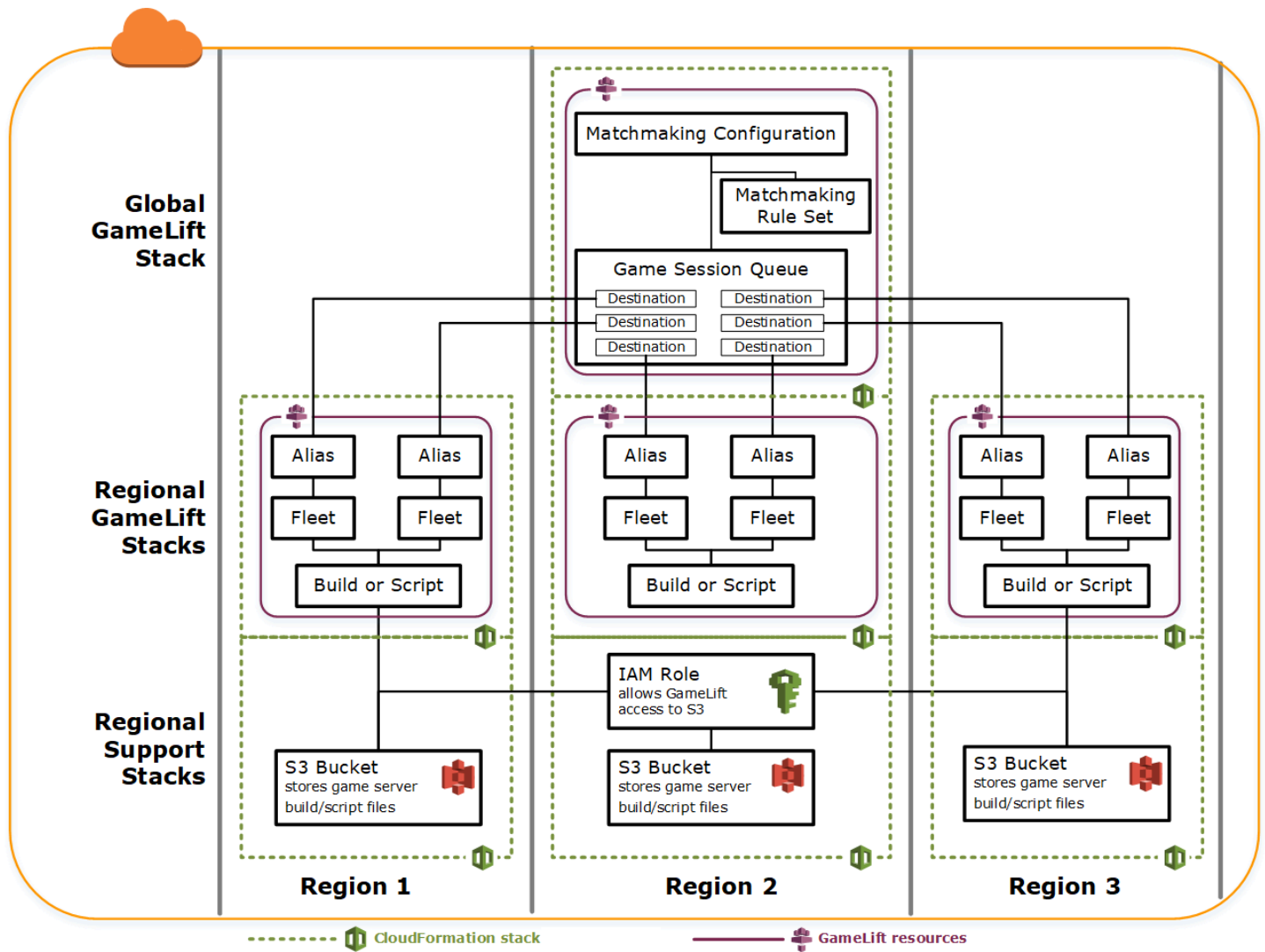


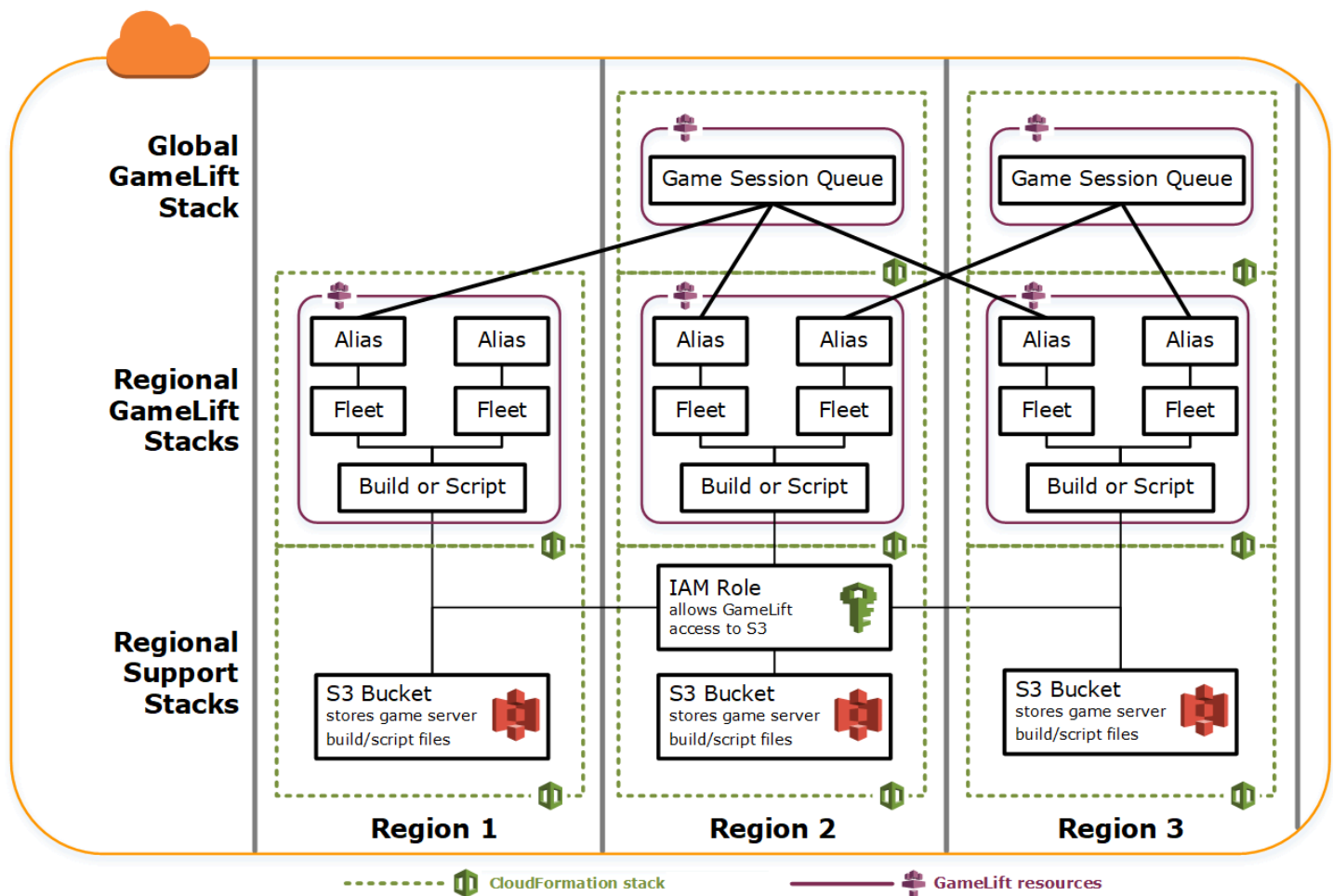
适用于多个区域的堆栈

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 Servers 队组）只能引用同一区域中的其他资源。其他资源（例如 Amazon GameLift Servers 队列）与区域无关。要管理多个区域中的 Amazon GameLift Servers 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 Servers 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。它还可能包含用于您的游戏的其他 AWS 资源，例如 DynamoDB 表、Amazon Redshift 集群和 Lambda 函数。其中许多资源特定于区域，因此您必须在各个区域中创建它们。Amazon GameLift Servers 还需要允许访问这些支持资源的 IAM 角色。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 Servers 堆栈-此堆栈包含部署游戏的每个区域中必须存在的 Amazon GameLift Servers 资源，包括版本或脚本、一组队列和别名。CloudFormation 使用文件在 S3 存储桶位置创建生成或脚本资源，并将构建或脚本部署到一个或多个队列资源。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 Servers 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。

下图说明了用于在多个 AWS 区域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。





更新构建

Amazon GameLift Servers 生成包是不可变的，就像生成包与队列之间的关系。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、队列和别名资源的堆栈时，默认 CloudFormation 行为是按顺序自动执行这些步骤。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，修改 CloudFormation 构建模板以指向新的 S3 位置。使用新 S3 位置更新堆栈时，这将触发以下 CloudFormation 序列：

1. 从 S3 检索新文件、验证文件并创建新 Amazon GameLift Servers 生成包。
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，自动扩缩会处理在每组实例集内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自动扩缩。

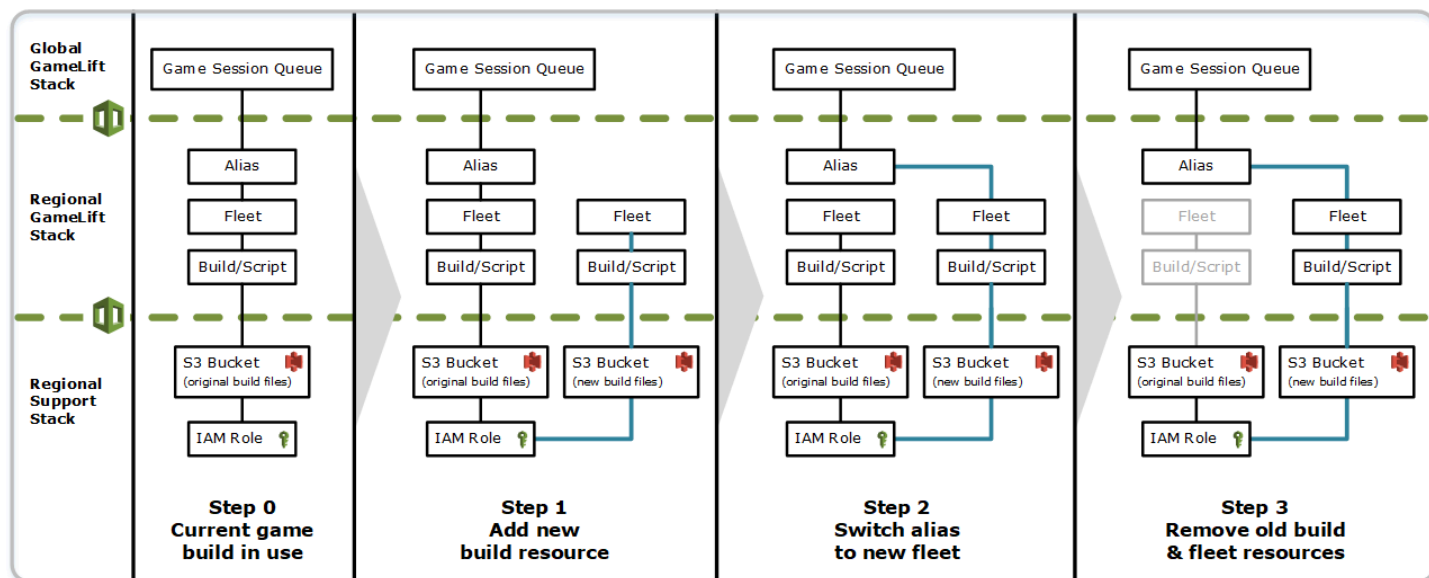
您也可以 CloudFormation 保留资源而不是将其删除。有关更多信息，请参阅《AWS CloudFormation API Reference》中的 [RetainResources](#)。

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 Servers 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。更新模板时，CloudFormation 会创建第二个构建资源和相应的队列。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，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 Servers 生成包和队组，在该队组处于活动状态之后，别名切换回新队组。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](#)。

DDoS 保护带有 Amazon GameLift Servers

Amazon GameLift Servers 玩家网关是一个基于中继的网络，用于在游戏客户端和托管的游戏服务器之间路由 UDP 流量。Amazon GameLift Servers Player Gateway 通过在流量到达游戏服务器之前对其进行验证、限制玩家流量、向公众隐藏游戏服务器 IP 地址以及在中继端点变得不健康时提供更新的端点来提供主动的 DDoS 保护。此功能有助于保护您的游戏服务器免受容量攻击，延迟可以忽略不计。

玩家网关要求游戏服务器在基于 Linux 的 Amazon GameLift Servers 托管 EC2 队列或容器队列上运行。您必须在舰队创建期间启用玩家网关，并更新游戏客户端和游戏后端才能使用此功能。

主要优势

玩家网关具有以下安全性和运营优势：

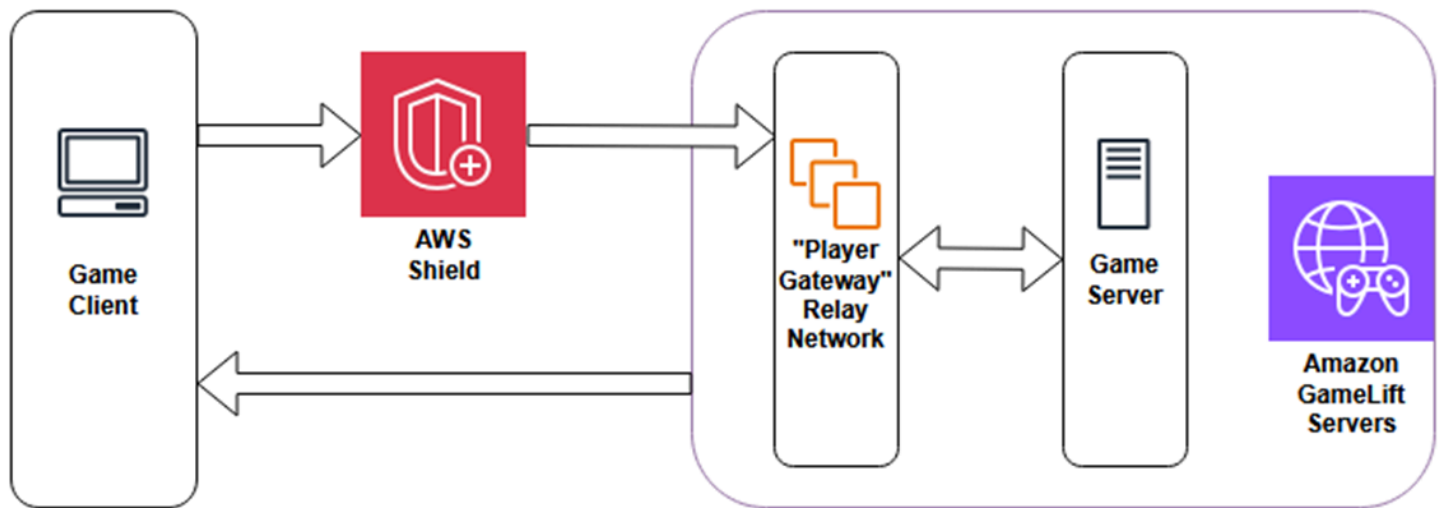
- 隐藏游戏服务器 IP 地址 — 游戏客户端通过中继端点而不是直接连接到游戏服务器，从而向公众隐藏您的游戏服务器地址。
- 验证流量-所有通过玩家网关的流量都需要玩家网关令牌，仅允许来自经过身份验证的玩家的流量到达您的游戏服务器。
- 动态端点替换 — 当中继端点变得不健康时，Amazon GameLift Servers 可以快速替换端点，并在下次调用时提供更新的健康端点以刷新玩家连接详细信息。
- 分配玩家流量-中继端点因玩家而异，从而减少单个不健康的中继端点对同一游戏会话中其他玩家的影响。

玩家网关的工作原理

玩家网关使用中继网络在游戏客户端和游戏服务器之间路由 UDP 流量。这通过在流量到达游戏服务器之前对其进行验证、限制玩家流量、向公众隐藏游戏服务器 IP 地址以及在中继端点变得不健康时提供更新的端点来防御 DDoS 攻击。

流量流

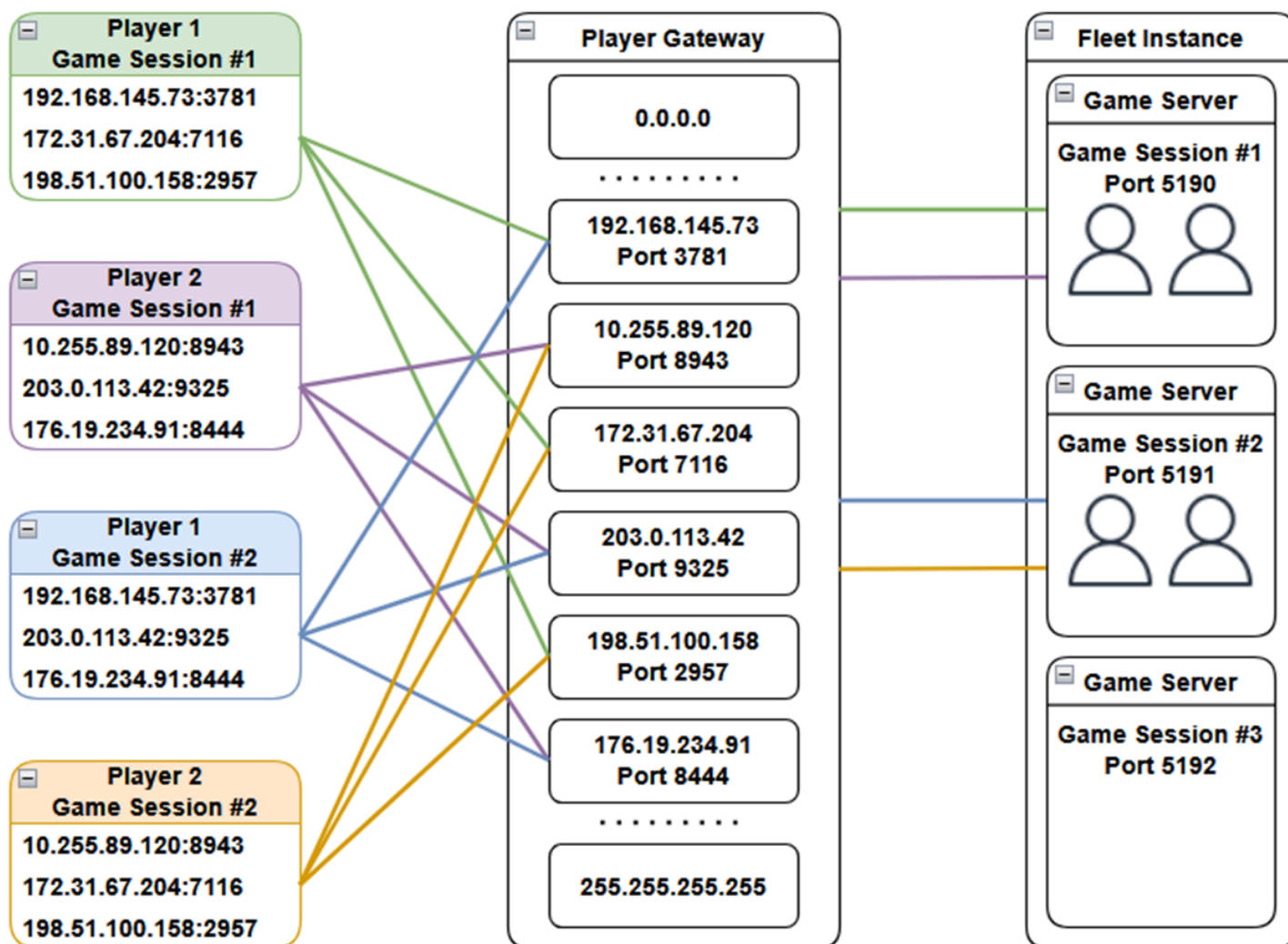
当玩家连接到游戏会话时，您的游戏后端会从 `GetPlayerConnectionDetails` API 中检索中继端点和玩家网关令牌，并将其发送到游戏客户端。游戏客户端将玩家网关令牌预置在 UDP 数据包中，并将数据包发送到中继端点。中继网络验证令牌并将合法流量路由到游戏服务器。在交付之前，中继网络会剥离玩家网关令牌，以便游戏服务器接收原始游戏客户端有效负载，并且不需要更改代码。从游戏服务器返回到游戏客户端的通信通过中继网络返回，无需修改。



核心概念

中继端点

中继端点是游戏客户端用来通过玩家网关发送流量的 IP 地址和端口组合。每个玩家都会收到多个端点，这些端点因玩家而异，以分配流量并减少攻击对同一游戏会话中其他玩家的影响。



玩家网关代币

玩家网关令牌是加密的字节数组，授权玩家通过玩家网关向游戏会话发送流量。GetPlayerConnectionDetailsAPI 以 base64 编码的字符串形式返回令牌。游戏客户端必须在每个 UDP 数据包中预先添加玩家网关令牌。中继网络验证令牌并拒绝带有无效或缺失令牌的数据包。

⚠ Important

玩家网关令牌不得加密，并且必须出现在游戏客户端发送的每个 UDP 数据包的开头。如果您的游戏对有效负载进行加密，请在加密游戏数据后预先添加未修改的玩家网关令牌。

玩家网关代币的有效期至少为 3 分钟。我们建议每 60 秒刷新一次令牌，以确保玩家在中继端点变得不健康时收到更新的端点。

GetPlayerConnectionDetails API

您的游戏后端调用 `GetPlayerConnectionDetails` API 来检索游戏会话中玩家的连接详细信息。API 返回中继端点和玩家网关令牌，或者回退到游戏服务器的 IP 地址和端口进行直接连接。您的游戏客户端应设计为可以处理这两种连接类型。要在中继端点变得不健康时接收更新的端点，请在整个游戏会话中定期调用此 API（建议每 60 秒调用一次）。

有关更多信息，请参阅 [GetPlayerConnectionDetails API](#)。

监控玩家网关

Player gateway CloudWatch 向 Amazon 发布指标，以帮助您监控网络流量模式、识别潜在的 DDoS 攻击并跟踪中继性能。指标包括入/出的数据包和字节、限制的流量以及玩家会话。有关玩家网关指标的完整列表，请参阅 [DDoS 保护 \(玩家网关 \) 指标](#)。

IPv4 和 IPv6 兼容性

游戏客户端使用进行通信 IPv4。玩家网关 IPv6 用于与游戏服务器通信。Amazon GameLift Servers IPv6 根据您的舰队配置自动处理 IPv4 和之间的转换。

有关在舰队上配置玩家网关的更多信息，请参阅 [CreateFleet](#) 或 [CreateContainerFleet](#)。有关 IPv4 和 IPv6 支持的更多信息，请参阅 [IPv4 和 IPv6 兼容性](#)。

在舰队上启用玩家网关

在创建队列时启用玩家网关，通过中继端点路由游戏流量，这些中继端点验证并限制传入游戏服务器的 UDP 流量。

先决条件

在启用玩家网关之前，请验证以下要求：

- 支持的地点-有关支持玩家网关 AWS 的地点列表，请参阅 [Amazon GameLift Servers 服务位置](#)。
- 舰队类型 — 玩家网关支持基于 Linux 的托管 EC2 队列和托管容器队列。
- 服务器 SDK — 玩家网关需要 Amazon GameLift Servers 服务器 SDK 5.0 或更高版本。

玩家网关模式

使用以下 `PlayerGatewayMode` 属性为您的舰队配置玩家网关兼容性：

已禁用（默认）

舰队与玩家网关不兼容。

已启用

Fleet 在支持的地方与玩家网关兼容。不支持该功能的舰队位置仍然可以直接使用游戏服务器 IP 地址添加到舰队中，但不会使用玩家网关。

REQUIRED

舰队与玩家网关兼容。不支持该功能的舰队位置无法添加到舰队中。

IPv4 和 IPv6 兼容性

游戏客户端必须通过 IPv4 与玩家网关通信。玩家网关中继网络使用 IPv6 与游戏服务器通信。托管容器队列会自动将 IPv6 流量转换为 IPv4。对于托管 EC2 队列，请使用以下 `GameServerIpProtocolSupported` 属性配置队列处理 IPv6 流量的方式：`PlayerGatewayConfiguration`

IPv4（默认）

您的游戏服务器仅接受传入 IPv4 流量。将在您的实例上安装和执行轻量级 IP 转换软件，以接收和转换传入的 IPv6 流量为 IPv4。

DUAL_STACK

您的游戏服务器与传入 IPv6 流量兼容。不会在您的实例上安装或执行 IP 转换软件。IPv6 原生支持的游戏服务器的性能可能稍高一些。

将玩家网关集成到游戏中

创建支持玩家网关的舰队后，必须更新游戏客户端和游戏后端才能与玩家网关集成。您的游戏后端会检索中继端点和玩家网关令牌，然后将其提供给游戏客户端。游戏客户端将所有 UDP 流量发送到这些中继端点。

集成遵循以下调用路径：

1. 您的游戏后端调用 `GetPlayerConnectionDetails` API 来检索游戏会话中每位玩家的中继端点和玩家网关令牌。
2. 您的游戏后端将中继端点和玩家网关令牌发送到游戏客户端。
3. 您的游戏客户端将玩家网关令牌预先添加到所有传出的 UDP 数据包中，并将数据包发送到中继端点。
4. 中继网络验证玩家网关令牌并将流量路由到您的游戏服务器。
5. 您的游戏服务器通过相同的中继路径将流量发送回游戏客户端。

后端集成

您的游戏后端必须调用 `GetPlayerConnectionDetails` API 来检索玩家的中继端点和玩家网关令牌。然后，后端将这些信息提供给游戏客户端。

GetPlayerConnectionDetails API

`GetPlayerConnectionDetailsAPI` 会返回游戏会话中玩家的连接详情：

中继端点和令牌

当该位置启用并支持玩家网关时，将返回：

- 中继端点 — 多个中继端点（IP 地址和端口）因玩家而异
- 玩家网关令牌 — 客户端必须在所有 UDP 数据包之前添加的令牌（有效期至少 3 分钟）
- 过期-玩家网关令牌到期时间戳

直接连接

如果该位置未启用或不支持玩家网关，则返回游戏服务器的 IP 地址和端口

您的游戏客户端应设计为可以处理这两种连接类型。

API 调用示例：

```
// C++ example using AWS SDK
Aws::GameLift::GameLiftClient client;
Aws::GameLift::Model::GetPlayerConnectionDetailsRequest request;
request.SetGameSessionId(gameSessionId);
request.SetPlayerIds(playerIds); // Vector of player IDs
```

```
auto outcome = client.GetPlayerConnectionDetails(request);
if (outcome.IsSuccess()) {
    auto result = outcome.GetResult();
    auto connectionDetails = result.GetPlayerConnectionDetails();

    // Process each player's connection details
    for (const auto& detail : connectionDetails) {
        std::string playerId = detail.GetPlayerId();

        // Get relay endpoints (IP address and port)
        auto endpoints = detail.GetEndpoints();
        for (const auto& endpoint : endpoints) {
            std::string ipAddress = endpoint.GetIpAddress();
            int port = endpoint.GetPort();
        }

        // Get player gateway token
        auto token = detail.GetPlayerGatewayToken();

        // Get expiration time
        auto expiration = detail.GetExpiration();

        // Send endpoints and token to game client
    }
}
```

请参阅[GetPlayerConnectionDetails](#) 《Amazon GameLift Servers API 参考》。

刷新中继端点

当中继端点变得不健康时，请GetPlayerConnectionDetails定期致电为玩家提供更新的端点。我们建议每隔 60 秒为游戏会话中的所有玩家调用 API。要优化 API 的使用，请在单个 API 调用 IDs 中批量处理多个玩家。

Important

定期刷新调用是向玩家提供更新的端点的主要机制。虽然玩家网关令牌的有效期至少为 3 分钟，但每 60 秒刷新一次可确保玩家定期收到更新的终端节点。

客户端集成

您的游戏客户端必须在所有传出的 UDP 数据包中预先添加玩家网关令牌，并将数据包发送到提供的中继端点。C++ for C++ 和虚幻引擎的Amazon GameLift Servers客户端 SDK 提供了简化这种集成的实用工具。

客户端要求

要通过玩家网关路由流量，您的游戏客户端必须：

- 预置玩家网关令牌 — 将玩家网关令牌预置到所有传出的 UDP 数据包中。玩家网关令牌不得加密，必须出现在每个数据包的开头。没有有效玩家网关令牌的数据包将被丢弃。
- 发送到中继端点-将所有 UDP 数据包发送到提供的中继端点。
- 保持连接活动 — 确保游戏客户端向游戏服务器发送数据包，或者游戏服务器至少每 30 秒向玩家发送一次数据包。本活动通过中继网络维护连接。
- 处理端点更新-从您的后端接受更新的中继端点和玩家网关令牌（建议每 60 秒一次），并在不中断连接的情况下过渡到新的端点。

客户端 SDK

C++ for C++ 和虚幻引擎的Amazon GameLift Servers客户端 SDK 提供了简化玩家网关集成的实用工具：

- 令牌管理 — 将玩家网关令牌预置到所有传出的 UDP 数据包中。
- 端点选择-使用可配置算法将流量路由到中继端点。
- 端点刷新-计划定期回调，以从您的后端检索更新的中继端点和玩家网关令牌。

端点选择算法

客户端 SDK 提供了两种内置算法，用于选择要使用的中继端点：

后备算法

使用一个端点直到其运行状况不佳，然后切换到另一个端点。最适合可以短暂中断的菜单、大厅和回合制游戏。在端点故障转移期间，数据包可能会在配置的超时时间内丢失（默认值：2 秒）。

预测性旋转算法

持续轮换所有可用端点，并在故障发生之前对其进行预测。最适合实时游戏玩法，例如第一人称射击游戏和赛车游戏，在这些游戏中，持续的数据包传输至关重要。要求游戏服务器以一致的频率发送消息。

您还可以通过扩展 SDK 的基础算法类来实现自定义算法。

客户端 SDK 资源

有关完整的集成说明和示例代码，请参阅以下资源：

C++

适用于 C++ 的客户端 SDK 可在适用于 C++ 的 [Amazon GameLift Servers 客户端 SDK](#) 中找到。

Unreal Engine

虚幻引擎的插件可在虚幻引擎的 [Amazon GameLift Servers 客户端 SDK](#) 中找到。访问权限需要在 Epic Games 组织中的成员资格 GitHub。详情请 [访问虚幻引擎](#)。GitHub

测试您的集成。

在部署到队列之前，请使用玩家网关测试工具在本地测试您的玩家网关集成。该工具模拟玩家网关 UDP 代理基础架构，可帮助您验证游戏客户端是否正确预置了玩家网关令牌、通过多个端点路由流量以及处理网络退化问题。

有关设置和使用说明，请参阅上的 [Player Gateway 测试应用程序](#) GitHub。

最佳实践

集成玩家网关时，请遵循以下最佳实践：

- 每 60 秒刷新一次端点 — `GetPlayerConnectionDetails` 每 60 秒调用一次，以确保玩家定期收到更新的端点。
- Batch API 调用 — 在同一个游戏会话中 `GetPlayerConnectionDetails` 为多个玩家调用时，批量调用以减少 API 开销。
- 保持连接活动-确保流量至少每 30 秒在游戏客户端和游戏服务器之间流动一次。这可以是 client-to-server 或 server-to-client 流量。

- 优雅地处理端点更新 — 当您的游戏客户端从后端收到更新的端点时，在不中断玩家连接的情况下过渡到新的端点。
- 选择正确的算法 — 选择最符合游戏要求的端点选择算法。回合制游戏使用后备算法，实时游戏使用预测轮换算法。

Amazon GameLift Servers 测试和故障排除

本节提供有关诊断和解决您的Amazon GameLift 托管解决方案问题的指导。无论您是在开发、部署还是生产运营过程中遇到问题，这些资源都将帮助您确定根本原因并实施有效的解决方案。

- **开发和测试环境：**设置与生产条件一致的迭代开发工作流程，使您能够在开发周期的早期发现并解决集成问题。
- **实例集部署和管理：**诊断常见的实例集创建失败问题、服务器进程激活问题、扩缩问题以及部署错误，这些问题可能会导致您的游戏服务器无法正常运行。
- **运行时诊断和监控：**访问实时实例集实例以进行实时故障排除、性能分析和配置验证，从而解决影响玩家体验的问题。

将这些资源与 Amazon GameLift 的监控和记录功能结合使用，为您的多人游戏保持最佳性能和可用性。

借助 Amazon GameLift Servers Anywhere 为迭代开发做好准备

Amazon GameLift Servers 提供多种工具和解决方案，协助您设置在游戏开发过程中使用的托管测试环境。利用这些工具，您可以创建一个环境，该环境可以反映玩家使用 Amazon GameLift Servers 进行托管式托管的现实体验，并支持快速的迭代开发流程。

有了单独的测试环境，您就能省去测试期间的 Amazon GameLift Servers 托管式实例集的开销。您不用再上传每个新游戏服务器生成包迭代，为其创建一个新实例集，然后等待 15 分钟以上让它激活。相反，您可以创建一个新生成包，用该新生成包快速更新测试实例集，启动它，然后开始测试。

借助 Amazon GameLift Servers Anywhere 实例集，您可以使用本地设备（如开发工作站）设置测试环境。您也可以使用基于云的托管资源来设置测试环境。

设置 Anywhere 测试环境以开发和测试一系列场景，包括：

- 使用 Amazon GameLift Servers 服务器 SDK 测试游戏服务器集成。即使没有正常运行的游戏客户端，也可以使用 AWS CLI 调用来启动新的游戏会话并跟踪游戏会话事件，从而进行测试。
- 在为游戏开发组件时，测试游戏客户端、后端服务和 Amazon GameLift Servers 服务之间的交互。微调玩家加入游戏的体验。
- 试验您的 FlexMatch 对战构建器设计。试用规则集变体和其他对战功能实施。设置并测试对战回填。

- 试用其他 Amazon GameLift Servers 托管功能，例如用于游戏服务器生命周期管理的运行时配置设置（使用 Amazon GameLift Servers 代理）。
- 在实时托管环境中快速构建、测试并重复验证游戏玩家体验的各个方面，包括多人互动。

稍后，在准备游戏发布时，您需要添加 Amazon GameLift Servers 托管式实例集以微调您的托管配置和测试其他场景，这包括以下场景：

- 试验和测试游戏会话队列设计，包括使用多位置实例集、竞价型实例集和按需型实例集以及多种实例类型。
- 试用使用托管式实例集的游戏会话放置选项，包括使用可选的延迟策略和实例集优先级设置。
- 使用自动或手动扩展选项配置容量扩展以满足玩家需求。
- 设置 AWS CloudFormation Amazon GameLift Servers 托管车队以长期管理您的托管资源。

Fast Build Update Tool（仅用于开发）

对于托管 EC2 舰队，要部署游戏服务器版本更新，您需要将每个新版本上传到该 Amazon GameLift Servers 队列并为其创建新的舰队。

Fast Build Update Tool 可让您在开发过程中绕过这些步骤，从而节省时间并加快开发迭代速度。利用此工具，您可以在现有实例集的所有计算中快速更新游戏生成包文件。此工具有多个选项；您可以替换整个游戏生成包或更改 6 个特定文件，也可以管理更新后重新启动游戏服务器进程的方式。您还可以使用它来更新实例集中的单个计算。

要获取 Fast Build Update Tool 并详细了解其使用方法，请访问 Github 中的 Amazon GameLift Servers Toolkit 存储库，查看关于 [Fast Build Update Tool](#) 的页面。

主题

- [构建基于云的测试环境](#)
- [使用 Amazon GameLift Servers Anywhere 设置本地测试](#)
- [使用 Amazon GameLift Servers Local 测试您的集成。](#)

构建基于云的测试环境

Note

本主题涵盖针对已集成 Amazon GameLift Servers 服务器 SDK 5.x 的游戏所进行的迭代测试。如果您的游戏使用的是服务器 SDK 版本 4.x 或更早版本，请参阅[使用 Amazon GameLift Servers Local 测试您的集成](#)。

使用 Amazon GameLift Servers Anywhere 实例集在基于云的托管环境中迭代构建和测试游戏组件。创建具有托管资源并连接到 Amazon GameLift Servers 服务的 Anywhere 实例集，在其上运行游戏服务器，并根据需要测试游戏功能。

使用 Amazon GameLift Servers 代理部署 Anywhere 实例集

如果您的游戏服务器生成包与 Amazon GameLift Servers SDK 5.x 或更高版本集成，您可以使用 Amazon GameLift Servers 代理将其部署到基于云的 Anywhere 实例集。该代理是一个后台进程，用于管理游戏服务器生命周期以及实例集中每个计算上的其他任务。这些任务包括向 Anywhere 队列注册计算、获取身份验证令牌以及基于一组指令的 starting/stopping 游戏服务器进程。该代理由实例集的运行配置控制，在实例集生命周期内，您可以随时更新这些配置。（该代理会自动部署到托管式 EC2 实例集。）要了解更多信息并下载代理，请参阅[Amazon GameLift Servers GitHub 存储库](#)。

使用 Amazon EC2 设置迭代测试

使用此 [Amazon GameLift Servers Toolkit 解决方案](#) 中的引导式工作流程来设置基于云的托管环境，以反映使用 Amazon GameLift Servers 的托管式托管体验。

GitHub 存储库提供了一组脚本，这些脚本可以自动执行使用 Amazon GameLift Servers Anywhere 和 A Amazon GameLift Servers gent 设置测试环境的大部分过程。它还针对在有新游戏服务器生成包要测试时更新环境提供指导。您可以运行一个脚本来部署具有示例游戏服务器生成包的测试环境，也可以使用您自己的游戏服务器生成包逐步设置它。

在此工作流程中，您将完全使用 AWS CloudShell 来运行脚本和完成命令行任务。AWS 管理控制台

Note

对于本教程中的任务，您需要具有以下服务权限的 AWS 账户用户：Amazon GameLift Servers、Amazon S3 AWS CloudShell、AWS Systems Manager、Amazon EC2 和 AWS

Identity and Access Management。对 AWS 账户拥有管理员级别访问权限的用户已经拥有所需的权限。

此工作流程包含以下任务：

- 为 Amazon GameLift Servers 打包游戏服务器生成包。此工作流程提供了一个用于构建示例 C++ 游戏服务器的脚本，该游戏服务器已与 Amazon GameLift Servers 服务器 SDK 5.x 集成，随时可以进行托管。如果您已完成集成，也可以使用自己的游戏项目。
- 设置 Amazon Simple Storage Service 存储桶以存储游戏服务器生成包和依赖项。当您生成新版本的游戏生成包时，您可以将其存储在 S3 中，并使用脚本更新 Anywhere 实例集以进行游戏测试。
- 获取并构建 Amazon GameLift Servers 代理。该代理将根据您的配置来管理托管资源上的游戏服务器进程。它使用的逻辑和行为与 Amazon GameLift Servers 托管式 EC2 托管相同。
- 为托管资源设置 Anywhere 实例集。有了 Anywhere 实例集，您就可以将 Amazon GameLift Servers 服务用于不受 Amazon GameLift Servers 管理的托管资源。在此步骤中，您还将配置运行时配置，该配置用于指示 Amazon GameLift Servers 代理何时以及如何启动游戏服务器进程。
- 设置 Amazon EC2 实例。这是用于迭代测试的测试环境。使用标准 EC2 实例比使用完全托管式 Amazon GameLift Servers 实例（针对生产级使用进行了优化）速度要快得多。使用标准 EC2 实例时，您可以根据需要快速且持续地更新游戏服务器。
- 将游戏服务器生成包和 Amazon GameLift Servers 代理部署到 Amazon EC2 实例。此工作流程提供了一个脚本，用于获取最新版本的游戏生成包和所有依赖项，并将其安装到您的 EC2 实例上。在此工作流程中，依赖关系包括 Amazon GameLift Servers 代理和 CloudWatch 代理。
- 启动 Amazon GameLift Servers 代理。安装后，该代理会自动启动并开始执行指令。这些指令包括：
 - 将 EC2 实例注册为 Amazon GameLift Servers Anywhere 实例集内的计算。
 - 与 Amazon GameLift Servers 服务建立 WebSocket 连接并获取最新的运行时配置。
 - 根据运行时配置中的指令启动游戏服务器进程。在此工作流程中，该代理会按照指令启动游戏服务器可执行文件的单个进程。
- 测试游戏场景。设置好测试环境并安装好最新的游戏服务器生成包后，就可以开始测试了。此工作流程将引导您完成多个测试步骤，包括启动游戏会话。访问 CloudWatch 游戏服务器日志以跟踪游戏会话启动并准备接受玩家时的进度。

在开发游戏组件（包括游戏客户端和客户端后端服务）时，可以将其包含在测试场景中。使用游戏客户端请求游戏会话，从 Amazon GameLift Servers 服务中检索连接信息，然后直接连接到游戏会话。

- 部署新的游戏服务器生成包并重复测试。在开发游戏时，您可以生成新的游戏服务器生成包，然后快速将其部署到 EC2 测试环境进行测试。将它们上传到 Amazon S3 存储桶，然后使用工作流程脚本更新测试环境。

将您的游戏过渡到 Amazon GameLift Servers 托管式实例集

在完成开发测试并准备好发布后，就可以切换到 Amazon GameLift Servers 托管式实例集了。使用托管式实例集来微调和测试游戏托管资源。实施游戏会话放置解决方案（队列和对战构建器），选择最佳托管硬件（包括竞价型实例集）和位置，然后选择容量扩展策略。您可能还想开始使用 AWS CloudFormation 来更有效地管理所有游戏托管资源的生命周期，包括舰队、队列和媒人。

从基于云的 Anywhere 测试实例集过渡到 Amazon GameLift Servers 托管式实例集的工作量很少。您无需更改任何游戏代码，而且可以重复使用相同的队列和对战构建器。请执行以下任务：

- 创建 Amazon GameLift Servers 生成包资源。使用 Anywhere 测试实例集时，您需要手动将游戏服务器生成包和依赖项部署到每个实例集计算。使用托管式实例集时，请将您的游戏生成包上传到 Amazon GameLift Servers，后者会自动将生成包部署到所有实例集计算。请参阅[为 Amazon GameLift Servers 创建游戏服务器生成包](#)，详细了解如何打包游戏生成包文件以及在 Amazon S3 存储桶中创建包含文件的生成包资源。
- 创建托管式实例集。使用控制台或 AWS CLI 创建一个实例集，并指定 EC2 托管式实例集。这种类型的实例集需要额外的配置设置，包括指定生成包资源和实例类型。您可以使用相同的运行时配置来管理每个实例集计算上的游戏服务器生命周期。请参阅[创建 Amazon GameLift Servers 托管 EC2 舰队](#)，详细了解如何创建托管式实例集。
- 重定向实例集别名（可选）。如果您设置了要用于 Anywhere 实例集的别名，则可以对托管式实例集使用相同的别名。请参阅[创建 Amazon GameLift Servers 别名](#)，详细了解如何创建或更新别名。

使用 Amazon GameLift Servers Anywhere 设置本地测试

Note

本主题涵盖针对已集成 Amazon GameLift Servers SDK 5.x 版本的游戏所进行的本地测试。如果您的游戏使用的是服务器 SDK 版本 4.x 或更早版本，请参阅[使用 Amazon GameLift Servers Local 测试您的集成](#)。

使用 Amazon GameLift Servers Anywhere 实例集和您自己的硬件在模拟托管环境中迭代构建和测试您的游戏组件。设置 Anywhere 实例集并注册本地设备，以与 Amazon GameLift Servers 服务建立连

接。将游戏服务器生成包安装到设备上，启动游戏服务器进程，并根据需要测试游戏功能。您可以按照所需的频率更新游戏服务器生成包，以测试每个新生成包迭代。

使用 Anywhere 队列，您可以使用 AWS CLI 或测试脚本进行测试。如果您已将游戏客户端与 Amazon GameLift Servers 集成，则可以在同一台本地设备或其他设备上运行该客户端。

使用 Anywhere 实例集在本地进行测试对于测试游戏服务器与 Amazon GameLift Servers 的集成特别有用。您可以全面了解本地计算机上的所有托管活动以及事件和日志记录数据。

Note

您使用的是适用于 Unreal Engine 或 Unity 的 Amazon GameLift Servers 插件吗？这些工具包含使用 Anywhere 实例集设置本地测试的引导式工作流程。请按照文档 [适用于 Unity 的插件：使用 Amazon GameLift Servers Anywhere 设置本地测试](#) 或 [适用于 Unreal 的插件：使用 Amazon GameLift Servers Anywhere 在本地托管游戏](#) 进行操作。

主题

- [设置本地 Anywhere 实例集](#)
- [更新并安装游戏服务器](#)
- [测试游戏会话活动](#)
- [在游戏服务器上迭代](#)
- [将您的游戏过渡到 Amazon GameLift Servers 托管式实例集](#)

设置本地 Anywhere 实例集

按照以下步骤为您的本地工作站创建 Anywhere 实例集。有关使用 AWS CLI 或 for 的详细说明 Amazon GameLift Servers，AWS 管理控制台 请参阅[创建 Amazon GameLift Servers Anywhere 实例集](#)。

创建 Anywhere 实例集

1. 为本地工作站创建自定义位置。（AWS CLI 或控制台）。自定义位置只是您计划包含在 Anywhere 实例集内的计算资源的一个标签。自定义位置名称必须以 custom- 开头。例如：custom-my_laptop。请参阅[创建自定义位置](#)。
2. 创建 Anywhere 队列（AWS CLI 或控制台）。在此步骤中，请使用本地工作站的自定义位置创建实例集资源。请参阅[创建 Anywhere 实例集](#)。

记下新实例集的 ID 或 ARN 值。在下一个步骤中，您需要用到该值。

3. 将您的本地工作站注册为队列计算（仅限AWS CLI）。Anywhere 实例集必须至少有一个计算资源来托管您的游戏服务器。请参阅[向实例集添加计算](#)。要向实例集添加计算，您需要以下信息：
 - 计算名称。实例集中的每个计算都必须有一个唯一名称。
 - Anywhere 实例集标识符。您可以使用 FleetID 或 FleetArn。
 - 计算的连接信息。指定 IPAddress 或 DnsName。这就是 Amazon GameLift Servers 和游戏客户端连接到游戏服务器的方式。
 - Anywhere 实例集内的自定义位置。

记下 GameLiftServiceSdkEndpoint 返回值。当您更新游戏服务器以在 Anywhere 实例集上运行时，您将需要此值。

更新并安装游戏服务器

此任务假定您已将游戏服务器生成包与 Amazon GameLift Servers 服务器 SDK 5.x 集成。集成过程包括向游戏服务器添加代码，使其能够与 Amazon GameLift Servers 服务交互以启动和管理游戏会话。

对于 Anywhere 实例集，您需要手动配置某些游戏服务器设置。在 Amazon GameLift Servers 托管式实例集内，这些设置会自动配置。

为 Anywhere 实例集准备游戏服务器

1. 获取身份验证令牌。您的游戏服务器在与 Amazon GameLift Servers 服务进行每次通信时，都必须包含身份验证令牌。Amazon GameLift Servers 身份验证令牌有效期较短，必须定期刷新。

作为最佳实践，请创建脚本来完成以下任务：

- 调用 C AWS CLI 操作 `get-compute-auth-token`。
- 将返回的令牌值存储在游戏服务器进程可以检索到的地方，例如本地计算上的环境变量中。

将脚本与游戏服务器一起安装在计算上。将脚本设置为在启动第一个游戏服务器进程之前运行。当游戏服务器进程处于活动状态时，请定期运行脚本以保持身份验证令牌有效。计算上的所有游戏服务器进程都可以使用相同的身份验证令牌。

2. 更新 Amazon GameLift Servers 游戏服务器代码。当您将游戏服务器代码与适用于 Amazon GameLift Servers 的服务器 SDK 集成时，您添加了对 `InitSdk()` 操作的调用。当游戏服务器在

Anywhere 实例集上运行时，此调用需要额外的服务器参数。有关更多信息，请参阅[初始化服务器进程](#)和适用于您开发语言的 [Amazon GameLift Servers 服务器 SDK 5.x](#)。服务器参数是：

- `websocketUrl` – 将此参数设置为向实例集注册计算时返回的 `GameLiftServiceSdkEndpoint` 值。
- `hostId` – 将此参数设置为向 Anywhere 实例集注册计算时指定的计算名称。
- `fleetId` – 将此参数设置为 Anywhere 实例集的 ID。
- `authToken` – 将此参数设置为响应检索计算的身份验证令牌的请求而返回的令牌。
- `processId` – 设置此参数以标识在本地计算上运行的游戏服务器进程。每个并发的游戏服务器进程都必须有一个唯一的进程 ID。

每个游戏服务器进程使用的服务器参数值都必须特定于运行该进程的 Anywhere 实例集计算。有关如何为计算获取适当值的详细信息，请参阅[向实例集添加计算](#)。作为最佳实践，请将 `websocketUrl`、`hostId`、`fleetId` 和 `authToken` 设置为本地计算上的环境变量。在该计算上运行的所有服务器进程都将使用这些值。

3. 在本地计算上安装游戏服务器生成包。加入运行游戏服务器所需的所有依赖项。
4. 启动在本地计算上运行的一个或多个游戏服务器进程。当游戏服务器进程调用服务器 SDK 操作 `ProcessReady()` 时，则说明该进程已准备好托管游戏会话。

测试游戏会话活动

通过使用游戏会话来测试游戏服务器集成。如果您的游戏客户端没有与 Amazon GameLift Servers 功能集成，则可以使用 AWS CLI 启动游戏会话。请尝试以下场景：

- 创建游戏会话。调用 [create-game-session](#) 命令（或 [CreateGameSession](#) API 操作）。指定 Anywhere 实例集的 ID 和自定义位置。此调用将返回新游戏会话的唯一标识符。
- 检查游戏会话状态。调用 [describe-game-sessions](#) 命令（或 [DescribeGameSessions](#) API 操作）。指定游戏会话 ID。此调用将返回详细的游戏会话信息，包括游戏会话状态。处于活动状态的游戏会话已准备就绪，可供玩家连接。要获取舰队所有游戏会话的列表，请调用 [list-game-sessions](#) command（或 [ListGameSessions](#) API 操作）。
- 连接到游戏会话。如果您的游戏客户端可以加入游戏会话，请使用游戏会话信息中包含的连接信息。

在游戏服务器上迭代

您可以使用相同的 Anywhere 实例集和计算来测试其他版本的游戏服务器生成包。

1. 清理现有的 **GameSession**。如果游戏服务器进程崩溃或无法调用 `ProcessEnding()`，Amazon GameLift Servers 会在游戏服务器停止发送运行状况检查后清理 **GameSession**。
2. 生成新的游戏服务器生成包。对游戏服务器进行更改并打包修改后的生成包。
3. 更新本地计算上的游戏服务器生成包。您之前的 Anywhere 实例集仍处于活动状态，您的笔记本电脑仍注册为实例集内的计算资源。
4. 获取更新后的授权令牌。调用 [get-compute-auth-token](#) CLI 命令并将令牌存储在本地计算机上。
5. 启动在本地计算上运行的一个或多个游戏服务器进程。当游戏服务器进程调用 `ProcessReady()` 时，它就可以用于测试了。

将您的游戏过渡到 Amazon GameLift Servers 托管式实例集

在完成开发测试并准备好发布后，就可以切换到 Amazon GameLift Servers 托管式实例集了。使用托管式实例集来微调和测试游戏托管资源。实施游戏会话放置解决方案（队列和对战构建器），选择最佳托管硬件（包括竞价型实例集）和位置，然后选择容量扩展策略。您可能还想开始使用 AWS CloudFormation 来更有效地管理所有游戏托管资源的生命周期，包括舰队、队列和媒人。

要从本地 Anywhere 测试实例集过渡到 Amazon GameLift Servers 托管式实例集，您需要做一些细微修改。您可以重复使用相同的队列和对战构建器。请执行以下任务：

- 将游戏服务器代码调用更改为 **InitSdk()**。删除服务器参数。对于托管式实例集，Amazon GameLift Servers 会自动跟踪此信息。
- 创建 Amazon GameLift Servers 生成包资源。使用 Anywhere 测试实例集时，您需要手动将游戏服务器生成包和依赖项部署到每个实例集计算。使用托管式实例集时，您需要创建游戏生成包并把它上传到 Amazon GameLift Servers，后者会自动将生成包部署到所有实例集计算。请参阅[Amazon GameLift Servers 创建游戏服务器生成包](#)，详细了解如何打包游戏生成包文件以及在 Amazon S3 存储桶中创建包含文件的生成包资源。请勿包含注册计算并获取身份验证令牌的脚本，因为 Amazon GameLift Servers 会通过托管式实例集自动处理这些任务。
- 创建托管式实例集。使用控制台或 AWS CLI 创建队列，指定 EC2 托管队列。这种类型的实例集需要额外的配置设置，包括指定生成包资源和实例类型。您还需要设置运行时配置来管理每个实例集计算上的游戏服务器生命周期。请参阅[创建 Amazon GameLift Servers 托管 EC2 舰队](#)，详细了解如何创建托管式实例集。
- 重定向实例集别名（可选）。如果您设置了要用于 Anywhere 实例集的别名，则可以对托管式实例集使用相同的别名。请参阅[创建 Amazon GameLift Servers 别名](#)，详细了解如何创建或更新别名。

使用 Amazon GameLift Servers Local 测试您的集成。

Note

本主题介绍仅与 Amazon GameLift Servers 服务器 SDK 版本 3.x 和 4.x 的 SDK 集成的游戏测试。您的服务器 SDK 软件包中包含兼容版本的 Amazon GameLift Servers Local。如果您使用的是服务器 SDK 版本 5.x，请参阅[使用 GameLift Server SDK 5.x 进行本地测试](#)以了解如何通过 Amazon GameLift Servers Anywhere 实例集进行本地测试。有关从 SDK 4.x 迁移到 5.x 的指南，请参阅[迁移到 GameLift Server SDK 5.x](#)。

使用 Amazon GameLift Servers Local 可以在本地设备上运行托管 Amazon GameLift Servers 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。替代方案是将每个新生成包上传到 Amazon GameLift Servers 并配置实例集来托管游戏，每次都需要数分钟或更长时间。

通过 Amazon GameLift Servers 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 Amazon GameLift Servers 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 Amazon GameLift Servers 的 AWS SDK 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。

Amazon GameLift Servers Local 是一个命令行工具，可启动托管 Amazon GameLift Servers 服务的独立版本。Amazon GameLift ServersLocal 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 ServersLocal 可识别适用于 Amazon GameLift Servers 的 AWS SDK 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 Amazon GameLift Servers 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 Servers Local 的可执行文件。游戏会话完成后，您应该调用 `GameLiftServerSDK::ProcessEnding` 然后退出该进程。使用 Amazon GameLift Servers 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 Servers 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 Servers Local 支持以下 API：

- `CreateGameSession()`

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

设置 Amazon GameLift Servers Local

Amazon GameLift Servers Local 作为与 .jar 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 Amazon GameLift Servers 支持的语言。对于 SDK 版本 4.0.2，您可以从 [官方 GitHub 发布页面](#) 下载。GameLiftLocal.jar 文件包含在 GameLift-CSharp-ServerSDK-4.0.2.zip 构件中。

在运行 Local 之前，您还必须已安装以下各项。

- 适用于 Amazon GameLift Servers 版本 3.1.5 至 4.x 的服务器 SDK 生成包版本。
- 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 AWS CLI 来模拟游戏客户端对 Amazon GameLift Servers 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 Amazon GameLift Servers 服务器 SDK。
- 在启动过程中，游戏服务器通知 Amazon GameLift Servers，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 Amazon GameLift Servers。
- 游戏服务器响应请求，启动新游戏会话。

1. 启动 Amazon GameLift Servers 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 Amazon GameLift Servers 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 Amazon GameLift Servers Local 服务。这意味着游戏服务器已成功初始化适用于 Amazon GameLift Servers 的服务器 SDK (使用 `InitSDK()`)。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，Amazon GameLift Servers 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 Servers Web 服务终端节点。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 Servers 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: gameId:
```

```
"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 Servers 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 Amazon GameLift Servers 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 AWS Local 服务发出 Amazon GameLift Servers 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 Servers 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 Servers 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 Servers 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[设置 Amazon GameLift Servers API](#) 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 GameLiftClient，并且已成功与 Amazon GameLift Servers 服务通信。

4. 测试游戏客户端对 Amazon GameLift Servers 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 Servers 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 Servers 服务。

要让 Amazon GameLift Servers 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 Servers。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 Servers Local 日志消息应显示游戏服务器调用了 `RemovePlayerSession()`。对 AWS CLI 的 `DescribePlayerSessions()` 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 Servers 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()`。此方法在 Amazon GameLift Servers Server SDK v4.0.1 中已弃用。请参阅[结束游戏会话](#)。

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

Local 的变化

使用 Amazon GameLift Servers Local 时，请记住以下内容：

- 与 Amazon GameLift Servers 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
```

调试 Amazon GameLift Servers 队组问题

本主题提供有关如何解决 Amazon GameLift Servers 托管车 EC2 队问题的指导。

实例集创建问题

当您创建托管 EC2 队列时，该 Amazon GameLift Servers 服务会启动创建队列的工作流程，在安装了游戏服务器版本的情况下部署 EC2 实例，并在每个实例上启动游戏服务器进程。有关详细说明，请参阅。实例集在进入活动状态之前无法托管游戏会话和玩家。

您可以通过确定出现问题的实例集创建阶段并查看实例集创建事件和日志，来调试阻止实例集激活的问题。如果日志未提供有用信息，则问题可能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，请尝试重新上传游戏生成包，以解决可能的文件损坏问题。您还可以联系 Amazon GameLift Servers 支持或在论坛上发布问题。

下载和验证生成包

在此阶段，Amazon GameLift Servers 会获取您上传的游戏服务器生成包，提取文件并运行所有安装脚本。如果实例集创建在这些阶段失败，请查看实例集事件和日志以查明问题。可能的原因包括：

- Amazon GameLift Servers 无法获取压缩的生成包文件 (事件 `FLEET_BINARY_DOWNLOAD_FAILED`)。验证版本的存储位置是否可以访问，是否正在创建与版本相同的 AWS 区域 队列，以及是否 Amazon GameLift Servers 具有访问该队列的正确权限。
- Amazon GameLift Servers 无法提取生成包文件 (事件 `FLEET_CREATION_EXTRACTING_BUILD`)。
- 生成包文件中的安装脚本未能成功完成 (事件 `FLEET_CREATION_FAILED_INSTALLER`)。

构建实例集资源

此阶段的问题通常涉及实例集资源的分配和部署。可能的原因包括：

- 请求的实例类型不可用。
- 请求的实例集类型（竞价型或按需型）不可用。

激活游戏服务器进程

在此阶段，Amazon GameLift Servers 会尝试执行大量任务和测试关键元素，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 SDK 与 Amazon GameLift Servers 服务连接的能力。

Note

在此阶段，您可以远程访问实例集实例以进一步调查问题。请参阅[远程连接到 Amazon GameLift Servers 实例集实例](#)。

可能的问题包括：

- 服务器进程未开始运行。这表明实例集的运行时配置设置有问题（事件 `FLEET_VALIDATION_LAUNCH_PATH_NOT_FOUND` 或 `FLEET_VALIDATION_EXECUTABLE_RUNTIME_FAILURE`）。请验证您是否正确设置了启动路径和可选启动参数。
- 服务器进程开始运行，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与 Amazon GameLift Servers 服务通信。请验证您的游戏服务器是否正在进行以下正确的服务器 SDK 调用（请参阅[初始化服务器进程](#)）：
 - 服务器进程无法初始化（事件 `SERVER_PROCESS_SDK_INITIALIZATION_TIMEOUT`）。服务器进程调用 `InitSdk()` 失败。
 - 服务器进程在准备好托管游戏会话时无法通知 Amazon GameLift Servers（事件 `SERVER_PROCESS_PROCESS_READY_TIMEOUT`）。服务器进程已初始化，但没有及时调用 `ProcessReady()`。
- 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。您可以通过调用[describe-vpc-peering-connections](#)用来跟踪对等互连请求的成功或失败。请务必检查是否存在有效的 VPC 对等授权（[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 Servers 控制台（编辑队列的容量分配设置）或通过调用 AWS CLI 命令来更改队列的运行时配置[update-runtime-configuration](#)。

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。请务必关闭自动扩缩，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 对等连接的队列（请参阅[为 Amazon GameLift Servers 设置 VPC 对等连接](#)）。之所以出现这种情况，是因为删除队列的过程还包括删除队列的 VPC 和所有 VPC 对等连接。您必须首先通过调用 Amazon GameLift Servers [CreateVpcPeeringAuthorization\(\)](#) 的服务 API 或使用 AWS CLI 命令来获得授权 `create-vpc-peering-authorization`。获得授权之后，您就可以删除该实例集。

远程连接到 Amazon GameLift Servers 实例集实例

您可以连接到活动的 Amazon GameLift Servers 托管式 EC2 或容器实例集中的任何实例。远程访问实例的常见原因包括：

- 排查游戏服务器集成问题。
- 微调运行时配置和其他特定于实例集的设置。
- 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。

连接到实例时，请考虑以下潜在问题：

- 您可以连接到活动实例集中的任何实例。通常，您无法连接到非活动实例集，例如正在激活或处于错误状态的实例集。（这些实例集的可用性可能会在短时间内受到限制。）有关实例集激活问题的帮助，请参阅[调试 Amazon GameLift Servers 队组问题](#)。
- 连接到一个活动实例不会影响该实例的托管活动。该实例会继续基于运行时配置启动和停止服务器进程。它会激活并运行游戏会话。该实例可能会因缩减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实例的活动游戏会话和连接的玩家。

通过控制台进行远程访问

您可以通过 Amazon GameLift Servers 控制台使用 Amazon EC2 Systems Manager (SSM) 直接连接到实例集实例。此方法无需额外设置或凭证管理即可提供安全访问。

1. 在 Amazon GameLift Servers 控制台中，从导航窗格中选择托管式 EC2 或托管容器，然后选择实例集。
2. 选择包含您要访问的实例的实例集 ID。
3. 在实例集详细信息页面上，选择实例选项卡以查看实例集的所有计算实例。
4. 选择要连接到的实例，然后选择连接。此操作将弹出“连接到实例”对话框，该对话框会显示连接详细信息，同时允许您查看用于连接到实例的脚本。再次选择连接进行确认。
5. 在连接对话框中，选择运行以创建新的 SSM 会话。系统通过 AWS 密钥管理服务 (AWS KMS) 对您的会话进行身份验证，并在浏览器中打开终端。

Note

基于控制台的远程访问适用于运行服务器 SDK 5.x 版本的实例集。对于运行较早 SDK 版本的舰队，请使用下一节中描述的 AWS CLI 方法。

使用 AWS CLI 进行远程访问

以下说明介绍了如何使用 AWS 命令行界面 (CLI) 远程连接到实例。您还可以使用 AWS SDK 进行编程调用，如[服务 API 参考](#)中所述 Amazon GameLift Servers。

收集实例数据

要连接到 Amazon GameLift Servers 托管式 EC2 实例集实例，您需要以下信息：

- 要连接到的实例的 ID。您可以使用实例 ID 或 ARN。
- 实例上使用的 Amazon GameLift Servers 服务器 SDK 版本。服务器 SDK 与实例上运行的游戏生成包集成。

以下说明描述了如何使用 AWS CLI 完成这些任务。您必须知道要连接到的实例的实例集 ID。

1. 获取计算名称。获取实例集中所有活动计算的列表。使用实例集 ID 或 ARN 调用 [list-compute](#)。对于单位置实例集，请仅指定实例集标识符。对于多位置实例集，请指定实例集标识符和位置。对于托管式 EC2 实例集，list-compute 将返回实例集实例的列表，属性 ComputeName 为实例 ID。查找要访问的计算。

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_2023",  
      "Type": "c4.large"  
    }  
  ]  
}
```

```
}
```

2. 查找服务器 SDK 版本。要获取此信息，您需要查找部署到实例集的生成包。服务器 SDK 版本是一个生成包属性。
 - a. [describe-fleet-attributes](#) 使用舰队 ID 或 ARN 致电以获取舰队的构建 ID 和 ARN。
 - b. 使用生成包 ID 或 ARN 调用 [describe-build](#)，以获取生成包的服务器 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_2023",  
  "ServerSdkVersion": "5.1.1",
```

```
. . .  
}
```

连接到实例 (服务器 SDK 5)

如果要连接到的实例正在运行使用服务器 SDK 版本 5.x 的游戏生成包，请使用 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。

开始之前：

完成 SSM 设置步骤并在本地计算机上安装 SSM 插件。有关更多信息，请参阅 Amazon EC2 Systems Manager 用户指南中的[设置 SSM](#) 和为 AWS CLI 安装会话管理器[插件](#)。

1. 请求实例的访问凭证。[get-compute-access](#)使用您要连接的实例的队列 ID 和计算名称进行调用。Amazon GameLift Servers返回一组用于访问实例的临时证书。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（可选）。您可以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. 连接到实例集实例。使用要连接到的实例启动 SSM 会话。包括实例的 AWS 地区或位置。有关更多信息，包括如何设置 SSM 和 SSM 插件，请参阅 Amazon EC2 Systems Manager 用户 [指南中的启动会话 \(AWS CLI\)](#)。

start-session 请求将自动使用您在步骤 1 中获得的凭证。

```
aws ssm start-session \
--target i-11111111a222b333c \
--region us-west-2 \
```

Note

如果您遇到拒绝访问错误，则可能将 AWS_PROFILE 环境变量设置为 AWS 配置文件，这会导致 AWS CLI 使用错误的凭据进行远程访问。要解决此问题，请暂时取消设置您的 AWS_PROFILE 环境变量。或者，您可以为远程访问凭证创建自定义 AWS 配置文件，并将 --profile 命令行参数添加到 start-session 请求中。

连接到实例（服务器 SDK 4.x 或更早版本）

如果要连接到的实例正在运行使用服务器 SDK 版本 4 或更早版本的游戏生成包，请按照以下说明进行操作。您可以连接到在 Windows 或 Linux 上运行的实例。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 GameLift Servers 将返回实例的操作系统、IP 地址和一组凭证（用户名和私有密钥）。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 Servers 发布 RSA 私钥并将其作为单个字符串返回，同时使用换行符 (\n) 表示换行符。要使私有密钥可用，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-----
nEXAMPLEKEYKCAQEAY7WZhaDsR1W3mR1QtvhwyORRX8gnxgDAfRt/gx42kWXsT4rXE/b5CpSgie/
\nvBoU7jLxx92pNHoFnByP+Dc21eyyz6CvjTmWA0JwfWiW5/akH7i05dSrvC7dQkW2duV5QuUdE0QW
\nZ/aNxMniGQE6XAgfwlnXVBwrerrQo+ZWQeqiUwwMkuEbLeJFLhMCvYURpUMSC1oehm449i1x9X1F
\nG50TCFe0zf18dqqCP6GzbPaIjiU19xX/az0R9V+tpU0zEL+wmXnZt3/nHPQ5xvD20JH67km6SuPW
\noPzev/D8V+x4+bHthfSjR9Y7DvQFjFBVwHXigBdtZcU2/wei8D/HYwIDAQABaoIBAGZ1kaEvnrrqu
\n/uler7vgIn5m71N5LKw4hJLAIW6tUT/fzvtcHK0SkbQCQXuriHmQ2MqyJX/0kn2NfjLV/
ufGxbL1\nmb5qwmGUnEpJaZD6QSSs3kICLwUyUiGfc0uiSbmJoap/
GTLU0W5Mfcv36PaBUNy5p53V6G7hXb2\nbahyWyJNfjLe4M86yd2YK3V2CmK+X/
B0sShnJ36+hjrXPPWmV3N9zEmCdJjA+K15DYmhm/
tJWSD9\n81oGk9TopEp7CkIfatEATyyZiVqoRq6k64iuM9JkA30zdXzMqexXVJ1TLZVEH0E7bh1Y9d801ozR
\noQs/FiZNAx2iijCWyv01pjE73+kCgYEA9mZtyhkHkFDpwrSM1APaL8oNAbbjwEy7Z5Mqfq1
+lIp1\nYkriL0DbLXlvRAH+yHPRit2hH0jtUNZh4Axv+cpq09qbUI3+43eEy24B7G/Uh
+GTfbjsXs0xQx/x\np9otyVwc7hsQ5TA5PZb
+mvkJ50BEKzet9XcKw0NBVELGhnEPe7cCgYEA06Vgov6YHleHui9kHuws
\nayav0elc5zkxjF9nfHFJRry21R1trw2Vdpn+9g481URipzWV0Eihvm+xTtmaZ1Sp//1kq75XDwnU
\nWA8gkn603QE3fq2yN98BURsAKdJfJ5RL1HvGQvTe10HLYYXpJnEkHv+Unl2ajLivWUt5pbBrKbUC
\nngYBjb0+0Zk0sCcpZ29sbzjYjpIddErySIyRX5gV2uNQwAjLdp9PfN295yQ+BxMBXiIycWVQiw0bH
\nnoMo7yykABY70zd5wQewBQ4AdSlWSX4nGDtsiFxiWiI5sKuAAe0CbTosy1s8w8fxoJ5Tz1sdoxNeGs
\nArq6Wv/G16zQuAE9zK9vwwKBgF+09VI/1wJBirsDGz9whVwfFPrTkJNvJZzYt69qezx1sjgFKshy
\nWBhd4xHZtmCqpBP1AymEjr/T0lboxyARMXmnIOWIANXMGb4KGSyl1mzSVAoQ+fqR+cJ3d0dyP11j
\nnjjb0Ed/NY8frlNDxAVHE8BSkdsx2f6ELEyBKJSRr9snRAoGAMrTwYneXzvTskF/S5Fyu0i0egLda
\nNWUH38v/nDCgEpIXD5Hn3qAEcju1IjmbwlvT+nY2jVhv7UGd8MjwUTNGItDb6nsYqM2asrnF3qS
\nVRkAKKKYeGjKpUfVTrW0YFjXkfcrR/V+QFL50ndHAKJXjW7a4ejJLncTzmZSpYzwApc=\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。您可以通过实例集配置中授权的任何端口访问 Amazon GameLift Servers 实例集中的实例。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。创建实例集后，您在实例集激活前无法更新端口设置。如果您遇到困难，请在端口设置打开的情况下重新创建实例集。

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)。示例：

```
$ AWS gamelift update-fleet-port-settings
--fleet-id "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
--inbound-permission-authorizations
"FromPort=22,ToPort=22,IpRange=54.186.139.221/32,Protocol=TCP"
```

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

无论您是通过控制台还是 AWS CLI 远程连接到实例，您都拥有对该实例的完全用户和管理访问权限。这意味着您也有权限对游戏托管服务进行操作，因此可能引发服务异常或故障。如果实例托管有活动玩

家的游戏，您可能面临着以下风险：游戏会话崩溃且玩家掉线，或者中断游戏关闭进程，从而导致已保存游戏数据和日志中出错。

在托管实例上查找以下资源：

- 游戏生成包文件。这些文件是您上传到 Amazon GameLift Servers 的游戏生成包。它们包含一个或多个游戏服务器可执行文件、资产和依赖项。游戏生成包文件位于名为 game 的根目录下：
 - 在 Windows 上：c:\game
 - 在 Linux 上：/local/game
- 游戏日志文件。在您指定的任何目录路径下的 game 根目录中查找游戏服务器生成的日志文件。
- Amazon GameLift Servers 托管资源。根目录 Whitewater 包含 Amazon GameLift Servers 服务用于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。请勿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 文件包含实例所属的实例集的相关信息，供在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 证书生成的实例集中，请在以下位置查找证书文件，包括证书、证书链、私有密钥和根证书：
 - 在 Windows 上：c:\GameMetadata\Certificates
 - 在 Linux 上：/local/gamemetadata/certificates/

利用 Amazon GameLift Servers 扩展游戏托管容量

托管容量（以实例为单位）表示 Amazon GameLift Servers 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行玩家数量。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。

容量在实例集位置级别进行调整。所有舰队都至少有一个地点：舰队的所在 AWS 地区。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。

您可以使用 Amazon GameLift Servers 控制台执行大多数实例集扩展活动。您也可以将 S AWS DK 或 AWS Command Line Interface (AWS CLI) 与[服务 API 配合](#)使用 Amazon GameLift Servers。

主题

- [在控制台中管理实例集容量](#)
- [设置 Amazon GameLift Servers 容量限制](#)
- [手动设置 Amazon GameLift Servers 舰队的容量](#)
- [通过 Amazon GameLift Servers 自动扩展队组容量](#)
- [管理缩放 Amazon GameLift Servers 舰队 To/From Zero](#)
- [扩缩 Amazon GameLift Servers 容器实例集](#)

在控制台中管理实例集容量

1. 打开 [Amazon GameLift Servers 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。

设置 Amazon GameLift Servers 容量限制

手动或通过 auto Scaling 扩展 Amazon GameLift Servers 舰队位置的托管容量时，请考虑该地点的扩展限制。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 Servers 根据玩家需求扩展实例集位置，同时还可以防止托管成本失控，例如在 DDOS 攻击期间。设置 A [Amazon CloudWatch 警报](#)，以便在容量接近最大限制时通知您，这样您就可以评估情况并根据需要进行手动调整。（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 Servers 控制台](#)中或使用 AWS Command Line Interface (AWS CLI) 为舰队的位置设置容量限制。

设置容量限制

Console

1. 打开 [Amazon GameLift Servers 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，使用包含要更改容量的舰队 ID 和位置的[describe-fleet-location-capacity](#)命令。此命令返回一个包含该地点当前容量设置的[FleetCapacity](#)对象。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 Servers 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 Servers 将返回错误。

手动设置Amazon GameLift Servers舰队的容量

当您创建新的实例集时，Amazon GameLift Servers 会自动将所需实例设置为每个实例集位置中一个实例。然后，Amazon GameLift Servers 会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。

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 Servers控制台中手动设置容量，也可以使用 AWS Command Line Interface (AWS CLI)。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 Servers 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 Servers 保持活动状态且随时可托管游戏会话的实例数量。

Amazon GameLift Servers 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 Servers 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，使用包含要更改容量的舰队 ID 和位置的[describe-fleet-location-capacity](#)命令。此命令返回一个包含该地点当前容量设置的[FleetCapacity](#)对象。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。使用带有舰队 ID、位置和所需实例的新值的[update-fleet-capacity](#)命令。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 Servers 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 Servers 会返回错误。

通过 Amazon GameLift Servers 自动扩展队组容量

使用 Amazon GameLift Servers 中的自动扩缩功能可以动态扩缩实例集容量，以应对游戏服务器活动。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。

要使用自动扩缩功能，您需要创建扩展策略，告知 Amazon GameLift Servers 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 Servers 控制台、AWS Command Line Interface (AWS CLI) 或 AWS SDK 管理基于目标的自动扩展。尽管可以在控制台中查看基于规则的扩展策略，但您只能使用 AWS CLI 或 AWS SDK 管理基于规则的自动扩展。

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 Servers 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 Servers 会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 Servers 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，设置要保持在的百分比，然后选择确认。在您确认设置后，Amazon GameLift Servers 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 Servers 容量限制](#)。
2. 创建新策略。打开命令行窗口，使用带有策略参数设置的 [put-scaling-policy](#) 命令。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 Servers 中基于规则的扩缩策略会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

使用 AWS SDK 或带有[服务 API](#) 的 AWS Command Line Interface (AWS CLI) 创建、更新或删除基于规则的策略。Amazon GameLift Servers 您可以在 Amazon GameLift Servers 控制台中查看所有活动的策略。

要暂时停止队列的所有扩展策略，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](#)命令设置一个或两个极限值。有关更多信息，请参阅[设置 Amazon GameLift Servers 容量限制](#)。
2. 创建新策略。打开命令行窗口，使用带有策略参数设置的[put-scaling-policy](#)命令。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，使用带有舰队 ID 和策略名称的[delete-scaling-policy](#)命令。

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<metric name>仍<comparison operator><threshold value>为<evaluation period>，则使用更改舰队容量<adjustment type> to/by <adjustment value>。

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 Servers 队组指标](#)。

- 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

指示 Amazon GameLift Servers 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 Servers 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。

队列容量还受到队列实例类型限制和您的服务配额的限制 AWS 账户。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

在更改容量以响应自动扩缩策略之后，Amazon GameLift Servers 将等待 10 分钟，然后再响应该相同策略的触发器。这段等待时间能让 Amazon GameLift Servers 有足够时间添加新实例，启动游戏服务器，连接玩家，以及开始从新实例收集数据。在此期间，Amazon GameLift Servers 会依据相关指标对策略进行评估，并跟踪该策略的评估周期（该周期会在发生扩缩事件后重新开始计算）。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。

管理缩放Amazon GameLift Servers舰队 To/From Zero

Amazon GameLift Servers支持根据游戏会话活动自动缩放到零实例和从零实例缩放。此托管容量选项允许您的舰队位置在规定的无游戏会话活动期限后缩减到零实例，并在请求游戏会话时自动向外扩展。

扩展到零实例和从零实例扩展有以下几个优点：

- 成本优化 — 通过在没有游戏会话活动时运行零个实例，从而消除非活动期间的计算成本。
- 自动响应式横向扩展 — 当请求游戏会话时，舰队位置会自动向外扩展到一个实例，无需手动干预。
- 简化管理 — 无需根据预期的玩家需求或开发需求手动将舰队容量调整 to/from 为零。

当您在队列上启用 Scale To/From Zero 时，Amazon GameLift Servers 会监控游戏会话活动并自动调整队列容量：

- 缩减到零 — 在配置的时间段之后，没有游戏会话活动，将舰队Amazon GameLift Servers位置缩减为零。
- 从零向外扩展 — 收到游戏会话创建请求后，将 Fleet Amazon GameLift Servers 位置扩展到一个实例，从而允许恢复自动缩放。
- 继续扩展 — 扩展后，队列将使用配置的 auto Scaling 策略恢复容量。

缩小行为

Amazon GameLift Servers在配置的闲置时间已过且没有游戏会话活动之后，开始缩减舰队位置的规模。这被定义为一个时期，其中：

- 舰队所在地没有活跃的游戏会话。
- 尚未收到在舰队所在地创建新游戏会话的请求。

在缩减期间，Amazon GameLift Servers会将舰队所在地的最低和所需容量设置为零，并迅速扩大规模以节省成本。

横向扩展行为

在舰队位置为零的情况下收到游戏会话创建请求时：

- Amazon GameLift Servers立即启动一个实例的横向扩展。

- 尝试在其他舰队或舰队位置放置游戏会话可能会继续，具体取决于队列的配置（如果使用）。

Note

从零向外扩展需要时间来配置和初始化实例。玩家在闲置一段时间后，等待第一个游戏会话的时间可能会更长。因此，此功能最好与多地点舰队队列 and/or 搭配使用。

配置 To/From 零比例

通过更新现有队列来配置缩放 To/From 零。

Console

1. 打开 [Amazon GameLift Servers 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“缩放”下，选择要为其配置 to/from 零比例的每个位置，然后选择“编辑”。
5. 在“编辑扩展容量”对话框中，将“最小容量策略”选择“自动”，在“之后将最小容量设置为 0”的首选值（以分钟为单位），然后选择确认。

Amazon GameLift Servers 然后，一旦在配置的持续时间内没有游戏会话活动，就会将选定的舰队位置缩减为 0 个实例。此后，当在此位置发出游戏会话请求时，Amazon GameLift Servers 将尽快扩展一个实例。这个过程需要一点时间。

AWS CLI

- 配置 to/from 零比例。在命令行窗口中，使用带有舰队 ID、位置和托管容量配置的 [update-fleet-capacity](#) 命令来配置 to/from 零规模。

```
aws gamelift update-fleet-capacity \
    --fleet-id <fleet identifier> \
    --location <location name> \
    --managed-capacity-configuration
    ScaleInAfterInactivityMinutes=60,ZeroCapacityStrategy=SCALE_TO_AND_FROM_ZERO
```

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 \
    --managed-capacity-configuration
    ScaleInAfterInactivityMinutes=60,ZeroCapacityStrategy=SCALE_TO_AND_FROM_ZERO
```

如果您的请求成功，则Amazon GameLift Servers返回更新的容量配置。Amazon GameLift Servers然后，一旦在配置的持续时间内没有游戏会话活动，就会将选定的舰队位置缩减为 0 个实例。此后，当在此位置发出游戏会话请求时，Amazon GameLift Servers将尽快扩展一个实例。这个过程需要一点时间。

最佳实践

使用 Scale To/From Zero 时，请考虑以下建议：

- 设置适当的闲置时间-在成本节省与缩小/向外扩展周期的频率之间取得平衡。缩短缩至零的等待时间可以最大限度地节省开支，但会导致更频繁的冷启动。
- 与可预测的工作负载一起使用 — Scale To/From Zero 最适合处于明显非活动状态的游戏，例如非高峰时段明显的 development/test 环境或游戏。
- 监控 CloudWatch 指标-跟踪队列扩展事件和游戏会话放置时间，以优化您的配置。
- 与扩展策略结合使用 Scale To/From Zero 以及基于目标或基于规则的自动缩放，实现全面的容量管理。

扩缩 Amazon GameLift Servers 容器实例集

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。在托管式容器实例集中，您可以通过添加或移除实例集实例来扩缩实例集容量。

当您创建新实例集时，Amazon GameLift Servers 会将该实例集的所需容量设置为一个实例，并在该实例集的主区域部署一个实例。对于多位置实例集，Amazon GameLift Servers 会在主区域以及每个远程位置各部署一个实例。实例集状态变为 ACTIVE 后，您可以提高所需容量以扩容，或降低所需容量以缩减规模。

您可以使用 Amazon GameLift Servers 扩缩功能手动更改容量，也可以根据玩家需求设置自动扩缩：

- 使用目标跟踪设置自动扩缩。请参阅[基于目标的自动扩缩](#)。
- 手动更改实例集容量。请参阅[手动设置Amazon GameLift Servers舰队的容量](#)。

在扩缩容器实例集时，需考虑添加或删除实例对实例集托管游戏会话和玩家容量的影响。

- 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可托管一个游戏会话的能力。
 - 使用以下公式计算一个容器实例集实例上可同时运行的游戏会话数：

```
[Game sessions per instance] = [# of game server processes per game server container] * [# of game server container groups per instance]
```

如果您的容器架构在游戏服务器容器中并发运行一个游戏服务器进程，则每个实例的游戏会话数等于每个实例的游戏服务器容器组数量。

- 对于每个实例的游戏服务器容器组数量，可调用 [DescribeContainerFleet](#) 以获取 `GameServerContainerGroupsPerInstance` 或 `MaximumGameServerContainerGroupsPerInstance` 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席位数量。根据您的托管解决方案处理游戏会话放置的方式，您可以在对战配置中或发起游戏会话放置的调用中定义每个游戏会话的玩家数。
 - 使用以下公式计算一个容器实例集实例上可同时参与游戏的玩家数量：

```
[Players per instance] = [# of game sessions per instance] * [# of player slots per game session]
```

要获取容器实例集当前的总容量，可调用 [DescribeFleetCapacity](#) 或 [DescribeFleetLocation Capacity](#) 来获取实例集中游戏服务器容器组的数量。其中，活跃组指当前正在托管游戏会话的容器组，空闲组指已就绪可托管新游戏会话的容器组。将这些值乘以每个游戏服务器容器组的服务器进程数，即可得到总容量。

监控 Amazon GameLift Servers

监控是维护和其他 AWS 解决方案的可靠性、可用性和性能的重要组成部分。Amazon GameLift Servers 指标有三个主要用途：监控系统运行状况和设置警报；跟踪游戏服务器的性能和使用情况；以及使用手动或自动扩展功能管理容量。

AWS 提供以下监控工具 Amazon GameLift Servers，供您监视、报告问题并在适当时自动采取措施：

- Amazon GameLift Servers 控制台 – 使用图形界面管理您的 Amazon GameLift Servers 资源并跟踪游戏托管活动。
- 服务器遥测指标 – Amazon GameLift Servers 支持您通过 SDK 和插件集成，直接从游戏服务器发送自定义指标。您可以定义并跟踪自己的游戏特定指标以及内置的性能、网络、内存和计时数据。所有指标都可以发布到 [Amazon Managed Service for Prometheus](#)，并使用完全可自定义的控制面板在 Amazon Grafana 中进行监控。您也可以发布到 Amazon CloudWatch 以与其他 AWS 服务集成。遥测系统可完全自定义，除内置指标外，您还可以创建自定义 Prometheus 查询，以获取更多指标。有关特定技术栈的实施指南，请参阅 [the section called “使用服务器遥测指标进行监控”](#)。
- Amazon CloudWatch — 您可以实时监控 Amazon GameLift Servers 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。Amazon CloudWatch 提供了一套监控功能，包括用于创建自定义控制面板的工具，以及设置警报以在指标达到指定阈值时发出通知或采取行动的功能。
- AWS CloudTrail — 捕获由您的账户或代表您的 AWS 账户为和其他 AWS 服务进行的所有 API 调用 Amazon GameLift Servers 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。

跨监控源的指标比较

Amazon GameLift Servers 通过三个主要来源提供指标：Amazon GameLift Servers 控制台舰队活动指标、服务器遥测指标和 Amazon CloudWatch Amazon GameLift Servers 指标。了解每个来源的重叠情况和独特功能，可以帮助您根据需求选择合适的监控方法。

按来源划分的指标可用性

下表按指标类别显示了不同监控来源可用的指标。

实例指标

实例集容量和运行状况监控的实例级指标：

实例指标可用性

指标	控制台	CloudWatch	遥测
活动实例	✓	✓	✓
空闲实例	✓	✓	✓*
空闲实例所占百分比	✓	✓	✓*
预期实例数	✓	✓	
最大实例数	✓	✓	
最小实例数	✓	✓	
CPU 使用率		✓	✓
网络输入/输出量		✓	✓
Disk/Storage read/write		✓	✓
实例中断次数	✓	✓	
实例回收次数	✓	✓	
运行不正常的实例替换次数		✓	

*可通过使用遥测数据的自定义 Prometheus 查询获取。

游戏服务器指标

服务器进程和游戏会话指标：

游戏服务器指标可用性

指标	控制台	CloudWatch	遥测
有效游戏会话	✓	✓	✓*
激活游戏会话	✓	✓	

指标	控制台	CloudWatch	遥测
可用的游戏会话	✓	✓	
可用游戏会话所占百分比	✓	✓	
并发可激活游戏会话数	✓	✓	
游戏会话中断次数	✓	✓	
活动服务器进程数			✓
运行状况良好的游戏服务器数			✓
崩溃的游戏会话数			✓

*可通过使用遥测数据的自定义 Prometheus 查询获取。

玩家使用情况指标

玩家会话和并发用户指标：

玩家使用情况指标可用性

指标	控制台	CloudWatch	遥测
当前玩家会话	✓		
玩家会话激活次数	✓		
全局并发用户数 (CCU)			✓
位置并发用户数 (CCU)			✓
位置容量			✓

容器实例集指标

特定于容器的指标（仅 CloudWatch 适用于集装箱舰队）：

容器实例集指标可用性

指标	控制台	CloudWatch	遥测
活动容器组数		✓	✓
空闲容器组数		✓	✓
容器 CPU/Memory 利用率		✓	✓
容器网络流量		✓	✓
容器存储操作次数		✓	✓

选择合适的监控源

根据具体需求选择监控方法：

- 使用服务器遥测指标来监控来自游戏服务器的全面性能数据，并发出特定于游戏的自定义指标。这些指标可以详细了解所有游戏引擎和服务器的服务器性能、网络活动、内存使用情况和计时数据 SDKs。您可针对游戏事件、业务逻辑性能和特定于应用程序的数据点定义自定义指标。所有控制面板均可完全自定义，您可以创建自定义 Prometheus 查询，从收集的数据中获取更多指标。有关更多信息，请参阅 [the section called “使用服务器遥测指标进行监控”](#)。
- 使用 Amazon GameLift Servers 控制台实现实例集管理、容量规划和一般运营监督。该控制台提供实例集运行状况与玩家活动的集成视图。有关更多信息，请参阅 [the section called “管理游戏托管资源”](#)。
- 使用 Amazon CloudWatch 进行自动监控、警报以及与其他 AWS 服务的集成。CloudWatch 启用自定义仪表板和基于警报的自动化。有关更多信息，请参阅 [the section called “使用监视器 CloudWatch”](#)。
- 使用 [Amazon Managed Service for Prometheus](#) 实现高性能指标收集与存储，并支持 PromQL 查询功能。Prometheus 为服务器遥测指标提供可扩展的时间序列数据存储。
- 使用 [Amazon Managed Grafana](#) 实现高级可视化和完全可自定义的控制面板管理。Grafana 提供您可以自定义和扩展的 GameLift 预建仪表板，还支持创建包含多个数据源的完全自定义的仪表板。您可以构建自定义查询和可视化工具，以追踪游戏中任何重要的指标。
- 使用多个来源实现全面监控。将控制台监督、服务器遥测指标和自动化警报结合起来，全面了解游戏托管基础设施。

要获得其他监控功能，还可使用以下方法：

- [the section called “记录 API 调用”](#) – 跟踪 API 调用和相关事件，满足审计与合规性要求。
- [the section called “记录服务器消息”](#) – 捕获自定义服务器消息和游戏会话日志。

主题

- [the section called “管理游戏托管资源”](#)
- [the section called “使用监视器 CloudWatch”](#)
- [the section called “使用服务器遥测指标进行监控”](#)
- [the section called “记录 API 调用”](#)
- [the section called “记录服务器消息”](#)

Amazon GameLift Servers使用 Amazon 进行监控 CloudWatch

您可以Amazon GameLift Servers使用 Amazon 进行监控 CloudWatch，该 AWS 服务可收集原始数据并将其处理为可读的近乎实时的指标。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 Servers 托管游戏服务器的性能。您可以设置用于监控特定阈值的警报，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](#)。

为了通过详细的遥测指标增强服务器端的可观测性，Amazon GameLift Servers 还支持通过 [Amazon Managed Service for Prometheus](#) 和 [Amazon Managed Grafana](#) 进行监控。这些遥测指标提供了详细的服务器性能数据，包括：

- 服务器计时：服务器增量时间（服务器节拍率一致性）、服务器节拍率（每秒更新）、服务器节拍时间（每个节拍的处理时间）和服务器世界节拍时间（游戏世界更新时间）
- 网络指标：连接（网络连接总数）、以字节和数据包为单位的网络 I/O（数据传输统计信息）和数据包丢失（传输失败百分比）
- 内存指标：以单位表示的内存使用量（总内存消耗）和物理内存使用百分比（内存利用率）
- CPU 使用率：CPU 使用百分比（按游戏服务器进程显示的资源利用率）

有关特定于您的 SDK 或插件的实施指南，请参阅[the section called “使用服务器遥测指标进行监控”](#)。

以下各表列出 Amazon GameLift Servers 的指标和维度。中可用的所有指标 CloudWatch 也可在控制台中找到，Amazon GameLift Servers 控制台以一组可自定义的图表形式提供数据。要访问游戏的 CloudWatch 指标，请 AWS 管理控制台使用 AWS CLI、或 CloudWatch API。

如果指标没有位置，则使用起始位置。

Amazon GameLift Servers 指标的维度

Amazon GameLift Servers 支持按以下维度来筛选指标。

维度	说明
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。通过将指标组名称添加到队列的属性中，将队列添加到指标组（请参阅 UpdateFleetAttributes() ）。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。

维度	说明
ContainerGroupType	筛选集装箱船队 GameServer ContainerGroup 指标，以区分船队位置 PerInstance ContainerGroup 指标和船队位置指标。此维度包含在某些容器实例集指标（例如 ContainerNetworkIn 、 ContainerStorageReadBytes ）中，可让您区分不同容器组类型的指标。

Amazon GameLift Servers 队组指标

Amazon 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。队组与托管 Amazon GameLift Servers 解决方案一起使用。该 Amazon GameLift Servers 服务 CloudWatch 每分钟向发送一次指标。

实例

指标	说明
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 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置
DesiredInstances	Amazon GameLift Servers 在队组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

指标	说明
	相关 CloudWatch 统计数据：平均值、最小值、最大值 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 相关 CloudWatch 统计数据：平均值、最小值、最大值 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 CloudWatch 统计数据：平均值、最小值、最大值 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 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置

指标	说明
PercentIdleInstances	处于空闲状态的所有活动实例的百分比 (计算公式为：$\text{IdleInstances} / \text{ActiveInstances}$)。该指标可与自动扩展功能配合使用。 单位：百分比 相关 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置
RecycledInstances	已回收和替换的竞价型实例的数量。Amazon GameLift Servers 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。此指标仅适用于托管式 EC2 实例集。 单位：计数 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 维度：位置
RecycledSpotInstances	已回收和替换的竞价型实例的数量。Amazon GameLift Servers 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。此指标仅适用于容器实例集。 单位：计数 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 维度：位置

指标	说明
InstanceSpotInterruptions	已中断的竞价型实例的数量。 单位：计数 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 维度：位置
CPUUtilization	EC2 指标。对于 Amazon GameLift Servers，此指标表示跨实例集位置中所有活跃实例的硬件性能。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。CloudWatch 由于传统设备模拟、非传统设备配置、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有所不同。 单位：百分比
NetworkIn	EC2 指标。对于 Amazon GameLift Servers，此指标表示跨实例集位置中所有活跃实例的硬件性能。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
NetworkOut	EC2 指标。对于 Amazon GameLift Servers，此指标表示跨实例集位置中所有活跃实例的硬件性能。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

指标	说明
DiskReadBytes	EC2 指标。对于 Amazon GameLift Servers，此指标表示跨实例集位置中所有活跃实例的硬件性能。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 Servers，此指标表示跨实例集位置中所有活跃实例的硬件性能。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 Servers，此指标表示跨实例集位置中所有活跃实例的硬件性能。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。要计算该时段内的平均每秒 I/O 操作数 (IOPS)，请将该时段内的总操作数除以该时段内的秒数。 单位：计数
DiskWriteOps	EC2 指标。对于 Amazon GameLift Servers，此指标表示跨实例集位置中所有活跃实例的硬件性能。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。要计算该时段内的平均每秒 I/O 操作数 (IOPS)，请将该时段内的总操作数除以该时段内的秒数。 单位：计数

服务器进程

指标	说明
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 CloudWatch 统计数据：平均值、最小值、最大值 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 CloudWatch 统计数据：平均值、最小值、最大值 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：$\text{HealthyServerProcesses} / \text{ActiveServerProcesses}$)。 单位：百分比 相关 CloudWatch 统计数据：平均值、最小值、最大值 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 Servers 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

指标	说明
	检查失败或没有干净地终止（通过调用 ProcessEnding()）时，就会发生这种情况。 单位：计数 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 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 CloudWatch 统计数据：总和、平均值、最小值、最大值 维度：位置
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 CloudWatch 统计数据：总和、平均值、最小值、最大值 维度：位置

游戏会话

指标	说明
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

指标	说明
	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 CloudWatch 统计数据：平均值、最小值、最大值 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 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置

指标	说明
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 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置
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。（参见舰队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 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置

指标	说明
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：$\text{PercentAvailableGameSessions} = \frac{\text{AvailableGameSessions}}{[\text{ActiveGameSessions} + \text{AvailableGameSessions} + \text{unhealthy server processes}]}$)。该指标可与自动扩展功能配合使用。 单位：百分比 相关 CloudWatch 统计数据：平均 维度：位置
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 CloudWatch 统计数据：总和、平均值、最小值、最大值 维度：位置

玩家会话

指标	说明
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 CloudWatch 统计数据：平均值、最小值、最大值

指标	说明
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 CloudWatch 统计数据：总和、平均值、最小值、最大值

容器实例集指标

容器实例集指标为基于容器的游戏托管提供监控数据，包括容器组状态、资源利用率和性能指标。

Note

使用遥测指标解决方案时，可通过 [Amazon Managed Grafana](#) 获得用于监控容器实例集的增强型控制面板。有关设置说明，请参阅[the section called “使用服务器遥测指标进行监控”](#)。

指标	说明
ActiveGameServerContainerGroups	当前处于活动状态并已准备好托管游戏会话的容器组。 单位：计数 相关 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置
IdleGameServerContainerGroups	处于活动状态但当前未托管游戏会话的容器组。 单位：计数 相关 CloudWatch 统计数据：平均值、最小值、最大值

指标	说明
	维度：位置
PendingGameServerContainerGroups	正处于启动过程中的容器组。 单位：计数 相关 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置
TerminatingGameServerContainerGroups	正处于关闭过程中的容器组。 单位：计数 相关 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置
UnhealthyGameServerContainerGroupsReplaced	由于运行状况检查失败而被替换的容器组。 单位：计数 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 维度：位置
ContainerCPUUtilizationPerInstance	实例上所有容器的平均 CPU 利用率。 单位：百分比 相关 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置

指标	说明
ContainerCPUReservation	容器预留的 CPU 资源的百分比。 单位：百分比 相关 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置
ContainerMemoryReservation	容器预留的内存资源的百分比。 单位：百分比 相关 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置
ContainerMemoryUtilization	容器正在使用的预留内存百分比。 单位：百分比 相关 CloudWatch 统计数据：平均值、最小值、最大值 维度：位置
ContainerNetworkIn	容器收到的网络字节数。 单位：字节 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 维度：位置

指标	说明
ContainerNetworkOut	容器发送的网络字节数。 单位：字节 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 维度：位置
ContainerStorageReadBytes	容器从存储中读取的字节数。 单位：字节 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 维度：位置
ContainerStorageWriteBytes	容器写入存储的字节数。 单位：字节 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 维度：位置

DDoS 保护（玩家网关）指标

Amazon GameLift Servers 当设置为 `ENABLED` 或 `REQUIRED` 时，玩家网关指标会跟踪流经 DDoS 保护层 `PlayerGatewayMode` 的 UDP 流量。这些指标可帮助您监控流量模式、识别潜在的 DDoS 攻击并跟踪中继性能。

指标	说明
PlayerGatewayPacketsIn	从玩家那里收到并通过 DDoS 保护层转发到游戏服务器的 UDP 数据包的数量。

指标	说明
	单位：计数 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 尺寸：FleetId，位置
PlayerGatewayBytesIn	从玩家那里接收并通过 DDoS 保护层转发到游戏服务器的字节数。 单位：字节 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 尺寸：FleetId，位置
PlayerGatewayPacketsOut	从游戏服务器接收并通过 DDoS 保护层转发给玩家的 UDP 数据包的数量。 单位：计数 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 尺寸：FleetId，位置
PlayerGatewayBytesOut	从游戏服务器接收并通过 DDoS 保护层转发给玩家的字节数。 单位：字节 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 尺寸：FleetId，位置

指标	说明
PlayerGatewayPacketsThrottled	由于 DDo S 保护层的速率限制而丢弃的入站数据包数量。 单位：计数 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 尺寸：FleetId，位置
PlayerGatewayBytesThrottled	由于 DDo S 保护层的速率限制而丢弃的入站字节数。 单位：字节 相关 CloudWatch 统计数据：总和、平均值、最小值、最大值 尺寸：FleetId，位置
PlayerGatewayPlayerSessions	当前受 DDo S 保护层保护的玩家会话。 单位：计数 相关 CloudWatch 统计数据：平均值、最小值、最大值 尺寸：FleetId，位置

Amazon GameLift Servers 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 Servers 解决方案一起使用。该 Amazon GameLift Servers 服务 CloudWatch 每分钟向发送一次指标。

指标	说明
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。

指标	说明
	单位：秒 相关 CloudWatch 统计数据：平均值、最小值、最大值、总和 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 优先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 CloudWatch 统计数据：平均值、最小值、最大值、总和
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 优先级选择的第一个实例集。 单位：计数 相关 CloudWatch 统计数据：平均值、最小值、最大值、总和
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 相关 CloudWatch 统计数据：平均值、最小值、最大值、总和

指标	说明
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 CloudWatch 统计数据：平均值、最小值、最大值、总和
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 CloudWatch 统计数据：总和
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 CloudWatch 统计数据：平均值、最小值、最大值、总和
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 CloudWatch 统计数据：平均值、最小值、最大值、总和

指标	说明
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 CloudWatch 统计数据：平均值、最小值、最大值、总和
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 CloudWatch 统计数据：平均值、最小值、最大值、总和
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 CloudWatch 统计数据：平均值、最小值、最大值、总和
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 CloudWatch 统计数据：平均值、最小值、最大值、总和 维度：位置

用于对战的 Amazon GameLift Servers 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 Amazon GameLift Servers 解决方案一起使用。该 Amazon GameLift Servers 服务 CloudWatch 每分钟向发送一次指标。

有关对战活动序列的更多信息，请参阅 [Amazon GameLift Servers FlexMatch 的工作原理](#)。

对战配置

指标	说明
CurrentTickets	当前正在处理或等待处理的对战请求。 单位：计数 相关 CloudWatch 统计数据：平均值、最小值、最大值、总和
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 CloudWatch 统计数据：总和
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 CloudWatch 统计数据：总和
MatchesCreated	上次报告后创建的潜在对战游戏。 单位：计数 相关 CloudWatch 统计数据：总和
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	说明
	相关 CloudWatch 统计数据：总和
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 CloudWatch 统计数据：总和
PlayersStarted	上次报告后在对战票证中添加的玩家。 单位：计数 相关 CloudWatch 统计数据：总和
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 CloudWatch 统计数据：总和
TicketsStarted	上次报告后创建的新对战请求。 单位：计数 相关 CloudWatch 统计数据：总和
TicketsTimedOut	上次报告后达到超时限制的对战请求。 单位：计数 相关 CloudWatch 统计数据：总和
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 CloudWatch 统计数据：数据样本、平均值、最小值、最大值

指标	说明
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 CloudWatch 统计数据：数据样本、平均值、最小值、最大值
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 CloudWatch 统计数据：数据样本、平均值、最小值、最大值

对战规则

指标	说明
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 CloudWatch 统计数据：总和
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 CloudWatch 统计数据：总和

Amazon GameLift ServersFleetIQ的 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。该Amazon GameLift Servers服务 CloudWatch 每分钟向发送一次指标。另请参阅 Amazon EC2 Auto Scaling 用户指南 CloudWatch中的使用亚马逊监控您的 Auto Scaling [组和实例](#)。

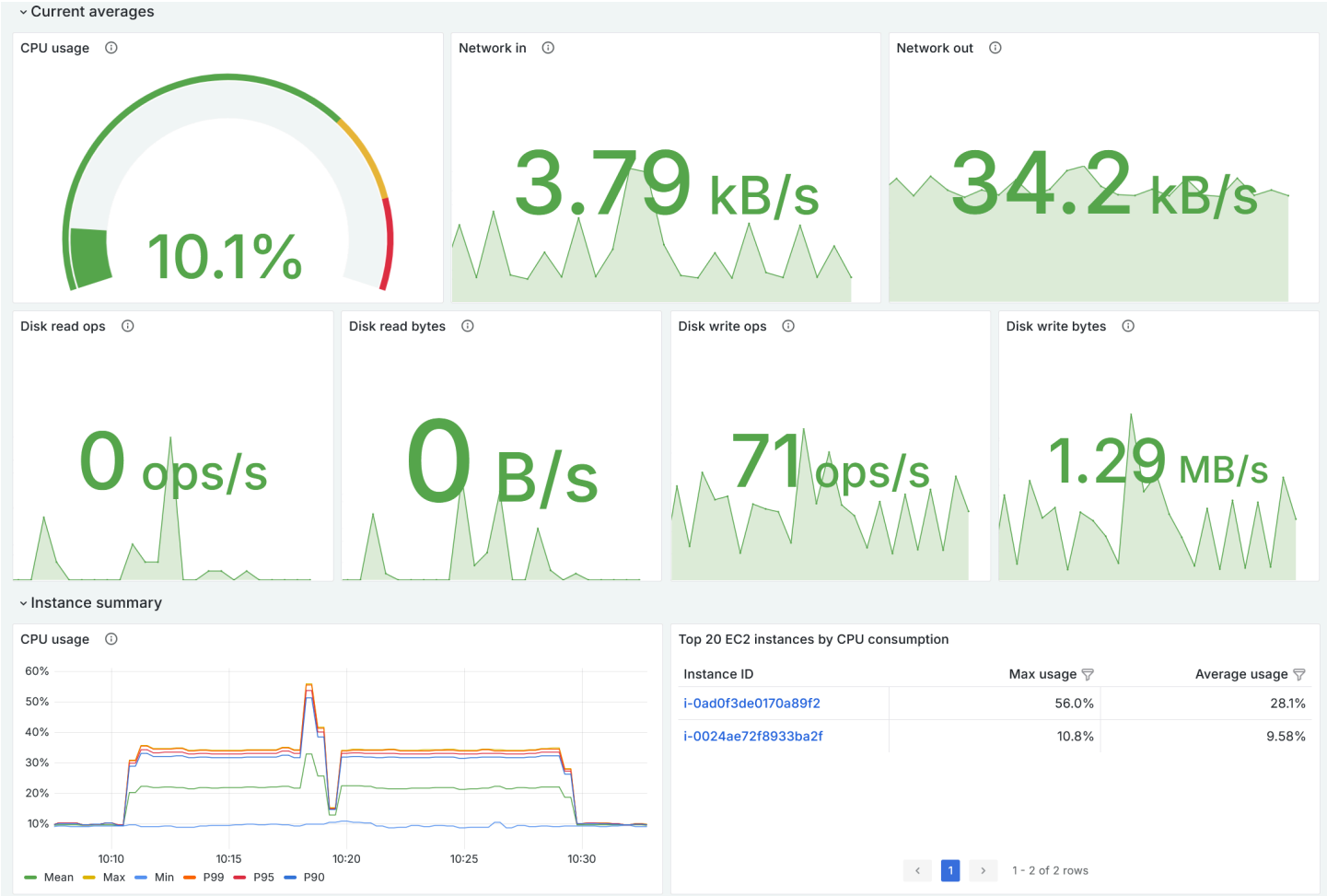
指标	说明
AvailableGameServers	可用于运行游戏执行但当前未被玩游戏占用的游戏服务器数量。此数字包括已认领但仍处于 AVAILABLE（可用）状态的游戏服务器。 单位：计数 亚马逊的相关 CloudWatch 统计数据：总和 尺寸：GameServerGroup
UtilizedGameServers	当前被游戏占用的游戏服务器数量。此数字包括处于 UTILIZED 状态的游戏服务器。 单位：计数 亚马逊的相关 CloudWatch 统计数据：总和 尺寸：GameServerGroup
DrainingAvailableGameServers	当前不支持玩游戏且计划终止的实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 亚马逊的相关 CloudWatch 统计数据：总和 尺寸：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	说明
	单位：计数 亚马逊的相关 CloudWatch 统计数据：总和 尺寸：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 Amazon 的相关 CloudWatch 统计数据：平均值、最小值、最大值 尺寸：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 亚马逊的相关 CloudWatch 统计数据：总和 尺寸：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 亚马逊的相关 CloudWatch 统计数据：总和 尺寸：GameServerGroup，InstanceType

使用服务器遥测指标进行监控

Amazon GameLift Servers 可以配置为收集和发布在托管 Amazon EC2 和容器队列上运行的游戏服务器的遥测指标。在将遥测收集器与服务器生成包一起部署后，即可使用这些指标。该指标系统支持所

有插件 SDKs（C++、C#、Go）、所有插件（虚幻、Unity）和Amazon GameLift Servers游戏服务器包装器。指标数据流向 [Amazon Managed Service for Prometheus](#)、[the section called “使用监视器 CloudWatch”](#) 和 [Amazon Managed Grafana](#) 控制面板（推荐用于可视化）。



使用 Amazon Managed Grafana 或亚马逊中预建的控制面板监控游戏服务器的性能。 CloudWatch

遥测指标的优势

遥测指标系统具有五个主要优势：

- 特定于游戏引擎的指标：游戏引擎插件（Unreal、Unity）提供与特定于引擎的性能指标的原生集成，例如服务器节拍时间、帧速率和引擎级别的资源利用率，这些指标对于游戏性能优化至关重要。
- 自定义指标支持：使用服务器 SDK 函数调用定义和跟踪您自己的游戏特定指标，以监控与您的游戏相关的自定义游戏事件、业务逻辑性能和特定于应用程序的数据点。
- 自动收集 — 在部署遥测收集器后，指标会自动流动，无需其他工具，并且可以直接与 [Prometheus](#) 和 [Amazon 的亚马逊托管服务](#) 集成。 CloudWatch

- 多级别监控：实例集级指标可监控容量和扩缩事宜，实例级指标可反映资源利用率，游戏会话指标则可跟踪性能。
- 通用兼容性-适用于所有Amazon GameLift Servers支持的开发环境，与所有服务器集成 SDKs，游戏引擎插件中提供原生支持。

Note

遥测指标适用于运行亚马逊 Linux 2023 EC2 或 Windows 的Amazon GameLift Servers托管亚马逊或集装箱舰队。

开始前的准备工作

所需 AWS 资源

- AWS 为配置的帐户Amazon GameLift Servers。
- 在以下位置运行的托管式实例集：
 - 支持操作系统的亚马逊 EC2，或者
 - 安装了 Amazon Linux 2023 的容器
- 适当的 IAM 权限

IAM 要求

仅当您计划使用相应服务时，才需要具备以下 IAM 权限：

- [Amazon Managed Service for Prometheus](#) (仅在向 Prometheus 发布指标时需要使用该服务)
 - `aps:RemoteWrite` 许可
 - 您的 Prometheus 工作区的访问权限
- 亚马逊 CloudWatch (仅在向亚马逊发布指标时才需要 CloudWatch)
 - `cloudwatch:PutMetricData` 许可
 - 指标命名空间的访问权限
- [Amazon Managed Grafana](#) (仅在使用 Grafana 控制面板时需要使用该服务)
 - `grafana:Read` 许可
 - 用于访问控制面板的 SSO 配置

实施

根据您的开发环境选择您的实现路径：

SDK 实现

SDK 类型	SDK 设置	自定义 指标	API 参考
Go SDK	完整设置指南	Go 指标 API	Go 操作和数据类型
C# SDK	完整设置指南	C# 指标 API	C# 操作和数据类型
C++ SDK	完整设置指南	C++ 指标 API	C++ 操作和数据类型

插件实现

插件	插件设置	自定义 指标	API 参考
Unreal	完整设置指南	Unreal 指标 API	Unreal 操作和数据类型
Unity	完整设置指南	Unity 指标 API	C# 操作和数据类型

实现工作流程

每个实现都需遵循一个两步流程：

1. 完整设置指南 (Metrics.md) — 基础架构部署、AWS 基础架构配置、队列设置和 Grafana 控制面板配置。
2. API 实现指南 (CUSTOM_METRICS.md)：特定于语言的 SDK 用法、指标类型、自定义指标创建和高级配置。

验证

1. 通过检查您的[亚马逊托管服务 Prometheus 工作空间](#)或 [CloudWatch 亚马逊控制](#)台中是否有传入的遥测数据，验证指标流。
2. 使用预构建的控制面板在 [Amazon Managed Grafana](#) 中检查控制面板可见性。
3. 通过验证自定义指标是否出现在您的监控控制面板中来测试这些指标。

 Note

完成实现后，返回此页面并转到[可用指标](#)页面。

可用指标

指标分为以下三个类别：

- 自动收集的指标
- SDK 提供的指标
- 自定义指标

自动指标收集

这些指标无需更改代码：

实例指标

指标类型	说明	使用场景
CPU	每个实例的利用率百分比	资源监控
内存	物理内存使用量和百分比	容量规划
网络 I/O	发送/接收的字节数和数据包数	连接运行状况
磁盘 I/O	读/写操作次数和吞吐量	存储性能

机群指标

指标类型	说明	使用场景
活动实例	正在运行的实例计数	实例集扩展
游戏会话	活动与可用的会话数量	容量管理

指标类型	说明	使用场景
崩溃的游戏会话数	已崩溃的游戏会话数	错误监控

SDK 提供的指标

需要在代码中调用 SDK 函数：

服务器计时指标

指标	说明	实施
服务器增量时间	当前服务器节拍和上一个服务器节拍之间的时间差。衡量服务器节拍率的一致性	致电 GetDeltaTime ()
服务器节拍率	显示服务器每秒处理更新的次数	自动计算
服务器节拍时间	服务器处理单个节拍或更新所用的时间	致电 GetTickTime ()
服务器世界节拍时间	服务器每个节拍更新游戏世界所用的时间	致电 GetWorldUpdateTime ()

实现：对于与引擎无关的 SDKs（C++、C#、Go），您可以通过使用计算出的计时值从游戏循环中调用 SDK 函数来实现这些指标。对于引擎插件（Unreal、Unity），这些指标将通过引擎集成自动捕获。

网络指标

指标	说明	实施
Connections	服务器已建立的网络连接总数	在 InitMetrics () 之后自动
网络 I/O（字节）	服务器通过网络发送和接收的字节总数	在 InitMetrics () 之后自动
网络 I/O（数据包）	服务器发送和接收的网络数据表总数	在 InitMetrics () 之后自动
数据包丢失	传输过程中丢失的网络数据包的百分比	在 InitMetrics () 之后自动

实现：将 SDK 函数调用与您的网络库集成。SDK 为不同的网络实现提供了指导。

进程指标

指标	说明	实施
CPU 使用率 (%)	游戏服务器进程使用的 CPU 资源的百分比	在 InitMetrics () 之后自动
内存使用量 (单位)	服务器进程使用的内存总量	在 InitMetrics () 之后自动
物理内存使用率 (%)	当前正在使用的服务器总物理内存的百分比	在 InitMetrics () 之后自动
服务器状态	游戏服务器运行状况信息	在 InitMetrics () 之后自动

实现：这些指标由 SDK 在每次游戏会话进程中自动收集。

每个进程的控制面板组织

每个进程的指标可在两个专门的控制面板中找到：

- 服务器性能控制面板 — 包含服务器计时 (增量时间、滴答率、滴答时间、世界滴答时间)、网络指标 (连接、I/O 字节/数据包、数据包丢失)、内存使用情况和单个游戏会话的 CPU 使用率。
- “实例性能”控制面板：包含“内存消耗最大的前 N 个游戏会话”和“CPU 消耗最大的前 N 个游戏会话”表，可帮助确定哪些进程消耗的实例资源最多。点击“游戏会话”链接可以更深入地调查详细指标。

每个进程的指标使用案例

每进程/ per-game-session 指标支持以下监控场景：

- 深入的性能调查 — 当由于特定进程或游戏会话而导致性能下降时，每个进程的指标可通过“占用 CPU 和内存占用量最高的游戏会话”表来帮助确定哪个进程导致了问题。host/instance
- 游戏服务器崩溃调查：当游戏会话崩溃时，这些指标可帮助确定崩溃原因是否源于内存不足、CPU 过载或网络带宽问题。
- 调查玩家报告的问题：当玩家报告称游戏过程中出现延迟或中断时，每个进程的指标有助于识别 CPU、内存、网络、节拍时间或世界更新时间中的瓶颈。

- 识别不同生成包中的性能变化：借助节拍时间、节拍率 and 世界更新时间指标，开发人员能够衡量不同服务器生成包中游戏性能的变化情况。
- 检测游戏过程中的延迟和缓慢情况：节拍时间、节拍率 and 世界更新时间指标反映了服务器更新游戏的速度，直接影响客户体验。
- 基准测试：根据玩家人数、游戏模式及其他变量等因素，识别不同游戏场景对服务器性能的影响。

控制面板组织

[Amazon Managed Grafana](#) 将指标按不同监控场景组织到专门的控制面板中。可用的控制面板取决于您的实例集类型：

EC2 舰队仪表板

- EC2 舰队概述仪表板 — 高级舰队容量、扩展见解、并发玩家 (CCU)、实例、玩家容量和游戏会话崩溃。
- “实例概述”控制面板：涵盖所有实例的主机级汇总指标，包括平均 CPU、内存、网络 and 磁盘利用率。
- “实例性能”控制面板：提供单个实例的详细指标，其中包含用于识别资源密集型进程的“内存消耗最大的前 N 个游戏会话”和“CPU 消耗最大的前 N 个游戏会话”表。
- 服务器性能控制面板 (EC2)- EC2 实例上单个游戏会话的游戏循环时间、网络性能、内存和 CPU 指标。

容器实例集控制面板

- “容器实例集概述”控制面板：容器实例集资源利用率的高级概述，包括 CPU 预留、内存利用率和容器组状态。
- “容器性能”控制面板：特定 ECS 任务中各个容器的详细指标，包括 CPU 利用率、内存使用率、网络 I/O 和存储性能。
- “服务器性能”控制面板 (容器)：提供容器中单个游戏会话的游戏循环计时、网络性能、内存及 CPU 指标。

有关控制面板的详细信息和使用说明，请参阅[the section called “控制面板组织结构和 Usage”](#)。

工作原理

遥测指标系统遵循从游戏服务器到可视化控制面板的简单四阶段数据流。

集合：您的游戏服务器与 GameLift 服务器 SDK 或插件集成，会自动向在同一实例上运行的本地遥测收集器发送指标。该 SDK 既能捕获自动指标（服务器生命周期、资源使用情况），也能捕获您在代码中定义的自定义指标。

处理：遥测收集器汇总来自游戏服务器的指标，并将其与实例级性能数据（CPU、内存、网络、磁盘使用情况）相结合。这全面展示了游戏的性能和底层基础设施。

存储：处理过的指标将导出到您选择的指标仓库中：[适用于Prometheus的亚马逊托管服务用于](#)高性能的时间序列存储，CloudWatch 亚马逊 AWS 用于服务集成，或者两者兼而有之。所有数据传输都经过身份验证和加密处理。

可视化：[Amazon Managed Grafana](#) 连接到您的指标仓库以显示预先构建的仪表板。GameLift 这些控制面板提供实例集概述、服务器性能详细信息和容器监控视图，可帮助您监控游戏托管基础设施并对其进行问题排查。

 Note

出于安全考虑，游戏服务器与遥测收集器之间的所有指标传输均在实例本地进行。只有收集器才能使用正确的身份验证与 AWS 服务通信。

控制面板组织结构和使用方式

在 [Amazon Managed Grafana](#) 的综合性控制面板上查看您的指标。可用的控制面板取决于您的实例集类型：

按实例集类型划分的控制面板可用性

下表显示了每种实例集类型可用的控制面板：

控制面板	实例集类型	说明
EC2 舰队概述	EC2 舰队	显示有关并发玩家（CCU）、实例和玩家容量的信息
实例概述	EC2 舰队	显示实例集中所有实例的平均 CPU、内存和网络利用率
实例性能	EC2 舰队	显示单个实例的详细指标（CPU、内存、磁盘、网络）

控制面板	实例集类型	说明
容器实例集概述	容器实例集	显示托管式容器实例集中所有容器的平均资源利用率
容器性能	容器实例集	显示特定 ECS 任务中各个容器的详细指标
服务器性能	二者	显示指定游戏服务器进程（和容器队列的不同版本）的网络、内存 EC2 和运行时性能

托管 EC2 车队：

- EC2 Fleet Overview 提供高级舰队容量和扩展见解。
- 使用“实例概述”和“实例性能”控制面板进行主机级监控。
- 提供通过 hostmetrics 接收器收集的指标，用于实现系统级可见性。
- 重点关注 EC2 实例资源利用率和性能。
- 服务器性能 (EC2) 独立于底层基础架构监控游戏服务器应用程序指标。

托管式容器实例集：

- 使用“容器实例集概述”和“容器性能”控制面板进行 ECS 任务和容器级监控。
- 提供通过 ECS Container Receiver 收集的指标，用于实现容器化工作负载可见性。
- 重点关注任务级聚合和容器资源隔离。
- “服务器性能”（容器）可独立于底层基础架构监控游戏服务器应用程序指标。

EC2 舰队概述控制面板

此控制面板简要概述了您的实例集在全球以及各个位置的利用率和容量。它包含图表，显示了游戏服务器停止、启动和崩溃的次数，以及运行状况良好的游戏服务器的百分比。您可以按实例集 ID 和位置进行筛选。

“实例集概述”各指标

下表显示了“实例集概述”控制面板提供的指标：

控制面板	指标名称	定义
全球 CCU 和容量	CCU	显示所有位置所有实例上所有游戏服务器中的并发用户数量
	全球 CCU	显示全球所有实例上所有游戏服务器中的并发用户数量
	活动实例	显示实例集中处于活动状态的实例总数
	活动进程	显示已准备好托管游戏会话的活动游戏服务器进程数量
	游戏服务器启动	显示实例集中已开始的游戏会话数量
全球 CCU 和容量	运行状况良好的游戏服务器	显示实例集中向 Amazon GameLift Servers 报告运行状况良好的游戏服务器的平均百分比
	崩溃的游戏会话	显示崩溃的游戏会话 IDs 的游戏会话。点击游戏会话 ARN 的链接可导航到“服务器性能”控制面板
位置 CCU	位置玩家	显示某个位置的并发玩家数量，包括该位置所有实例上所有游戏服务器中的并发玩家
	位置容量	显示某个位置的容量利用率（%），以及该位置正在使用的游戏服务器的百分比

Note

CCU 指标需要在游戏服务器代码中实现。这些指标不会自动收集，必须由您的应用程序实现并报告。

“实例概述”控制面板

此控制面板提供实例集中所有实例的主机级汇总指标。当前平均值显示了实例的总体运行状况。当性能下降时，请检查 CPU 使用率、内存消耗、网络 and 磁盘消耗是否存在瓶颈。您可以按实例集 ID 和位置进行筛选。

“实例概述”各指标

下表显示了“实例概述”控制面板提供的指标：

控制面板	指标名称	定义
实例摘要	平均 CPU 使用率	“实例摘要”跟踪实例的平均资源使用情况。该指标显示了实例集中所有实例正在使用的 CPU 资源的平均百分比。若出现高利用率模式，需要检查各个实例指标和资源分配情况
	CPU 使用率峰值	实例集中任意实例在某个时间段内记录的最高 CPU 利用率
	按实例列的 CPU 使用率	实例集中每个实例的单独 CPU 利用率，有助于识别资源热点
实例摘要	平均内存使用率	实例集中所有实例正在使用的平均 RAM 百分比。若出现高利用率模式，需要检查各个实例指标和资源分配情况，例如内存消耗
	内存使用率峰值	实例集中任意实例在某个时间段内记录的最高内存利用率
	按实例列的内存使用率	实例集中每个实例的单独内存利用率，有助于识别内存密集型工作负载
网络摘要	网络 I/O（字节）	“网络摘要”反映了平均实例连接情况。该指标显示了实例集中所有实例的平均网络流量（发送和接收）。对于延迟或连接问题，请检查各个实例指标，例如网络 I/O 和数据包速率
	网络 I/O（数据包）	实例集中所有实例的平均网络数据包速率（发送和接收）

控制面板	指标名称	定义
	按实例列的网络使用率	每个实例的单独网络利用率，有助于识别网络瓶颈
磁盘摘要	磁盘 I/O 操作	“磁盘摘要”显示实例的平均磁盘性能。所有队列实例的平均磁盘 read/write 操作量。响应时间慢建议在单个实例级别检查磁盘操作、I/O 等待时间和待处理操作
	磁盘 I/O 吞吐量	所有队列实例的平均磁盘 read/write 吞吐量
	按实例列的磁盘使用率	每个实例的单独磁盘利用率，有助于识别存储性能问题

Note

实例级指标通过 hostmetrics 接收器收集，提供对实例集基础架构性能的系统级可见性。可使用此控制面板来识别实例集的整体运行状况趋势，并在检测到性能问题时深入查看单个实例的情况。

“实例性能”控制面板

此控制面板提供各个实例的详细性能指标。当前平均值显示了实例的整体运行状况。当性能下降时，请检查 CPU 使用率、内存消耗和文件系统消耗是否存在瓶颈。它包含“内存消耗最大的前 N 个游戏会话”和“CPU 消耗最大的前 N 个游戏会话”表，可帮助确定哪些进程消耗的实例资源最多。点击“游戏会话”链接可以更深入地调查详细指标。您可以按具体实例 ID 进行筛选。

“实例性能”各指标

下表显示了“实例性能”控制面板提供的指标：

控制面板	指标名称	定义
加载摘要	内存消耗最大的前 N 个游戏会话	“负载摘要”跟踪游戏服务器的资源使用情况。该指标显示了此实例上消耗最多内存资源的游戏会话排名列表。若出现高利用率模式，需要检查各

控制面板	指标名称	定义
		个会话指标和资源分配情况。点击“游戏会话”链接可以更深入地调查详细指标
	CPU 消耗最大的前 N 个游戏会话	此实例上消耗最多 CPU 资源的游戏会话排名列表。若出现高利用率模式，需要检查各个会话指标和资源分配情况，例如 CPU 和内存。点击“游戏会话”链接可以更深入地调查详细指标
	每个游戏会话的 CPU 使用率	显示每个活跃游戏会话资源消耗情况的各个 CPU 利用率明细
	每个游戏会话的内存使用率	显示每个活跃游戏会话资源消耗情况的各个内存利用率明细
当前平均值	实例 CPU 使用率	所选实例在一段时间内的总体 CPU 利用率
	实例内存使用率	所选实例在一段时间内的总体内存利用率
	实例文件系统使用率	所选实例的文件系统消耗情况，有助于识别存储容量问题
Network	实例网络 I/O	“网络”反映了实例连接情况。该指标显示了所选实例的网络流量和数据包速率。对于延迟或连接问题，请调查网络 I/O、数据包速率和错误计数
	网络连接	所选实例上的活动网络连接数量
	网络错误计数	用于确定连接问题的网络错误统计信息
磁盘	磁盘 I/O 操作	“磁盘”显示了磁盘性能。所选实例的磁盘 read/write 活动和利用率。响应时间慢建议检查磁盘操作、I/O 等待时间和待处理操作
	磁盘 I/O 等待时间	等待磁盘操作完成所花费的时间
	待处理磁盘操作	等待处理的磁盘操作数量

 Note

“内存消耗最大的前 N 个游戏会话”和“CPU 消耗最大的前 N 个游戏会话”表，对于识别可能影响整体实例性能的性能瓶颈和资源密集型进程至关重要。这些排名可让您快速确定有问题的游戏会话，以便进一步调查。

“容器实例集概述”控制面板

此控制面板简要概述了您的容器实例集的资源利用率和容量。它显示托管式容器实例集中所有容器的平均资源利用率，包括 CPU 预留、内存利用率和容器组状态。您可以按实例集 ID 和位置进行筛选。

“容器实例集概述”各指标

下表显示了“容器实例集概述”控制面板提供的指标：

控制面板	指标名称	定义
容器组状态	活动游戏服务器容器组	当前处于活动状态并已准备好托管游戏会话的容器组
	空闲游戏服务器容器组	处于活动状态但当前未托管游戏会话的容器组
	待处理游戏服务器容器组	正处于启动过程中的容器组
	正在终止游戏服务器容器组	正处于关闭过程中的容器组
资源利用率	容器 CPU 利用率	实例集中所有容器的平均 CPU 利用率
	容器内存利用率	实例集中所有容器的平均内存利用率
	容器 CPU 预留	实例集中容器预留的 CPU 资源的百分比
网络活动	容器网络输入	实例集中容器接收的平均网络字节数
	容器网络输出	实例集中容器发送的平均网络字节数

 Note

容器实例集指标通过 ECS Container Receiver 收集，提供重点关注任务级聚合和容器资源隔离的容器化工作负载可见性。

“容器性能”控制面板

此控制面板提供特定 ECS 任务中各个容器的详细性能指标。它显示各个容器的详细指标，包括 CPU 利用率、内存使用率、网络 I/O 和存储性能。您可以按具体的容器 ID 或 ECS 任务进行筛选。

“容器性能”各指标

下表显示了“容器性能”控制面板提供的指标：

控制面板	指标名称	定义
资源性能	容器 CPU 利用率	所选容器在一段时间内的 CPU 利用率
	容器内存利用率	所选容器在一段时间内的内存利用率
	容器内存预留	所选容器预留的内存资源的百分比
网络性能	容器网络 I/O	所选容器的网络流量（输入/输出字节）
	网络连接	所选容器的活动网络连接数量
存储性能	容器存储读/写	所选容器的存储读取和写入活动
	容器存储 I/O 操作	选定容器执行的存储 I/O 操作数

 Note

容器性能指标提供了关于 ECS 任务中各个容器的资源消耗和性能特征的详细信息。

“服务器性能”控制面板

“服务器性能”控制面板显示与各个游戏会话的服务器计时、网络活动、内存和 CPU 使用率相关的指标。您可以按游戏会话 ID 进行筛选，并将指标直接导出到亚马逊 CloudWatch 或[亚马逊托管 Grafana](#)。

“服务器性能”各指标

下表显示了“服务器性能”控制面板提供的指标：

控制面板	指标名称	定义
服务器计时	服务器增量时间	当前服务器节拍和上一个服务器节拍之间的时间差。该指标用于衡量服务器节拍率的一致性
	服务器节拍率	显示服务器每秒处理更新的次数
	服务器节拍时间	服务器处理单个节拍或更新所用的时间
	服务器世界节拍时间	服务器每个节拍更新游戏世界所用的时间
Network	Connections	服务器已建立的网络连接总数
	网络 I/O（字节）	服务器通过网络发送和接收的字节总数
	网络 I/O（数据包）	服务器发送和接收的网络数据表总数
	数据包丢失	传输过程中丢失的网络数据包的百分比
内存	内存使用量（单位）	服务器进程使用的内存总量
	物理内存使用率（%）	当前正在使用的服务器总物理内存的百分比
CPU 使用率	CPU 使用率（%）	游戏服务器进程使用的 CPU 资源的百分比

常见监控场景

深入的性能调查

场景：A host/instance 由于特定进程或游戏会话而导致性能下降

调查步骤：

- 访问“实例性能”控制面板。
- 查看“内存消耗最大的前 N 个游戏会话”表，确定哪些进程消耗的实例内存最多。
- 查看“CPU 消耗最大的前 N 个游戏会话”表，确定哪些进程使用的实例 CPU 最多。
- 点击“游戏会话”链接，更深入地调查详细指标。
- 分析服务器计时（服务器增量时间、服务器节拍率、服务器节拍时间、服务器世界节拍时间）以识别性能瓶颈。

游戏服务器崩溃调查

场景：一个游戏会话发生了崩溃，您需要确定根本原因

调查步骤：

- 访问“服务器性能”控制面板，查看崩溃的游戏会话。
- 检查“内存使用量（单位）”和“物理内存使用率（%）”，确定崩溃是否是内存不足所致。
- 查看“CPU 使用率（%）”，确定崩溃是否是 CPU 过载所致。
- 分析网络 I/O（字节）和网络 I/O（数据包），以确定网络带宽问题是否导致崩溃。
- 检查“数据包丢失”百分比以识别网络相关问题。

调查玩家报告的问题

场景：玩家报告称游戏过程中出现延迟或中断

调查步骤：

- 访问“服务器性能”控制面板，查看受影响的游戏会话。
- 查看“服务器节拍时间”和“服务器世界节拍时间”，确定游戏更新是否有延迟。
- 检查“服务器节拍率”，确保服务器更新频率保持一致。
- 分析“CPU 使用率（%）”以识别处理瓶颈。
- 查看“内存使用率”各指标以识别与内存相关的性能问题。
- 检查网络 I/O 指标和数据包丢失以确定网络瓶颈。

识别不同游戏服务器生成包中的性能变化

场景：您想衡量不同服务器生成包中游戏性能的变化情况

调查步骤：

- 比较不同生成包之间的“服务器节拍时间”指标，以衡量处理效率的变化。
- 分析不同生成包之间的“服务器节拍率”一致性，确定是否有性能下降情况。
- 查看“服务器世界节拍时间”以衡量游戏世界更新性能的变化。
- 比较不同生成包之间的内存使用模式，确定是否有内存优化改进或倒退情况。
- 监控 CPU 使用率趋势以评测计算效率变化。

检测游戏过程中的延迟和缓慢情况

场景：您需要监控服务器响应速度和游戏更新速度

调查步骤：

- 监控“服务器节拍时间”，以衡量服务器处理每个更新周期的速度。
- 跟踪“服务器节拍率”，确保每秒游戏状态更新的一致性。
- 分析“服务器世界节拍时间”以衡量游戏世界更新速度，这会直接影响客户体验。
- 针对“服务器增量时间”变化设置警报，以检测服务器性能不一致的情况。

对不同的游戏场景进行基准测试

场景：您想确定不同的游戏场景如何影响服务器性能

调查步骤：

- 比较不同玩家数量的服务器性能指标，以了解扩缩的影响。
- 查看“服务器节拍时间”和“CPU 使用率”指标，分析不同游戏模式之间的性能差异。
- 监控不同游戏场景中的内存使用模式，以识别资源密集型功能。
- 跟踪网络 I/O 指标以了解不同游戏场景的带宽需求。
- 使用“实例性能”控制面板，确定哪些游戏场景产生了资源消耗量最大的游戏会话。

对高资源利用情况的响应

场景：异常的资源消耗峰值 (CPU > 85% , 内存 > 90%)

调查步骤：

确定受影响的资源

- 使用 DescribeGameSessionDetails API。
- 如果需要，按状态筛选。
- 记录受影响的实例。

分析资源使用情况

- 查看“实例概述”控制面板。
- 比较实例集中实例的利用率。
- 查看历史模式。

监控游戏服务器影响

- 检查“服务器性能”各指标。
- 查看节拍时间和数据表丢失情况。
- 监控内存泄漏。

解决步骤

- 下载会话日志。
- 解决生成包问题。
- 监控改进。

游戏服务器崩溃分析

场景：整个实例集出现多个错误状态的游戏会话

调查步骤：

初步评测

- 访问“实例集概述”控制面板。
- 查看崩溃的会话表。
- 注意时间/位置方面的模式。

执行分析

- 检查服务器计时指标。
- 查看资源利用情况。
- 监控网络性能。

基础架构审查

- 验证实例集容量。
- 检查实例运行状况。
- 查看扩缩策略。

解决方案路径

- 分析服务器日志。
- 查看代码优化情况。
- 实施修复。

实例集容量优化

场景：游戏发布或基准研究

分析步骤：

资源利用情况

- 按位置筛选。
- 查看 P50/P95/P99 指标。
- 分析使用模式。

实例类型分析

- 按类型比较性能。
- 确定扩缩候选项。
- 记录资源利用模式。

优化操作

- 调整扩缩策略。
- 修改实例类型。
- 更新实例集配置。

问题排查指南

常见问题和解决步骤

指标缺失或不完整

症状

- 控制面板中未显示任何指标。
- 部分指标收集。
- 指标更新延迟。

解决步骤

A. 验证收集器状态

检查 systemd 服务：

```
sudo systemctl status gamelift-telemetry-collector
```

审核收集器日志：

```
sudo journalctl -u gamelift-telemetry-collector
```

- 确认收集器配置。

B. IAM 权限验证

- 检查实例角色权限。
- 验证所需的策略：
 - `aps:RemoteWrite`
 - `cloudwatch:PutMetricData`
- 验证角色信任关系。

C. 网络连接

- 验证端点访问权限。
- 检查安全组规则：
- 评论网络 ACLs。

身份验证错误

症状

- SigV4 身份验证失败。
- “访问被拒绝”消息。
- 凭证刷新问题。

解决步骤

A. SigV4 身份验证

- 验证临时凭证。
- 检查凭证轮换。
- 验证实例配置文件。

B. AMP 访问权限

- 审核工作空间配置。
- 验证远程写入 URL。
- 检查 IAM 角色绑定。

控制面板问题

症状

- 控制面板为空。
- 数据点缺失。
- 身份验证失败。

解决步骤

A. 数据来源配置

- 验证 Prometheus 连接。
- 查看亚马逊 CloudWatch 整合。
- 测试数据来源权限。

B. Grafana 访问权限

- 确认 SSO 配置。
- 如果需要，请验证 2FA 设置。
- 检查用户权限

特定于 Windows 的问题

症状

- 服务启动失败。
- 指标采集缺失。
- 权限错误。

解决步骤

- 验证 Windows 服务状态。
- 检查 Windows 事件日志。
- 审核收集器配置。
- 验证特定于 Windows 的路径。

使用记录 Amazon GameLift Servers API 调用 AWS CloudTrail

Amazon GameLift Servers与 AWS CloudTrail一项服务集成，该服务提供用户、角色或 AWS 服务在中执行的操作的记录Amazon GameLift Servers。CloudTrail 将所有 API 调用捕获Amazon GameLift Servers为事件。捕获的调用包含来自 Amazon GameLift Servers 控制台和代码的 Amazon GameLift Servers API 操作调用。如果您创建了跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括的事件Amazon GameLift Servers。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的事件历史记录中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向哪个请求发出Amazon GameLift Servers、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅[AWS CloudTrail 用户指南](#)。

Amazon GameLift Servers信息在 CloudTrail

CloudTrail 在您创建账户 AWS 账户 时已在您的账户上启用。当活动发生在中时Amazon GameLift Servers，该活动会与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在中查看、搜索和下载最近发生的事件 AWS 账户。有关更多信息，请参阅[使用事件历史记录查看 CloudTrail 事件](#)。

对于 AWS 账户中的事件的持续记录（包括 Amazon GameLift Servers 的事件），请创建跟踪记录。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。默认情况下，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。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅以下内容：

- [创建跟踪记录概述](#)
- [CloudTrail 支持的服务和集成](#)
- [配置 Amazon SNS 通知 CloudTrail](#)
- [接收来自多个地区的 CloudTrail 日志文件](#)和[接收来自多个账户的 CloudTrail 日志文件](#)

所有Amazon GameLift Servers操作均由 API 参考记录 CloudTrail 并记录在 [Amazon GameLift ServersAPI 参考](#)中。例如，调

用CreateGameSessionCreatePlayerSession和UpdateGameSession操作会在 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。身份信息有助于您确定以下内容：

- 请求是使用根证书还是 AWS Identity and Access Management (IAM) 用户凭证发出。

- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅 [CloudTrail userIdentity 元素](#)。

了解 Amazon GameLift Servers 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序出现。

以下示例显示了一个演示CreateFleet和DescribeFleetAttributes操作的 CloudTrail 日志条目。

```
{
  "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"
    ],
    "e2InstanceType": "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 Servers 中记录服务器消息

您可以在日志文件中捕获来自 Amazon GameLift Servers 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 Amazon GameLift ServersRealtime，以及您使用的是托管 EC2 舰队还是集装箱舰队（请参阅本章中的相应小节）。

主题

- [日志差异：托管舰队 EC2 与容器舰队](#)
- [记录服务器消息（自定义服务器）](#)

日志差异：托管舰队 EC2 与容器舰队

托管 EC2 船队和集装箱船队之间的日志记录行为差异很大。在从容器迁移 EC2 到容器或根据日志要求选择正确的舰队类型时，了解这些差异至关重要。

主要区别概述

功能	托管 EC2 车队	容器实例集
日志存储	游戏会话结束后自动上传到 Amazon S3	必须配置 Amazon CloudWatch 日志或自定义日志解决方案

功能	托管 EC2 车队	容器实例集
GetGameSessionLogUrl API	可用：返回用于下载日志的 Amazon S3 URL	不可用：日志不会自动存储在 Amazon S3 中
日志保留	在 Amazon S3 中保留 14 天（自动）	取决于日志记录配置
实时监控	有限：日志仅在游戏会话结束后可用	与 Amazon CloudWatch Logs 集成一起可用
设置复杂性	自动：无需额外配置	需要明确的日志记录配置

托管 EC2 舰队日志

对于托管 EC2 舰队，Amazon GameLift Servers 提供自动日志管理：

- 自动上传：游戏会话结束后，服务器日志会自动上传到 Amazon S3
- GetGameSessionLogUrl API：使用此 API 检索用于从 Amazon S3 下载日志的预签名 URL
- 14 天保留期：日志在 Amazon S3 中保留 14 天，然后自动删除
- 大小限制：每个游戏会话的日志文件都有大小限制（请参阅 [Amazon GameLift Servers 端点和配额](#)）

有关更多信息，请参阅 [the section called “自定义服务器的日志记录”](#) 和 [GetGameSessionLogUrl API 参考](#)。

容器实例集日志记录

容器实例集需要您明确配置日志记录：

- 不自动上传到 Amazon S3：日志不会自动上传到 Amazon S3
- GetGameSessionLogUrl 不可用：此 API 不适用于集装箱舰队
- Amazon CloudWatch Logs 集成：将您的容器配置为将日志发送到 Amazon CloudWatch Logs 以进行集中记录
- 自定义日志记录解决方案：使用日志驱动程序或附加容器实施自己的日志记录基础架构
- 实时访问：通过适当的配置，可以在游戏会话期间实时访问日志

有关详细的容器日志记录选项，请参阅[容器实例集的工作原理](#)。

迁移注意事项

从托管舰队迁移 EC2 到容器舰队时，请考虑以下日志变更：

- 更新日志检索代码：将 GetGameSessionLogUrl API 调用替换为 Amazon CloudWatch Logs 查询或您的自定义日志记录解决方案
- 配置日志保留：在 Amazon Lo CloudWatch gs 或您的日志系统中设置适当的保留策略
- 实现实时监控：利用实时日志访问来提高可观测性
- 查看日志量和成本：Amazon Lo CloudWatch gs 定价与托管 EC2 队列中包含的 Amazon S3 存储不同

推荐的容器日志记录设置

对于容器实例集，我们建议采用以下设置：

1. Amazon CloudWatch Logs 集成：配置您的容器定义以使用awslogs日志驱动程序
2. 结构化日志记录：使用结构化日志格式（JSON）提高可搜索性和分析能力
3. 日志级别：实施适当的日志级别以控制详细程度和成本
4. 保留策略：根据您的合规性和调试需求设置保留期

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 Amazon GameLift Servers 自定义服务器的自定义服务器消息。

Important

每个游戏会话的日志文件大小有限制（请参阅 AWS 一般参考中的 [Amazon GameLift Servers 端点和配额](#)）。游戏会话结束时，Amazon GameLift Servers 会将服务器日志上传到 Amazon Simple Storage Service（Amazon S3）。Amazon GameLift Servers 将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 Servers 自定义服务器时，您可以自行编写代码来执行日志记录，将其作为服务器进程配置的一部分进行配置。Amazon GameLift Servers 使用您的日志记录配置来识别在每个游戏会话结束时必须上传到 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. 提供你的矢量作为你的[ProcessParameters](#)对象[LogParameters](#)的向量。

```
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

日志访问因实例集类型而异：

托管式 EC2 实例集

游戏会话结束后，Amazon GameLift Servers 会自动将日志存储在 S3 存储桶中，并保留这些日志 14 天。要获取游戏会话日志的位置，可以使用 [GetGameSessionLogUri](#) API 操作。要下载日志，请使用操作返回的 URL。

或者，您可以将游戏服务器配置为将日志直接发送到您的首选日志记录服务或存储位置，从而设置自己的日志记录解决方案。有关更多信息，请参阅 [将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。

容器实例集

容器实例集可捕获所有容器中的标准输出和错误流。您可使用以下选项配置日志记录：

- 将容器输出另存为 CloudWatch 指定日志组中的日志流
- 将容器输出保存到 S3 存储桶
- 关闭日志记录（不保存容器输出）

有关在创建容器队列时配置日志记录选项的详细信息，请参阅 Amazon GameLift Servers API 参考 [LogConfiguration](#) 中的。

Amazon GameLift Servers 中的安全性

如果您将 Amazon GameLift Servers FleetIQ 作为独立功能与 Amazon EC2 搭配使用，请参阅《Amazon EC2 用户指南》中的 [Amazon EC2 中的安全性](#)。

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](#)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。要了解适用的合规计划 Amazon GameLift Servers，请参阅按合规计划划分的[范围内的 AWS 服务（按合规计划）](#)。
- 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。

此文档将帮助您了解如何在使用 Amazon GameLift Servers 时应用责任共担模型。以下主题说明如何配置 Amazon GameLift Servers 以实现您的安全性和合规性目标。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 Servers 资源。

主题

- [Amazon GameLift Servers 中的数据保护](#)
- [对 Amazon GameLift Servers 进行身份和访问管理](#)
- [使用 Amazon GameLift Servers 进行日志记录和监控](#)
- [的合规性验证 Amazon GameLift Servers](#)
- [Amazon GameLift Servers 中的故障恢复能力](#)
- [Amazon GameLift Servers 中的基础结构安全性](#)
- [Amazon GameLift Servers 中的配置和漏洞分析](#)
- [Amazon GameLift Servers 的安全最佳实践](#)

Amazon GameLift Servers 中的数据保护

如果您将亚马逊 Amazon GameLift Servers FleetIQ 作为独立功能使用 EC2，请参阅《亚马逊 EC2 用户指南》 EC2 中的“亚马逊[安全](#)”。

分 AWS [担责任模型](#) 适用于中的数据保护 Amazon GameLift Servers。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结构上的内容的控制。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 Security Blog 上的 [AWS Shared Responsibility Model and 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。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)。
- 用于 SSL/TLS 与 AWS 资源通信。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。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅《AWS CloudTrail 用户指南》中的[使用跟 CloudTrail 踪](#)。
- 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 140-3 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 140-3 版》<https://aws.amazon.com/compliance/fips/>。

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）。这包括您使用控制台、API Amazon GameLift Servers 或以其他 AWS 服务方式使用控制台 AWS CLI、API 或 AWS SDKs。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 URL，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。

Amazon GameLift Servers 特定数据的处理方式如下：

- 您上传到 Amazon GameLift Servers 的游戏服务器生成包和脚本存储在 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 Servers 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 Amazon GameLift Servers 控制台中的链接或者通过调用服务 API 来访问日志数据。

- 指标和事件数据存储在 Amazon GameLift Servers 中，可以通过 Amazon GameLift Servers 控制台或者通过调用服务 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 Servers 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 IDs 中提供了自定义玩家，则这些值应该是匿名的，UUIDs 并且不包含任何可识别玩家的信息。

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 Servers 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在 Amazon GameLift Servers 中。

传输中加密

与的 Amazon GameLift Servers APIs 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：

- 对于在 Amazon GameLift Servers 资源上托管的自定义游戏服务器，通信不涉及该 Amazon GameLift Servers 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。
- 启用 TLS 的车队的证书与车队同时创建，其到期日期基于舰队的创建日期。
- 对于启用了 TLS 证书生成功能的 Amazon GameLift Servers Realtime，游戏客户端与使用 Realtime 客户端开发工具包的 Realtime 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。

 Note

为了满足证书颁发机构对 TLS 证书的要求，Amazon GameLift Servers 将调整配置为生成证书的队列的最大证书生命周期。证书有效期从创建队列时开始，并将按以下时间表进行更改：

- 在 2026 年 3 月 11 日之前，颁发的 TLS 证书的最长有效期为 398 天。
- 自 2026 年 3 月 1 日起，颁发的 TLS 证书的最长有效期为 200 天。
- 自 2027 年 3 月 1 日起，颁发的 TLS 证书的最长有效期为 100 天。
- 自 2029 年 3 月 1 日起，颁发的 TLS 证书的最长有效期为 47 天。

为了确保您的证书得到续订并维护 up-to-date 游戏服务器的运行时环境，Amazon GameLift Servers 建议您定期更换游戏服务器舰队。

互连网络流量隐私

您可以安全地远程访问您的 Amazon GameLift Servers 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 Servers FleetIQ，使用 S AWS ystems Manager 会话管理器和运行命令对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。有关连接到托管 Amazon GameLift Servers 实例的帮助，请参阅[远程连接到 Amazon GameLift Servers 实例集实例](#)。

对 Amazon GameLift Servers 进行身份和访问管理

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以通过身份验证（登录）和授权（具有权限）来使用 Amazon GameLift Servers 资源。您可以使用 IAM AWS 服务，无需支付额外费用。

主题

- [受众](#)
- [使用身份进行身份验证](#)
- [使用策略管理访问](#)
- [Amazon GameLift Servers 如何与 IAM 协同工作](#)
- [Amazon GameLift Servers 基于身份的策略示例](#)
- [排除 Amazon GameLift Servers 身份和访问问题](#)

- [AWS 的托管策略 Amazon GameLift Servers](#)

受众

您的使用方式 AWS Identity and Access Management (IAM) 因您的角色而异：

- 服务用户：如果您无法访问功能，请从管理员处请求权限（请参阅[排除 Amazon GameLift Servers 身份和访问问题](#)）
- 服务管理员：确定用户访问权限并提交权限请求（请参阅[Amazon GameLift Servers 如何与 IAM 协同工作](#)）
- IAM 管理员：编写用于管理访问权限的策略（请参阅[Amazon GameLift Servers 基于身份的策略示例](#)）

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份进行身份验证 AWS 账户根用户，或者通过担任 IAM 角色进行身份验证。

您可以使用来自身份源的证书 AWS IAM Identity Center（例如（IAM Identity Center）、单点登录身份验证或 Google/Facebook 证书，以联合身份登录。有关登录的更多信息，请参阅《AWS 登录 用户指南》中的[如何登录您的 AWS 账户](#)。

对于编程访问，AWS 提供 SDK 和 CLI 来对请求进行加密签名。有关更多信息，请参阅《IAM 用户指南》中的[适用于 API 请求的 AWS 签名版本 4](#)。

AWS 账户 root 用户

创建时 AWS 账户，首先会有一个名为 AWS 账户 root 用户的登录身份，该身份可以完全访问所有资源 AWS 服务 和资源。我们强烈建议不要使用根用户进行日常任务。有关需要根用户凭证的任务，请参阅《IAM 用户指南》中的[需要根用户凭证的任务](#)。

联合身份

作为最佳实践，要求人类用户使用与身份提供商的联合身份验证才能 AWS 服务 使用临时证书进行访问。

联合身份是指来自您的企业目录、Web 身份提供商的用户 Directory Service，或者 AWS 服务 使用来自身份源的凭据进行访问的用户。联合身份代入可提供临时凭证的角色。

要集中管理访问权限，建议使用。AWS IAM Identity Center 有关更多信息，请参阅《AWS IAM Identity Center 用户指南》中的[什么是 IAM Identity Center？](#)。

IAM 用户和群组

[IAM 用户](#)是对某个人员或应用程序具有特定权限的一个身份。建议使用临时凭证，而非具有长期凭证的 IAM 用户。有关更多信息，请参阅 IAM 用户指南中的[要求人类用户使用身份提供商的联合身份验证才能 AWS 使用临时证书进行访问](#)。

[IAM 组](#)指定一组 IAM 用户，便于更轻松地对大量用户进行权限管理。有关更多信息，请参阅《IAM 用户指南》中的[IAM 用户使用案例](#)。

IAM 角色

[IAM 角色](#)是具有特定权限的身份，可提供临时凭证。您可以通过[从用户切换到 IAM 角色（控制台）](#)或调用 AWS CLI 或 AWS API 操作来代入角色。有关更多信息，请参阅《IAM 用户指南》中的[担任角色的方法](#)。

IAM 角色对于联合用户访问、临时 IAM 用户权限、跨账户访问、跨服务访问以及在 Amazon EC2 上运行的应用程序非常有用。有关更多信息，请参阅《IAM 用户指南》中的[IAM 中的跨账户资源访问](#)。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略定义了与身份或资源关联时的权限。AWS 在委托人提出请求时评估这些政策。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的更多信息，请参阅《IAM 用户指南》中的[JSON 策略概述](#)。

管理员使用策略，通过定义哪个主体可以在什么条件下对哪些资源执行哪些操作来指定谁有权访问什么。

默认情况下，用户和角色没有权限。IAM 管理员创建 IAM 策略并将其添加到角色中，然后用户可以担任这些角色。IAM 策略定义权限，与执行操作所用的方法无关。

基于身份的策略

基于身份的策略是您附加到身份（用户、组或角色）的 JSON 权限策略文档。这些策略控制身份可以执行什么操作、对哪些资源执行以及在什么条件下执行。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的[使用客户管理型策略定义自定义 IAM 权限](#)。

基于身份的策略可以是内联策略（直接嵌入到单个身份中）或托管策略（附加到多个身份的独立策略）。要了解如何在托管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。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。您必须在基于资源的策略中[指定主体](#)。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

其他策略类型

AWS 支持其他策略类型，这些策略类型可以设置更常见的策略类型授予的最大权限：

- 权限边界 – 设置基于身份的策略可以授予 IAM 实体的最大权限。有关更多信息，请参阅《IAM 用户指南》中的[IAM 实体的权限边界](#)。
- 服务控制策略 (SCPs)-在中指定组织或组织单位的最大权限 AWS Organizations。有关更多信息，请参阅《AWS Organizations 用户指南》中的[服务控制策略](#)。
- 资源控制策略 (RCPs)-设置账户中资源的最大可用权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的[资源控制策略 \(RCPs\)](#)。
- 会话策略 – 在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。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。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的[策略评估逻辑](#)。

Amazon GameLift Servers 如何与 IAM 协同工作

在使用 IAM 管理对 Amazon GameLift Servers 的访问之前，您应该了解哪些 IAM 功能可用于 Amazon GameLift Servers。

可以与 Amazon GameLift Servers 搭配使用的 IAM 功能

IAM 功能	Amazon GameLift Servers 支持
基于身份的策略	是

IAM 功能	Amazon GameLift Servers 支持
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
ACLs	否
ABAC（策略中的标签）	是
临时凭证	是
主体权限	是
服务角色	是
服务关联角色	否

要全面了解Amazon GameLift Servers以及其他 AWS 服务如何与大多数 IAM 功能配合使用，请参阅 IAM 用户指南中的与 IAM [配合使用的AWS 服务](#)。

适用于 Amazon GameLift Servers 的基于身份的策略

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[使用客户管理型策略定义自定义 IAM 权限](#)。

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。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 GameLift Servers 基于身份的策略示例

要查看 Amazon GameLift Servers 基于身份的策略的示例，请参阅 [Amazon GameLift Servers 基于身份的策略示例](#)

Amazon GameLift Servers 内基于资源的策略

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。您必须在基于资源的策略中[指定主体](#)。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户访问，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。有关更多信息，请参阅《IAM 用户指南》中的[IAM 中的跨账户资源访问](#)。

适用于 Amazon GameLift Servers 的策略操作

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。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。在策略中包含操作以授予执行关联操作的权限。

要查看 Amazon GameLift Servers 操作的列表，请参阅服务授权参考中的[Amazon GameLift Servers 定义的操作](#)。

Amazon GameLift Servers 中的策略操作在操作前使用以下前缀：

```
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 Servers 基于身份的策略的示例，请参阅[Amazon GameLift Servers 基于身份的策略示例](#)。

Amazon GameLift Servers 的策略资源

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。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。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。作为最佳实践，请使用其 [Amazon 资源名称 \(ARN\)](#) 指定资源。对于不支持资源级权限的操作，请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

有关 Amazon GameLift Servers 资源类型及其列表 ARNs，请参阅《服务授权参考》[Amazon GameLift Servers 中定义的资源](#)。要了解可以在哪些操作中指定每个资源的 ARN，请参阅 [Amazon GameLift Servers 定义的操作](#)。

有些 Amazon GameLift Servers 资源具有 ARN 值，借助这些值，可使用 IAM 策略对这些资源的访问权限进行管理。Amazon GameLift Servers 队组资源具有采用以下语法的 ARN：

```
arn:${Partition}:gamelift:${Region}:${Account}:fleet/${FleetId}
```

有关格式的更多信息 ARNs，请参阅中的 [Amazon 资源名称 \(ARNs\) AWS 一般参考](#)。

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 Servers 基于身份的策略的示例，请参阅 [Amazon GameLift Servers 基于身份的策略示例](#)

Amazon GameLift Servers 的策略条件键

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。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。

Condition 元素根据定义的条件指定语句何时执行。您可以创建使用[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。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的[AWS 全局条件上下文密钥](#)。

有关 Amazon GameLift Servers 条件密钥的列表，请参阅服务授权参考中的 [Amazon GameLift Servers 的条件密钥](#)。要了解您可以对哪些操作和资源使用条件键，请参阅 [Amazon GameLift Servers 定义的操作](#)。

要查看 Amazon GameLift Servers 基于身份的策略的示例，请参阅 [Amazon GameLift Servers 基于身份的策略示例](#)

ACLs 在 Amazon GameLift Servers

支持 ACLs：否

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。ACLs 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

带有 Amazon GameLift Servers 的 ABAC

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，该策略基于称为标签的属性来定义权限。您可以将标签附加到 IAM 实体和 AWS 资源，然后设计 ABAC 策略以允许在委托人的标签与资源上的标签匹配时进行操作。

要基于标签控制访问，您需要使用 `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[使用 ABAC 授权定义权限](#)。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 Servers 队组](#)。

将临时凭证用于 Amazon GameLift Servers

支持临时凭证：是

临时证书提供对 AWS 资源的短期访问权限，并且是在您使用联合身份或切换角色时自动创建的。AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅《IAM 用户指南》中的[IAM 中的临时安全凭证](#)和[使用 IAM 的 AWS 服务](#)。

Amazon GameLift Servers 的跨服务主体权限

支持转发访问会话 (FAS)：是

转发访问会话 (FAS) 使用调用主体的权限 AWS 服务，再加上 AWS 服务 向下游服务发出请求的请求。有关发出 FAS 请求时的策略详细信息，请参阅[转发访问会话](#)。

Amazon GameLift Servers 的服务角色

支持服务角色：是

服务角色是由一项服务担任、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破坏 Amazon GameLift Servers 的功能。只有在 Amazon GameLift Servers 提供指导时，才能编辑服务角色。

允许 Amazon GameLift Servers 托管的游戏服务器访问其他 AWS 资源，例如 AWS Lambda 函数或 Amazon DynamoDB 数据库。由于游戏服务器托管在 Amazon GameLift Servers 管理的队列上，因此您需要一个服务角色来 Amazon GameLift Servers 限制对其他 AWS 资源的访问权限。有关更多信息，请参阅 [将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。

Amazon GameLift Servers 的服务相关角色

支持服务相关角色：否

服务相关角色是一种与服务相关联的 AWS 服务角色。服务可以代入代表您执行操作的角色。服务相关角色出现在您的 AWS 账户，并且归服务所有。IAM 管理员可以查看但不能编辑服务关联角色的权限。

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 GameLift Servers 基于身份的策略示例

默认情况下，用户和角色没有创建或修改 Amazon GameLift Servers 资源的权限。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。

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 策略（控制台）](#)。

有关由 Amazon GameLift Servers 定义的操作和资源类型（包括每种资源类型的格式）的详细信息，请参阅《服务授权参考》Amazon GameLift Servers 中的[操作、资源和条件密钥](#)。ARNs

主题

- [策略最佳实践](#)
- [使用 Amazon GameLift Servers 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 Servers 队列](#)
- [根据标签查看 Amazon GameLift Servers 群组](#)
- [在 Amazon S3 中访问游戏构建文件](#)

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 Amazon GameLift Servers 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，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的 AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管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 托管策略](#)或[工作职能的 AWS 托管策略](#)。

- 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，您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，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定 AWS 服务的（例如）使用的，则也可以使用条件来授予对服务操作的访问权限 CloudFormation。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，以帮助您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 [使用 IAM Access Analyzer 验证策略](#)。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的 [使用 MFA 保护 API 访问](#)。

有关 IAM 中的最佳实操的更多信息，请参阅 IAM 用户指南中的 [IAM 中的安全最佳实操](#)。

使用 Amazon GameLift Servers 控制台

要访问 Amazon GameLift Servers 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 Servers 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。

要确保这些实体仍然可以使用 Amazon GameLift Servers 控制台，请使用以下示例和 [管理权限示例](#) 中的语法向用户和组添加权限。有关更多信息，请参阅 [为 Amazon GameLift Servers 设置用户权限](#)。

使用 Amazon GameLift Servers 直通 AWS CLI 或 AWS API 操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 Amazon GameLift Servers 控制台所有功能所需的权限的信息，请参阅 [管理权限示例](#) 中的管理员权限语法。

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和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Sid": "ViewOwnUserInfo",
    "Effect": "Allow",
    "Action": [
      "iam:GetUserPolicy",
      "iam:ListGroupsWithUser",
      "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 Servers 队列

以下示例为用户提供了访问特定的 Amazon GameLift Servers 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。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewSpecificQueueInfo",
      "Effect": "Allow",
      "Action": [
        "gamelift:DescribeGameSessionQueues"
      ],
      "Resource": "arn:aws:gamelift:us-east-1:555555555555:gamesessionqueue/examplequeue123"
    },
    {
      "Sid": "ManageSpecificQueue",
      "Effect": "Allow",
      "Action": [
        "gamelift:UpdateGameSessionQueue",
        "gamelift>DeleteGameSessionQueue"
      ],
      "Resource": "arn:aws:gamelift:us-east-1:111122223333:gamesessionqueue/examplequeue123"
    }
  ]
}
```

根据标签查看 Amazon GameLift Servers 群组

您可以在基于身份的策略中使用条件，以便基于标签控制对 Amazon GameLift Servers 资源的访问。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。

JSON

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

```

    "Statement": [
      {
        "Sid": "ListFleetsInConsole",
        "Effect": "Allow",
        "Action": "gamelift:ListFleets",
        "Resource": "*"
      },
      {
        "Sid": "ViewFleetIfOwner",
        "Effect": "Allow",
        "Action": "gamelift:DescribeFleetAttributes",
        "Resource": "arn:aws:gamelift:*:*:fleet/*",
        "Condition": {
          "StringEquals": {"aws:ResourceAccount": "${aws:username}"}
        }
      }
    ]
  }
}

```

在 Amazon S3 中访问游戏构建文件

将游戏服务器与 Amazon GameLift Servers 集成后，将生成包文件上传到 Amazon S3。要让 Amazon GameLift Servers 访问生成包文件，请使用以下策略。

JSON

```

{
  "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 Servers 游戏文件的更多信息，请参阅[为 Amazon GameLift Servers 创建游戏服务器生成包](#)。

排除 Amazon GameLift Servers 身份和访问问题

使用以下信息来帮助您诊断和修复使用 Amazon GameLift Servers 和 AWS Identity and Access Management (IAM) 时可能遇到的常见问题。

主题

- [我无权在 Amazon GameLift Servers 中执行操作](#)
- [我无权执行 iam : PassRole](#)
- [我想允许我以外的人 AWS 账户 访问我的 Amazon GameLift Servers 资源](#)

我无权在 Amazon GameLift Servers 中执行操作

如果 AWS 管理控制台 告诉您您无权执行某项操作，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

如果您收到一个错误，表明您无权执行 iam:PassRole 操作，则必须更新策略以允许您将角色传递给。Amazon GameLift Servers

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的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 marymajor 的 IAM 用户尝试使用控制台在 Amazon GameLift Servers 中执行操作时，会发生以下示例错误。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。

如果您需要帮助，请联系您的 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 AWS 账户 访问我的 Amazon GameLift Servers 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。您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 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon GameLift Servers 是否支持这些特征，请参阅 [Amazon GameLift Servers 如何与 IAM 协同工作](#)
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户，请参阅 [IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问](#) 权限。
- 要了解如何向第三方提供对您的资源的访问 [权限 AWS 账户](#)，请参阅 [IAM 用户指南中的向第三方提供访问](#) 权限。AWS 账户
- 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 中的跨账户资源访问](#)。

AWS 的托管策略 Amazon GameLift Servers

AWS 托管策略是由创建和管理的独立策略 AWS。AWS 托管策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住，AWS 托管策略可能不会为您的特定用例授予最低权限权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于使用案例的 [客户管理型策略](#) 来进一步减少权限。

您无法更改 AWS 托管策略中定义的权限。如果 AWS 更新 AWS 托管策略中定义的权限，则更新会影响该策略所关联的所有委托人身份（用户、组和角色）。AWS 最有可能在启动新的 API 或现有服务可以使用新 AWS 服务的 API 操作时更新 AWS 托管策略。

有关更多信息，请参阅《IAM 用户指南》中的 [AWS 托管策略](#)。

AWS 托管策略：GameLiftContainerFleetPolicy

您可以将 GameLiftContainerFleetPolicy 附加到 IAM 角色。

该策略授予在 Amazon GameLift Servers 容器实例集中执行计算操作的权限。容器舰队是一组为您 Amazon GameLift Servers 管理的托管资源。Amazon GameLift Servers 需要权限才能代表您连接到 Amazon GameLift Servers AWS 服务和其他服务。

使用创建容器队列时 Amazon GameLift Servers，请提供一个附加 GameLiftContainerFleetPolicy 托管策略的 IAM 服务角色。有关如何创建服务角色的说明，请参阅 [为 Amazon GameLift Servers 设置 IAM 服务角色](#)。

有关更多信息，请参阅 [GameLiftContainerFleetPolicy](#)。

权限详细信息

该策略包含以下权限。

- `cloudwatch`— Amazon GameLift Servers 允许将游戏会话日志写入您 AWS 账户中的 Amazon EventBridge 日志流。
- `cloudwatch` — Amazon GameLift Servers 允许创建 CloudWatch 日志组来组织日志流中的游戏会话数据。
- `s3`— Amazon GameLift Servers 允许将游戏会话日志写入您 AWS 账户中的 Amazon 简单存储服务存储桶。
- `s3`— Amazon GameLift Servers 允许使用 API 操作检索指定的 Amazon S3 存储桶所在的位置 `s3:GetBucketLocation`。AWS 区域
- `gamelift` : 允许 Amazon GameLift Servers 检索身份验证令牌，该令牌允许托管游戏服务器通过您的 AWS 账户与 Amazon GameLift Servers 服务进行通信。

Amazon GameLift Servers AWS 托管策略的更新

查看 Amazon GameLift Servers 自该服务开始跟踪这些更改以来 AWS 托管策略更新的详细信息。要获得有关此页面更改的自动提示，请订阅 [Amazon GameLift Servers 发布说明](#) 页面上的 RSS 馈送。

更改	描述	日期
GameLiftContainerFleetPolicy – 更改	Amazon GameLift Servers 添加了检索 Amazon S3 存储桶的新权限。AWS 区域	2024 年 2 月 5 日
GameLiftContainerFleetPolicy : 新策略	Amazon GameLift Servers 添加了新的权限，使游戏服务器容器能够在 Amazon GameLift Servers 托管实例集上运行。	2024 年 11 月 12 日
Amazon GameLift Servers 开启了跟踪更改	Amazon GameLift Servers 开始跟踪其 AWS 托管策略的更改。	2024 年 11 月 12 日

使用 Amazon GameLift Servers 进行日志记录和监控

监控是保持 Amazon GameLift Servers 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 Servers 提供了多种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 自动扩缩策略发送通知。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 [Amazon GameLift Servers使用 Amazon 进行监控 CloudWatch](#)。

AWS CloudTrail 日志

CloudTrail 提供了用户、角色或 AWS 服务在 Amazon GameLift Servers 中所执行操作的记录。使用 CloudTrail 收集的信息，您可以确定向 Amazon GameLift Servers 发出了什么请求、发出请求的 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 [使用记录 Amazon GameLift Servers API 调用 AWS CloudTrail](#)。

的合规性验证 Amazon GameLift Servers

Amazon GameLift Servers 不在任何 AWS 合规性计划的范围内。

要了解某个 AWS 服务是否在特定合规性计划范围内，请参阅[合规性计划范围内的 AWS 服务](#)，然后选择您感兴趣的合规性计划。有关常规信息，请参阅[AWS 合规性计划](#)、。

您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参阅[在 AWS Artifact 中下载报告](#)、。

您在使用 AWS 服务 时的合规性责任由您的数据的敏感性、您公司的合规性目标以及适用的法律法规决定。有关您在使用 AWS 服务时的合规责任的更多信息，请参阅[AWS 安全性文档](#)。

Amazon GameLift Servers 中的故障恢复能力

如果您将 Amazon GameLift Servers FleetIQ 作为独立功能与 Amazon EC2 搭配使用，请参阅《Amazon EC2 用户指南》中的[Amazon EC2 中的安全性](#)。

AWS 全球基础架构围绕 AWS 区域和可用区构建。AWS 区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。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、容错能力和可扩展性。

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 全球基础设施之外，Amazon GameLift Servers 还提供了以下功能，以帮助支持您的数据弹性需求。

- 多区域队列 - Amazon GameLift Servers 游戏会话队列用于放置具有可用托管资源的新游戏会话。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[自定义游戏会话队列](#)。
- 自动容量扩展 - 通过使用 Amazon GameLift Servers 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[利用 Amazon GameLift Servers 扩展游戏托管容量](#)。
- 跨实例分配 - Amazon GameLift Servers 根据实例集大小在多个实例之间分配传入流量。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 Amazon GameLift Servers 的游戏服务器构建和脚本使用标准存储类别存储在 Amazon S3 中，该类别使用多数据中心复制来提升恢复能力。游戏会话日志也使用标准存储类别存储在 Amazon S3 中。

Amazon GameLift Servers 中的基础结构安全性

如果您将 Amazon GameLift Servers FleetIQ 作为独立功能与 Amazon EC2 搭配使用，请参阅《Amazon EC2 用户指南》中的 [Amazon EC2 中的安全性](#)。

作为一项托管式服务，Amazon GameLift Servers 由 [Amazon Web Services：安全流程概览](#) 白皮书中所述的AWS全球网络安全程序提供保护。

您可以使用 AWS 发布的 API 调用通过网络访问 Amazon GameLift Servers。客户端必须支持传输层安全性 (TLS) 1.2 或更高版本。我们建议使用 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 主体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 Servers 服务将所有队组放置到 Amazon Virtual Private Cloud (VPC) 中，以便每个队组都存在于 AWS 云中逻辑隔离的区域内。您还可以使用 Amazon GameLift Servers 策略来控制来自特定 VPC 终端节点或特定 VPC 的访问。事实上，这隔离了在 Amazon GameLift Servers 网络中仅从特定 VPC 到给定 AWS 资源的网络访问。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 GameLift Servers 中的配置和漏洞分析

如果您将 Amazon GameLift Servers FleetIQ 作为独立功能与 Amazon EC2 搭配使用，请参阅《Amazon EC2 用户指南》中的 [Amazon EC2 中的安全性](#)。

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处理 Amazon GameLift Servers 中的配置和漏洞分析：

- 客户负责管理部署到用于游戏托管的 Amazon GameLift Servers 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 Servers 或提供新的容器映像，为其创建新实例集，然后将流量重定向到新实例集。

- 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, 请定期替换实例集, 无论游戏服务器是否更新。生产实例集至少应每年替换一次。有关更多信息, 请参阅[确保实例集运行时环境保持最新状态](#)。
- 客户应考虑定期使用最新的 SDK 版本更新游戏, 包括 AWS SDK、Amazon GameLift Servers 服务器 SDK 和适用于实时服务器的 Amazon GameLift Servers 客户端 SDK。

Amazon GameLift Servers 的安全最佳实践

如果您将 Amazon GameLift Servers FleetIQ 作为独立功能与 Amazon EC2 搭配使用, 请参阅《Amazon EC2 用户指南》中的 [Amazon EC2 中的安全性](#)。

Amazon GameLift Servers 提供了在您开发和实施自己的安全策略时需要考虑的大量安全功能。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。

确保实例集运行时环境保持最新状态

Amazon GameLift Servers 强烈建议您定期更换托管的实例集 (包括托管 EC2 和托管的容器实例集), 以维护游戏服务器的安全运行时环境。长时间运行而未更新运行时的实例集可能包含过时的依赖项和安全漏洞, 这些漏洞可能会危及您的游戏服务器。有关部署到 Amazon GameLift Servers 实例集的软件责任共担详情, 请参阅 [Amazon GameLift Servers 中的配置和漏洞分析](#)。

托管实例集的运行时环境由其亚马逊机器映像 (AMI) 版本决定。创建新实例集时, Amazon GameLift Servers 会将最新的可用 AMI 版本分配给该实例集, 并且该实例集内的所有计算实例都将使用该版本进行部署。要更新 AMI 版本, 必须创建新的实例集。有关当前 AMI 版本的详细信息, 请参阅 [Amazon GameLift Servers AMI 版本](#)。

建议实操:

- 监控实例集使用期限并更换超过 30 天的实例集 – 您可以在 Amazon GameLift Servers 控制台中跟踪实例集的创建日期, 或使用 CLI 检索实例集属性。Amazon GameLift Servers 会在控制台中针对超过 90 天的实例集显示警告, 并通过电子邮件通知账户持有人其实例集使用期限已超过一年。

Note

更新队列 (例如使用 [UpdateFleetAttributes](#) 或 [UpdateContainerFleet](#)) 不会更改 AMI 版本。您必须创建新的实例集。

- 根据安全状况定期更换实例集 – 制定定期计划，以创建新实例集并停用旧实例集。可以考虑使用 [Amazon Q](#) 这样的服务，查看当前的 AMI 版本的游戏代码，检测安全问题并提出补救措施建议。
- 在部署之前，使用最新 AMI 版本测试服务器生成包 – 在创建新实例集之前，您可能需要修改服务器生成包，并将其上传到 Amazon GameLift Servers。
- 管理 AWS 账户的车队配额-如果需要，您可以申请提高限额以创建替换车队。有关更多信息，请参阅 [Amazon GameLift Servers 端点和限额](#)。
- 考虑自动更换实例集 – 您可以自动创建新实例集，和从旧实例集迁移玩家流量。例如：
 - AWS CloudFormation 用于自动创建和管理车队。将您的队列配置作为 CloudFormation 模板进行维护，并使用它们来启动资源堆栈。
 - 利用 Amazon GameLift Servers 别名功能来抽象特定的舰队 IDs。实例集别名可以轻松地将玩家流量从现有实例集转移到新实例集，而不会中断正在进行的游戏会话。有关更多信息，请参阅 [使用别名抽象化 Amazon GameLift Servers 实例集名称](#)。
 - 使用 blue/green 部署策略来降低迁移风险并保持零停机时间。利用两个相同的生产环境，您可以在类似生产的环境中进行全面测试，更好地控制迁移过程，并确保即时回滚。

保护您的端口配置

我们强烈建议不要开放到互联网的端口，因为这样做会带来安全风险。例如，以下配置会打开远程桌面端口，允许互联网上的任何人访问该实例：

```
{
  "FleetId": "<fleet identifier>",
  "InboundPermissionAuthorizations": [
    {
      "FromPort": 3389,
      "IpRange": "0.0.0.0/0",
      "Protocol": "RDP",
      "ToPort": 3389
    }
  ]
}
```

而是使用 [UpdateFleetPortSettings](#) 打开具有特定 IP 地址或地址范围的端口，如以下示例所示：

```
{
  "FleetId": "<fleet identifier>",
  "InboundPermissionAuthorizations": [
    {
```

```
        "FromPort": 3389,  
        "IpRange": "54.186.139.221/32",  
        "Protocol": "TCP",  
        "ToPort": 3389  
    }  
]  
}
```

其他安全资源

有关如何更安全地使用 Amazon GameLift Servers 的更多信息，请参阅 [AWS Well-Architected Tool 安全性支柱](#)。

Amazon GameLift Servers 定价和成本规划

了解 Amazon GameLift Servers 定价结构并在游戏托管解决方案中实施具有成本效益的策略对于优化游戏基础设施预算至关重要。有关详细的服务信息，包括示例定价方案和定价工具，请参阅 [Amazon GameLift Servers 定价](#)。

根据免费套餐，新 AWS 客户最多可以使用 12 个月，而 Amazon GameLift Servers 无需支付任何费用。请确保使用量不超过免费套餐限额，以避免产生费用。

定价模型

Amazon GameLift Servers 提供多种定价模型，旨在满足不同使用场景和业务需求。每种模型的成本计算基准不同，您可通过调整各模型中的成本影响因素来控制成本。

托管主机

托管 EC2 和托管集装箱船队使用 Amazon GameLift Servers 管理的 Amazon EC2 实例。在设置游戏托管时，您可以选择要使用的 Amazon EC2 实例类型。

- 计算成本 — 为托管游戏会话的每小时 EC2 实例使用量付费。影响实例成本的因素包括：
 - AWS 区域
 - 实例系列、类型和大小
 - 使用竞价型实例与按需型实例
 - 操作系统（托管式容器实例集仅在 Linux 上运行）
- 数据传输：按游戏客户端与托管游戏服务器之间的流量付费

随时随地托管

Anywhere 托管运行于您自行提供并管理的游戏托管资源之上，因此无计算成本。其成本基于以下两项计费：

- 游戏会话：按部署在 Anywhere 计算资源上的游戏会话数量付费
- 连接分钟数：按服务器进程的连接时长（分钟）付费

您可使用适用于 Amazon GameLift Servers 的定价计算器，估算不同 Amazon GameLift Servers 配置下的成本。在设计游戏托管解决方案时，或需预测配置更改后的成本时，均可使用该计算器。请参阅 [生成 Amazon GameLift Servers 定价估算值](#)。

生成 Amazon GameLift Servers 定价估算值

使用 AWS 定价计算器，您可以[为创建估算价格 Amazon GameLift Servers](#)。您无需具备 AWS 账户 或 深入的知识 AWS 即可使用计算器。

AWS 定价计算器 计算器会引导你做出影响服务成本的决策，让你大致了解游戏项目 Amazon GameLift Servers 可能要花多少钱。如果您还不确定计划如何使用 Amazon GameLift Servers，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。

估算 Amazon GameLift Servers 托管式托管

此选项提供了在 Amazon GameLift Servers 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。使用 Amazon GameLift Servers 托管式托管时，FlexMatch 对战不会产生额外费用。

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 Servers 实例

本部分可帮助您估算为玩家举办游戏会话所需的计算资源类型和数量。Amazon GameLift Servers 使用[亚马逊弹性计算云 \(Amazon EC2\) 实例](#)来管理游戏服务器。在 Amazon GameLift Servers 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，请打开的 [Amazon GameLift Servers“配置” 页面](#) AWS 定价计算器。添加描述，选择区域，然后选择估计 Amazon GameLift Servers 托管（实例 + 数据传出）。在 Amazon GameLift Servers 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 Servers 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 Servers 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 Servers 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 Servers 如何使用竞价型实例的更多信息，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 Servers 将维护 60 个按需型实例和 40 个竞价型实例。

- 实例类型

Amazon GameLift Servers 队列可以使用一系列 Amazon EC2 实例类型，这些实例类型在计算能力、内存、存储和联网能力方面各不相同。配置 Amazon GameLift Servers 队列时，请选择最适合游戏需求的实例类型。有关使用 Amazon GameLift Servers 选择实例类型的信息，请参阅[为托管式实例集选择计算资源](#)。

如果您知道自己正在使用或计划在 Amazon GameLift Servers 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“[配置 Amazon GameLift Servers](#)”页面上 AWS 定价计算器，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 Servers 定价估算值后，选择添加到我的估算值。有关在中创建和管理估算值的更多信息 AWS 定价计算器，请参阅《AWS 定价计算器 用户指南》中的[创建估算、配置服务和添加更多服务](#)。

成本优化策略

将这些策略与托管式托管解决方案结合使用，有助于降低云托管成本，同时保持高性能和优质玩家体验。

最佳实践

请遵循以下建议，为您的游戏托管纳入成本优化实践。我们至少建议您定期查看您的 Amazon GameLift Servers 成本与使用模式。如果您有与定价相关的问题，请联系

实例集配置

- 使用定价计算器：将此工具与您的游戏托管数据结合使用，探索各种选项并测试潜在的配置方案，以实现成本节约。请参阅[适用于 Amazon GameLift Servers 的AWS 定价计算器](#)。
- 节省开发阶段成本：在开发和测试阶段，选择规格更小、成本更低的实例类型，且仅在需要时运行实例。
- 将@@ 实例类型与游戏匹配 — 为您的队列选择最适合您的游戏要求的 Amazon EC2 实例系列、类型和大小。请参阅[为托管式实例集选择计算资源](#)中的“实例类型”部分。
- 使用 Graviton 实例类型：Graviton 实例由基于 ARM 的处理器提供支持。在按需型实例定价模型下，相比性能相当的 x86 架构实例，它们具备更高的性价比、更强的能效，且成本更低。
- 使用竞价型实例：竞价型实例的成本通常低于按需型实例，但可能并非随时可用。可将竞价型实例集与按需型实例集结合使用，在低成本与高可用性之间取得平衡（建议竞价型实例使用率控制在 50%-85%）。可将竞价型实例用于非关键游戏模式或低峰时段。要监控竞价型实例的可行性，可跟踪 InstanceInterruptions 和 GameServerInterruptions 等指标。请参阅[按需型实例和竞价型实例](#)。
- 在 Linux 上运行服务器：将游戏服务器运行时构建为 Linux 版本。通过 Linux 部署的实例通常比通过 Windows 部署的实例更具成本效益。
- 优化资源利用率：在维持性能的前提下，通过配置实例集支持尽可能多的并发游戏服务器进程，实现资源利用效率最大化。请参阅[在托管式 Amazon GameLift Servers 上优化游戏服务器运行时配置](#)。
- 制定多区域策略：将游戏托管部署到多个位置，既能为玩家提供更低延迟的体验，也能获得更可靠的备份选项。为平衡区域覆盖范围和成本效率，建议优先在玩家数量最多的市场部署托管资源，将次要位置用作高峰期的溢出容量。请参阅[建立多位置队列](#)。

实例集容量扩展

- 缩放到零：当实例集处于未使用状态时，手动将实例集容量设置为零，以避免产生不必要的费用。请参阅[手动设置Amazon GameLift Servers舰队的容量](#)。
- 添加自动扩缩：使用自动扩缩调整游戏托管容量，避免托管资源过度预调配。自动扩缩可根据玩家需求波动及其他关键指标动态匹配容量。请参阅[通过 Amazon GameLift Servers 自动扩展队组容量](#)。
- 维持缓冲容量：为应对突发的玩家需求峰值，避免玩家等待，可通过目标跟踪功能维持一定数量的闲置游戏服务器作为缓冲容量。可根据玩家群体规模及使用模式自定义缓冲容量大小。请参阅[基于目标的自动扩缩](#)。

游戏会话放置

- 使用放置队列：Amazon GameLift Servers 队列基于托管资源成本及其他因素，通过算法确定游戏会话的“最优”托管位置。请参阅[配置游戏会话放置](#)。
- 自定义队列优先级：您可以调整队列在放置游戏会话时对托管成本的优先级权重。请参阅[优先考虑游戏会话放置](#)。
- 监控放置指标：跟踪队列选择托管位置的具体情况，从中识别成本优化机会。请参阅[Amazon GameLift Servers 队列指标](#)。

数据传输

- 跟踪数据传输：监控游戏客户端与服务器之间的数据传输情况，并采取措施优化数据传输行为。
- 使用数据压缩：建议为游戏客户端与服务器之间的传输数据实施数据压缩技术。压缩技术在降低带宽使用量以及提升游戏运行性能、改善网络延迟方面发挥着关键作用。

资源成本和利用率工具

探索如何使用 AWS 工具来监控和优化游戏托管成本Amazon GameLift Servers。有关其他工具的信息，请参阅 [AWS 账单与成本管理](#)。

AWS 成本管理工具

- 账单控制台：查看您的 AWS 账单和使用情况。请参阅[开始设置账单](#)。
- 免费套餐用量提醒：在接近免费套餐限额时设置通知提醒。请参阅[跟踪您的 AWS 免费套餐使用情况](#)。

- Amazon CloudWatch 账单提醒 — 在使用量达到自定义阈值时配置提醒。请参阅[创建账单警报以监控 AWS 预估费用](#)。

成本分配和跟踪

- 成本分配标签：标记您的实例集及其他资源以组织和跟踪托管成本。请参阅[使用成本分配标签组织和跟踪 AWS 成本](#)。
- 成本报告：创建按分配的标签分类的报告。请参阅[使用月度成本分配报告](#)。
- AWS Cost Explorer：通过筛选和可自定义的视图分析成本、趋势和预测。请参阅使用[分析您的成本和使用情况 AWS Cost Explorer](#)。
- AWS Budgets：跟踪 AWS 成本和使用情况并采取相应措施。请参阅[使用管理成本 AWS Budgets](#)。

使用Amazon GameLift Servers和 Amazon 进行绩效监控 CloudWatch

监控以下关键指标以优化资源利用率。在Amazon GameLift Servers控制台中查看指标或使用 Amazon 控制 CloudWatch 面板。有关 Amazon GameLift Servers 的所有可用指标的详细信息，请参阅[Amazon GameLift Servers使用 Amazon 进行监控 CloudWatch](#)。

- 实例指标：ActiveInstances、IdleInstances、PercentIdleInstances
- 服务器进程指标：ActiveServerProcesses、HealthyServerProcesses
- 游戏会话指标：ActiveGameSessions、AvailableGameSessions
- 玩家会话指标：CurrentPlayerSessions
- 队列指标：AverageWaitTime、QueueDepth
- 对战指标：CurrentTickets、MatchesPlaced
- 硬件性能：CPUUtilization、NetworkIn/NetworkOut、DiskReadBytes/DiskWriteBytes
- 资源利用率指标
 - 使用 PercentIdleInstances 确定最优实例集大小。
 - 跟踪 PercentAvailableGameSessions 以确保有足够的游戏托管容量。
 - 监控 InstanceInterruptions 和 GameServerInterruptions 以确定竞价型实例的可行性。

Amazon GameLift Servers 参考指南

本部分包含使用 Amazon GameLift Servers 的参考文档。

主题

- [Amazon GameLift Servers 服务 API](#)
- [Amazon GameLift Servers 服务器 SDK 5.x](#)
- [Amazon GameLift Servers 服务器 SDK 版本 4 及更早版本](#)
- [游戏会话放置事件](#)
- [Amazon GameLift Servers AMI 版本](#)
- [Amazon GameLift Servers 服务位置](#)
- [Amazon GameLift Servers 端点和限额](#)
- [亚马逊 GameLift 服务器 API 限制](#)
- [Amazon GameLift Servers SDK API 限制](#)
- [Amazon GameLift Servers UDP ping 信标](#)

Amazon GameLift Servers 服务 API

在构建 Amazon GameLift Servers 游戏托管解决方案和其他功能时，使用此基于任务的列表查找 API 操作。S AWS DK 在 `aws.gamelift` 命名空间中包含这些操作。[下载 AWS 软件开发工具包](#) 或 [查看 Amazon GameLift Servers API 参考文档](#)。您还可以将 API 与 AWS 命令行接口 (AWS CLI) 一起使用，如 [AWS CLI 命令参考](#) 中所述。

该 API 包括两组用于游戏托管的操作：

- [管理 Amazon GameLift Servers 托管资源](#)
- [开始游戏会话并加入玩家行列](#)

Amazon GameLift Servers 服务 API 还包含用于其他 Amazon GameLift Servers 工具和解决方案的操作。有关列表 FleetIQ APIs，请参阅 [FleetIQ API 操作](#)。有关配对的 FlexMatch APIs 列表，请参阅 [FlexMatch API 操作](#)。

管理 Amazon GameLift Servers 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、访问性能和使用情况指标，等等。使用 Amazon GameLift Servers 托管游戏服务器（包括 Amazon GameLift Servers Realtime）时，请使用这些 API 操作。您还可以在 [Amazon GameLift Servers 控制台](#) 中执行大多数资源管理任务，也可以使用 AWS Command Line Interface（AWS CLI）工具进行调用。

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](#)：从本地路径上传生成包文件并创建新的 Amazon GameLift Servers 生成包资源。此操作可用作 AWS CLI 命令，是上传游戏服务器版本的最常用方法。
- [CreateBuild](#)— 使用存储在 Amazon S3 存储桶中的文件创建新版本。
- [ListBuilds](#)— 获取上传到某个 Amazon GameLift Servers 区域的所有版本的列表。
- [DescribeBuild](#)— 检索与版本相关的信息。
- [UpdateBuild](#)— 更改版本元数据，包括版本名称和版本。
- [DeleteBuild](#)— 从中移除构建 Amazon GameLift Servers。

管理 Amazon GameLift Servers Realtime 配置脚本

- [CreateScript](#)— 上传 JavaScript 文件并创建新的 Amazon GameLift Servers 脚本资源。
- [ListScripts](#)— 获取上传到某个 Amazon GameLift Servers 区域的所有 Realtime 脚本的列表。
- [DescribeScript](#)— 检索与 Realtime 脚本相关的信息。
- [UpdateScript](#)— 更改脚本元数据并上传修改后的脚本内容。
- [DeleteScript](#)— 从中删除 Realtime 脚本 Amazon GameLift Servers。

设置用于托管的计算资源

配置托管资源，并通过游戏服务器生成包或 Realtime 配置脚本进行部署。

创建和管理实例集

- [CreateFleet](#)— 配置和部署新的计算资源 Amazon GameLift Servers 队列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](#)— 获取一个Amazon GameLift Servers地区内所有舰队的列表。
- [DeleteFleet](#)— 移除不再运行游戏服务器或托管玩家的舰队。
- 查看/更新实例集位置。
 - [CreateFleetLocations](#)— 将远程位置添加到支持多个地点的现有舰队中
 - [DescribeFleetLocationAttributes](#)— 获取舰队所有远程位置的列表并查看每个位置的当前状态。
 - [DeleteFleetLocations](#)— 从支持多个位置的舰队中移除远程位置。
- 查看/更新实例集配置。
 - [DescribeFleetAttributes/UpdateFleetAttributes](#)— 查看或更改舰队的元数据以及游戏会话保护和资源创建限制的设置。
 - [DescribeFleetPortSettings/UpdateFleetPortSettings](#)— 查看或更改队列允许的入站权限 (IP 地址和端口设置范围) 。
 - [DescribeRuntimeConfiguration/UpdateRuntimeConfiguration](#)— 查看或更改队列中每个实例上要运行的服务器进程 (以及多少个) 。

管理实例集容量

- [描述 EC2 InstanceLimits](#)-检索当前 AWS 账户允许的最大实例数和当前使用级别。
- [DescribeFleetCapacity](#)— 检索舰队所在区域的当前容量设置。
- [DescribeFleetLocationCapacity](#)— 检索多地点车队中每个地点的当前容量设置。
- [UpdateFleetCapacity](#)— 手动调整车队的容量设置。
- 设置 :
 - [PutScalingPolicy](#)— 开启基于目标的自动缩放或创建自定义自动缩放策略，或更新现有策略。
 - [DescribeScalingPolicies](#)— 检索现有的 auto Scaling 策略。
 - [DeleteScalingPolicy](#)— 删除 auto Scaling 策略并阻止其影响队列的容量。
 - [StartFleetActions](#)— 重启队列的 auto Scaling 策略。
 - [StopFleetActions](#)— 暂停队列的 auto Scaling 政策。

监控实例集活动。

- [DescribeFleetUtilization](#)— 检索队列中当前活跃的服务器进程、游戏会话和玩家数量的统计信息。
- [DescribeFleetLocationUtilization](#)— 检索多地点车队中每个位置的利用率统计信息。
- [DescribeFleetEvents](#)— 查看队列在指定时间段内记录的事件。

- [DescribeGameSessions](#)— 检索游戏会话元数据，包括游戏的运行时间和当前玩家人数。

为游戏会话放置设置队列

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](#)— 创建队列，以便在处理游戏会话放置请求时使用。
- [DescribeGameSessionQueues](#)— 检索在某个Amazon GameLift Servers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](#)— 从该区域移除游戏会话队列。

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](#)— 定义新别名并可选择将其分配给舰队。
- [ListAliases](#)— 获取在某个Amazon GameLift Servers区域中定义的所有舰队别名。
- [DescribeAlias](#)— 检索有关现有别名的信息。
- [UpdateAlias](#)— 更改别名的设置，例如将其从一个舰队重定向到另一个舰队。
- [DeleteAlias](#)— 从该区域删除别名。
- [ResolveAlias](#)— 获取指定别名指向的舰队 ID。

连接到托管式托管实例

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](#)— 获取队列中每个实例的信息，包括实例 ID、IP 地址、位置和状态。
- [GetInstanceAccess](#)— 请求远程连接到队列中指定实例所需的访问凭证。

设置 VPC 对等连接

创建和管理您的Amazon GameLift Servers托管资源与其他资源之间的 VPC 对等 AWS 连接。

- [CreateVpcPeeringAuthorization](#)— 授权与您的 VPCs 其中一个建立对等连接。
- [DescribeVpcPeeringAuthorizations](#)— 检索有效的对等连接授权。
- [DeleteVpcPeeringAuthorization](#)— 删除对等连接授权。
- [CreateVpcPeeringConnection](#)— 在队列的 VPC 与您的 Amazon GameLift Servers VPCs 队伍的 VPC 之间建立对等连接。
- [DescribeVpcPeeringConnections](#)— 检索与 Amazon GameLift Servers 队列的活动或待定 VPC 对等连接的信息。
- [DeleteVpcPeeringConnection](#)— 删除与 Amazon GameLift Servers 队列的 VPC 对等连接。

开始游戏会话并加入玩家行列

通过后端服务调用这些操作即可启动新的游戏会话、获取有关现有游戏会话的信息以及让玩家加入游戏会话。这些操作适用于托管在 Amazon GameLift Servers 上的自定义游戏服务器。如果您正在使用 Amazon GameLift Servers Realtime，请使用 [Amazon GameLift Servers Realtime 客户端 API \(C#\)](#) 管理游戏会话。

- 为一个或多个玩家启动新游戏会话。
 - [StartGameSessionPlacement](#)— Amazon GameLift Servers 要求找到最好的可用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。该方法依靠游戏会话队列来跟踪多个区域的托管可用性，并使用 FleetIQ 算法根据玩家延迟、托管成本、位置等对放置位置进行优先排序。
 - [DescribeGameSessionPlacement](#)— 获取安置申请的详细信息和状态。
 - [StopGameSessionPlacement](#)— 取消安置申请。
 - [CreateGameSession](#)— 在特定的舰队位置开始新的空白游戏会话。此操作使您能够更好地控制游戏会话的启动位置，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](#)— 为玩家预留空位以加入游戏会话。
 - [CreatePlayerSessions](#)— 为多名玩家预留空位以加入游戏会话。
- 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游戏会话列表。
 - [DescribeGameSessions](#)— 检索特定游戏会话的元数据，包括活跃时间长度和当前玩家人数。
 - [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，例如最大玩家人数和加入政策。
- [GetGameSessionLogUrl](#)— 获取游戏会话保存日志的位置。

Amazon GameLift Servers 服务器 SDK 5.x

本节提供了 Amazon GameLift Servers 服务器 SDK 5.x 的参考文档。服务器 SDK 提供游戏服务器用于与 Amazon GameLift Servers 服务交互的核心功能。例如，游戏服务器会接收该服务发送的提示来启动和停止游戏会话，并定期向该服务提供游戏会话状态更新。在部署游戏服务器进行托管之前，请将其与服务器 SDK 集成。

使用此服务器 SDK 参考将要托管的自定义多人游戏服务器与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

Amazon GameLift Servers 服务器 SDK 的最新主要版本是 5.x。以下托管功能需要使用版本 5.x：

- Amazon GameLift Servers Anywhere
- 适用于 Unreal Engine 和 Unity 的 Amazon GameLift Servers 插件

Note

如果您需要使用服务器 SDK 版本 4.x 或更早版本，请参阅[Amazon GameLift Servers 服务器 SDK 版本 4 及更早版本](#)以获取文档和下载信息。

主题

- [Amazon GameLift Servers 服务器 SDK 5 中的更新](#)
- [迁移到 Amazon GameLift Servers 服务器 SDK 5.x](#)
- [适用于 Amazon GameLift Servers 的 C++ 服务器 SDK 5.x – 操作](#)
- [适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 操作](#)
- [适用于 Amazon GameLift Servers 的 Go 服务器 SDK – 操作](#)
- [适用于 Amazon GameLift Servers 的 C++ \(Unreal \) 服务器 SDK 5.x – 操作](#)

Amazon GameLift Servers 服务器 SDK 5 中的更新

托管的游戏服务器使用 Amazon GameLift Servers 服务器 SDK 与 Amazon GameLift Servers 服务通信，以启动和管理玩家的游戏会话。最新版本的 Amazon GameLift Servers 服务器 SDK 5 提供了大量改进功能并支持 Amazon GameLift Servers 的新功能。如果您的游戏服务器生成包目前使用的是 Amazon GameLift Servers 服务器 SDK 4 或更早版本，请按照本主题中的指南更新您的游戏。

Amazon GameLift Servers 服务器 SDK 5.0.0 及更高版本包含以下更新：

- 扩展的语言 - 库支持以下语言：C++、C#、Go。您可以构建适用于 Unreal Engine 的 C++ 库。
- 游戏引擎插件支持 - 适用于 Unreal Engine 和 Unity 的 Amazon GameLift Servers 独立插件需要 Amazon GameLift Servers 服务器 SDK 5 库。这些插件提供用于集成、测试和将游戏部署到 Amazon GameLift Servers 进行托管的引导式工作流程。请参阅[Amazon GameLift Servers 适用于 Unity 的插件 \(服务器 SDK 5.x\)](#) 和 [Amazon GameLift Servers 虚幻引擎插件](#) 文档。
- Amazon GameLift Servers Anywhere 支持 - 利用 Anywhere 实例集，您可以设置自己的托管资源，以使用 Amazon GameLift Servers 功能（包括对战）。添加 Amazon GameLift Servers 代理以自动执行游戏会话生命周期管理。使用 Anywhere 实例集通过本地硬件进行生产托管，或为快速迭代游戏开发设置测试环境。设置 [Anywhere 托管](#) 和 [Amazon GameLift Servers 代理](#)。
- 更新的测试工具 - Amazon GameLift Servers Anywhere 功能可让您为游戏设置本地或基于云的测试环境。设置使用或不使用 Amazon GameLift Servers 代理的测试。这些工具取代了 Amazon GameLift Servers Local。请参阅[使用 Amazon GameLift Servers Anywhere 设置本地测试](#)。
- 适用于 C# 的整合 .NET 解决方案 - C# 服务器 SDK 5.1+ 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。
- 新 Compute 资源 - 这种新资源整合了不同类型的托管资源。它包含基于云的托管资源（托管式 EC2 或容器实例集）和客户控制的托管资源（Anywhere 实例集）。它包含以下更新：
 - 针对 Compute 资源的新 API 调用包括：[ListCompute\(\)](#)、[DescribeCompute\(\)](#) 和 [GetComputeAccess\(\)](#)。这些操作将返回任何类型 Amazon GameLift Servers 实例集的托管资源信息。通常，对于具有使用服务器 SDK 5.x 的游戏服务器的实例集，请使用特定于计算的操作来替换特定于实例的操作。此外，这些操作适用于没有 Amazon GameLift Servers 代理的 Anywhere 实例集：[RegisterCompute\(\)](#)、[DeregisterCompute\(\)](#) 和 [GetComputeAuthToken\(\)](#)。
 - 包含 CloudWatch 维度 FleetId、Location 和 ComputeType 的新指标 ActiveCompute。该指标取代了之前的指标 ActiveInstances。
- 用于远程访问的 Amazon EC2 Systems Manager (SSM) - 为了提高安全性，在连接到 Amazon GameLift Servers 托管式实例集中的实例时，请使用 SSM 而不是 SSH。请参阅[远程连接到 Amazon GameLift Servers 实例集实例](#)。

迁移到 Amazon GameLift Servers 服务器 SDK 5.x

要将游戏项目更新为使用服务器软件开发工具包版本 5.x，请进行以下更改：

Note

如果您需要继续使用服务器 SDK 版本 4.x 或更早版本，请参阅 [Amazon GameLift Servers 服务器 SDK 版本 4 及更早版本](#) 以获取文档和下载信息。特别是对于 SDK 版本 4.0.2，您可以从[官方 GitHub 版本](#)中下载。

1. 获取适用于您的开发环境的最新 Amazon GameLift Servers 服务器 SDK 软件包 [[下载网站](#)]。按照 Readme 文件中针对下载的软件包和版本的安装说明进行操作。有关在游戏项目中使用服务器的信息 SDKs，请参阅以下说明。
 - [针对使用 C++、C# 或 Go 的开发环境](#)
 - [针对 Unreal Engine 项目（仅限适用于 Unreal 库的 C++ 服务器 SDK）](#)
 - [针对 Unity 项目（仅限适用于 Unity 库的 C# 服务器 SDK）](#)
 - [针对与适用于 Unreal Engine 的 Amazon GameLift Servers 插件结合使用](#)
 - [针对与适用于 Unity 的 Amazon GameLift Servers 插件结合使用](#)
2. 按如下方式更新服务器代码：
 - 将服务器代码回调函数 `onCreateGameSession()` 更改为 `onStartGameSession()`。
 - 根据需要更新 `InitSDK()` 输入：
 - 如果您计划在有 Amazon GameLift Servers 代理的情况下将游戏服务器生成包部署到 Amazon GameLift Servers 托管式 EC2 实例集或 Anywhere 实例集：

调用 `InitSDK()` 而无需使用参数（[C++](#)）（[C#](#)）（[Unreal](#)）。此调用设置计算环境和与 Amazon GameLift Servers 服务的 WebSocket 连接。
 - 如果您计划在没有 Amazon GameLift Servers 代理的情况下将游戏服务器生成包部署到 Anywhere 实例集：

使用服务器参数调用 `InitSDK()`（[C++](#)）（[C#](#)）（[Unreal](#)）。游戏服务器进程使用这些参数与 Amazon GameLift Servers 服务建立连接。
3. 如果您的游戏服务器生成包或其他托管应用程序在运行时与其他 AWS 资源通信，您需要更改应用程序访问这些资源的方式。使用新服务器 SDK 操作 `GetFleetRoleCredentials()` 替换 `AssumeRoleCredentials`（对于游戏服务器），或使用共享凭证（对于其他应用程序）。有关

如何实施此更改的更多信息，请参阅[将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。

- 如果您的项目调用了服务器 SDK 操作 `GetInstanceCertificate()` 来检索 TLS 证书，请将代码修改为使用新的 `GetComputeCertificate()` ([C++](#)) ([C#](#)) ([Unreal](#))。
- 将游戏版本上传到时 Amazon GameLift Servers (例如使用 [upload-build](#) 或 [CreateBuild\(\)](#))，请将 `ServerSdkVersion` 参数设置为您正在使用的 5.x 版本 (此参数当前默认为 4.0.2)。该参数必须与游戏服务器生成包中的实际服务器 SDK 库匹配。如果为上传的游戏服务器生成包指定了错误的版本，那么使用该生成包创建的所有实例集都将失败。请参阅[为 Amazon GameLift Servers 创建游戏服务器生成包](#)。

以下示例说明了如何指定服务器 SDK 版本：

```
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
```

- 如果您使用脚本远程连接到托管式实例集，请更新脚本以使用新流程，如[远程连接到 Amazon GameLift Servers 实例集实例](#)中所述。

适用于 Amazon GameLift Servers 的 C++ 服务器 SDK 5.x – 操作

使用服务器 SDK 5.x 参考，将要托管的多人游戏与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

Note

本主题介绍使用 C++ 标准库 (`std`) 构建时可以使用的 Amazon GameLift Servers 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。

适用于 Amazon GameLift Servers 的 C++ 服务器 SDK 5.x – 数据类型

使用 Amazon GameLift Servers C++ 服务器 SDK 5.x 参考，将要托管的多人游戏与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

 Note

本主题介绍使用 C++ 标准库 (std) 构建时可以使用的 Amazon GameLift Servers 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。

适用于 Amazon GameLift Servers 的 C++ 服务器 SDK 5.x – 操作

数据类型

- [LogParameters](#)
- [MetricsParameters](#)
- [ProcessParameters](#)
- [UpdateGameSession](#)
- [GameSession](#)
- [ServerParameters](#)
- [StartMatchBackfillRequest](#)
- [玩家](#)
- [DescribePlayerSessionsRequest](#)
- [StopMatchBackfillRequest](#)
- [AttributeValue](#)
- [GetFleetRoleCredentialsRequest](#)
- [AwsLongOutcome](#)
- [AwsStringOutcome](#)
- [DescribePlayerSessionsOutcome](#)
- [DescribePlayerSessionsResult](#)
- [GenericOutcome](#)
- [GenericOutcomeCallable](#)
- [PlayerSession](#)
- [StartMatchBackfillOutcome](#)
- [StartMatchBackfillResult](#)
- [GetComputeCertificateOutcome](#)

- [GetComputeCertificateResult](#)
- [GetFleetRoleCredentialsOutcome](#)
- [GetFleetRoleCredentialsResult](#)
- [初始化 SDKOutcome](#)
- [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 Servers 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，作为 ProcessParameters 对象的一部分向 Amazon GameLift Servers 提供 LogParameters。

属性	描述
LogPaths	您希望 Amazon GameLift Servers 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：std::vector<std::string> 必需：否

MetricsParameters

使用此数据类型配置游戏服务器的指标收集和崩溃报告功能。游戏服务器通过 [InitMetrics\(\)](#) 调用将 MetricsParameters 传达给 Amazon GameLift Servers。

属性	描述

StatsdHost	用于指标收集的 StatsD 服务器的主机名或 IP 地址。 类型：<code>std::string</code> 必需：否
StatsdPort	用于收集指标的 StatsD 服务器的端口号。 类型：<code>int</code> 必需：否
CrashReporterHost	崩溃报告服务器的主机名或 IP 地址。 类型：<code>std::string</code> 必需：否
CrashReporterPort	崩溃报告服务器的端口号。 类型：<code>int</code> 必需：否
FlushIntervalMs	将指标数据刷新到服务器的时间间隔（以毫秒为单位）。 类型：<code>int</code> 必需：否
MaxPacketSize	发送到服务器的指标数据包的最大大小（以字节为单位）。 类型：<code>int</code> 必需：否

ProcessParameters

此数据类型包含一组在 [ProcessReady\(\)](#) 中发送到 Amazon GameLift Servers 的参数。

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 GameLift Servers 会复制并存储这些文件，以便将来访问。 类型：Aws::GameLift::Server:: LogParameters 必需：否
OnHealthCheck	Amazon GameLift Servers 为向服务器进程请求运行状况报告而调用的回调函数。Amazon GameLift Servers 每隔 60 秒调用此函数一次，并等待 60 秒以获取响应。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。如果没有返回任何响应，Amazon GameLift Servers 会将服务器进程记录为运行状况不佳。 类型：std::function<bool()> onHealthCheck 必需：否
OnProcessTerminate	Amazon GameLift Servers 为强制关闭服务器进程而调用的回调函数。调用此函数之后，Amazon GameLift Servers 会等待 5 分钟，以确保服务器进程关闭并通过 ProcessEnding() 调用作出响应，随后才会终止该服务器进程。 类型：std::function<void()> onProcessTerminate 是否必需：是

OnStartGameSession

调Amazon GameLift Servers用以激活新游戏会话的回调函数。Amazon GameLift Servers应客户端请求调用此函数[CreateGameSession](#)。回调函数将传递一个 [GameSession](#) 对象。

类型：`const std::function<void (Aws::GameLift::Model::GameSession)> onStartGameSession`

是否必需：是

OnUpdateGameSession

Amazon GameLift Servers 为向服务器进程传递更新的游戏会话对象而调用的回调函数。当对战回填请求处理完毕时，Amazon GameLift Servers 会调用此函数以提供更新的对战构建器数据。它将传递一个 [GameSession](#) 对象、一个状态更新（`updateReason`）以及对战回填票证 ID。

类型：`std::function<void(Aws::GameLift::Server::Model::UpdateGameSession)> onUpdateGameSession`

必需：否

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会使用该端口号。

类型：`Integer`

是否必需：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）。

Properties	描述
GameSession	一个 GameSession 对象。该GameSession 对象包含描述游戏会话的属性。 类型：Aws::GameLift::Server::GameSession 是否必需：是
UpdateReason	更新游戏会话的原因。 类型：Aws::GameLift::Server::UpdateReason 是否必需：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。

Properties	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：std::string

Properties	描述
	必需：否
Name	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 Servers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 Servers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

Properties	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：<code>std::string</code> 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：<code>std::string</code> 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：<code>std::vector<GameProperty></code> 必需：否

Properties	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：<code><unique identifier>.<region identifier>.amazon gamelift.com</code> - Non-TLS-enabled 舰队：<code>ec2-<unique identifier>.compute.amazonaws.com</code>。 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：<code>std::string</code> 必需：否

ServerParameters

游戏服务器进程用来与 Amazon GameLift Servers 服务建立连接的信息。仅当游戏服务器生成包将在没有 Amazon GameLift Servers 代理的情况下部署到 Anywhere 实例集或容器实例集时，才需在调用 [InitSDK\(\)](#) 时包含这些参数。对于所有其他部署场景，请调用 [InitSDK\(\)](#) 而不使用参数。

Properties	描述
webSocketUrl	当您为 Amazon GameLift Servers Anywhere 计算资源调用 RegisterCompute 时，<code>GameLiftServerSdkEndpoint</code> Amazon GameLift Servers 会返回结果。 类型：<code>std::string</code> 是否必需：是
ProcessID	注册到托管游戏的服务器进程的唯一标识符。

Properties	描述
	类型 : <code>std::string</code> 是否必需 : 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 : <code>std::string</code> 是否必需 : 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 : <code>std::string</code> 是否必需 : 是
AuthToken	由 Amazon GameLift Servers 生成的身份验证令牌，用于向 Amazon GameLift Servers 对您的服务器进行身份验证。有关更多信息，请参阅 GetComputeAuthToken 。 类型 : <code>std::string</code> 是否必需 : 是

StartMatchBackfillRequest

用于创建对战回填请求的信息。游戏服务器通过调用 [StartMatchBackfill\(\)](#) 将这些信息传达给 Amazon GameLift Servers。

Properties	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。

Properties	描述
	类型： <code>std::string</code> 是否必需：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： <code>std::string</code> 是否必需：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： <code>std::vector<Player></code> 是否必需：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 Servers 会生成值。使用此标识符可跟踪对战回填票证状态或取消请求（如需要）。 类型： <code>std::string</code> 必需：否

玩家

此数据类型代表对战中的玩家。启动对战请求时，玩家会有玩家 ID、属性数据以及可能的延迟数据。对战结束后，Amazon GameLift Servers 会添加团队信息。

Properties	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
Team	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。服务器进程通过向 Amazon GameLift Servers 调用 [DescribePlayerSessions\(\)](#) 来提供此信息。

Properties	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 <code>arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string></code>。GameSessionId 是自定义 ID 字符串或 类型：<code>std::string</code> 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：<code>std::string</code> 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：<code>std::string</code> 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。

Properties	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。游戏服务器通过调用 [StopMatchBackfill\(\)](#) 将此信息传达给 Amazon GameLift Servers 服务。

Properties	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。

Properties	描述
	类型：char[] 必需：否
TicketId	要取消的回调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。

Properties	说明
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
N	表示属性值。 类型：double

Properties	说明
	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 Amazon GameLift Servers 设置 IAM 服务角色](#)。

Properties	描述
RoleArn	服务角色的 Amazon 资源名称 (ARN)，用于扩展对您的 AWS 资源的有限访问权限。 类型：std::string 必需：否
RoleSessionName	可用于唯一标识会话的角色会 AWS Security Token Service AssumeRole 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：

Properties	说明
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 是否必需：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：

Properties	说明
结果	操作的结果。 类型：std::string

Properties	说明
	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 是否必需：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：

Properties	说明
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。 类型：Aws::GameLift::Server::Model::DescribePlayerSessionsResult&&

Properties	说明
	必需：否
成功	操作是否成功。 类型：bool 是否必需：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。

Properties	说明
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 是否必需：是
PlayerSessions	类型：IList< the section called “PlayerSession” > 必填
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

Properties	说明
成功	操作是否成功。 类型：bool 是否必需：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：

Properties	说明
成功	操作是否成功。 类型：bool 是否必需：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：

Properties	说明
成功	操作是否成功。

Properties	说明
	类型：bool 是否必需：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 Servers 传递给游戏服务器的玩家会话。有关更多信息，请参阅 [PlayerSession](#)。

Properties	说明
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
PlayerData	类型：std::string 必需：否
PlayerId	类型：std::string

Properties	说明
	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
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：

Properties	说明
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：<code>StartMatchBackfillResult&&</code> 必需：否
成功	操作是否成功。 类型：<code>bool</code> 是否必需：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：

Properties	说明
TicketId	对战配置的唯一标识符。如果此处未指定任何票证 ID，则 Amazon GameLift Servers 将生成一个采用 UUID 形式的 ID。使用此标识符跟踪对战回填单状态并检索匹配结果。 类型：<code>std::string</code> 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：

Properties	说明
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：<code>Aws::GameLift::Server::Model::GetComputeCertificateResult&&</code> 必需：否
成功	操作是否成功。 类型：<code>bool</code> 是否必需：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。

Properties	说明
CertificatePath	计算资源上 TLS 证书的路径。使用 Amazon GameLift Servers 托管式实例集时，此路径包含：

Properties	说明
	• <code>certificate.pem</code> : 最终用户证书。完整的证书链是<code>certificateChain.pem</code> 附加到此证书的组合。 • <code>certificateChain.pem</code> : 包含根证书和中间证书的证书链。 • 根证书。 • <code>privateKey.pem</code> : 最终用户证书的私钥。 类型 : <code>std::string</code> 必需 : 否
ComputeName	您的计算资源的名称。 类型 : <code>std::string</code> 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：

Properties	说明
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 : <code>Aws::GameLift::Server::Model::GetFleetRoleCredentialsResult</code> 必需 : 否

Properties	说明
成功	操作是否成功。 类型：bool 是否必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLiftError” 必需：否

GetFleetRoleCredentialsResult

Properties	说明
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。 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

Properties	说明
	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 是否必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLiftError” 必需：否

初始化 SDKOutcome

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。

Properties	说明
成功	操作是否成功。 类型：bool

Properties	说明
	是否必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLiftError” 必需：否

GameLiftError

Properties	说明
ErrorType	错误的类型。 类型： GameLiftErrorType 枚举 。 必需：否
ErrorMessage	错误类型的名称。 类型： <code>std::string</code> 必需：否
ErrorMessage	错误消息。 类型： <code>std::string</code> 必需：否

枚举

为适用于 Amazon GameLift Servers (C++) 的服务器 SDK 定义了以下枚举类型：

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 Amazon GameLift Servers 的本地连接失败。
- 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_
- ALREADY_INITIALIZED – Amazon GameLift Servers 服务器或客户端已使用 Initialize() 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 Servers 客户端尚未初始化。
- GAMELIFT_SERVER_NOT_INITIALIZED – Amazon GameLift Servers 服务器尚未初始化。
- GAME_SESSION_ENDED_FAILED – 适用于 Amazon GameLift Servers 的服务器 SDK 无法联系服务部门报告游戏会话已结束。
- GAME_SESSION_NOT_READY – Amazon GameLift Servers 服务器游戏会话未激活。
- GAME_SESSION_READY_FAILED – 适用于 Amazon GameLift Servers 的服务器 SDK 无法联系服务部门报告游戏会话已准备就绪。
- INITIALIZATION_MISMATCH – 在 Server::Initialization() 之后调用了客户端方法，反之亦然。
- NOT_INITIALIZED – Amazon GameLift Servers 服务器或客户端尚未使用 Initialize() 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 – 适用于 Amazon GameLift Servers 的服务器 SDK 无法联系服务部门报告流程即将结束。
- PROCESS_NOT_ACTIVE — 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_READY – 服务器进程尚未准备好激活。

- PROCESS_READY_FAILED – 适用于 Amazon GameLift Servers 的服务器 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 服务器后端组件初始化失败。
- 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。

适用于 Amazon GameLift Servers 的 C++ 服务器 SDK 5.x – 数据类型

主题

- [GetSdkVersion\(\)](#)
- [InitMetrics\(\)](#)
- [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 SDK 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();
```

InitMetrics()

初始化指标系统，用于收集和报告服务器性能数据。为了获得最佳结果，请在调用 [InitSDK\(\)](#) 之前调用此方法，以便在 SDK 初始化期间启用指标收集。

语法

```
Aws::GameLift::GenericOutcome InitMetrics();  
Aws::GameLift::GenericOutcome InitMetrics(const  
    Aws::GameLift::Server::MetricsParameters &metricsParameters);
```

参数

[MetricsParameters](#) (可选)

指标收集的配置参数。如果未提供，则使用可以被环境变量覆盖的默认值。

返回值

如果成功，则返回一个表示成功的 [the section called “GenericOutcome”](#) 对象。如果不成功，将返回错误消息。

示例

```
// Initialize with default parameters (uses environment variables if available)  
// Defaults: localhost:8125 for StatsD, localhost:8126 for crash reporter  
// FlushInterval: 10000ms, MaxPacketSize: 512 bytes  
Aws::GameLift::GenericOutcome outcome = Aws::GameLift::Server::InitMetrics();  
if (outcome.IsSuccess()) {  
    // Metrics system initialized successfully  
}  
  
// Initialize with custom parameters  
Aws::GameLift::Server::MetricsParameters metricsParams("localhost", 8125, "crash-host",  
    8126, 5000, 1024);  
Aws::GameLift::GenericOutcome customOutcome =  
    Aws::GameLift::Server::InitMetrics(metricsParams);  
if (customOutcome.IsSuccess()) {  
    // Metrics system initialized with custom parameters  
}
```

InitSDK()

初始化 Amazon GameLift Servers 开发工具包。在与 Amazon GameLift Servers 相关的任何其他初始化步骤之前执行，在启动时调用此方法。此操作从主机环境读取服务器参数，以设置游戏服务器进程和 Amazon GameLift Servers 服务之间的通信。它使用幂等性令牌，因此当调用失败时，您可以放心地重试此调用。

如果游戏服务器生成包将在没有 Amazon GameLift Servers 代理的情况下部署到 Amazon GameLift Servers Anywhere 实例集或容器实例集，请调用 [InitSDK\(\)](#) 并指定一组服务器参数。

语法

```
Server::InitSDKOutcome Server::initSdkOutcome = InitSDK();
```

返回值

返回一个[the section called “初始化 SDKOutcome”](#)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 Amazon GameLift Servers Agent
to enable further communication.
Aws::GameLift::Server::InitSDKOutcome initSdkOutcome =
    Aws::GameLift::Server::InitSDK();
```

InitSDK()

初始化 Amazon GameLift Servers 开发工具包。在与 Amazon GameLift Servers 相关的任何其他初始化步骤之前执行，在启动时调用此方法。此操作需要一组服务器参数来设置游戏服务器进程和 Amazon GameLift Servers 服务之间的通信。它使用幂等性令牌，因此当调用失败时，您可以放心地重试此调用。

如果游戏服务器生成包将在有 Amazon GameLift Servers 代理的情况下部署到 Amazon GameLift Servers 托管式 EC2 实例集，或者部署到 Amazon GameLift Servers Anywhere 实例集或容器实例集，请调用 [InitSDK\(\)](#) 而不使用服务器参数。

语法

```
Server::InitSDKOutcome Server::initSdkOutcome = InitSDK(serverParameters);
```

参数

[ServerParameters](#)

要在 Amazon GameLift Servers Anywhere 实例集上初始化游戏服务器，请使用以下信息构造 `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 Amazon GameLift Servers Anywhere 计算的 Amazon GameLift Servers 实例集的 ID。
- Amazon GameLift Servers 操作生成的授权令牌。

返回值

返回一个[the section called “初始化 SDKOutcome”](#)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

Amazon GameLift Servers 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 Amazon GameLift Servers Agent
to enable further communication.
Aws::GameLift::Server::InitSDKOutcome initSdkOutcome =
    Aws::GameLift::Server::InitSDK(serverParameters);
```

ProcessReady()

通知 Amazon GameLift Servers，该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 Servers 服务调用其与服务器进程通信。
- 服务器进程正在侦听的端口号。
- 您希望 Amazon GameLift Servers 捕获和存储的任何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 Servers 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。这些参数指定了 Amazon GameLift Servers 在特定环境下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：

- 游戏服务器代码中实现的回调方法的名称，Amazon GameLift Servers 服务调用其与服务器进程通信。
- 服务器进程正在侦听的端口号。
- 您希望 Amazon GameLift Servers 捕获和存储的任何游戏会话特定文件的路径。

是否必需：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 Amazon GameLift Servers，该服务器进程即将终止。请在已完成所有其他清理任务（包括关闭活动游戏会话）但尚未终止进程时调用此方法。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，则生成的实例集事件为 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 Servers，该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

```
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](#)。

是否必需：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。服务器进程在收到来自 Amazon GameLift Servers 的 `onProcessTerminate()` 回调后执行此操作。Amazon GameLift Servers 会出于以下原因调用 `onProcessTerminate()`：

- 服务器进程报告运行状况不佳或未响应 Amazon GameLift Servers。
- 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 6373559680000000000 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 Servers，具有指定玩家会话 ID 的玩家已连接到服务器进程并且需要进行验证。Amazon GameLift Servers 将验证该玩家会话 ID 是否有效。玩家会话经过验证后，Amazon GameLift Servers 会将玩家位置的状态从 `RESERVED` 更改为 `ACTIVE`。

语法

```
GenericOutcome AcceptPlayerSession(String playerId)
```

参数

playerSessionId

创建新玩家会话时由 Amazon GameLift Servers 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。IDs

```
void ReceiveConnectingPlayerSessionID (Connection& connection, const std::string&
playerSessionId)
{
    Aws::GameLift::GenericOutcome connectOutcome =
    Aws::GameLift::Server::AcceptPlayerSession(playerSessionId);
    if(connectOutcome.IsSuccess())
    {
        connectionToSessionMap.emplace(connection, playerId);
        connection.Accept();
    }
    else
    {
        connection.Reject(connectOutcome.GetError().GetMessage());
    }
}
```

RemovePlayerSession()

通知 Amazon GameLift Servers，有玩家已断开与服务器进程的连接。作为回应，Amazon GameLift Servers 会将玩家位置更改为可用。

语法

```
GenericOutcome RemovePlayerSession(String playerId)
```

参数

playerSessionId

创建新玩家会话时由 Amazon GameLift Servers 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。省略NextToken并将Limit值设置为10，即可Amazon GameLift Servers返回与请求匹配的前10个玩家会话记录。

```
// 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。如果成功匹配了新玩家，则 Amazon GameLift Servers 会使用回调函数 OnUpdateGameSession() 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对战回填请求。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](#)

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；如果未提供任何 ID，则 Amazon GameLift Servers 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。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()

取消活动的对战回填请求。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 Amazon GameLift Servers Anywhere 计算资源和 Amazon GameLift Servers 之间的网络连接。当您将计算设备注册到 Amazon GameLift Servers Anywhere 实例集时，您可以使用该证书路径。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 Servers 与其他 AWS 服务互动的 IAM 角色凭证。有关更多信息，请参阅[将您的Amazon GameLift Servers托管游戏服务器连接到其他 AWS 资源](#)。

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 *[host-id]* “*[fleet-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 Servers 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 实例集，并且没有在每次游戏会话后终止服务器进程，请先调用 `Destroy()`，再通过 `InitSDK()` 重新初始化，最后使用 `ProcessReady()` 通知 Amazon GameLift Servers 该进程已准备好托管游戏会话。

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);  
}
```

适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 操作

使用服务器 SDK 5.x 参考，将要托管的多人游戏与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。如果您使用的是适用于 Unity 的 Amazon GameLift Servers 插件，还可以参阅[Amazon GameLift Servers 适用于 Unity 的插件 \(服务器 SDK 5.x \)](#)。

适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 数据类型

使用 Amazon GameLift Servers C# 服务器 SDK 5.x 参考，将要托管的多人游戏与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。如果您使用的是适用于 Unity 的 Amazon GameLift Servers 插件，还可以参阅[Amazon GameLift Servers 适用于 Unity 的插件 \(服务器 SDK 5.x \)](#)。

[适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 操作](#)

数据类型

- [LogParameters](#)
- [MetricsParameters](#)
- [ProcessParameters](#)
- [UpdateGameSession](#)
- [GameSession](#)
- [ServerParameters](#)
- [StartMatchBackfillRequest](#)
- [玩家](#)
- [DescribePlayerSessionsRequest](#)
- [StopMatchBackfillRequest](#)
- [GetFleetRoleCredentialsRequest](#)
- [AttributeValue](#)
- [AwsStringOutcome](#)
- [GenericOutcome](#)
- [MetricsManagerOutcome](#)
- [DescribePlayerSessionsOutcome](#)

- [DescribePlayerSessionsResult](#)
- [PlayerSession](#)
- [StartMatchBackfillOutcome](#)
- [StartMatchBackfillResult](#)
- [GetComputeCertificateOutcome](#)
- [GetComputeCertificateResult](#)
- [GetFleetRoleCredentialsOutcome](#)
- [GetFleetRoleCredentialsResult](#)
- [AwsDateTimeOutcome](#)
- [GameLiftError](#)
- [枚举](#)

LogParameters

使用此数据类型来标识游戏会话期间生成的哪些文件需要在游戏会话结束后由游戏服务器上传到 Amazon GameLift Servers。游戏服务器通过 [ProcessReady\(\)](#) 调用将 LogParameters 传达给 Amazon GameLift Servers。

属性	描述
LogPaths	您希望 Amazon GameLift Servers 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

MetricsParameters

使用此数据类型配置游戏服务器的指标收集和崩溃报告功能。游戏服务器通过 [InitMetrics\(\)](#) 调用将 MetricsParameters 传达给 Amazon GameLift Servers。

属性	描述
StatsdHost	用于指标收集的 StatsD 服务器的主机名或 IP 地址。 类型 : String 必需 : 否
StatsdPort	用于收集指标的 StatsD 服务器的端口号。 类型 : Integer 必需 : 否
CrashReporterHost	崩溃报告服务器的主机名或 IP 地址。 类型 : String 必需 : 否
CrashReporterPort	崩溃报告服务器的端口号。 类型 : Integer 必需 : 否
FlushIntervalMs	将指标数据刷新到服务器的时间间隔 (以毫秒为单位)。 类型 : Integer 必需 : 否
MaxPacketSize	发送到服务器的指标数据包的最大大小 (以字节为单位)。

类型：Integer

必需：否

ProcessParameters

此数据类型包含一组在 [ProcessReady\(\)](#) 调用中发送到 Amazon GameLift Servers 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 是否必需：是
OnHealthCheck	Amazon GameLift Servers 为向服务器进程请求运行状况报告而调用的回调函数的名称。 Amazon GameLift Servers 每隔 60 秒调用此函数一次。调用此函数后，Amazon GameLift Servers 将等待 60 秒接收响应，如果未收到响应，则 Amazon GameLift Servers 会将该服务器进程记录为运行状况不佳。 类型：void OnHealthCheckDelegate() 是否必需：是
OnProcessTerminate	Amazon GameLift Servers 为强制关闭服务器进程而调用的回调函数的名称。调用此函数之后，Amazon GameLift Servers 等待五分钟以便服务器进程关闭，然后使用 ProcessEnding() 调用响应，再关闭服务器进程。 类型：void OnProcessTerminate Delegate() 是否必需：是

OnStartGameSession

为激活新游戏会话而Amazon GameLift Servers调用的回调函数的名称。Amazon GameLift Servers调用此函数是为了响应客户端的请求[CreateGameSession](#)。回调函数将采用一个[GameSession](#)对象。

类型：void OnStartGameSession
Delegate([GameSession](#))

是否必需：是

OnUpdateGameSession

Amazon GameLift Servers 为向服务器进程传递更新的游戏会话对象而调用的回调函数的名称。当对战回填请求处理完毕时，Amazon GameLift Servers 会调用此函数以提供更新的对战构建器数据。它将传递一个 [GameSession](#) 对象、一个状态更新 (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会使用该端口号。

类型：Integer

是否必需：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。

Properties	描述
GameSession	一个 GameSession 对象。该GameSession 对象包含描述游戏会话的属性。 类型 : <code>GameSession GameSession()</code> 是否必需 : 是
UpdateReason	更新游戏会话的原因。 类型 : <code>UpdateReason UpdateReason()</code> 是否必需 : 是
BackfillTicketId	尝试更新游戏会话的回填票证的 ID。 类型 : <code>String</code> 是否必需 : 是

GameSession

游戏会话的详细信息。

Properties	描述
GameSessionId	会话的唯一标识符。游戏会话ARN具有以下格式: <code>arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token></code> 类型 : <code>String</code> 必需 : 否
Name	游戏会话的描述性标签。 类型 : <code>String</code>

Properties	描述
	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
端口：	游戏会话的端口号。要连接到Amazon GameLift Servers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 Servers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

Properties	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：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
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>.amazon gamelift.com • Non-TLS-enabled 舰队：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 Amazon GameLift Servers Anywhere 服务器与 Amazon GameLift Servers 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 Servers 托管式 EC2 实例上的服务器，请使用空对象。

Properties	描述
WebSocketUrl	这是在 Amazon GameLift Servers Anywhere 中，当您执行 RegisterCompute 时返回的 GameLiftServerSdkEndpoint 。 类型：String 是否必需：是
ProcessId	注册到托管游戏的服务器进程的唯一标识符。 类型：String 是否必需：是
HostId	托管游戏的服务器进程的主机的唯一标识符。HostID 是您注册 ComputeName 计算机时使用的。欲了解更多信息，请参阅 RegisterCompute 类型：String 是否必需：是
FleetId	计算注册到的队列的实例集 ID。有关更多信息，请参阅RegisterCompute。 类型：String 是否必需：是
AuthToken	由 Amazon GameLift Servers 生成的身份验证令牌，用于向 Amazon GameLift Servers 对您

Properties	描述
	的服务器进行身份验证。有关更多信息，请参阅 GetComputeAuthToken 。
	类型：String
	是否必需：是

StartMatchBackfillRequest

用于创建对战回填请求的信息。游戏服务器通过调用 [StartMatchBackfill\(\)](#) 将这些信息传达给 Amazon GameLift Servers。

Properties	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 是否必需：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 是否必需：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>

Properties	描述
	是否必需：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 Servers 会生成值。使用此标识符可跟踪对战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。启动对战请求时，玩家会有玩家 ID、属性数据以及可能的延迟数据。对战结束后，Amazon GameLift Servers 会添加团队信息。

Properties	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：Dictionary<string, Attribute Value>

Properties	描述
	必需：否
PlayerId	玩家的唯一标识符。 类型：String 必需：否
Team	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 a PlayerSessionId 以请求特定的玩家会话；(2) 提供 a GameSessionId 以请求指定游戏会话中的所有玩家会话；或 (3) 提供 a 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。

Properties	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。

Properties	描述
	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 and/or 已通过验证。 • 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 — 已收到玩家会话请求，但未在超时限制（60 秒）内验证未连接 and/or 的玩家。 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

Properties	描述
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。游戏服务器通过调用 [StopMatchBackfill\(\)](#) 将此信息传达给 Amazon GameLift Servers 服务。

Properties	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 是否必需：是
MatchmakingConfigurationArn	此请求发送到的对战构建器的唯一标识符。 类型：string 是否必需：是
TicketId	要取消的回填请求票证的唯一标识符。 类型：string 是否必需：是

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 Amazon GameLift Servers 设置 IAM 服务角色](#)。

Properties	描述
RoleArn	服务角色的 Amazon 资源名称 (ARN)，用于扩展对您的 AWS 资源的有限访问权限。 类型：string 是否必需：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。

Properties	说明
属性类型	指定属性值的类型。 类型：一个 AttrType 枚举 值。 必需：否
S	表示字符串属性值。 类型：string 是否必需：是
N	表示属性值。 类型：double 是否必需：是
sl	表示字符串属性值的数组。

Properties	说明
	类型： <code>string[]</code> 是否必需：是
SDM	表示字符串键和双精度值的字典。 类型： <code>Dictionary<string, double></code> 是否必需：是

AwsStringOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

Properties	说明
结果	操作的结果。 类型： <code>string</code> 必需：否
成功	操作是否成功。 类型： <code>bool</code> 是否必需：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：

Properties	说明
成功	操作是否成功。 类型：bool 是否必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLiftError” 必需：否

MetricsManagerOutcome

表示调用 [InitMetrics\(\)](#) 的结果。包含成功时的 MetricsManager 实例或失败时的错误信息。

结果	用于收集和报告指标的 MetricsManager 实例。 类型：MetricsManager 必需：否
成功	操作是否成功。 类型：bool 是否必需：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：

Properties	说明
结果	操作的结果。 类型：the section called “DescribePlayerSessionsResult” 必需：否
成功	操作是否成功。 类型：bool 是否必需：是
错误	操作失败时发生的错误。 类型：the section called “GameLiftError” 必需：否

DescribePlayerSessionsResult

Properties	说明
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 是否必需：是
PlayerSessions	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

Properties	说明
	是否必需：是
错误	操作失败时发生的错误。 类型：the section called “GameLiftError” 必需：否

PlayerSession

Properties	说明
CreationTime	类型：long 是否必需：是
FleetId	类型：string 是否必需：是
GameSessionId	类型：string 是否必需：是
IpAddress	类型：string 是否必需：是
PlayerData	类型：string 是否必需：是
PlayerId	类型：string 是否必需：是
PlayerSessionId	类型：string 是否必需：是

Properties	说明
端口：	类型：int 是否必需：是
Status	类型： PlayerSessionStatus 枚举 。 是否必需：是
TerminationTime	类型：long 是否必需：是
DnsName	类型：string 是否必需：是

StartMatchBackfillOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

Properties	说明
结果	操作的结果。 类型： the section called “StartMatchBackfillResult” 必需：否
成功	操作是否成功。 类型：bool 是否必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLiftError” 必需：否

StartMatchBackfillResult

Properties	说明
TicketId	类型：string 是否必需：是

GetComputeCertificateOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

Properties	说明
结果	操作的结果。 类型： the section called “GetComputeCertificateResult” 必需：否
成功	操作是否成功。 类型：bool 是否必需：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。

Properties	说明
CertificatePath	类型：string

Properties	说明
	是否必需：是
ComputeName	类型：string 是否必需：是

GetFleetRoleCredentialsOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

Properties	说明
结果	操作的结果。 类型： the section called “GetFleetRoleCredentialsResult” 必需：否
成功	操作是否成功。 类型：bool 是否必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLiftError” 必需：否

GetFleetRoleCredentialsResult

Properties	说明
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。 类型：string

Properties	说明
	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 是否必需：是

Properties	说明
错误	操作失败时发生的错误。 类型： the section called “GameLiftError” 必需：否

AwsDateTimeOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

Properties	说明
结果	操作的结果。 类型：DateTime 必需：否
成功	操作是否成功。 类型：bool 是否必需：是
错误	操作失败时发生的错误。 类型： the section called “GameLiftError” 必需：否

GameLiftError

Properties	说明
ErrorType	错误的类型。 类型： GameLiftErrorType 枚举 。

Properties	说明
	必需：否
ErrorMessage	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

为适用于 Amazon GameLift Servers (C#) 的服务器 SDK 定义了以下枚举类型：

AttrType

- 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 Amazon GameLift Servers 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
- ALREADY_INITIALIZED – Amazon GameLift Servers 服务器或客户端已使用 Initialize() 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 Servers 客户端尚未初始化。
- GAMELIFT_SERVER_NOT_INITIALIZED – Amazon GameLift Servers 服务器尚未初始化。
- GAME_SESSION_ENDED_FAILED – 适用于 Amazon GameLift Servers 的服务器 SDK 无法联系服务部门报告游戏会话已结束。
- GAME_SESSION_NOT_READY – Amazon GameLift Servers 服务器游戏会话未激活。
- GAME_SESSION_READY_FAILED – 适用于 Amazon GameLift Servers 的服务器 SDK 无法联系服务部门报告游戏会话已准备就绪。
- INITIALIZATION_MISMATCH – 在 Server::Initialization() 之后调用了客户端方法，反之亦然。
- NOT_INITIALIZED – Amazon GameLift Servers 服务器或客户端尚未使用 Initialize() 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 – 适用于 Amazon GameLift Servers 的服务器 SDK 无法联系服务部门报告流程即将结束。
- PROCESS_NOT_ACTIVE — 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_READY – 服务器进程尚未准备好激活。
- PROCESS_READY_FAILED – 适用于 Amazon GameLift Servers 的服务器 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 服务器后端组件初始化失败。
- UNEXPECTED_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_URL
- WEBSOCKET_CONNECT_FAILURE_
- WEBSOCKET_RETRYABLE_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。

PlayerSessionStatus

- ACTIVE (处于活动状态)
- COMPLETED
- NOT_SET
- 预留
- 超时

[适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 数据类型](#)

主题

- [GetSdkVersion\(\)](#)
- [InitMetrics\(\)](#)
- [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 “AwsStringOutcome”](#) 对象返回当前 SDK 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();
```

InitMetrics()

初始化指标系统，用于收集和报告服务器性能数据。为了获得最佳结果，请在调用 [InitSDK\(\)](#) 之前调用此方法，以便在 SDK 初始化期间启用指标收集。

语法

```
MetricsManagerOutcome InitMetrics();  
MetricsManagerOutcome InitMetrics(MetricsParameters metricsParameters);
```

参数

[MetricsParameters](#) (可选)

指标收集的配置参数。如果未提供，则使用可以被环境变量覆盖的默认值。

返回值

如果成功，则返回包含该 MetricsManager 实例的[the section called “MetricsManagerOutcome”](#)对象。如果不成功，将返回错误消息。

示例

```
// Initialize with default parameters (uses environment variables if available)
var outcome = GameLiftServerAPI.InitMetrics();
if (outcome.Success) {
    var metricsManager = outcome.Result;
} else {
    Console.WriteLine("Failed to initialize metrics: " + outcome.Error.ErrorMessage);
}

// Initialize with custom parameters
var metricsParams = new MetricsParameters("localhost", 8125, "crash-host", 9999, 1000,
    1024);
var customOutcome = GameLiftServerAPI.InitMetrics(metricsParams);
if (customOutcome.Success) {
    var metricsManager = customOutcome.Result;
}
```

InitSDK()

为托管式 EC2 实例集初始化 Amazon GameLift Servers SDK。在与 Amazon GameLift Servers 相关的任何其他初始化发生之前，在启动时调用此方法。此方法从主机环境读取服务器参数，以设置服务器与 Amazon GameLift Servers 服务之间的通信。它使用幂等性令牌，因此当调用失败时，您可以放心地重试此调用。

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 实例集初始化 Amazon GameLift Servers SDK。在与 Amazon GameLift Servers 相关的任何其他初始化发生之前，在启动时调用此方法。此方法需要明确的服务器参数来设置服务器和 Amazon GameLift Servers 服务之间的通信。它使用幂等性令牌，因此当调用失败时，您可以放心地重试此调用。

语法

```
GenericOutcome InitSDK(ServerParameters serverParameters);
```

参数

[ServerParameters](#)

要在 Amazon GameLift Servers Anywhere 实例集上初始化游戏服务器，请使用以下信息构造 `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 Amazon GameLift Servers Anywhere 计算的 Amazon GameLift Servers 实例集的 ID。
- Amazon GameLift Servers 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 Amazon GameLift Servers，该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 Amazon GameLift Servers，该服务器进程即将终止。请在已完成所有其他清理任务（包括关闭活动游戏会话）但尚未终止进程时调用此方法。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，则生成的实例集事件为 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 Servers，该服务器进程已激活游戏会话，现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()
```

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 “GameLiftError”](#)。

语法

```
AwsStringOutcome GetGameSessionId()
```

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。

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。服务器进程在收到来自 Amazon GameLift Servers 的 `onProcessTerminate()` 回调后执行此操作。Amazon GameLift Servers 会出于以下原因调用 `onProcessTerminate()`：

- 服务器进程报告运行状况不佳或未响应 Amazon GameLift Servers。
- 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 “AwsDateTimeOutcome”](#)对象形式返回终止时间。该值是终止时间，以后经过的刻度表示。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 Servers，具有指定玩家会话 ID 的玩家已连接到服务器进程并且需要进行验证。Amazon GameLift Servers 将验证该玩家会话 ID 是否有效。玩家会话经过验证后，Amazon GameLift Servers 会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 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

此示例说明了处理连接请求的函数，包括验证和拒绝无效的玩家会话。IDs

```
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 Servers，有玩家已断开与服务器进程的连接。作为回应，Amazon GameLift Servers 会将玩家位置更改为可用。

语法

```
GenericOutcome RemovePlayerSession(String playerSessionId)
```

参数

playerSessionId

创建新玩家会话时由 Amazon GameLift Servers 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 “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。省略NextToken并将Limit值设置为10，Amazon GameLift Servers将返回与请求匹配的前10个玩家会话记录。

```
// 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。如果成功匹配了新玩家，则 Amazon GameLift Servers 会使用回调函数 `OnUpdateGameSession()` 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对战回填请求。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()

取消活动的对战回填请求。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 Servers Anywhere 实例集时，您可以使用该证书路径。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 Servers 托管式实例集时，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 Servers 与其他 AWS 服务互动的 IAM 角色凭证。有关更多信息，请参阅[将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。

语法

```
GetFleetRoleCredentialsOutcome GetFleetRoleCredentials(GetFleetRoleCredentialsRequest request);
```

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 [the section called “GetFleetRoleCredentialsOutcome”](#) 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 Servers 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 实例集，并且没有在每次游戏会话后终止服务器进程，请先调用 `Destroy()`，再通过 `InitSDK()` 重新初始化，最后使用 `ProcessReady()` 通知 Amazon GameLift Servers 该进程已准备好托管游戏会话。

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 Amazon GameLift Servers 的 Go 服务器 SDK – 操作

使用服务器 SDK 5.x 参考，将要托管的多人游戏与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

GameLiftServerAPI.go 定义了 Go 服务器软件开发工具包 操作。

适用于 Amazon GameLift Servers 的 Go 服务器 SDK – 数据类型

使用服务器 SDK 参考，将要托管的多人游戏与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

GameLiftServerAPI.go 定义了 Go 服务器软件开发工具包 操作。

[适用于 Amazon GameLift Servers 的 Go 服务器 SDK – 操作](#)

数据类型

- [LogParameters](#)
- [MetricsParameters](#)
- [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 Servers 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，作为 ProcessParameters 对象的一部分向 Amazon GameLift Servers 提供 LogParameters。

属性	描述
LogPaths	您希望 Amazon GameLift Servers 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：[]string 必需：否

MetricsParameters

用于初始化指标系统的配置参数通信对象。此配置用于设置 StatsD 报告、崩溃报告和指标处理行为。游戏服务器通过调用 [InitMetrics\(\)](#) 向 Amazon GameLift Servers 提供 MetricsParameters。

属性	描述
StatsdHost	用于指标报告的 StatsD 服务器主机 (例如“localhost”)。

	类型：string 是否必需：是
StatsdPort	用于指标报告的 StatsD 服务器端口（例如 8125）。 类型：int 是否必需：是
CrashReporterHost	用于崩溃跟踪和进程监控的崩溃报告器主机。 类型：string 是否必需：是
CrashReporterPort	用于崩溃跟踪和进程监控的崩溃报告器端口。 类型：int 是否必需：是
FlushIntervals	指标刷新闻隔（以毫秒为单位）。控制将指标发送到 StatsD 的频率。 类型：int 是否必需：是
MaxPacketSize	指标的最大数据包大小（以字节为单位）。限制发送到 StatsD 的 UDP 数据包的大小。 类型：int 是否必需：是

ProcessParameters

描述服务器进程与 Amazon GameLift Servers 之间通信的对象。服务器进程通过调用 [ProcessReady\(\)](#) 向 Amazon GameLift Servers 提供此信息。

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 GameLift Servers 会复制并存储这些文件，以便将来访问。 类型：LogParameters 必需：否
OnHealthCheck	Amazon GameLift Servers 为向服务器进程请求运行状况报告而调用的回调函数。Amazon GameLift Servers 每隔 60 秒调用此函数一次，并等待 60 秒以获取响应。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。如果没有返回任何响应，Amazon GameLift Servers 会将服务器进程记录为运行状况不佳。 类型：<code>OnHealthCheck func() bool</code> 必需：否
OnProcessTerminate	Amazon GameLift Servers 为强制关闭服务器进程而调用的回调函数。调用此函数之后，Amazon GameLift Servers 会等待 5 分钟，以确保服务器进程关闭并通过 ProcessEnding() 调用作出响应，随后才会终止该服务器进程。 类型：<code>OnProcessTerminate func()</code> 是否必需：是
OnStartGameSession	Amazon GameLift Servers 为向服务器进程传递更新的游戏会话对象而调用的回调函数。当对战回填请求处理完毕时，Amazon GameLift Servers 会调用此函数以提供更新的对战构建器数据。它将传递一个 GameSession 对象、一个状态更新（<code>updateReason</code>）以及对战回填票证 ID。 类型：<code>OnStartGameSession func (model.GameSession)</code> 是否必需：是

OnUpdateGameSession	Amazon GameLift Servers 为向服务器进程传递更新的游戏会话信息而调用的回调函数。当对战回填请求处理完毕时，Amazon GameLift Servers 会调用此函数以提供更新的对战构建器数据。 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会使用该端口号。 类型：int 是否必需：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（如果使用回填填充填充游戏会话中的玩家会话）。

Properties	描述
GameSession	一个 GameSession 对象。该GameSession 对象包含描述游戏会话的属性。 类型：GameSession GameSession() 是否必需：是
UpdateReason	更新游戏会话的原因。 类型：UpdateReason UpdateReason() 是否必需：是
BackfillTicketId	尝试更新游戏会话的回填票证的 ID。 类型：String

Properties	描述
	必需：否

GameSession

游戏会话的详细信息。

Properties	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
Name	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
端口：	游戏会话的端口号。要连接到Amazon GameLift Servers游戏服务器，应用程序需要 IP 地址和端口号。 类型：Integer

Properties	描述
	必需：否
IpAddress	游戏会话的 IP 地址。要连接到Amazon GameLift Servers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

Properties	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 - Non-TLS-enabled 舰队：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 Amazon GameLift Servers Anywhere 服务器与 Amazon GameLift Servers 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 Servers 托管式 EC2 实例上的服务器，请使用空对象。

Properties	描述
WebSocket URL	当您为 Amazon GameLift Servers Anywhere 计算资源调用 RegisterCompute 时，GameLiftServerSdkEndpoint 返回 Amazon GameLift Servers 会返回结果。 类型：string 是否必需：是
ProcessID	注册到托管游戏的服务器进程的唯一标识符。 类型：string 是否必需：是
HostID	托管新服务器进程的计算资源的唯一标识符。

Properties	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 是否必需：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 是否必需：是
AuthToken	由 Amazon GameLift Servers 生成的身份验证令牌，用于向 Amazon GameLift Servers 对您的服务器进行身份验证。有关更多信息，请参阅 GetComputeAuthToken。 类型：string 是否必需：是

StartMatchBackfillRequest

用于创建对战回填请求的信息。游戏服务器通过调用 [StartMatchBackfill\(\)](#) 将这些信息传达给 Amazon GameLift Servers。

Properties	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 是否必需：是

Properties	描述
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。 类型：String 是否必需：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 是否必需：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 Servers 会生成值。使用此标识符可跟踪对战回填票证状态或取消请求（如果需要）。 类型：String 必需：否

玩家

表示对战中的玩家的对象。启动对战请求时，玩家会有玩家 ID、属性数据以及可能的延迟数据。对战结束后，Amazon GameLift Servers 会添加团队信息。

Properties	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

Properties	描述
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 使用的密钥相匹配。 有关玩家属性的更多信息，请参阅AttributeValue。 类型：<code>map[string] AttributeValue</code> 必需：否
PlayerId	玩家的唯一标识符。 类型：<code>String</code> 必需：否
Team	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：<code>String</code> 必需：否

DescribePlayerSessionsRequest

一个对象，用于指定要检索哪些玩家会话。服务器进程通过向 Amazon GameLift Servers 调用 [DescribePlayerSessions\(\)](#) 来提供此信息。

Properties	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 <code>arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string></code>。GameSessionID 是自定义 ID 字符串或生成的字符串。 类型：<code>String</code> 必需：否

Properties	描述
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
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。游戏服务器通过调用 [StopMatchBackfill\(\)](#) 将此信息传达给 Amazon GameLift Servers 服务。

Properties	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 Amazon GameLift Servers 设置 IAM 服务角色](#)。

Properties	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 是否必需：是
RoleSessionName	描述角色证书使用情况的会话名称。

Properties	描述
	类型 : string
	是否必需 : 是

适用于 Amazon GameLift Servers 的 Go 服务器 SDK – 数据类型

操作

- [GetSdkVersion\(\)](#)
- [InitMetrics\(\)](#)
- [InitMetricsFromEnvironment\(\)](#)
- [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()
```

InitMetrics()

初始化 Amazon GameLift Servers SDK 的指标收集。此方法设置指标报告，有助于监控服务器性能和
运行状况。请在 [InitSDK\(\)](#) 之后、[ProcessReady\(\)](#) 之前调用此方法。

语法

```
func InitMetrics() error  
func InitMetrics(metricsParameters MetricsParameters) error
```

参数

MetricsParameters (可选)

用于配置指标收集的 MetricsParameters 对象。如果未提供，则使用默认指标配置。该
MetricsParameters结构包含以下字段：

- StatsdHost – StatsD 服务器的主机名或 IP 地址。
- StatsdPort – StatsD 服务器进程的端口号。
- CrashReporterHost – 崩溃报告服务的主机名或 IP 地址。
- CrashReporterPort – 崩溃报告服务的端口号。
- FlushIntervalMs – 刷新指标数据的时间间隔 (以毫秒为单位)。
- MaxPacketSize – 指标数据包的最大大小 (以字节为单位)。

有关 MetricsParameters 结构的更多信息，请参阅适用于 [C# 数据类型的服务器 SDK 5.x](#)。

返回值

如果成功，则返回 `nil` 错误，表示指标收集已成功初始化。

示例

使用默认配置初始化指标：

```
err := server.InitMetrics()
```

使用自定义配置初始化指标：

```
metricsParams := MetricsParameters{
    StatsdHost:      "localhost",
    StatsdPort:      8125,
    CrashReporterHost: "localhost",
    CrashReporterPort: 9125,
    FlushIntervalMs:  5000,
    MaxPacketSize:    1024,
}

err := server.InitMetrics(metricsParams)
```

InitMetricsFromEnvironment()

使用环境变量中的配置初始化 Amazon GameLift Servers SDK 的指标收集。此方法使用从运行时环境中派生的默认设置来设置指标报告。

请在 [InitSDK\(\)](#) 之后、[ProcessReady\(\)](#) 之前调用此方法。

语法

```
func InitMetricsFromEnvironment() error
```

返回值

如果成功，则返回 `nil` 错误，表示已使用环境配置成功初始化指标收集。

示例

```
err := server.InitMetricsFromEnvironment()
```

InitSDK()

初始化 Amazon GameLift Servers 开发工具包。在与 Amazon GameLift Servers 相关的任何其他初始化发生之前，在启动时调用此方法。此方法在服务器和 Amazon GameLift Servers 服务之间设置通信。

语法

```
func InitSDK(params ServerParameters) error
```

参数

[ServerParameters](#)

要在 Amazon GameLift Servers Anywhere 实例集上初始化游戏服务器，请使用以下信息构造 `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 Amazon GameLift Servers Anywhere 计算的 Amazon GameLift Servers 实例集的 ID。
- Amazon GameLift Servers 操作生成的授权令牌。

要在 Amazon GameLift Servers 托管式 EC2 实例集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 Servers 代理为您设置计算环境并自动连接到 Amazon GameLift Servers 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

Amazon GameLift Servers 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 Amazon GameLift Servers Agent
to enable further communication.
err := server.InitSDK(serverParameters)
```

Amazon GameLift Servers 托管式 EC2 示例

```
//Define the server parameters
serverParameters := ServerParameters {}

//Call InitSDK to establish a local connection with the Amazon GameLift Servers Agent
to enable further communication.
err := server.InitSDK(serverParameters)
```

ProcessReady()

通知 Amazon GameLift Servers，该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 Servers 服务调用其与服务器进程通信。
- 服务器进程正在侦听的端口号。
- [LogParameters](#) 数据类型包含您希望 Amazon GameLift Servers 服务器捕获并存储的任何游戏会话特定文件的路径。

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 Amazon GameLift Servers，该服务器进程即将终止。请在已完成所有其他清理任务（包括关闭活动游戏会话）但尚未终止进程时调用此方法。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，则生成的实例集事件为 `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 Servers，该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

```
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。服务器进程在收到来自 Amazon GameLift Servers 的 onProcessTerminate() 回调后执行此操作。Amazon GameLift Servers 会出于以下原因调用 onProcessTerminate()：

- 服务器进程报告运行状况不佳或未响应 Amazon GameLift Servers。
- 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 Servers，具有指定玩家会话 ID 的玩家已连接到服务器进程并且需要进行验证。Amazon GameLift Servers 将验证该玩家会话 ID 是否有效。玩家会话经过验证后，Amazon GameLift Servers 会将玩家位置的状态从 RESERVED 更改为 ACTIVE。

语法

```
func AcceptPlayerSession(playerSessionID string) error
```

参数

playerSessionId

创建新玩家会话时由 Amazon GameLift Servers 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。IDs

```
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 Servers，有玩家已断开与服务器进程的连接。作为回应，Amazon GameLift Servers 会将玩家位置更改为可用。

语法

```
func RemovePlayerSession(playerSessionID string) error
```

参数

playerSessionId

创建新玩家会话时由 Amazon GameLift Servers 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。省略 `NextToken` 并将 `Limit` 值设置为 10，即可 Amazon GameLift Servers 返回与请求匹配的前 10 个玩家会话记录。

```
// 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。如果成功匹配了新玩家，则 Amazon GameLift Servers 会使用回调函数 `OnUpdateGameSession()` 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对战回填请求。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；如果未提供任何 ID，则 Amazon GameLift Servers 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。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()

取消活动的对战回填请求。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 Servers Anywhere 实例集时，您可以使用该证书路径。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 TLS 证书的路径。使用 Amazon GameLift Servers 托管式实例集时，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()

检索您创建的用于将权限扩展 AWS 服务 到Amazon GameLift Servers其他角色的服务角色证书。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 Amazon GameLift Servers 设置 IAM 服务角色](#)。

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 Servers 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 实例集，并且没有在每次游戏会话后终止服务器进程，请先调用 `Destroy()`，再通过 `InitSDK()` 重新初始化，最后使用 `ProcessReady()` 通知 Amazon GameLift Servers 该进程已准备好托管游戏会话。

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)
    }
}
```

适用于 Amazon GameLift Servers 的 C++ (Unreal) 服务器 SDK 5.x – 操作

使用 Amazon GameLift Servers Unreal 服务器 SDK 5.x 参考，帮助您为多人游戏准备好与 Amazon GameLift Servers 配合使用。有关集成过程的详细信息，请参阅[借助服务器 SDK 将 Amazon GameLift](#)

[Servers 添加到游戏服务器](#)。如果您使用的是适用于 Unreal 的 Amazon GameLift Servers 插件，还可以参阅[Amazon GameLift Servers 虚幻引擎插件](#)。

 Note

本主题介绍在为 Unreal Engine 构建时可以使用的 Amazon GameLift Servers 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。

适用于 Amazon GameLift Servers 的 C++ (Unreal) 服务器 SDK 5.x – 数据类型

使用 Amazon GameLift Servers Unreal 服务器 SDK 5.x 参考，帮助您为多人游戏准备好与 Amazon GameLift Servers 配合使用。有关集成过程的详细信息，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。如果您使用的是适用于 Unreal 的 Amazon GameLift Servers 插件，还可以参阅[Amazon GameLift Servers 虚幻引擎插件](#)。

 Note

本主题介绍在为 Unreal Engine 构建时可以使用的 Amazon GameLift Servers 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。

[适用于 Amazon GameLift Servers 的 C++ \(Unreal \) 服务器 SDK 5.x – 操作](#)

数据类型

- [FProcess 参数](#)
- [UpdateGameSession](#)
- [GameSession](#)
- [FServer 参数](#)
- [FStartMatchBackfillRequest](#)
- [FPlayer](#)
- [FGameLiftDescribePlayerSessionsRequest](#)
- [FStopMatchBackfillRequest](#)
- [FAttribute 价值](#)
- [FGameLiftGetFleetRoleCredentialsRequest](#)

- [FGameLiftLongOutcome](#)
- [FGameLiftStringOutcome](#)
- [FGameLiftDescribePlayerSessionsOutcome](#)
- [FGameLiftDescribePlayerSessionsResult](#)
- [FGeneric结果](#)
- [FGameLiftPlayerSession](#)
- [FGameLiftGetComputeCertificateOutcome](#)
- [FGameLiftGetComputeCertificateResult](#)
- [FGameLiftGetFleetRoleCredentialsOutcome](#)
- [FGetFleetRoleCredentialsResult](#)
- [FGameLiftError](#)
- [枚举](#)

FProcess参数

此数据类型包含一组在 [ProcessReady\(\)](#) 中发送到 Amazon GameLift Servers 的参数。

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 GameLift Servers 会复制并存储这些文件，以便将来访问。 类型：TArray<FString> 必需：否
OnHealthCheck	Amazon GameLift Servers 为向服务器进程请求运行状况报告而调用的回调函数。Amazon GameLift Servers 每隔 60 秒调用此函数一次，并等待 60 秒以获取响应。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。如果没有返回任何响应，Amazon GameLift Servers 会将服务器进程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 GameLift Servers 为强制关闭服务器进程而调用的回调函数。调用此函数之后，Amazon GameLift Servers 会等待 5 分钟，以确保服务器进程关闭并通过 [ProcessEnding\(\)](#) 调用作出响应，随后才会终止该服务器进程。

类型：FSimpleDelegate

是否必需：是

OnStartGameSession

调Amazon GameLift Servers用以激活新游戏会话的回调函数。Amazon GameLift Servers应客户端请求调用此函数[CreateGameSession](#)。回调函数将传递一个 [GameSession](#) 对象。

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

是否必需：是

OnUpdateGameSession	Amazon GameLift Servers 为向服务器进程传递更新的游戏会话对象而调用的回调函数。当对战回填请求处理完毕时，Amazon GameLift Servers 会调用此函数以提供更新的对战构建器数据。它将传递一个 GameSession 对象、一个状态更新 (updateReason) 以及对战回填票证 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会使用该端口号。 类型：int 是否必需：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) 。

Properties	描述
GameSession	一个 GameSession 对象。该GameSession 对象包含描述游戏会话的属性。

Properties	描述
	类型 : <code>Aws::GameLift::Server::GameSession</code> 必需 : 否
UpdateReason	更新游戏会话的原因。 类型 : <code>enum class UpdateReason</code> • 对战数据已更新 • 回填失败 • <code>BACKFILL_TIMED_OUT</code> • 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 : <code>char[]</code> 必需 : 否

GameSession

此数据类型提供游戏会话的详细信息。

Properties	描述
GameSessionId	会话的唯一标识符。游戏会话ARN具有以下格式: <code>arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token></code> 类型 : <code>char[]</code> 必需 : 否

Properties	描述
Name	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 Servers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 Servers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

Properties	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>.amazon gamelift.com • Non-TLS-enabled 舰队：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

FServer参数

用于维护 Amazon GameLift Servers Anywhere 服务器与 Amazon GameLift Servers 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 Servers 托管式 EC2 实例上的服务器，请使用空对象。

Properties	描述
webSocketUrl	当您为 Amazon GameLift Servers Anywhere 计算资源调用 RegisterCompute 时，GameLiftServerSdkEndpoint Amazon GameLift Servers 会返回结果。 类型：char[] 是否必需：是
ProcessID	注册到托管游戏的服务器进程的唯一标识符。 类型：char[] 是否必需：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[] 是否必需：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[] 是否必需：是
AuthToken	由 Amazon GameLift Servers 生成的身份验证令牌，用于向 Amazon GameLift Servers 对您

Properties	描述
	的服务器进行身份验证。有关更多信息，请参阅 GetComputeAuthToken 。
	类型：char[]
	是否必需：是

FStartMatchBackfillRequest

用于创建对战回填请求的信息。游戏服务器通过调用 [StartMatchBackfill\(\)](#) 将这些信息传达给 Amazon GameLift Servers。

Properties	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[] 是否必需：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[] 是否必需：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>

Properties	描述
	是否必需：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 Servers 会生成值。使用此标识符可跟踪对战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。启动对战请求时，玩家会有玩家 ID、属性数据以及可能的延迟数据。对战结束后，Amazon GameLift Servers 会添加团队信息。

Properties	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<

Properties	描述
	必需：否
PlayerId	玩家的唯一标识符。 类型：std::string 必需：否
Team	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

FGameLiftDescribePlayerSessionsRequest

一个对象，用于指定要检索哪些玩家会话。服务器进程通过向 Amazon GameLift Servers 调用 [DescribePlayerSessions\(\)](#) 来提供此信息。

Properties	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

Properties	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

FStopMatchBackfillRequest

用于取消对战回填请求的信息。游戏服务器通过调用 [StopMatchBackfill\(\)](#) 将此信息传达给 Amazon GameLift Servers 服务。

Properties	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 是否必需：是
MatchmakingConfigurationArn	此请求发送到的对战构建器的唯一标识符。 类型：FString 是否必需：是
TicketId	要取消的回填请求票证的唯一标识符。 类型：FString 是否必需：是

FAttribute价值

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。

Properties	说明
属性类型	指定属性值的类型。 类型：一个 FAttributeType 枚举 值。 必需：否
S	表示字符串属性值。 类型：FString

Properties	说明
	必需：否
N	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

FGameLiftGetFleetRoleCredentialsRequest

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 Amazon GameLift Servers 设置 IAM 服务角色](#)。

Properties	描述
RoleArn	服务角色的 Amazon 资源名称 (ARN)，用于扩展对您的 AWS 资源的有限访问权限。 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

FGameLiftLongOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

Properties	说明
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 是否必需：是
错误	操作失败时发生的错误。 类型： the section called “FGameLiftError” 必需：否

FGameLiftStringOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

Properties	说明
结果	操作的结果。 类型：FString

Properties	说明
	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 是否必需：是
错误	操作失败时发生的错误。 类型：the section called “FGameLiftError” 必需：否

FGameLiftDescribePlayerSessionsOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

Properties	说明
结果	操作的结果。 类型：the section called “FGameLiftDescribePlayerSessionsResult” 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&&

Properties	说明
	必需：否
成功	操作是否成功。 类型：bool 是否必需：是
错误	操作失败时发生的错误。 类型： the section called “FGameLiftError” 必需：否

FGameLiftDescribePlayerSessionsResult

Properties	说明
PlayerSessions	类型：TArray<FGameLiftPlayerSession> 是否必需：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 是否必需：是
错误	操作失败时发生的错误。 类型： the section called “FGameLiftError”

Properties	说明
	必需：否

FGeneric结果

此数据类型由操作生成，并生成具有以下属性的对象：

Properties	说明
成功	操作是否成功。 类型：bool 是否必需：是
错误	操作失败时发生的错误。 类型： the section called “FGameLiftError” 必需：否

FGameLiftPlayerSession

Properties	说明
CreationTime	类型：long 是否必需：是
FleetId	类型：FString 是否必需：是
GameSessionId	类型：FString 是否必需：是
IpAddress	类型：FString

Properties	说明
	是否必需：是
PlayerData	类型：FString 是否必需：是
PlayerId	类型：FString 是否必需：是
PlayerSessionId	类型：FString 是否必需：是
端口：	类型：int 是否必需：是
Status	类型： PlayerSessionStatus 枚举 。 是否必需：是
TerminationTime	类型：long 是否必需：是
DnsName	类型：FString 是否必需：是

FGameLiftGetComputeCertificateOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

Properties	说明
结果	操作的结果。

Properties	说明
	类型： the section called “FGameLiftGetComputeCertificateResult” 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 是否必需：是
错误	操作失败时发生的错误。 类型： the section called “FGameLiftError” 必需：否

FGameLiftGetComputeCertificateResult

计算机上 TLS 证书的路径和计算机的主机名。

Properties	说明
CertificatePath	类型：FString 是否必需：是
ComputeName	类型：FString 是否必需：是

FGameLiftGetFleetRoleCredentialsOutcome

此数据类型由操作生成，并生成具有以下属性的对象：

Properties	说明
结果	操作的结果。 类型：the section called “FGetFleetRoleCredentialsResult” 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 是否必需：是
错误	操作失败时发生的错误。 类型：the section called “FGameLiftError” 必需：否

FGetFleetRoleCredentialsResult

Properties	说明
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。 类型：FString

Properties	说明
	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 是否必需：是

Properties	说明
错误	操作失败时发生的错误。 类型： the section called “GameLiftError” 必需：否

FGameLiftError

Properties	说明
ErrorType	错误的类型。 类型： GameLiftErrorType 枚举 。 必需：否
ErrorMessage	错误类型的名称。 类型： <code>std::string</code> 必需：否
ErrorName	错误消息。 类型： <code>std::string</code> 必需：否

枚举

为适用于 Amazon GameLift Servers (Unreal) 的服务器 SDK 定义了以下枚举类型：

FAttribute类型

- 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 Amazon GameLift Servers 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
- ALREADY_INITIALIZED – Amazon GameLift Servers 服务器或客户端已使用 Initialize() 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 Servers 客户端尚未初始化。
- GAMELIFT_SERVER_NOT_INITIALIZED – Amazon GameLift Servers 服务器尚未初始化。
- GAME_SESSION_ENDED_FAILED – 适用于 Amazon GameLift Servers 的服务器 SDK 无法联系服务部门报告游戏会话已结束。
- GAME_SESSION_NOT_READY – Amazon GameLift Servers 服务器游戏会话未激活。
- GAME_SESSION_READY_FAILED – 适用于 Amazon GameLift Servers 的服务器 SDK 无法联系服务部门报告游戏会话已准备就绪。
- INITIALIZATION_MISMATCH – 在 Server::Initialization() 之后调用了客户端方法，反之亦然。
- NOT_INITIALIZED – Amazon GameLift Servers 服务器或客户端尚未使用 Initialize() 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 – 适用于 Amazon GameLift Servers 的服务器 SDK 无法联系服务部门报告流程即将结束。
- PROCESS_NOT_ACTIVE — 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_READY – 服务器进程尚未准备好激活。

- PROCESS_READY_FAILED – 适用于 Amazon GameLift Servers 的服务器 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 服务器后端组件初始化失败。
- 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。

EPlayerSessionCreationPolicy

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。

EPlayerSessionStatus

- ACTIVE (处于活动状态)
- COMPLETED
- NOT_SET
- 预留
- 超时

[适用于 Amazon GameLift Servers 的 C++ \(Unreal \) 服务器 SDK 5.x – 数据类型](#)

主题

- [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

```
FGameLiftStringOutcome GetSdkVersion();
```

返回值

如果成功，返回以 [the section called “FGameLiftStringOutcome”](#) 对象返回当前 SDK 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()

为托管式 EC2 实例集初始化 Amazon GameLift Servers SDK。在与 Amazon GameLift Servers 相关的任何其他初始化发生之前，在启动时调用此方法。此方法从主机环境读取服务器参数，以设置服务器与 Amazon GameLift Servers 服务之间的通信。它使用幂等性令牌，因此当调用失败时，您可以放心地重试此调用。

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 Amazon GameLift Servers Agent  
to enable further communication.  
FGameLiftGenericOutcome initSdkOutcome = GameLiftSdkModule->InitSDK();
```

InitSDK()

为 Anywhere 实例集或托管式容器实例集初始化 Amazon GameLift Servers SDK。在与 Amazon GameLift Servers 相关的任何其他初始化发生之前，在启动时调用此方法。此方法需要明确的服务器参数来设置服务器和 Amazon GameLift Servers 服务之间的通信。它使用幂等性令牌，因此当调用失败时，您可以放心地重试此调用。

语法

```
FGameLiftGenericOutcome InitSDK(serverParameters)
```

参数

[FServer](#)参数

要在 Amazon GameLift Servers Anywhere 实例集上初始化游戏服务器，请使用以下信息构造 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 ID。
- 包含您的 Amazon GameLift Servers Anywhere 计算的 Amazon GameLift Servers 实例集的 ID。
- Amazon GameLift Servers 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 Amazon GameLift Servers Agent
to enable further communication.
FGameLiftGenericOutcome initSdkOutcome = GameLiftSdkModule->InitSDK(serverParameters);
```

ProcessReady()

通知 Amazon GameLift Servers，该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

[FProcess参数](#) 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 Servers 服务调用其与服务器进程通信。
- 服务器进程正在侦听的端口号。
- 您希望 Amazon GameLift Servers 捕获和存储的任何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 Amazon GameLift Servers this game server is ready to
receive incoming game sessions!
UE_LOG(GameServerLog, Log, TEXT("Calling Process Ready"));
FGameLiftGenericOutcome processReadyOutcome = GameLiftSdkModule-
>ProcessReady(*params);
```

ProcessEnding()

通知 Amazon GameLift Servers，该服务器进程即将终止。请在已完成所有其他清理任务（包括关闭活动游戏会话）但尚未终止进程时调用此方法。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，则生成的实例集事件为 `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. Amazon GameLift Servers will invoke this callback before
//shutting down an instance hosting this game server.
//It gives this game server a chance to save its state, communicate with services,
//etc., before being shut down.
//In this case, we simply tell Amazon GameLift Servers we are indeed going to shutdown.
params->OnTerminate.BindLambda(=[]() {
    UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
    GameLiftSdkModule->ProcessEnding();
});
```

ActivateGameSession()

通知 Amazon GameLift Servers，该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()
```

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, Amazon GameLift Servers sends an activation request
//to the game server and passes along the game session object containing game properties
//and other settings.
//Here is where a game server should take action based on the game session object.
//Once the game server is ready to receive incoming player connections, it should
//invoke GameLiftServerAPI.ActivateGameSession()
auto onGameSession = [=](Aws::GameLift::Server::Model::GameSession gameSession)
{
    FString gameSessionId = FString(gameSession.GetGameSessionId());
    UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"), *gameSessionId);
    GameLiftSdkModule->ActivateGameSession();
};
```

UpdatePlayerSessionCreationPolicy()

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_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 “FGameLiftError”](#)。

语法

```
FGameLiftStringOutcome GetGameSessionId()
```

参数

此操作没有参数。

返回值

如果成功，以 [the section called “FGameLiftStringOutcome”](#) 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, Amazon GameLift Servers sends an activation request
//to the game server and passes along the game session object containing game properties
//and other settings.
//Here is where a game server should take action based on the game session object.
//Once the game server is ready to receive incoming player connections, it should
//invoke GameLiftServerAPI.ActivateGameSession()
auto onGameSession = [=](Aws::GameLift::Server::Model::GameSession gameSession)
{
    FString gameSessionId = FString(gameSession.GetGameSessionId());
    UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"), *gameSessionId);
    GameLiftSdkModule->ActivateGameSession();
};
```

GetTerminationTime()

如果终止时间可用，则返回安排服务器进程关闭的时间。服务器进程在收到来自 Amazon GameLift Servers 的 `onProcessTerminate()` 回调后执行此操作。Amazon GameLift Servers 会出于以下原因调用 `onProcessTerminate()`：

- 服务器进程报告运行状况不佳或未响应 Amazon GameLift Servers。
- 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 -0000Z 于 6373559680000000000 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 Servers，具有指定玩家会话 ID 的玩家已连接到服务器进程并且需要进行验证。Amazon GameLift Servers 将验证该玩家会话 ID 是否有效。玩家会话经过验证后，Amazon GameLift Servers 会将玩家位置的状态从 `RESERVED` 更改为 `ACTIVE`。

语法

```
FGameLiftGenericOutcome AcceptPlayerSession(const FString& playerId)
```

参数

playerSessionId

创建新玩家会话时由 Amazon GameLift Servers 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。IDs

```
bool GameLiftManager::AcceptPlayerSession(const FString& playerId, 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 Servers，有玩家已断开与服务器进程的连接。作为回应，Amazon GameLift Servers 会将玩家位置更改为可用。

语法

```
FGameLiftGenericOutcome RemovePlayerSession(const FString& playerSessionId)
```

参数

playerSessionId

创建新玩家会话时由 Amazon GameLift Servers 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& playerId)
{
    #if WITH_GAMELIFT
        UE_LOG(GameServerLog, Log, TEXT("Removing GameLift PlayerSession: %s"),
            *playerId);

        if (!GameLiftSdkModule->RemovePlayerSession(playerId).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(playerId);
        idToPlayerSessionMap.Remove(playerId);

        // 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

[FGameLiftDescribePlayerSessionsRequest](#)

[the section called “FGameLiftDescribePlayerSessionsRequest”](#) 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 “FGameLiftDescribePlayerSessionsOutcome”](#) 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。省略NextToken并将Limit值设置为10，即可Amazon GameLift Servers返回与请求匹配的前10个玩家会话记录。

```
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。如果成功匹配了新玩家，则 Amazon GameLift Servers 会使用回调函数 `OnUpdateGameSession()` 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对战回填请求。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](#)

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；如果未提供任何 ID，则 Amazon GameLift Servers 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。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()

取消活动的对战回填请求。有关更多信息，请参阅 [FlexMatch 回填功能](#)。

语法

```
FGameLiftGenericOutcome StopMatchBackfill (FStopMatchBackfillRequest  
    &stopBackfillRequest);
```

参数

[FStopMatchBackfillRequest](#)

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 Amazon GameLift Servers Anywhere 计算资源和 Amazon GameLift Servers 之间的网络连接。当您将计算设备注册到 Amazon GameLift Servers Anywhere 实例集时，您可以使用该证书路径。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 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_T0_TCHAR(outres.GetCertificatePath());
        result.m_computeName = UTF8_T0_TCHAR(outres.GetComputeName());
        return FGameLiftGetComputeCertificateOutcome(result);
    }
    else {
        return FGameLiftGetComputeCertificateOutcome(FGameLiftError(outcome.GetError()));
    }
}

```

```
#else
return FGameLiftGetComputeCertificateOutcome(FGameLiftGetComputeCertificateResult());
#endif
}
```

GetFleetRoleCredentials()

检索授权 Amazon GameLift Servers 与其他 AWS 服务互动的 IAM 角色凭证。有关更多信息，请参阅[将您的 Amazon GameLift Servers 托管游戏服务器连接到其他 AWS 资源](#)。

语法

```
FGameLiftGetFleetRoleCredentialsOutcome
FGameLiftServerSDKModule::GetFleetRoleCredentials(const
FGameLiftGetFleetRoleCredentialsRequest &request)
```

参数

[FGameLiftGetFleetRoleCredentialsRequest](#)

返回值

返回 [the section called “FGameLiftGetFleetRoleCredentialsOutcome”](#) 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_T0_TCHAR(outres.GetAccessKeyId());
result.m_secretAccessKey = UTF8_T0_TCHAR(outres.GetSecretAccessKey());
result.m_sessionToken = UTF8_T0_TCHAR(outres.GetSessionToken());
result.m_expiration = FDateTime::FromUnixTimestamp(outres.GetExpiration());
return FGameLiftGetFleetRoleCredentialsOutcome(result);
}
else {
    return FGameLiftGetFleetRoleCredentialsOutcome(FGameLiftError(outcome.GetError()));
}
#else
return
FGameLiftGetFleetRoleCredentialsOutcome(FGameLiftGetFleetRoleCredentialsResult());
#endif
}
```

Destroy

从内存中释放 Amazon GameLift Servers 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 实例集，并且没有在每次游戏会话后终止服务器进程，请先调用 `Destroy()`，再通过 `InitSDK()` 重新初始化，最后使用 `ProcessReady()` 通知 Amazon GameLift Servers 该进程已准备好托管游戏会话。

语法

```
FGameLiftGenericOutcome 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

```
// First call ProcessEnding()
FGameLiftGenericOutcome processEndingOutcome = GameLiftSdkModule->ProcessEnding();

// Then call Destroy() to free the SDK from memory
FGameLiftGenericOutcome destroyOutcome = GameLiftSdkModule->Destroy();
```

```
// Exit the process with success or failure
if (processEndingOutcome.IsSuccess() && destroyOutcome.IsSuccess())
{
    UE_LOG(GameServerLog, Log, TEXT("Server process ending successfully"));
}
else {
    if (!processEndingOutcome.IsSuccess()) {
        const FGameLiftError& error = processEndingOutcome.GetError();
        UE_LOG(GameServerLog, Error, TEXT("ProcessEnding() failed. Error: %s"),
            error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
            *error.m_errorMessage);
    }
    if (!destroyOutcome.IsSuccess()) {
        const FGameLiftError& error = destroyOutcome.GetError();
        UE_LOG(GameServerLog, Error, TEXT("Destroy() failed. Error: %s"),
            error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
            *error.m_errorMessage);
    }
}
```

Amazon GameLift Servers 服务器 SDK 版本 4 及更早版本

本参考文献记录了 Amazon GameLift Servers 服务器 SDK 版本 4.x 及更早版本。服务器 SDK 提供游戏服务器用于与 Amazon GameLift Servers 服务通信的核心功能。例如，游戏服务器会接收该服务发送的提示来启动和停止游戏会话，并定期向该服务提供游戏会话状态更新。在部署游戏服务器进行托管之前，请将其与服务器 SDK 集成。

使用此服务器 SDK 参考将要托管的自定义多人游戏服务器与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

对于 SDK 版本 4.0.2，你可以从[官方 GitHub 版本](#)中下载。包含 GameLiftLocal.jar 的完整 SDK 包可在 GameLift-CSharp-ServerSDK -4.0.2.zip 工件中找到。

Amazon GameLift Servers 服务器 SDK 的最新主要版本是 5.x。以下托管功能需要更新到版本 5.x：

- Amazon GameLift Servers Anywhere
- Amazon GameLift Servers 托管式容器
- 适用于 Unreal Engine 和 Unity 的 Amazon GameLift Servers 插件

主题

- [适用于 Amazon GameLift Servers 4.x 的 C++ 服务器 SDK – 操作](#)
- [适用于 Amazon GameLift Servers 4.x 的 C# 服务器 SDK – 操作](#)
- [适用于 Amazon GameLift Servers 的服务器 SDK \(Unreal \) – 操作](#)

适用于 Amazon GameLift Servers 4.x 的 C++ 服务器 SDK – 操作

使用服务器 SDK 参考，将要托管的多人游戏与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

Note

本参考适用于 Amazon GameLift Servers 服务器 SDK 的早期版本。有关最新版本，请参阅[适用于 Amazon GameLift Servers 的 C++ 服务器 SDK 5.x – 操作](#)。

适用于 Amazon GameLift Servers 4.x 的 C++ 服务器 SDK – 数据类型

使用服务器 SDK 参考，将要托管的多人游戏与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

Note

本参考适用于 Amazon GameLift Servers 服务器 SDK 的早期版本。有关最新版本，请参阅[适用于 Amazon GameLift Servers 的 C++ 服务器 SDK 5.x – 数据类型](#)。

此 API 在 `GameLiftServerAPI.h`、`LogParameters.h` 和 `ProcessParameters.h` 中定义。

[适用于 Amazon GameLift Servers 4.x 的 C++ 服务器 SDK – 操作](#)

`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 a `PlayerSessionId` 以请求特定的玩家会话。
- 提供 a `GameSessionId` 以请求指定游戏会话中的所有玩家会话。
- 提供 a `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。

内容

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。将此参数与一起NextToken使用可将结果作为一组连续页面获取。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 IDs 由开发者定义。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 and/or 已通过验证。
- 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 — 已收到玩家会话请求，但未在超时限制（60 秒）内验证未连接 and/or 的玩家。

类型：字符串

必需：否

LogParameters

此数据类型用于标识您希望 Amazon GameLift Servers 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 [ProcessReady\(\)](#) 调用中传递给 Amazon GameLift Servers 服务。

内容

logPaths

您希望 Amazon GameLift Servers 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 [ProcessReady\(\)](#) 调用中发送到 Amazon GameLift Servers 服务的参数。

内容

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会使用该端口号。

类型：整数

是否必需：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:: 服务器:: [LogParameters](#)

必需：否

onStartGameSession

Amazon GameLift Servers 服务为激活新游戏会话而调用的回调函数的名称。Amazon GameLift Servers 调用此函数是为了响应客户端的请求 [CreateGameSession](#)。回调函数传递一个 [GameSession](#) 对象（在 Amazon GameLift Servers 服务 API 参考中定义）。

类型：const std::function<void(Aws::GameLift::Model::GameSession)>
onStartGameSession

是否必需：是

onProcessTerminate

Amazon GameLift Servers 服务为强制关闭服务器进程而调用的回调函数的名称。调用此函数之后，Amazon GameLift Servers 等待五分钟以便服务器进程关闭，然后使用 [ProcessEnding\(\)](#) 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

Amazon GameLift Servers 服务为向服务器进程请求运行状况报告而调用的回调函数的名称。Amazon GameLift Servers 每隔 60 秒调用此函数一次。调用此函数后，Amazon GameLift Servers 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 Servers 服务为向服务器进程传递更新的游戏会话对象而调用的回调函数的名称。当[对战回填](#)请求处理完毕时，Amazon GameLift Servers 会调用此函数以提供更新的对战构建器数据。它将传递一个 [GameSession](#) 对象、一个状态更新（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 [StartMatchBackfill\(\)](#) 调用中传递给 Amazon GameLift Servers 服务。

内容

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

是否必需：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

是否必需：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 Servers API 参考指南》。要查找玩家属性和团队分配，请在游戏会话对象的匹配器数据属性中查找。IDs 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](#)>

是否必需：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 Servers 将生成一个采用 UUID 形式的值。使用此标识符可跟踪对战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 [StopMatchBackfill\(\)](#) 调用中传递给 Amazon GameLift Servers 服务。

内容

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

是否必需：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

是否必需：是

TicketId

要取消的回填请求票证的唯一标识符。

类型：字符串

是否必需：是

[适用于 Amazon GameLift Servers 4.x 的 C++ 服务器 SDK – 数据类型](#)

主题

- [AcceptPlayerSession\(\)](#)
- [ActivateGameSession\(\)](#)
- [DescribePlayerSessions\(\)](#)
- [GetGameSessionId\(\)](#)
- [GetInstanceCertificate\(\)](#)
- [GetSdkVersion\(\)](#)
- [GetTerminationTime\(\)](#)
- [InitSDK\(\)](#)
- [ProcessEnding\(\)](#)
- [ProcessReady\(\)](#)
- [ProcessReadyAsync\(\)](#)
- [RemovePlayerSession\(\)](#)
- [StartMatchBackfill\(\)](#)
- [StopMatchBackfill\(\)](#)
- [TerminateGameSession\(\)](#)
- [UpdatePlayerSessionCreationPolicy\(\)](#)
- [Destroy](#)

AcceptPlayerSession()

通知 Amazon GameLift Servers 服务，具有指定玩家会话 ID 的玩家已连接到服务器进程并且需要验证。Amazon GameLift Servers 将验证该玩家会话 ID 是否有效，即该玩家 ID 是否已在游戏会话

中预留玩家位置。通过验证后，Amazon GameLift Servers 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 Servers 服务为响应 AWS SDK Amazon GameLift Servers API 操作调用而颁发的唯一 ID [CreatePlayerSession](#)。连接到服务器进程时，游戏客户端会引用此 ID。

类型：std::string

是否必需：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

此示例说明了处理连接请求的函数，包括验证和拒绝无效的玩家会话。IDs

```
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 Amazon GameLift Servers 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

`describePlayerSessions` 请求

[DescribePlayerSessionsRequest](#) 对象描述要检索的玩家会话。

是否必需：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。玩家会话对象的结构与 AWS SDK Amazon GameLift Servers API [PlayerSession](#) 数据类型相同。

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` 并将 `Limit` 值设置为 10，Amazon GameLift Servers 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 [GetCertificate 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 SDK 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。服务器进程在收到来自 Amazon GameLift Servers 服务的 `onProcessTerminate()` 回调后执行此操作。Amazon GameLift Servers 可能出于以下原因调用 `onProcessTerminate()`：(1) 服务器进程报告运行状况不佳或未响应 Amazon GameLift Servers，(2) 在缩减事件期间终止实例，或 (3) 因[竞价型实例中断](#)而导致实例终止。

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 6373559680000000000 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 Servers 开发工具包。应在启动之后、进行任何其他 Amazon GameLift Servers 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，则返回一个 InitSdkOutcome 对象，表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 Amazon GameLift Servers 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 Amazon GameLift Servers 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：

- 游戏服务器代码中实现的回调方法的名称，Amazon GameLift Servers 服务调用其与服务器进程通信。
- 服务器进程正在侦听的端口号。
- 您希望 Amazon GameLift Servers 捕获和存储的任何游戏会话特定文件的路径。

是否必需：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 Amazon GameLift Servers 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 Amazon GameLift Servers 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：

- 游戏服务器代码中实现的回调方法的名称，Amazon GameLift Servers 服务调用其与服务器进程通信。
- 服务器进程正在侦听的端口号。
- 您希望 Amazon GameLift Servers 捕获和存储的任何游戏会话特定文件的路径。

是否必需：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 Amazon GameLift Servers 服务，具有指定玩家会话 ID 的玩家已从服务器进程断开连接。作为响应，Amazon GameLift Servers 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& playerId);
```

参数

playerSessionId

Amazon GameLift Servers 服务为响应 AWS SDK Amazon GameLift Servers API 操作调用而颁发的唯一 ID [CreatePlayerSession](#)。连接到服务器进程时，游戏客户端会引用此 ID。

类型：std::string

是否必需：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 FlexMatch 创建的游戏会话中的开放位置查找新玩家。另请参阅 [AWS SDK 操作 StartMatchBackfill\(\)](#)。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。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 Amazon GameLift Servers 服务会通过调用回调函数 `OnUpdateGameSession()` 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对战回填请求。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 Amazon GameLift Servers 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。

是否必需：是

返回值

返回带有匹配回填票证或失败并显示错误消息的 `StartMatchBackfillOutcome` 对象。可以使用 AWS SDK 操作 [DescribeMatchmaking\(\)](#) 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 [S AWS DK 操作 StopMatchmaking\(\)](#)。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

是否必需：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()

此方法在版本 4.0.1 中已弃用。相反，服务器进程应在游戏会话结束[ProcessEnding\(\)](#)后调用。

通知 Amazon GameLift Servers 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此操作，因为它会向 Amazon GameLift Servers 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 S AWS DK 操作 [UpdateGameSession\(\)](#)。

语法

```
GenericOutcome UpdatePlayerSessionCreationPolicy(  
    Aws::GameLift::Model::PlayerSessionCreationPolicy newPlayerSessionPolicy);
```

参数

newPlayerSession政策

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 enum。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。

是否必需：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);  
}
```

适用于 Amazon GameLift Servers 4.x 的 C# 服务器 SDK – 操作

使用服务器 SDK 参考，将要托管的多人游戏与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

Note

本参考适用于 Amazon GameLift Servers 服务器 SDK 的早期版本。有关最新版本，请参阅[适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 操作](#)。

适用于 Amazon GameLift Servers 4.x 的 C# 服务器 SDK – 数据类型

使用服务器 SDK 参考，将要托管的多人游戏与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

Note

本参考适用于 Amazon GameLift Servers 服务器 SDK 的早期版本。有关最新版本，请参阅[适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 数据类型](#)。

[适用于 Amazon GameLift Servers 4.x 的 C# 服务器 SDK – 操作](#)

主题

- [LogParameters](#)
- [DescribePlayerSessionsRequest](#)
- [ProcessParameters](#)
- [StopMatchBackfillRequest](#)

LogParameters

此数据类型用于标识您希望 Amazon GameLift Servers 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 [ProcessReady\(\)](#) 调用中传递给 Amazon GameLift Servers 服务。

内容

logPaths

您希望 Amazon GameLift Servers 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 a `PlayerSessionId` 以请求特定的玩家会话；(2) 提供 a `GameSessionId` 以请求指定游戏会话中的所有玩家会话；或 (3) 提供 a `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。

内容

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。将此参数与一起 `NextToken` 使用可将结果作为一组连续页面获取。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 IDs 由开发者定义。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 and/or 已通过验证。
- 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 — 已收到玩家会话请求，但未在超时限制（60 秒）内验证未连接 and/or 的玩家。

类型：字符串

必需：否

ProcessParameters

此数据类型包含一组在 [ProcessReady\(\)](#) 调用中发送到 Amazon GameLift Servers 服务的参数。

内容

端口

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会使用该端口号。

类型：整数

是否必需：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](#)

是否必需：是

onStartGameSession

Amazon GameLift Servers服务为激活新游戏会话而调用的回调函数的名称。Amazon GameLift Servers调用此函数是为了响应客户端的请求[CreateGameSession](#)。回调函数采用一个[GameSession](#)对象（在Amazon GameLift Servers服务 API 参考中定义）。

类型：void OnStartGameSessionDelegate(GameSession gameSession)

是否必需：是

onProcessTerminate

Amazon GameLift Servers 服务为强制关闭服务器进程而调用的回调函数的名称。调用此函数之后，Amazon GameLift Servers 等待五分钟以便服务器进程关闭，然后使用 [ProcessEnding\(\)](#) 调用响应，再关闭服务器进程。

类型：void OnProcessTerminateDelegate()

是否必需：是

onHealthCheck

Amazon GameLift Servers 服务为向服务器进程请求运行状况报告而调用的回调函数的名称。Amazon GameLift Servers 每隔 60 秒调用此函数一次。调用此函数后，Amazon GameLift Servers 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()

是否必需：是

onUpdateGameSession

Amazon GameLift Servers 服务为向服务器进程传递更新的游戏会话对象而调用的回调函数的名称。当[对战回填](#)请求处理完毕时，Amazon GameLift Servers 会调用此函数以提供更新的对战构建器数据。它将传递一个 [GameSession](#) 对象、一个状态更新 (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 [StartMatchBackfill\(\)](#) 调用中传递给 Amazon GameLift Servers 服务。

内容

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

是否必需：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

是否必需：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 Servers API 参考指南》。要查

找玩家属性和团队分配，请在游戏会话对象的匹配器数据属性中查找。IDs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\[\]](#)

是否必需：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 Servers 将生成一个采用 UUID 形式的值。使用此标识符可跟踪对战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 [StopMatchBackfill\(\)](#) 调用中传递给 Amazon GameLift Servers 服务。

内容

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

是否必需：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

是否必需：是

TicketId

要取消的回填请求票证的唯一标识符。

类型：字符串

是否必需：是

[适用于 Amazon GameLift Servers 4.x 的 C# 服务器 SDK – 数据类型](#)

主题

- [AcceptPlayerSession\(\)](#)
- [ActivateGameSession\(\)](#)
- [DescribePlayerSessions\(\)](#)
- [GetGameSessionId\(\)](#)
- [GetInstanceCertificate\(\)](#)
- [GetSdkVersion\(\)](#)
- [GetTerminationTime\(\)](#)
- [InitSDK\(\)](#)
- [ProcessEnding\(\)](#)
- [ProcessReady\(\)](#)
- [RemovePlayerSession\(\)](#)
- [StartMatchBackfill\(\)](#)
- [StopMatchBackfill\(\)](#)
- [TerminateGameSession\(\)](#)
- [UpdatePlayerSessionCreationPolicy\(\)](#)

AcceptPlayerSession()

通知 Amazon GameLift Servers 服务，具有指定玩家会话 ID 的玩家已连接到服务器进程并且需要验证。Amazon GameLift Servers 将验证该玩家会话 ID 是否有效，即该玩家 ID 是否已在游戏会话中预留玩家位置。通过验证后，Amazon GameLift Servers 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 Servers 颁发的唯一 ID。玩家会话 ID 是在对象中指定的，该 `PlayerSession` 对象是为了响应客户端对 GameLift API 操作的调用而返回的 [StartGameSessionPlacement CreateGameSession](#)、[DescribeGameSessionPlacement](#)、或 [DescribePlayerSessions](#)。

类型：字符串

是否必需：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

此示例说明了处理连接请求的函数，包括验证和拒绝无效的玩家会话。IDs

```
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 Amazon GameLift Servers 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

`describePlayerSessions` 请求

[DescribePlayerSessionsRequest](#) 对象描述要检索的玩家会话。

是否必需：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。玩家会话对象的结构与 AWS SDK Amazon GameLift Servers API [PlayerSession](#) 数据类型相同。

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。省略NextToken并将Limit值设置为10，Amazon GameLift Servers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。如果失败，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 [GetCertificate 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 SDK 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。服务器进程在收到来自 Amazon GameLift Servers 服务的 `onProcessTerminate()` 回调后执行此操作。Amazon GameLift Servers 可能出于以下原因调用 `onProcessTerminate()`：(1) 运行状况不佳 (服务器进程报告端口运行状况问题或未响应 Amazon GameLift Servers)，(2) 在缩减事件期间终止实例，或 (3) 因[竞价型实例中断](#)而导致实例终止。

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 Servers 开发工具包。应在启动之后、进行任何其他 Amazon GameLift Servers 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，则返回一个 InitSdkOutcome 对象，表示服务器进程已准备好调用[ProcessReady\(\)](#)。

示例

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

ProcessEnding()

通知 Amazon GameLift Servers 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 Amazon GameLift Servers 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：

- 游戏服务器代码中实现的回调方法的名称，Amazon GameLift Servers 服务调用其与服务器进程通信。
- 服务器进程正在侦听的端口号。
- 您希望 Amazon GameLift Servers 捕获和存储的任何游戏会话特定文件的路径。

是否必需：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 Amazon GameLift Servers 服务，具有指定玩家会话 ID 的玩家已从服务器进程断开连接。作为响应，Amazon GameLift Servers 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 playerId)
```

参数

playerSessionId

创建新玩家会话时由 Amazon GameLift Servers 颁发的唯一 ID。玩家会话 ID 是在对象中指定的，该 `PlayerSession` 对象是为了响应客户端对 GameLift API 操作的调用而返回的 [StartGameSessionPlacement CreateGameSession](#)、[DescribeGameSessionPlacement](#)、或 [DescribePlayerSessions](#)。

类型：字符串

是否必需：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 `FlexMatch` 创建的游戏会话中的开放位置查找新玩家。另请参阅 `S AWS DK` 操作 [StartMatchBackfill\(\)](#)。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。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 Amazon GameLift Servers 服务会使用回调函数 `OnUpdateGameSession()` 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对战回填请求。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 Amazon GameLift Servers 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。

是否必需：是

返回值

返回带有匹配回填票证 ID 的 StartMatchBackfillOutcome 对象，或者返回失败并显示错误消息的对象。

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 [S AWS DK 操作 StopMatchmaking\(\)](#)。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

是否必需：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()

此方法在版本 4.0.1 中已弃用。相反，服务器进程应在游戏会话结束[ProcessEnding\(\)](#)后调用。

通知 Amazon GameLift Servers 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此操作，因为它会向 Amazon GameLift Servers 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《Amazon GameLift Servers服务 API 参考》中的 [UpdateGameSession\(\)](#) 操作）。

语法

```
GenericOutcome UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy
playerSessionPolicy)
```

参数

newPlayerSession政策

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。

是否必需：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);
```

适用于 Amazon GameLift Servers 的服务器 SDK (Unreal) – 操作

使用适用于 Unreal 的服务器 SDK，将要托管的多人游戏与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

Note

本参考适用于 Amazon GameLift Servers 服务器 SDK 的早期版本。有关最新版本，请参阅[适用于 Amazon GameLift Servers 的 C++ \(Unreal \) 服务器 SDK 5.x – 操作](#)。

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 Servers 集成到 Unreal Engine 项目中](#)。

适用于 Amazon GameLift Servers 的服务器 SDK (Unreal) – 数据类型

使用适用于 Unreal 的 Amazon GameLift Servers 服务器 SDK 参考，将要托管的多人游戏与 Amazon GameLift Servers 集成。有关集成过程的指南，请参阅[借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)。

Note

本参考适用于 Amazon GameLift Servers 服务器 SDK 的早期版本。有关最新版本，请参阅[适用于 Amazon GameLift Servers 的 C++ \(Unreal \) 服务器 SDK 5.x – 数据类型](#)。

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 Servers 集成到 Unreal Engine 项目中](#)。

[适用于 Amazon GameLift Servers 的服务器 SDK \(Unreal \) – 操作](#)

主题

- [FDescribePlayerSessionsRequest](#)
- [FProcess参数](#)
- [FStartMatchBackfillRequest](#)
- [FStopMatchBackfillRequest](#)

FDescribePlayerSessionsRequest

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 a `PlayerSessionId` 以请求特定的玩家会话。
- 提供 a `GameSessionId` 以请求指定游戏会话中的所有玩家会话。
- 提供 a `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。

内容

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。将此参数与一起NextToken使用可将结果作为一组连续页面获取。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 IDs 由开发者定义。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 and/or 已通过验证。
- 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 — 已收到玩家会话请求，但未在超时限制（60 秒）内验证未连接 and/or 的玩家。

类型：字符串

必需：否

FProcess参数

此数据类型包含一组在 [ProcessReady\(\)](#) 调用中发送到 Amazon GameLift Servers 服务的参数。

内容

端口

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会使用该端口号。

类型：整数

是否必需：是

logParameters

包含游戏会话日志文件目录路径列表的对象。

键入：TArray< FString >

必需：否

onStartGameSession

Amazon GameLift Servers服务为激活新游戏会话而调用的回调函数的名称。Amazon GameLift Servers调用此函数是为了响应客户端的请求[CreateGameSession](#)。回调函数采用一个[GameSession](#)对象（在Amazon GameLift Servers服务 API 参考中定义）。

类型：FOnStartGameSession

是否必需：是

onProcessTerminate

Amazon GameLift Servers 服务为强制关闭服务器进程而调用的回调函数的名称。调用此函数之后，Amazon GameLift Servers 等待五分钟以便服务器进程关闭，然后使用 [ProcessEnding\(\)](#) 调用响应，再关闭服务器进程。

类型：FSimple代表

必需：否

onHealthCheck

Amazon GameLift Servers 服务为向服务器进程请求运行状况报告而调用的回调函数的名称。Amazon GameLift Servers 每隔 60 秒调用此函数一次。调用此函数后，Amazon GameLift Servers 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 Servers 服务为向服务器进程传递更新的游戏会话对象而调用的回调函数的名称。当[对战回填](#)请求处理完毕时，Amazon GameLift Servers 会调用此函数以提供更新的对战构建器数据。它将传递一个 [GameSession](#) 对象、一个状态更新 (updateReason) 以及对战回填票证 ID。

类型：FOnUpdateGameSession

必需：否

FStartMatchBackfillRequest

此数据类型用于发送对战回填请求。此信息在 [StartMatchBackfill\(\)](#) 调用中传递给 Amazon GameLift Servers 服务。

内容

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

是否必需： 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

是否必需： 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 Servers API 参考指南》。要查找玩家属性和团队分配，请在游戏会话对象的匹配器数据属性中查找。IDs 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。

键入： TArray<[FPlayer](#)>

是否必需： 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 Servers 将生成一个采用 UUID 形式的值。使用此标识符可跟踪对战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 [StopMatchBackfill\(\)](#) 调用中传递给 Amazon GameLift Servers 服务。

内容

GameSessionArn

与被取消的请求关联的唯一游戏会话标识符。

类型： FString

是否必需： 是

MatchmakingConfigurationArn

此请求发送到的对战构建器的唯一标识符。

类型： FString

是否必需： 是

TicketId

要取消的回填请求票证的唯一标识符。

类型： FString

是否必需： 是

适用于 Amazon GameLift Servers 的服务器 SDK (Unreal) – 数据类型

主题

- [AcceptPlayerSession\(\)](#)
- [ActivateGameSession\(\)](#)
- [DescribePlayerSessions\(\)](#)
- [GetGameSessionId\(\)](#)
- [GetInstanceCertificate\(\)](#)
- [GetSdkVersion\(\)](#)
- [InitSDK\(\)](#)
- [ProcessEnding\(\)](#)
- [ProcessReady\(\)](#)
- [RemovePlayerSession\(\)](#)
- [StartMatchBackfill\(\)](#)
- [StopMatchBackfill\(\)](#)
- [TerminateGameSession\(\)](#)
- [UpdatePlayerSessionCreationPolicy\(\)](#)

AcceptPlayerSession()

通知 Amazon GameLift Servers 服务，具有所指定玩家会话 ID 的玩家已连接到服务器进程并且需要验证。Amazon GameLift Servers 将验证该玩家会话 ID 是否有效，即该玩家 ID 是否已在游戏会话中预留玩家位置。通过验证后，Amazon GameLift Servers 将玩家位置的状态从 RESERVED 更改为 ACTIVE。

语法

```
FGameLiftGenericOutcome AcceptPlayerSession(const FString& playerId)
```

参数

playerSessionId

Amazon GameLift Servers 服务为响应 AWS SDK Amazon GameLift Servers API 操作调用而颁发的唯一 ID [CreatePlayerSession](#)。连接到服务器进程时，游戏客户端会引用此 ID。

类型：FString

是否必需：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 Amazon GameLift Servers 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

describePlayerSessions请求

[FDescribePlayerSessionsRequest](#) 对象描述要检索的玩家会话。

是否必需：是

返回值

如果成功，将返回一个 [FDescribePlayerSessionsRequest](#) 对象，包含一组与请求参数相匹配的玩家会话对象。玩家会话对象的结构与 AWS SDK Amazon GameLift Servers API [PlayerSession](#) 数据类型相同。

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 [GetCertificate 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 SDK 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 Servers 开发工具包。应在启动之后、进行任何其他 Amazon GameLift Servers 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 Amazon GameLift Servers 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 Amazon GameLift Servers 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

FProcess参数

[FProcess参数](#) 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 Servers 服务调用其与服务器进程通信。
- 服务器进程正在侦听的端口号。
- 您希望 Amazon GameLift Servers 捕获和存储的任何游戏会话特定文件的路径。

是否必需：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 Amazon GameLift Servers 服务，具有指定玩家会话 ID 的玩家已从服务器进程断开连接。作为响应，Amazon GameLift Servers 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& playerId)
```

参数

playerSessionId

Amazon GameLift Servers 服务为响应 AWS SDK Amazon GameLift Servers API 操作调用而颁发的唯一 ID [CreatePlayerSession](#)。连接到服务器进程时，游戏客户端会引用此 ID。

类型：FString

是否必需：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 FlexMatch 创建的游戏会话中的开放位置查找新玩家。另请参阅 S AWS DK 操作 [StartMatchBackfill\(\)](#)。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。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 Amazon GameLift Servers 服务会使用回调函数 `OnUpdateGameSession()` 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对战回填请求。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 Amazon GameLift Servers 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。

是否必需：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。可以使用 AWS SDK 操作 [DescribeMatchmaking\(\)](#) 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 S AWS DK 操作 [StopMatchmaking\(\)](#)。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

是否必需：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()

此方法在版本 4.0.1 中已弃用。相反，服务器进程应在游戏会话结束 [ProcessEnding\(\)](#) 后调用。

通知 Amazon GameLift Servers 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此操作，因为它会向 Amazon GameLift Servers 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 Amazon GameLift Servers 操作)。

语法

```
FGameLiftGenericOutcome UpdatePlayerSessionCreationPolicy(EPlayerSessionCreationPolicy policy)
```

参数

Policy

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。

是否必需：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。

游戏会话放置事件

Amazon GameLift Servers 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。这些事件还会尽力近乎实时地发送到 Amazon CloudWatch Events Amazon CloudWatch Events。

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 Servers API 参考》中的 [GameSessionPlacement](#)。

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](#)）：

version

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。

账户

用于管理 Amazon GameLift Servers 的 AWS 账户。

时间

事件时间戳

区域

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。

resources

正在处理放置请求的游戏会话队列的 ARN 值。

已完成展示位置

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。

端口

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。

游戏会话区域

托管游戏会话的实例集所在的 AWS 区域。这与 GameSessionArn 中的区域令牌对应。

PlacedPlayer 会话

为游戏会话放置请求中的每位玩家创建的会话集合。

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-1232131232324124123123"
      }
    ]
  }
}
```

放置已取消

[拨打 GameLift 服务 stopGameSessionPlacement 后，放置请求被取消。](#)

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"
  }
}
```

放置超时

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"
  }
}
```

放置失败

Amazon GameLift Servers 无法满足游戏会话请求。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"
  }
}
```

Amazon GameLift Servers AMI 版本

下表列出了用于托管 EC2 托管的最新 Amazon 计算机映像 (AMIs)。Amazon GameLift Servers 如[Amazon GameLift Servers 中的配置和漏洞分析](#)中所述，您必须定期创建新的 Amazon GameLift Servers 托管式 EC2 实例集，以部署最新的 AMI 版本更新。

AMIs 适用于适用于 Amazon GameLift Servers 5 岁以上的服务器 SDK

Amazon GameLift Servers使用以下内容 AMIs 来托管与Amazon GameLift Servers版本 5 的服务器 SDK 集成的游戏服务器。

Amazon GameLift Servers 映像	架构	最新补丁	基础 AWS 图片
Amazon Linux 2023 BASE_AMI_LINUX_2023	x86	2025-12-16	亚马逊 Linux 2023 AMI 2023.9.20 251208.0 x86_64 HVM kernel-6.1 (发行说明)
Amazon Linux 2 BASE_AMI_LINUX_2_SDK_5	x86	2025-12-16	亚马逊 Linux 2 内核 5.10 AMI 2.0.20251208.0 x86_64 HVM gp2 (发行说明)
Windows 2016 BASE_AMI_WINDOWS_2016_SDK_5	x86	2025-12-16	Windows_Server-2016-English-Full-Base-2025.12.10 (发行说明)
Amazon Linux 2023 BASE_AMI_LINUX_2023_ARM	ARM64	2025-12-16	亚马逊 Linux 2023 AMI 2023.9.20 251208.0 arm64 HVM kernel-6.1 (发行说明)
Amazon Linux 2 BASE_AMI_LINUX_2_ARM	ARM64	2025-12-16	亚马逊 Linux 2 LTS Arm64 内核 5.10 AMI 2.0.20251208.0 arm64 HVM gp2 (发行说明)
Windows 2022 BASE_AMI_WINDOWS_2022	x86	2025-12-16	Windows_Server-2022-English-Full-Base-2025.12.10 (发行说明)

AMIs 用于服务器 SDK for Amazon GameLift Servers 4

Amazon GameLift Servers使用以下内容 AMIs 来托管与Amazon GameLift Servers版本 4 或更早版本的服务器 SDK 集成的游戏服务器。

Amazon GameLift Servers 映像	架构	最新补丁	基础 AWS 图片
Amazon Linux 2 BASE_AMI_LINUX_2_SDK_4	x86	2025-12-16	亚马逊 Linux 2 AMI 2.0.20251208.0 x86_64 HVM gp2 (发行说明)
Amazon Linux 2023 BASE_AMI_LINUX_2023_SDK_4 仅在中国（北京）和中国（宁夏）区域可用。	x86	2025-12-16	亚马逊 Linux 2023 AMI 2023.9.20 251208.0 x86_64 HVM kernel-6.1 (发行说明)
Windows 2016 BASE_AMI_WINDOWS_2016_SDK_4	x86	2025-12-16	Windows_Server-2016-English-Full-Base-2025.12.10 (发行说明)

有关更多信息，请参阅以下资源：

- [Amazon Linux 2023 release notes](#)
- [Amazon Linux 2 release notes](#)
- [AWS Windows AMI 版本历史记录](#)
- [2024 软件更新服务和 Windows Server Update Services 内容更改的说明](#)

Amazon GameLift Servers 服务位置

Amazon GameLift Servers功能可在多个区域 AWS 区域 和 Local Zones 中使用。您可以设计一种托管解决方案，将游戏服务器部署在与玩家地理位置高度匹配的位置。

支持 AWS 的地点

下表列出了支持Amazon GameLift Servers资源的 AWS 区域 和 Local Zones。它表示您可以在每个位置创建的资源类型。

地理位置	位置代码	托管式实例集的主区域 (单位置)	托管式实例集的主区域 (多位置)	托管式实例集的远程位置 (多位置)	Anywhere 实例集	游戏会话队列	FlexMatch 对战构建器和规则集	DDoS 保护 (玩家网关)
美国东部 (弗吉尼亚州北部)	us-east-1	支持	是	是	是	是	是	是 [†]
美国东部 (俄亥俄州)	us-east-2	支持		是	是	是		
美国西部 (北加利福尼亚)	us-west-1	支持		是	是	是		
美国西部 (俄勒冈州)	us-west-2	支持	是	是	是	是	是	是 [†]
非洲 (开普敦)	af-south-1			是				
亚太地区 (泰国)	ap-southeast-7			是				
亚太地区 (香港)	ap-east-1			是				
亚太地区 (马来西亚)	ap-southeast-5			是				

地理位置	位置代码	托管式实例集的主区域 (单位置)	托管式实例集的主区域 (多位置)	托管式实例集的远程位置 (多位置)	Anywhere实例集	游戏会话队列	FlexMatch对战构建器和规则集	DDoS保护 (玩家网关)
亚太地区 (孟买)	ap-south-1	支持		是	是	是		
亚太地区 (大阪)	ap-northeast-3			是				
亚太地区 (首尔)	ap-northeast-2	支持	是	是	是	是	是	是 [†]
亚太地区 (新加坡)	ap-southeast-1	支持		是	是	是		
亚太地区 (悉尼)	ap-southeast-2	支持	是	是	是	是	是	是 [†]
亚太地区 (东京)	ap-northeast-1	支持	是	是	是	是	是	是 [†]
加拿大 (中部)	ca-central-1	支持		是	是	是		
欧洲地区 (法兰克福)	eu-central-1	支持	是	是	是	是	是	是 [†]

地理位置	位置代码	托管式实例集的主区域 (单位置)	托管式实例集的主区域 (多位置)	托管式实例集的远程位置 (多位置)	Anywhere实例集	游戏会话队列	FlexMatch对战构建器和规则集	DDoS保护 (玩家网关)
欧洲地区 (爱尔兰)	eu-west-1	支持	是	是	是	是	是	是 [†]
欧洲 (伦敦)	eu-west-2	支持		是	是	是		
欧洲地区 (米兰)	eu-south-1			是				
欧洲地区 (巴黎)	eu-west-3			是				
欧洲地区 (斯德哥尔摩)	eu-north-1			是				
中东 (巴林)	me-south-1			是				
南美洲 (圣保罗)	sa-east-1	支持		是	是	是		
亚特兰大本地区域	us-east-1-atl-1			是				
芝加哥本地区域	us-east-1-chi-1			是				

地理位置	位置代码	托管式实例集的主区域 (单位置)	托管式实例集的主区域 (多位置)	托管式实例集的远程位置 (多位置)	Anywhere实例集	游戏会话队列	FlexMatch对战构建器和规则集	DDoS保护 (玩家网关)
达拉斯本地区域*	us-east-1-dfw-1			是				
达拉斯本地区域 2	us-east-1-dfw-2			是				
丹佛本地区域	us-west-2-den-1			是				
休斯顿本地区域	us-east-1-iah-1			是				
堪萨斯城本地区域	us-east-1-mci-1			是				
洛杉矶本地区域	us-west-2-lax-1			是				
凤凰城本地区域	us-west-2-phx-1			是				
尼日利亚拉各斯本地区域	af-south-1-los-1			是				

* 适用于 AWS 账户 已选择加入的用户。

† 有效的舰队本土区域可获 DDoS 保护（玩家网关）。

Note

默认情况下 AWS 区域，某些和所有的 Local Zones 都未启用 AWS 账户。您必须先选择加入这些区域或 Local Zones，然后才能将游戏服务器部署到这些位置。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](#)。请参阅《AWS Local Zones 用户指南》中的[Local Zones 入门](#)。（在 2022 年 2 月 28 日之前创建的实例集不受此要求影响。）

此外，您必须更新 Amazon GameLift Servers 管理员策略以允许 `ec2:DescribeRegions` 操作。有关原定设置情况下未启用的区域政策示例，请参阅[管理权限示例](#)。

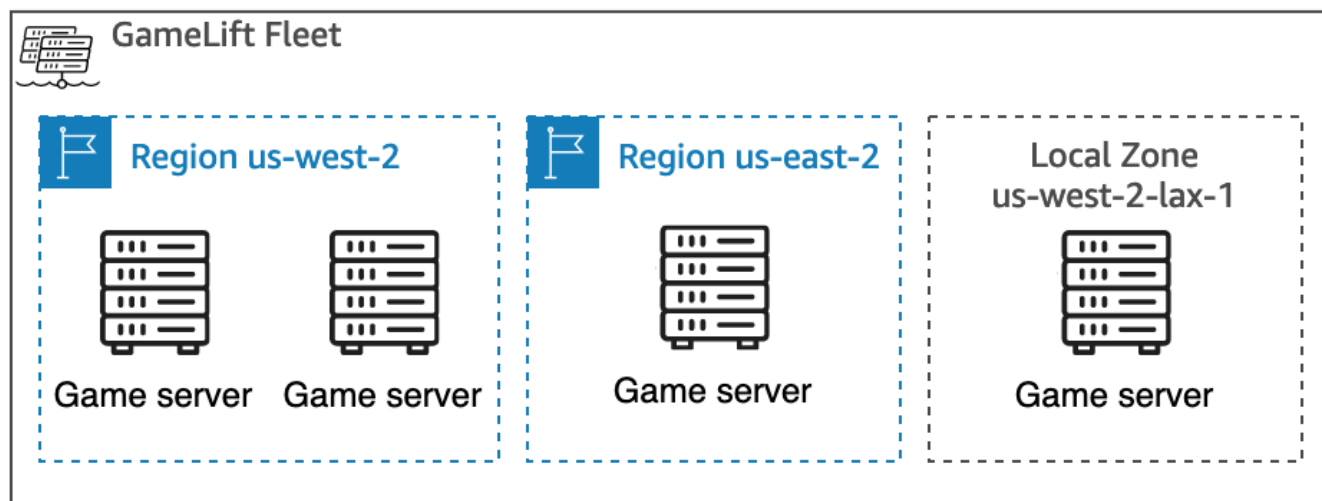
托管式托管的位置

Amazon GameLift Servers 托管式托管将部署游戏服务器资源的实例集。每支舰队都是在舰队所在区域创建的。AWS 区域实例集的主区域在实例集的 Amazon 资源编号（ARN）中引用。

您可以部署单区域实例集，仅在主区域中托管资源。或者，您也可以部署多位置实例集，在多个地理位置托管资源。多位置实例集有一个主区域和一个或多个偏远位置。在实例集中对位置进行分组时，不存在任何限制。管理多位置实例集的托管容量时，您可以为每个位置单独设置容量。

多地点舰队的远程位置可以是“其他”AWS 区域或“Local Zones”。本地区域是 AWS 区域的扩展。该区域可让您将计算资源部署在更靠近用户的位置，从而为玩家提供更低延迟的游戏体验。有关更多信息，请参阅[AWS Local Zones](#)。本地区域的位置代码由其所属父区域的代码，后跟一个物理位置标识符构成。例如，洛杉矶本地区域的代码是 `us-west-2-lax-1`。

下图展示了一个多位置实例集，其资源分布在两个 AWS 区域 和一个本地区域。该实例集的主区域是 `us-west-2`，包含两个远程位置：`us-east-2` 区域和 `us-west-2-lax-1` 本地区域。



除实例集资源外，基于 Amazon GameLift Servers 的托管式托管还可以使用以下类型的资源。您可以在支持 AWS 区域 该资源类型的特定环境中创建这些资源。

- **生成包**：这是一个专为托管于托管式 EC2 实例集而构建的游戏服务器。需在生成包待部署的实例集所在区域中创建相应的生成包资源。
- **脚本**：这是用于通过 Amazon GameLift Servers Realtime 托管游戏的配置脚本。需在脚本待部署的实例集所在区域中创建相应的生成包资源。
- **容器组定义**：这是在托管式容器实例集上运行容器的配置。该配置会指定一个或多个包含游戏服务器软件的容器映像，供部署到容器实例集中使用。需在容器组定义待部署的实例集所在区域中创建相应的容器组定义资源。
- **游戏会话队列**：此资源用于处理游戏会话请求并启动新的游戏会话。处理发生在队列 AWS 区域 所在的位置。为减少游戏会话放置过程中的延迟，建议在地理上靠近使用该队列的玩家群体的区域创建会话队列。

适用于 Amazon GameLift Servers Anywhere 的位置

Amazon GameLift Servers Anywhere 实例集是您提供的托管硬件的集合。您需负责管理托管资源上的所有活动，包括部署游戏服务器软件、维护软件更新及启动服务器进程。创建 Anywhere 实例集的目的，是将 Amazon GameLift Servers 服务与您的自管理式托管资源建立连接。Amazon GameLift Servers 负责管理游戏会话放置，具体包括处理玩家加入请求、查找可用的托管资源、启动新的游戏会话以及为游戏客户端提供连接信息。您可在所有支持 Anywhere 实例集的 AWS 区域 中创建该类型实例集。

您可以通过注册将托管硬件的实例添加到 Anywhere 实例集中。每个注册的实例都必须关联一个自定义位置。自定义位置与 AWS 区域 或 Local Zones 无关。它们用于表示硬件的实际物理位置。

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 Amazon GameLift Servers Anywhere 实例集](#)和[使用 Amazon GameLift Servers Anywhere 设置本地测试](#)。

适用于 Amazon GameLift Servers FlexMatch 的位置

FlexMatch 资源用于处理玩家的对战请求。其中包括对战配置资源和规则集资源。处理是在 FlexMatch 资源 AWS 区域 所在的地方进行的。为减少对战过程中的延迟，建议在地理上靠近使用这些资源的玩家群体的区域创建资源。某一对战配置及其所引用的规则集必须位于同一 AWS 区域。您可以在任何支持它们的 FlexMatch 资源中创建资源。AWS 区域

有关如何为您的托管解决方案设置 FlexMatch 的更多信息，请参阅《[Amazon GameLift Servers FlexMatch 开发人员指南](#)》。

Amazon GameLift Servers 在中国

Amazon GameLift Servers 在中国（北京）区域（由光环新网运营）或中国（宁夏）区域（由西云数据运营）时，您必须有一个单独的 AWS（中国）账户。请注意，某些特征在中国区域中不可用。有关在这些区域使用 Amazon GameLift Servers 的更多信息，请参阅以下资源：

- [亚马逊云科技中国](#)
- [Amazon GameLift Servers](#)（亚马逊云科技中国入门）

Amazon GameLift Servers 端点和限额

- 有关您可用于以编程方式连接服务的服务端点和信标端点，请参阅[Amazon GameLift Servers 服务端点](#)。
- 有关每个 AWS 账户使用服务资源或操作的 Amazon GameLift Servers 配额，请参阅[Amazon GameLift Servers 服务配额](#)。有关配额以及如何请求增加配额的更多信息，请参阅[AWS service quotas](#)。您可以使用 Amazon GameLift Servers 控制台请求增加配额。

亚马逊 GameLift 服务器 API 限制

本节中的表格列出了亚马逊 GameLift 服务器 API 操作的默认速率限制。这些限制有助于保障服务稳定性并确保公平使用。每个 API 操作根据其资源影响和使用模式设有特定限制。

默认情况下，每个客户账户的节流上限为突发请求 500 次、常规流量 500 次。此外，每个 API 都有自己的默认限制值对，如下所述。

API 限制参考

下表列出了亚马逊 GameLift 服务器 API 操作的默认速率限制。APIs 标有资源级限制可能会受到额外限制，以防止出现数据库热键问题。

API 限制参考

API 操作	突增限制	速率限制	资源级节流	注意
AcceptMatch	100	100	否	限制按账户维度生效。
ClaimGameServer	40	20	否	限制按账户维度生效。
CreateAlias	4	1	否	限制按账户维度生效。
CreateBuild	10	1	否	限制按账户维度生效。
CreateContainerFleet	4	1	否	限制按账户维度生效。
CreateContainerGroupDefinition	10	1	否	限制按账户维度生效。
CreateFleet	4	1	否	限制按账户维度生效。
CreateFleetLocations	4	1	否	限制按账户维度生效。
CreateGameServerGroup	5	0.1	否	限制按账户维度生效。
CreateGameSession	50	50	不可以	限制按账户维度生效。
CreateGameSessionQueue	4	1	否	限制按账户维度生效。
CreateLocation	4	1	否	限制按账户维度生效。
CreateMatchmakingConfiguration	4	1	否	限制按账户维度生效。

API 操作	突增限制	速率限制	资源级节流	注意
CreateMatchmakingRuleSet	4	1	否	限制按账户维度生效。
CreatePlayerSession	200	200	否	限制按账户维度生效。
CreatePlayerSessions	200	200	否	限制按账户维度生效。
CreateScript	10	1	否	限制按账户维度生效。
CreateVpcPeeringAuthorization	1	0.1	否	限制按账户维度生效。
CreateVpcPeeringConnection	1	0.1	否	限制按账户维度生效。
DeleteAlias	4	1	否	限制按账户维度生效。
DeleteBuild	10	1	否	限制按账户维度生效。
DeleteContainerFleet	4	1	否	限制按账户维度生效。
DeleteContainerGroupDefinition	10	1	否	限制按账户维度生效。
DeleteFleet	4	1	否	限制按账户维度生效。
DeleteFleetLocations	4	1	否	限制按账户维度生效。
DeleteGameServerGroup	5	0.1	否	限制按账户维度生效。
DeleteGameSessionQueue	4	1	否	限制按账户维度生效。
DeleteLocation	4	1	否	限制按账户维度生效。
DeleteMatchmakingConfiguration	4	1	否	限制按账户维度生效。

API 操作	突增限制	速率限制	资源级节流	注意
DeleteMatchmakingRuleSet	4	1	否	限制按账户维度生效。
DeleteScalingPolicy	4	1	否	限制按账户维度生效。
DeleteScript	10	1	否	限制按账户维度生效。
DeleteVpcPeeringAuthorization	1	0.1	否	限制按账户维度生效。
DeleteVpcPeeringConnection	1	0.1	否	限制按账户维度生效。
DeregisterGameServer	40	20	否	限制按账户维度生效。
DeregisterCompute	10	2	否	限制按账户维度生效。
DescribeAlias	10	2	否	限制按账户维度生效。
DescribeBuild	20	5	否	限制按账户维度生效。
DescribeCompute	10	2	否	限制按账户维度生效。
DescribeContainerFleet	20	5	否	限制按账户维度生效。
DescribeContainerGroupDefinition	20	5	否	限制按账户维度生效。
描述 EC2 InstanceLimits	40	15	否	限制按账户维度生效。
DescribeFleetAttributes	10	2	否	限制按账户维度生效。

API 操作	突增限制	速率限制	资源级节流	注意
DescribeFleetCapacity	4	1	否	限制按账户维度生效。
DescribeFleetDeployment	20	5	否	限制按账户维度生效。
DescribeFleetEvents	10	5	否	限制按账户维度生效。
DescribeFleetLocationAttributes	10	2	否	限制按账户维度生效。
DescribeFleetLocationCapacity	10	5	否	限制按账户维度生效。
DescribeFleetLocationUtilization	10	2	否	限制按账户维度生效。
DescribeFleetPortSettings	10	2	否	限制按账户维度生效。
DescribeFleetUtilization	10	2	否	限制按账户维度生效。
DescribeGameServer	40	20	否	限制按账户维度生效。
DescribeGameServerGroup	10	2	否	限制按账户维度生效。
DescribeGameServerInstances	10	2	否	限制按账户维度生效。
DescribeGameSessionDetails	10	5	否	限制按账户维度生效。请考虑 DescribeGameSessions 改用，因为它具有更高的默认限制并且支持资源级别限制。

API 操作	突增限制	速率限制	资源级节流	注意
DescribeGameSessionPlacement	75	75	是	限制按账户维度生效。受资源级节流约束。
DescribeGameSessionQueues	10	2	否	限制按账户维度生效。
DescribeGameSessions	20	20	是	限制按账户维度生效。受资源级节流约束。
DescribeInstances	10	2	否	限制按账户维度生效。
DescribeMatchmaking	200	200	是	限制按账户维度生效。受资源级节流约束。
DescribeMatchmakingConfigurations	15	7	否	限制按账户维度生效。
DescribeMatchmakingRuleSets	20	10	否	限制按账户维度生效。
DescribePlayerSessions	200	200	是	限制按账户维度生效。受资源级节流约束。
DescribeRuntimeConfiguration	4	1	否	限制按账户维度生效。
DescribeScalingPolicies	20	20	否	限制按账户维度生效。
DescribeScript	20	5	否	限制按账户维度生效。
DescribeVpcPeeringAuthorizations	1	0.1	否	限制按账户维度生效。
DescribeVpcPeeringConnections	1	0.1	否	限制按账户维度生效。

API 操作	突增限制	速率限制	资源级节流	注意
GetComputeAccess	4	1	否	限制按账户维度生效。
GetComputeAuthToken	10	5	否	限制按账户维度生效。
GetGameSessionLogUrl	4	1	否	限制按账户维度生效。
GetInstanceAccess	4	1	否	限制按账户维度生效。
ListAliases	10	2	否	限制按账户维度生效。
ListBuilds	20	5	否	限制按账户维度生效。
ListContainerFleets	20	5	否	限制按账户维度生效。
ListCompute	10	2	否	限制按账户维度生效。
ListContainerGroupDefinitions	20	5	否	限制按账户维度生效。
ListContainerGroupDefinitionVersions	20	5	否	限制按账户维度生效。
ListFleets	10	2	否	限制按账户维度生效。
ListFleetDeployments	20	5	否	限制按账户维度生效。
ListGameServerGroups	10	2	否	限制按账户维度生效。
ListGameServers	10	2	否	限制按账户维度生效。
ListLocations	10	2	否	限制按账户维度生效。
ListScripts	20	5	否	限制按账户维度生效。
ListTagsForResource	20	5	否	限制按账户维度生效。

API 操作	突增限制	速率限制	资源级节流	注意
PutScalingPolicy	4	1	否	限制按账户维度生效。
RegisterCompute	10	5	否	限制按账户维度生效。
RegisterGameServer	40	20	否	限制按账户维度生效。
RequestUploadCredentials	4	1	否	限制按账户维度生效。
ResolveAlias	4	1	否	限制按账户维度生效。
ResumeGameServerGroup	1	0.1	否	限制按账户维度生效。
SearchGameSessions	10	5	否	限制按账户维度生效。
StartFleetActions	1	0.1	否	限制按账户维度生效。
StartGameSessionPlacement	75	75	否	限制按账户维度生效。
StartMatchBackfill	70	70	否	限制按账户维度生效。
StartMatchmaking	100	100	否	限制按账户维度生效。
StopFleetActions	1	0.1	否	限制按账户维度生效。
StopGameSessionPlacement	20	20	否	限制按账户维度生效。
StopMatchmaking	100	100	否	限制按账户维度生效。
SuspendGameServerGroup	1	0.1	否	限制按账户维度生效。
TagResource	20	5	否	限制按账户维度生效。

API 操作	突增限制	速率限制	资源级节流	注意
TerminateGameSession	10	2	否	限制按账户维度生效。
UntagResource	20	5	否	限制按账户维度生效。
UpdateAlias	4	1	否	限制按账户维度生效。
UpdateBuild	10	1	否	限制按账户维度生效。
UpdateContainerGroupDefinition	10	1	否	限制按账户维度生效。
UpdateContainerFleet	4	1	否	限制按账户维度生效。
UpdateFleetAttributes	4	1	否	限制按账户维度生效。
UpdateFleetCapacity	4	1	否	限制按账户维度生效。
UpdateFleetPortSettings	4	1	否	限制按账户维度生效。
UpdateGameServer	40	20	否	限制按账户维度生效。
UpdateGameServerGroup	5	0.1	否	限制按账户维度生效。
UpdateGameSession	10	2	否	限制按账户维度生效。
UpdateGameSessionQueue	4	1	否	限制按账户维度生效。
UpdateMatchmakingConfiguration	4	1	否	限制按账户维度生效。
UpdateRuntimeConfiguration	4	1	否	限制按账户维度生效。
UpdateScript	10	1	否	限制按账户维度生效。

API 操作	突增限制	速率限制	资源级节流	注意
ValidateMatchmakingRuleSet	4	1	否	限制按账户维度生效。

资源级节流

部分 API 操作受资源级节流约束，以避免数据库热键问题。即便提高这些操作的 API 级限制，它们仍可能在资源级被节流。

以下 API 操作受资源级节流约束：

- DescribeGameSessionPlacement
- DescribeGameSessions
- DescribeMatchmaking
- DescribePlayerSessions

资源级节流通通过包含资源标识符的特定节流键实现。例如，DescribeGameSessionPlacement 使用密钥 “Operation:GameLift/DescribeGameSessionPlacement,aws-account-placement-id:” 来强制对每个账号和展示位置ID的组合进行限制。这可以防止客户因请求特定资源而使系统不堪重负。

Amazon GameLift Servers SDK API 限制

下表列出了 Amazon GameLift Servers SDK API 操作的默认速率限制。这些限制旨在优化游戏服务器环境中的性能并避免资源争用问题。了解这些限制对高效实现与 Amazon GameLift Servers 的服务器端集成至关重要。

使用包含账户和进程标识符的特定节流键强制执行服务器 SDK 限制。部分限制按玩家维度设置，以防止单个玩家的恶意滥用行为。

Note

本表内容可能随时更改。当前限制的权威依据为内部 SDC 配置。如需获取最新信息，请联系 AWS Support。

Server SDK API 限制参考

API 操作	突增限制	速率限制	节流键	备注
AcceptPlayerSession	3	1	aws-account-and-process-id-and-player-id	限制按玩家和进程维度生效；因按玩家维度控制，无需提升上限。
RemovePlayerSession	3	1	aws-account-and-process-id-and-player-id	限制按玩家和进程维度生效；因按玩家维度控制，无需提升上限。
ActivateGameSession	3	1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
ActivateHostProcessV2	3	1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
StartMatchBackfill	3	1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
DescribePlayerSessions()	10	5	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
GetComputeCertificate	3	0.1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
GetCustomRoleCredentials	3	0.1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。

API 操作	突增限制	速率限制	节流键	备注
InitSDK	3	0.1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
RefreshHostProcess	3	0.1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
StopMatchmaking	3	1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
ProcessEnding	3	1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
ProcessReady	3	1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
UpdateGameSession	3	1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
UpdateHostProcess	3	1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
GetGameSessionId	10	5	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。
GetTerminationTime	3	1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。

API 操作	突增限制	速率限制	节流键	备注
UpdatePlayerSessionCreationPolicy	3	1	aws-account-and-process-id	限制按服务器进程维度生效；因按进程维度控制，无需提升上限。

Note

这些是默认限制。如果您在使用此类 API 时遇到节流，则可能是由于账户级别的节流所致。如需申请提升限制额度，请联系 AWS Support。

进程级和玩家级限制的设计旨在适配常规使用场景，通常无需调整。如果您在使用此类 API 时遇到节流，建议先检查自身实现以挖掘优化空间，再申请提升限制额度。

账户级别的节流还可能通过以下额外规则触发：

- SdkWebSocket/AccountLevelHighUsage,aws-account
- SdkWebSocket/AccountLevelThrottle,aws-account

Amazon GameLift Servers UDP ping 信标

UDP ping 信标提供了一种方法，可用来测量玩家设备和 Amazon GameLift Servers 托管位置之间的网络延迟。借助 ping 信标，您可以收集准确的延迟数据，以便就游戏服务器的放置做出明智的决定，并根据延迟要求改善玩家对战质量。

UDP ping 信标的工作原理

Amazon GameLift Servers 在每个可以部署游戏服务器的托管位置，提供固定的 UDP 端点（ping 信标）。由于大多数游戏服务器都使用 UDP 进行通信，因此与使用 ICMP ping 信标测量延迟相比，使用 UDP ping 信标测量延迟可以获得更准确的结果。网络设备处理 ICMP 数据包的方式通常与 UDP 数据包不同，这可能导致延迟测量结果无法真实反映玩家将体验到的性能。

借助 UDP ping 信标，您的游戏客户端可以向这些端点发送 UDP 消息并接收异步响应，从而为您提供延迟测量结果，以更好地反映玩家设备和潜在托管位置之间的实际游戏流量状况。这些端点是永久性的，只要 Amazon GameLift Servers 在该位置支持游戏托管，它们就会一直可用。

UDP ping 信标的常见使用案例

您可以通过多种方式使用 UDP ping 信标来优化游戏的联网体验。

选择最佳托管位置

收集不同地理区域的延迟数据，为您的玩家群确定托管游戏服务器的最佳主位置和备用位置。

根据玩家延迟放置游戏会话

在请求新的游戏会话时包括玩家延迟数据，以帮助选择延迟最低，体验最佳的位置。

根据延迟提升对战质量

在请求对战时提供玩家延迟数据，以帮助匹配具有相似延迟特征的玩家，并将游戏会话放置在匹配玩家的最佳位置。

Note

创建对战请求时，请勿向没有实例集的位置提供延迟信息。否则，Amazon GameLift Servers 可能会尝试将游戏会话放置在没有实例集容量的位置，从而导致对战请求失败。

获取信标端点

要检索Amazon GameLift Servers位置的 ping 信标域和端口信息，请使用 [ListLocations](#) API 操作。此 API 返回的位置集取决于 AWS 区域您在调用它时指定的位置（如果您未指定，则取决于您的默认区域）。当您从以下位置调用 API 时：

- 支持多位置的实例集的主区域：API 返回“所有”托管位置的信息
- 支持单位置的实例集的主区域：API 返回该位置的信息

请注意，如果您调用此 API 时使用的位置只能作为多位置实例集内的远程位置，则 API 将返回错误，因为该类型的位置没有服务端点。

请查阅[支持 AWS 的地点](#)中支持的位置表，以确定支持单位置实例集和多位置实例集的主区域。

示例

```
aws gamelift list-locations --region ap-northeast-2
```

这 AWS 区域 支持多地点舰队，因此将返回多个地点。以下是其中一个返回值示例：

```
[...]  
{  
  "LocationName": "ap-northeast-1",  
  "PingBeacon": {  
    "UDPEndpoint": {  
      "Domain": "gamelift-ping.ap-northeast-1.api.aws",  
      "Port": 7770  
    }  
  }  
}
```

Important

缓存 ping 信标信息，而不是在每次测量延迟之前调用 `ListLocations`。域和端口信息是静态的，而且 API 并非设计用于处理大量请求。

实施延迟测量

在使用 UDP ping 信标实施延迟测量时，请遵循以下最佳实践：

1. 使用以下方法之一存储 ping 信标信息：

- 在游戏客户端中对端点进行硬编码。
- 在游戏后端缓存信息。
- 实施定期更新机制（每天/每周）以刷新信息。

2. 发送 UDP ping 消息：

- 消息正文可包含任意内容（不可为空），并且消息大小不超过 300 字节。
- 请遵守每个位置的以下速率限制：
 - 每个唯一发件人 IP 地址和端口组合：每秒 3 个事务（TPS）
 - 每个唯一发件人 IP 地址：每秒 1000 个事务

3. 计算延迟：

- 向每个位置发送多个 ping 以计算平均延迟。
- 考虑向多个位置发送并发 ping，以更快地获得结果。

- 由于 UDP 不能 100% 保证传送，因此对于任何未在短时间内（通常为 1-3 秒）返回的数据包，应根据需要使用重试逻辑发送新数据包。
- 计算发送消息和接收响应之间的时间差。
- 如果发送到某个位置的大部分 UDP ping 请求始终未收到响应，请使用 ICMP ping 针对标准 [Amazon GameLift Servers 服务端点](#) 计算延迟，作为备用方案。

Tip

我们建议您在记录玩家必须在本地网络上开放的端口列表时，添加端口 7770。这也正是您需要准备备用延迟测量方案（例如使用 ICMP）的另一个原因，以防此端口被阻塞。

代码示例

以下是一些简单的示例，展示了如何发送 UDP ping 并计算延迟。

C++

```
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <chrono>

int main() {
    // Replace with Amazon GameLift Servers UDP ping beacon domain for your desired
    location
    const char* domain = "gamelift-ping.ap-south-1.api.aws";
    const int port = 7770;
    const char* message = "Ping";           // Your message
    const int num_pings = 3;                // Number of pings to send

    // Create socket
    int sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock < 0) {
        std::cerr << "Error creating socket" << std::endl;
        return 1;
    }
```



```

}

// Resolve domain name to IP address
struct hostent* host = gethostbyname(domain);
if (host == nullptr) {
    std::cerr << "Error resolving hostname" << std::endl;
    close(sock);
    return 1;
}

// Set up the server address structure
struct sockaddr_in server_addr;
std::memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(port);
std::memcpy(&server_addr.sin_addr, host->h_addr, host->h_length);

// Set socket timeout
struct timeval tv;
tv.tv_sec = 1; // 1 second timeout
tv.tv_usec = 0;
if (setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv)) < 0) {
    std::cerr << "Error setting socket timeout" << std::endl;
    close(sock);
    return 1;
}

double total_latency = 0;
int successful_pings = 0;

for (int i = 0; i < num_pings; ++i) {
    auto start = std::chrono::high_resolution_clock::now();

    // Send the message
    ssize_t bytes_sent = sendto(sock, message, std::strlen(message), 0,
                                (struct sockaddr*)&server_addr,
sizeof(server_addr));

    if (bytes_sent < 0) {
        std::cerr << "Error sending message" << std::endl;
        continue;
    }

    // Receive response

```

```

    char buffer[1024];
    socklen_t server_addr_len = sizeof(server_addr);
    ssize_t bytes_received = recvfrom(sock, buffer, sizeof(buffer), 0,
                                      (struct sockaddr*)&server_addr,
    &server_addr_len);

    auto end = std::chrono::high_resolution_clock::now();

    if (bytes_received < 0) {
        std::cerr << "Error receiving response or timeout" << std::endl;
    } else {
        std::chrono::duration<double, std::milli> latency = end - start;
        total_latency += latency.count();
        successful_pings++;

        std::cout << "Received response, latency: " << latency.count() << " ms"
    << std::endl;
    }

    // Wait a bit before next ping
    usleep(1000000); // 1s
}

// Close the socket
close(sock);

if (successful_pings > 0) {
    double avg_latency = total_latency / successful_pings;
    std::cout << "Average latency: " << avg_latency << " ms" << std::endl;
} else {
    std::cout << "No successful pings" << std::endl;
}

return 0;
}

```

C#

```

using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Diagnostics;

```

```
using System.Threading.Tasks;

class UdpLatencyTest
{
    static async Task Main()
    {
        // Replace with Amazon GameLift Servers UDP ping beacon domain for your
        // desired location
        string domain = "gamelift-ping.ap-south-1.api.aws";
        int port = 7770;
        string message = "Ping";           // Your message
        int numPings = 3;                  // Number of pings to send
        int timeoutMs = 1000;              // Timeout in milliseconds

        await MeasureLatency(domain, port, message, numPings, timeoutMs);
    }

    static async Task MeasureLatency(string domain, int port, string message, int
numPings, int timeoutMs)
    {
        using (var udpClient = new UdpClient())
        {
            try
            {
                // Resolve domain name to IP address
                IPAddress[] addresses = await Dns.GetHostAddressesAsync(domain);
                if (addresses.Length == 0)
                {
                    Console.WriteLine("Could not resolve domain name.");
                    return;
                }

                IPEndPoint endPoint = new IPEndPoint(addresses[0], port);
                byte[] messageBytes = Encoding.UTF8.GetBytes(message);

                // Set receive timeout
                udpClient.Client.ReceiveTimeout = timeoutMs;

                double totalLatency = 0;
                int successfulPings = 0;
                var stopwatch = new Stopwatch();

                for (int i = 0; i < numPings; i++)
                {
```

```

        try
        {
            stopwatch.Restart();

            // Send message
            await udpClient.SendAsync(messageBytes, messageBytes.Length,
endPoint);

            // Wait for response
            UdpReceiveResult result = await
ReceiveWithTimeoutAsync(udpClient, timeoutMs);

            stopwatch.Stop();
            double latency = stopwatch.Elapsed.TotalMilliseconds;

            totalLatency += latency;
            successfulPings++;

            string response = Encoding.UTF8.GetString(result.Buffer);
            Console.WriteLine($"Ping {i + 1}: {latency:F2}ms - Response:
{response}");
        }
        catch (SocketException ex)
        {
            Console.WriteLine($"Ping {i + 1}: Failed - {ex.Message}");
        }
        catch (TimeoutException)
        {
            Console.WriteLine($"Ping {i + 1}: Timeout");
        }

        // Wait before next ping
        await Task.Delay(1000); // 1s between pings
    }

    if (successfulPings > 0)
    {
        double averageLatency = totalLatency / successfulPings;
        Console.WriteLine($"Summary:");
        Console.WriteLine($"Successful pings: {successfulPings}/
{numPings}");
        Console.WriteLine($"Average latency: {averageLatency:F2}ms");
    }
    else

```

```

        {
            Console.WriteLine("\nNo successful pings");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error: {ex.Message}");
    }
}

static async Task<UdpReceiveResult> ReceiveWithTimeoutAsync(UdpClient client,
int timeoutMs)
{
    using var cts = new System.Threading.CancellationTokenSource(timeoutMs);
    try
    {
        return await client.ReceiveAsync().WaitAsync(cts.Token);
    }
    catch (OperationCanceledException)
    {
        throw new TimeoutException("Receive operation timed out");
    }
}
}

```

Python

```

import socket
import time
import statistics
from datetime import datetime

def udp_ping(host, port, timeout=2):
    """
    Send a UDP ping and return the round trip time in milliseconds.
    Returns None if timeout occurs.
    """
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.settimeout(timeout)
    message = b'ping'

    try:

```

```
# Resolve hostname first
try:
    socket.gethostbyname(host)
except socket.gaierror as e:
    print(f"Could not resolve hostname: {e}")
    return None

start_time = time.time()
sock.sendto(message, (host, port))

# Wait for response
data, server = sock.recvfrom(1024)
end_time = time.time()

# Calculate round trip time in milliseconds
rtt = (end_time - start_time) * 1000
return rtt

except socket.timeout:
    print(f"Request timed out")
    return None
except Exception as e:
    print(f"Error: {type(e).__name__}: {e}")
    return None
finally:
    sock.close()

def main():
    # Replace with Amazon GameLift Servers UDP ping beacon domain for your desired
    location
    host = "gamelift-ping.ap-south-1.api.aws"
    port = 7770
    num_pings = 3
    latencies = []

    print(f"\nPinging {host}:{port} {num_pings} times...")
    print(f"Start time: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")

    for i in range(num_pings):
        print(f"Ping {i+1}:")
        rtt = udp_ping(host, port)

        if rtt is not None:
            print(f"Response from {host}: time={rtt:.2f}ms")
```

```
latencies.append(rtt)

# Wait 1 second between pings
if i < num_pings - 1:
    time.sleep(1)
print()

# Calculate and display statistics
print("-" * 50)
print(f"Ping statistics for {host}:")
print(f"    Packets: Sent = {num_pings}, Received = {len(latencies)}, "
      f"Lost = {num_pings - len(latencies)} "
      f"f"{{{(num_pings - len(latencies)) / num_pings * 100}:.1f}% loss}")

if latencies:
    print("\nRound-trip latency statistics:")
    print(f"    Minimum = {min(latencies):.2f}ms")
    print(f"    Maximum = {max(latencies):.2f}ms")
    print(f"    Average = {statistics.mean(latencies):.2f}ms")

print(f"\nEnd time: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")

if __name__ == "__main__":
    main()
```


先前版本

删除 Salttime	
删除	
删除	
删除 Unreal	
删除 SDK	
删除	
删除	
删除	
删除	
删除	
删除	

删除 Unreal (C	
删除 Unity	
删除 ++)	
删除 Unreal	

删除 10-2	
删除 5	
删除	
删除	
删除	
删除	
删除	
删除	

删除 10-2	
删除 5	
删除	
删除	
删除	
删除	
删除	
删除	

删除 10-0	
删除 5	
删除	

- 引擎Saltire
- 引擎
- 引擎
- 引擎Unreal
- 引擎SDK
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎Unreal (C
- 引擎Unity
- 引擎)
- 引擎Unreal
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎008-1
- 引擎5
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎006-2
- 引擎5
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎Saltire
- 引擎
- 引擎
- 引擎Unity
- 引擎SDK
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎Unreal (C
- 引擎Unity
- 引擎++)
- 引擎Unreal

- 引擎5-2
- 引擎5
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎0.1.1E
- 引擎5
- 引擎)
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎Saltire
- 引擎
- 引擎
- 引擎Unity
- 引擎SDK
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎Unreal (C
- 引擎Unity
- 引擎++)
- 引擎Unreal

- 引擎3-2
- 引擎5
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎3-1
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- Amazon Salttime
- 删除
- 端
- Amazon
- SDK
- 插
- 件
- 工
- 具
- 包

- Unreal (C
- Unity
- ++)
- Unreal

- [2018-1](#)
- 或
- 更
- 高
- 反
- 本

- [2018-0](#)
- 7
- 或
- 更
- 高
- 反
- 本

Amazon Salttime
删除
端
Control
SDK
插
件
工
具
包

Unreal (C
Unity
++)
Unreal

[2-1](#)
或
更
高
版
本

[3-1](#)
或
更
高
版
本

Amazon Salttime
删除
端
Amazon
SDK
插
件
工
具
包

Unreal (C
Unity
++)
Unreal

[20251](#)
成
更
高
版
本

[20251](#)
成
更
高
版
本

[20251](#)
成
更
高
版
本

Saltime

辨別

第

此補也 Kal

DK

插

耕

C

mk

包

```
349  Conreal ( C
```

Unity

 $\vdash \vdash)$

Inreal

2009131

2

13

五

反

7

ENDING 130

22

13

15

一

レ

BIBLIO

2

13

五

反

レ

- Amazon Salttime
- 删除
- 端
- Amazon
- SDK
- 插
- 件
- 工
- 具
- 包

- Unreal (C
- Unity
- ++)
- Unreal

- [5000041](#)
- 成
- 更
- 高
- 反
- 本

- [50000712](#)
- 成
- 更
- 高
- 反
- 本

- [50000612](#)
- 成
- 更
- 高
- 反
- 本

	Realtime
	顧客
	端
	OS / al
	SDK
	插 件
	工 具 包
	Unreal (C ++) Unity Unreal
	50205-1
	成 更 高 版 本
	50205-2
	成 更 高 版 本
	50209-2
	成 更 高 版 本

- 引擎Saltire
- 引擎
- 引擎
- 引擎Unity
- 引擎SDK
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎Unreal (C
- 引擎Unity
- 引擎++)
- 引擎Unreal

- [500000-1](#)
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- [500000-0](#)
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- [500000-3](#)
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎Saltire
- 引擎
- 引擎
- 引擎Unity
- 引擎SDK
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎Unreal (C
- 引擎Unity
- 引擎)
- 引擎Unreal

- 引擎[30000-NE](#)
- 引擎
- 引擎.NET
- 引擎)
- 引擎
- 引擎

- 引擎[30000-2](#)
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎[30000-2](#)
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- Amazon Salttime
- 删除
- 部署
- Amazon
- SDK
- 插
- 件
- 工
- 具
- 包

- Unreal (C
- Unity
- ++)
- Unreal

- [3.12.0-0](#)
- 或
- 更
- 高
- 反
- 本

- [3.12.0-2](#)
- 或
- 更
- 高
- 反
- 本

- [3.12.0-1](#)
- 或
- 更
- 高
- 反
- 本

Saltime

辨別

第

此補也 Kal

DK

插

耕

C

mk

包

```
349  Conreal ( C
```

Unity

 $\left(\frac{1}{2} + \frac{1}{2} \right)$

Inreal

~~3042B.0~~

12

13

五

反

レ

302B.02-2

12

III

五

一又

レ

30320-2

2

15

五

反

レ

- Amazon Salttime
- 删除
- 端
- Amazon
- SDK
- 插
- 件
- 工
- 具
- 包

- Unreal (C
- Unity
- ++)
- Unreal

- [3.12.0-1](#)
- 或
- 更
- 高
- 反
- 本

- [3.12.0-1](#)
- 或
- 更
- 高
- 反
- 本

- [3.12.0-2](#)
- 或
- 更
- 高
- 反
- 本

- Amazon Salttime
- 删除
- 部署
- Amazon
- SDK
- 插
- 件
- 工
- 具
- 包

- Unreal (C
- Unity
- ++)
- Unreal

- [3.12.0-1](#)
- 或
- 更
- 高
- 版
- 本

- [3.12.0-0](#)
- 或
- 更
- 高
- 版
- 本

- [3.12.0-1](#)
- 或
- 更
- 高
- 版
- 本

- Amazon Salttime
- 删除
- 端
- Amazon
- SDK
- 插
- 件
- 工
- 具
- 包

- Unreal (C
- Unity
- ++)
- Unreal

- [4.10.1](#)
- 或
- 更
- 高
- 反
- 本

- [4.10.2](#)
- 或
- 更
- 高
- 反
- 本

- [4.10.0](#)
- 或
- 更
- 高
- 反
- 本

- Amazon Salttime
- 删除
- 部署
- Amazon
- SDK
- 插
- 件
- 工
- 具
- 包

- Unreal (C
- Unity
- ++)
- Unreal

- [13900-0](#)
- 或
- 更
- 高
- 版
- 本

- [13900-2](#)
- 或
- 更
- 高
- 版
- 本

- [13903-0](#)
- 或
- 更
- 高
- 版
- 本

- 引擎Saltire
- 引擎
- 引擎
- 引擎Unity
- 引擎SDK
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎Unreal (C
- 引擎Unity
- 引擎++)
- 引擎Unreal

- 引擎3002-0
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎3002-1
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎22109-2
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

包

Inreal

ニ

レ

レ

- 引擎Saltire
- 引擎
- 引擎
- 引擎Unreal
- 引擎SDK
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎Unreal (C
- 引擎Unity
- 引擎++)
- 引擎Unreal

- 引擎2002-0
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎11709-0
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- 引擎11708-1
- 引擎
- 引擎
- 引擎
- 引擎
- 引擎

- Amazon Salttime
- 删除
- 部署
- Amazon
- SDK
- 插
- 件
- 工
- 具
- 包

- Unreal (C
- Unity
- ++)
- Unreal

- [X1.502-1](#)
- 或
- 更
- 高
- 版
- 本

- [X1.503-1](#)
- 或
- 更
- 高
- 版
- 本

- [X1.502-2](#)
- 或
- 更
- 高
- 版
- 本

包

Inreal

レ

7

レ

Salttime	10
删除	10
部署	10
引擎	10
SDK	10
插	10
件	10
工	10
具	10
包	10
Unreal (C	10
Unity	10
++)	10
Unreal	10
10.7.08-0	10
成	10
更	10
高	10
版	10
本	10

发行说明

以下发行说明按时间顺序排列，首先列出最新更新。Amazon GameLift Servers 于 2016 年首次发布。要了解早于此处所列的发行说明，请参阅[SDK 版本](#)中的发布日期链接。

2026 年 3 月 4 日：Amazon GameLift Servers现已为基于 Linux 的 EC2 和容器队列提供 DDoS 保护 SDKv5

Amazon GameLift Servers现在为基于 Linux 的 EC2 和容器队列提供 distributed-denial-of-service (DDoS) 保护。SDKv5玩家网关代理中继网络与您的游戏服务器实例位于同一位置，提供流量验证、每位玩家的速率限制和游戏服务器 IP 地址混淆功能，所有这些都增加了可以忽略不计的延迟，而且不会产生额外费用。此功能最初在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、欧洲（法兰克福）、欧洲（爱尔兰）、亚太地区（悉尼）、亚太地区（东京）、亚太地区（东京）、亚太地区（首尔）。有关服务地点的更多信息，请参阅[Amazon GameLift Servers 服务位置](#)。

了解更多：

- [博客文章：引入 Amazon GameLift Servers DDoS 保护](#)
- [DDoS 保护带有 Amazon GameLift Servers](#)，《Amazon GameLift Servers 开发人员指南》
- [玩家网关的工作原理](#)，《Amazon GameLift Servers 开发人员指南》
- [将玩家网关集成到游戏中](#)，《Amazon GameLift Servers 开发人员指南》
- [适用于 C++ 的客户端 SDK](#)
- [虚幻引擎客户端 SDK](#)。访问权限需要在 Epic Games 组织中的成员资格 GitHub。详情请[访问虚幻引擎](#)。GitHub

2026 年 2 月 24 日：Amazon GameLift Servers 插件现在支持 Unity 6.3 和虚幻引擎 5.7

更新了插件版本：

- [适用于 Unity 的插件](#)，版本 3.2.1（包括适用于 Unity 的服务器 SDK，版本 5.4.0）
- [虚幻引擎插件](#)，版本 3.1.1（包括虚幻引擎的服务器 SDK，版本 5.4.0）

Amazon GameLift Servers 发布了适用于虚幻引擎和 Unity 的插件和服务器 SDK 的新版本。这些最新版本包含以下更新：

- 它们现在支持在虚幻引擎 5.7 和 Unity 6.3 中使用。

了解更多：

- [将游戏与适用于 Unreal Engine 的 Amazon GameLift Servers 插件集成](#)，《Amazon GameLift Servers 开发人员指南》
- [将 Amazon GameLift Servers 集成到 Unreal Engine 项目中](#)，《Amazon GameLift Servers 开发人员指南》
- [将游戏与 Unity Amazon GameLift Servers 插件集成](#)，《Amazon GameLift Servers 开发人员指南》
- [Amazon GameLift Servers 集成到 Unity 引擎项目](#)中，《Amazon GameLift Servers 开发人员指南》

2026 年 1 月 29 日：Amazon GameLift Servers 现在支持缩放 Fleets to/from 零实例

Amazon GameLift Servers 现在支持根据游戏会话活动自动缩放到零实例和从零实例缩放。队列在规定的无游戏会话活动期限后缩小到零，在请求游戏会话时从零向上扩展，这为成本优化提供了一个选项。

Amazon GameLift Servers API 参考链接：

- [AWS SDK 操作 UpdateFleetCapacity](#)

了解更多：

- [管理扩展 Amazon GameLift 服务器队列 To/From 为零](#)

2026 年 1 月 22 日：Amazon GameLift ServersRealtime 现在支持 2023 年亚马逊 Linux 上的 Node.js 24.x

Amazon GameLift ServersRealtime现在在亚马逊 Linux 2023 上支持 Node.js 24.x。此更新允许在现代亚马逊 Linux 2023 操作系统上运行时访问最新的 Node.js 功能和安全改进。

现在，您还可以在创建Realtime队列时加入可选的安装脚本，允许在实例启动期间根据您的特定要求安装软件。要使用新的运行时版本，请在创建Realtime脚本时指定 Node.js 24.x。

Amazon GameLift Servers API 参考链接：

- [AWS SDK 操作 CreateScript](#)

了解更多：

- [Node.js 运行时版本](#)
- [添加安装脚本](#)

2025 年 12 月 9 日：Amazon GameLift Servers现在在 Web 控制台中提供基于人工智能的帮助

Amazon GameLift Servers现在在 Web 控制台中提供人工智能支持，利用 Amazon Q Developer 为游戏开发者提供量身定制的指导。这项新功能整合了专业Amazon GameLift Servers知识，可帮助客户更有效地浏览复杂的工作流程、解决问题并优化其游戏服务器部署。

这项新功能直接在Amazon GameLift Servers主机中为游戏服务器集成、舰队配置和性能优化提供人工智能辅助建议。此增强旨在简化决策流程，缩短故障排除时间，提高整体资源利用率，从而节省成本并改善玩家体验。

了解更多：

- [Amazon GameLift Servers 控制台](#)

2025 年 11 月 6 日：Amazon GameLift Servers 现在支持使用 Windows Server 2022 操作系统的游戏生成包

Amazon GameLift Servers 客户现在可以使用 Windows Server 2022 操作系统来托管其游戏服务器。相较于 Windows Server 2016，Windows Server 2022 提供了多项改进，包括安全性的提高。除中国地区外，该操作系统在所有 AWS 区域均可用。

待对 Windows Server 2016 的支持于 2027 年 1 月终止后，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。对 Windows Server 2022 的支持将持续到 2031 年 10 月 13 日。

Amazon GameLift Servers API 参考链接：

- [AWS SDK 操作 CreateBuild](#)
- [CLI 命令 upload-build](#)
- [CLI 命令 create-build](#)

2025 年 10 月 28 日：Amazon GameLift Servers 为所有服务器 SDKs 和插件添加 OpenTelemetry (OTel) 指标收集器

Amazon GameLift Servers 现在所有服务器 SDKs 和插件中都包含内置 OpenTelemetry (OTel) 指标集合，为游戏服务器性能监控提供了标准化的可观察性。通过这种集成，可以直接为您的游戏服务器添加行业标准的遥测功能，即无需其他工具或配置即可实现全面监控并排查问题。

通过 OpenTelemetry 集成，您现在可以：

- 使用标准化协议，自动捕获基本的游戏服务器指标、日志和跟踪数据
- 与现有监控平台和可观测性工具无缝集成
- 实时监控所有实例集实例的游戏服务器性能和运行状况
- 使用适用于不同监控解决方案的遥测数据，更快地排查问题

OpenTelemetry 是一个开源可观测性框架，它提供了一种收集、处理和导出遥测数据的标准化方法。

了解更多：

- [使用服务器遥测指标进行监控](#)，《Amazon GameLift Servers 开发人员指南》

2025 年 10 月 2 日：Amazon GameLift Servers 为实例集实例添加了基于控制台的远程访问功能

Amazon GameLift Servers 现在可以通过 Amazon GameLift Servers 控制台，直接远程访问托管的 EC2 实例和容器实例集实例，无需进行身份验证管理。这项新功能使用 Amazon EC2 Systems Manager (SSM)，提供基于浏览器的安全终端访问，无需额外设置或凭证管理。

借助基于控制台的远程访问功能，您现在可以：

- 实时排查游戏服务器集成问题
- 在活跃游戏会话期间监控游戏服务器活动并访问日志文件
- 使用实际玩家流量运行诊断和基准测试工具

此功能适用于运行服务器 SDK 版本 5.x 的实例集。对于运行较早 SDK 版本的舰队，您可以继续使用现有 AWS CLI 方法进行远程访问。

要远程连接到队列实例，请打开 Amazon GameLift Servers 控制台并找到您要访问的托管 EC2 或容器队列。在实例集的详细信息中，打开实例选项卡，从列表中选择一个实例，然后选择连接。

了解更多：

- [远程连接到 Amazon GameLift Servers 实例集实例](#)，《Amazon GameLift Servers 开发人员指南》

2025 年 9 月 23 日：Amazon GameLift Servers 增加了对德克萨斯州达拉斯新本地区域的支持

借助 Amazon GameLift Servers 托管式托管，您现在可以在德克萨斯州达拉斯的新本地区域 (us-east-1-dfw-2) 部署游戏服务器资源。借助 Local Zones，您可以将游戏服务器放置在地理上更靠近玩家的位置，以降低延迟并显著改善游戏体验。在这个新本地区域中，您可以使用以下 EC2 实例类型：C6gn、C6i、C6in、M6g、M6i、M6in、M8g 和 R6i。有关每个实例类型的详细说明，请参阅 [AWS EC2 实例类型](#)。

要开始在新的达拉斯本地区域托管游戏会话，请先为您的 AWS 账户选择启用本地区域。然后，可以将其作为远程位置添加到新的或现有的多位置实例集内。如果您的游戏使用 Amazon GameLift Servers

FlexMatch，请更新对战队列中的实例集，以包含新的本地区域。利用多位置实例集，您可以直接管理每个位置的托管容量。

新的德克萨斯州达拉斯本地区域的父 AWS 区域 区域是 us-east-1 (弗吉尼亚州)。原德克萨斯州达拉斯本地区域 (us-east-1-dfw-1) 已不再接受新的选择加入请求。

了解更多：

- [入门 AWS Local Zones](#)，《AWS Local Zones 用户指南》
- [Amazon GameLift Servers 服务位置](#)，《Amazon GameLift Servers 开发人员指南》
- [更新实例集位置](#)，《Amazon GameLift Servers 开发人员指南》

2025 年 8 月 12 日：适用于 Unreal Engine 的 Amazon GameLift Servers 插件现已支持 UE 版本 5.6

更新了插件版本：

- [适用于 Unreal 的插件](#)，版本 3.0.1 (包括适用于 Unreal 的服务器 SDK，版本 5.3.1)

Amazon GameLift Servers 发布了适用于 Unreal Engine 的插件新版本，和适用于 Unreal Engine 的服务器 SDK。这些最新版本包含以下更新：

- 现在支持与 Unreal Engine 5.6 结合使用。
- 解决了打包 Android 客户端目标时的编译问题。

了解更多：

- [将游戏与适用于 Unreal Engine 的 Amazon GameLift Servers 插件集成](#)，《Amazon GameLift Servers 开发人员指南》
- [将 Amazon GameLift Servers 集成到 Unreal Engine 项目中](#)，《Amazon GameLift Servers 开发人员指南》

2025 年 6 月 24 日：Amazon GameLift Servers 发布了 UDP ping 信标，以帮助测量多人游戏的实时网络延迟

更新了 SDK 版本：

- AWS SDK 1.11.595

Amazon GameLift Servers 正式发布了一组名为 UDP ping 信标的固定端点，用于帮助您准确测量玩家设备和游戏服务器位置之间的延迟。UDP ping 信标端点在 Amazon GameLift Servers 支持的所有 AWS 全球区域和 Local Zones 均可用，但 AWS 中国区域除外。

因为 UDP（用户数据报协议）在实时游戏方面具有性能优势，所以大多数多人游戏都将其作为主要的数据包传输协议。了解和缩短网络延迟对于提供最佳玩家体验至关重要。UDP ping 信标提供了一种一致而可靠的方式来测量玩家和游戏服务器之间的实际 UDP 数据包延迟，可帮助您更好地决定 player-to-server 匹配和游戏会话放置。

[ListLocations](#) API 已扩展为将端点域和端口信息作为其返回的位置列表的一部分，从而可以轻松地以编程方式访问端点。

您的游戏客户端可以向这些端点发送 UDP 消息，并接收包含相同数据的异步响应，从而为您提供延迟测量结果，以更好地反映玩家设备和潜在托管位置之间的实际游戏流量状况。这些端点是永久性的，只要 Amazon GameLift Servers 在该位置支持游戏托管，它们就会一直可用。

了解更多：

- [UDP ping 信标](#)，《Amazon GameLift Servers 开发人员指南》
- [ListLocations](#)，Amazon GameLift Servers API 参考

2025 年 6 月 24 日：Amazon GameLift Servers 增加了对泰国和马来西亚的位置支持

借助 Amazon GameLift Servers 托管式托管，您现在可以在泰国曼谷和马来西亚吉隆坡部署游戏服务器资源，从而将游戏的覆盖范围扩展到整个东南亚的玩家。这些新区域有助于减少延迟，改善这些区域的玩家游戏体验。

以下 AWS 区域是多地点舰队的远程地点。要开始在这些位置托管游戏会话，请将它们作为远程位置，添加到新的或现有多位置实例集内。利用多位置实例集，您可以直接管理每个位置的托管容量。

- 亚太地区（泰国）（ap-southeast-7）
- 亚太地区（马来西亚）（ap-southeast-5）

默认情况下 AWS 区域，这些功能不会为 AWS 账户启用。您必须先选择启用每个区域，才能在该区域部署 Amazon GameLift Servers 资源。

了解更多：

- [Amazon GameLift Servers 服务位置](#)，《Amazon GameLift Servers 托管开发人员指南》
- [更新实例集位置](#)，《Amazon GameLift Servers 托管开发人员指南》
- 在《[账户管理参考指南](#)》中启用或禁用AWS 账户中的 [AWS 区域](#)。

2025 年 5 月 29 日：Amazon GameLift Servers 发布了适用于 Unreal Engine 和 Unity 的服务器 SDK 及插件新版本

更新了 SDK 版本：

- [C++ 服务器 SDK](#)，版本 5.3.0
- [C# 服务器 SDK](#)，版本 5.3.0
- [Go 服务器 SDK](#)，版本 5.3.0
- [适用于 Unreal 的服务器 SDK](#)，版本 5.3.0
- [适用于 Unity 的服务器 SDK](#)，版本 5.3.0

更新了插件版本：

- [适用于 Unreal 的插件](#)，版本 3.0.0
- [适用于 Unity 的插件](#)，版本 3.1.0

适用于 C++、C#、Go、Unreal 和 Unity 的服务器 SDK 新版本，以及适用于 Unreal Engine 和 Unity 的插件新版本现在是开源的。它们都在[Amazon GameLift Servers GitHub 组织](#)中可用。有关详细的更新信息，请参阅每个存储库中的发行说明和自述文件。

密钥服务器 SDK 更新：

- 改进了所有服务器 SDKs 中的客户端验证和错误响应。
- [OnProcessTerminate](#) 回调函数现在具有结束游戏服务器进程的默认逻辑。
- 函数 [InitSDK\(\)](#) 现在使用幂等性令牌来支持多次重试。
- 回 [OnUpdateGameSession](#) 调现在可以为传递数据 `autoBackfillMode`。

主要插件更新：

- 现在，虚幻引擎插件的安装和设置流程更加简化，自动化程度更高，先决条件更少（ CMakeOpenSSL和虚幻交叉编译工具链 ）。
- 适用于 Unreal Engine 的插件改善了托管式 EC2 工作流程的用户界面体验，包括支持游戏客户端和服务端生成包路径中的空格。此外，您现在可以在从编辑器启动游戏客户端时添加命令行参数。
- 虚幻引擎的插件现在支持 ARM 服务器内置 UE5。

了解更多：

- [Amazon GameLift Servers虚幻引擎插件](#)，《Amazon GameLift Servers 开发人员指南》
- [Amazon GameLift Servers适用于 Unity 的插件（ 服务器 SDK 5.x ）](#)，《Amazon GameLift Servers 开发人员指南》
- [Amazon GameLift Servers 服务器 SDK 5.x](#)，《Amazon GameLift Servers 开发人员指南》
- [将 Amazon GameLift Servers 集成到 Unreal Engine 项目中](#)，《Amazon GameLift Servers 开发人员指南》
- [将 Amazon GameLift Servers 集成到 Unity 项目中](#)，《Amazon GameLift Servers 开发人员指南》

2025 年 4 月 24 日：适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5 现在支持 .NET 8

更新了 SDK 版本：

- C# 服务器开发工具包，版本 5.2.1

对于使用 C# 的游戏开发人员，您现在可以使用 .NET 8 作为 Amazon GameLift Servers 项目的目标框架。使用 .NET 8，您可以利用性能改进，包括改进的 just-in-time (JIT) 编译、内存使用优化和更快的启动时间。如果您当前使用的是 .NET 6，我们建议您计划迁移到 .NET 8，包括将 C# 服务器 SDK 更新到最新版本。Microsoft 宣布在 2026 年 11 月之前支持 .NET 8（包括提供安全补丁和技术更新）。

[下载适用于 Amazon GameLift Servers 的最新版 C# 服务器 SDK](#)

了解更多：

- [获取 Amazon GameLift Servers 开发工具](#)，《Amazon GameLift Servers 开发人员指南》

2025 年 3 月 27 日：Amazon GameLift Servers 扩展了对新一代 EC2 实例系列的支持

更新了 SDK 版本：

- AWS SDK 1.11.535

现在，您可以使用 Amazon GameLift Servers 通过以下方式微调游戏服务器托管：从第 5 代到第 8 代实例系列的更多 Amazon EC2 实例中进行选择。每一代新的 EC2 实例都在计算、内存和网络方面有所提升，其中第 8 代实例提供基于尖端 AWS Graviton4 和 Intel Xeon 的实例。新一代实例适用于以下实例系列：

- [通用型](#) (M 系列)
- [计算优化型](#) (C 系列)
- [内存优化型](#) (R 系列)

您还可以选择提供本地存储 (d)、增强联网 (n) 和特定处理器架构 (Intel/AMD/Graviton – i/a/g) 的变体。除 AWS 中国地区外 AWS 区域，所有支持的 Amazon GameLift Servers 区域均提供下一代实例。有关更多详细信息，请参阅[Amazon GameLift Servers 服务位置](#)。

将这些新实例类型与 Amazon GameLift Servers 托管的 EC2 实例集和托管式容器实例集结合使用。为现有游戏托管改用新的实例类型（相同架构）时，只需部署新的实例集即可，除实例类型外，所有配置设置均保持不变。

了解更多：

- [Amazon GameLift Servers 实例定价](#)
- [CreateFleetAmazon GameLift Servers](#) API 参考
- [CreateContainerFleet](#)，Amazon GameLift Servers API 参考

2025 年 3 月 11 日：Amazon GameLift Servers 推出全新游戏服务器封装器，便于快速入门

适用于 Amazon GameLift Servers 新的游戏服务器包装器大大缩短了您在 Amazon GameLift Servers 上托管游戏服务器所需的时间。无需更改代码，即可使用包装器向游戏添加基本的游戏会话管理功能，并将其部署到 Amazon GameLift Servers Anywhere 实例集、托管的 EC2 实例集或托管容器实例集内。如果您希望通过自有游戏服务器，或示例游戏中的服务器对 Amazon GameLift Servers 功能进行亲自评估，该工具是理想选择。它还可用于快速部署游戏服务器迭代，例如用于快速原型开发或测试。

借助基本的游戏会话管理功能，您的游戏服务器可以初始化与 Amazon GameLift Servers 服务的连接，响应启动和停止游戏会话的提示，并在游戏会话完成时关闭。

[从下载游戏服务器包装器](#)。 [GitHub](#) 请参阅自述文件， [GitHub](#) 了解如何在游戏项目中安装和使用包装器。

2025 年 3 月 6 日：亚马逊现在 GameLift Amazon GameLift Servers 是 Amazon GameLift Streams

Amazon GameLift 是一项完全托管的服务，致力于帮助开发者构建、扩展和交付世界上要求最苛刻的游戏。随着 Amazon Streams 的正式发布，亚马逊 GameLift 现在提供了高扩展的游戏服务器和 GameLift 流畅的游戏直播功能。

借助 [Amazon GameLift Servers](#)，游戏开发人员能够部署、运行和扩展专用游戏服务器。您可以在几分钟内在云端部署高性能游戏服务器，并根据玩家需求进行扩展或缩减。建立在 AWS 久经考验的计算环境之上，在单个游戏中 Amazon GameLift Servers 支持 1 亿并发玩家，每秒支持 10 万名玩家，每分钟支持 9 千个新计算实例。而且，它具有企业级安全性、适合最大人群的配对以及 pay-as-you-go 灵活性，无论你是在研究新的游戏创意，还是运行一款拥有数百万玩家的游戏，它都能帮助你入门。

[Amazon GameLift Stream](#) s 可帮助游戏开发者提供分辨率高达 1080p 和 60 frames-per-second (fps) 的游戏直播体验，在 iOS、Android、FireOS 等设备和 PCs 游戏玩家之间没有明显的延迟。使用单一 AWS 产品，发行商无需修改即可在几分钟内将其游戏内容部署到完全托管的基于云的 GPU 实例上，然后通过 AWS Network Backbone 将其直接交付到任何装有 Web 浏览器的终端玩家设备上。玩家可以在短短几秒钟内开始游戏，无需等待下载或安装，而且它提供的游戏体验与在本地 PC 或游戏机上玩游戏几乎没有区别。

2025 年 1 月 14 日：Amazon GameLift Servers 发布了对游戏会话放置优先级设置的扩展支持

更新了 SDK 版本：

- AWS SDK 1.11.485

为了回应客户的反馈，我们发布了新功能，允许您为单个游戏会话放置请求设置位置优先级。对于配置为按位置确定放置优先级的队列，现在可以为每个放置请求提供自定义的优先位置列表。

借助这项新功能，客户可以根据需要动态更改每个放置请求的位置优先级。额外的灵活性意味着您可以更好地应对不断变化的情况，例如玩家位置、实例集负载或服务器运行状况。它还可以为想要进一步自定义放置位置选择方式的客户提供支持。

了解更多：

- [优先考虑游戏会话放置](#)，《Amazon GameLift Servers 开发人员指南》
- [StartGameSessionPlacement](#)，Amazon GameLift ServersAPI 参考

2025 年 1 月 2 日：Amazon GameLift Servers 发布终止游戏会话的新支持

更新了 SDK 版本：

- AWS SDK 1.11.477

为了回应客户的反馈，我们发布了新功能，可让您更轻松地终止单个游戏会话。在此版本中，您现在可以直接在Amazon GameLift Servers主机中终止游戏会话，也可以使用 AWS CLI 或 AWS SDK 终止游戏会话Amazon GameLift Servers。

这项新功能解决了仍处于活跃状态但状态不良的游戏会话问题，而此类问题会导致计算资源无法托管新的游戏会话。以前，客户需要远程访问计算资源才能手动终止游戏会话。

您有两种终止方法可供选择。第一种方法尝试使用游戏的自定义关闭序列，优雅地终止游戏会话，其中可能包括通知玩家和解析游戏数据的操作。第二种方法强制服务器进程停止，这会立即终止游戏会话。即使服务器进程无响应，第二种方法也能确保游戏会话结束。

了解更多：

- [使用 Amazon GameLift Servers 控制台关闭游戏会话](#)，《Amazon GameLift Servers 开发人员指南》
- [TerminateGameSession](#)，Amazon GameLift ServersAPI 参考

2024 年 12 月 19 日：Amazon GameLift Servers 发布了对托管容器的游戏引擎插件支持

更新了插件版本：

适用于 Unreal Engine 的 Amazon GameLift Servers 插件，版本 2.0.0

- 已升级为支持托管容器的 C++ 服务器 SDK 5.2.0。
- 增加了对 Unreal Engine 5.4 和 5.5 的支持。

适用于 Unity 的 Amazon GameLift Servers 插件，版本 3.0.0

- 已升级为支持托管容器的 C++ 服务器 SDK 5.2.0。
- 支持适用于 Windows 和 Mac OS 的 Unity 2021.3 LTS 和 2022.3 LTS。

适用于 Unreal 和 Unity 游戏引擎的 Amazon GameLift Servers 插件提供了工具和工作流程，可简化您使用 Amazon GameLift Servers 启动和运行游戏的步骤。Amazon GameLift Servers 是一项完全托管的云托管服务，游戏开发人员可以使用它来管理和扩展基于会话的多人游戏的专用游戏服务器。

最新的插件版本提供了以下增强功能：

- 使用托管容器进行托管的指导式工作流程。此工作流程将引导您完成以下步骤：使用游戏服务器软件设置容器映像；为游戏服务器部署基于云的托管解决方案。该工作流程提供了两种不同的部署方案：一种是简单的部署，另一种是使用游戏会话放置队列和 FlexMatch 对战构建器进行更完整的部署。每个场景都会生成 Amazon GameLift Servers 集装箱舰队和支持 AWS 资源。
- 改进了设置 AWS 用户配置文件和管理插件使用的 AWS 访问凭据的流程。您可以维护多个个人资料以处理不同的 AWS 账户、账户用户和区域。
- 新增了更新现有容器实例集的功能。您可以部署新的容器镜像（例如用于游戏服务器版本更新）和更改实例集配置设置，而不必从头开始。
- 改进了使用 Amazon GameLift Servers Anywhere 实例集和托管的 EC2 进行托管的工作流程。根据客户反馈进行的改进包括提供更好的指导，其中包含提示和指向有用资源的链接。

托管容器和托管 EC2 解决方案的部署方案使用 AWS CloudFormation 模板为每个场景创建和部署 AWS 资源。这些模板包含在 Amazon GameLift Servers 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：

- [适用于 Unreal 的插件：将游戏部署到托管式容器实例集](#)，《Amazon GameLift Servers 开发人员指南》
- [适用于 Unity 的插件：将游戏部署到托管式容器实例集](#)，《Amazon GameLift Servers 开发人员指南》
- [从以下网址下载虚幻引擎插件 GitHub](#)
- [从以下网址下载 Unity 的插件 GitHub](#)
- [使用 Amazon GameLift Servers 托管游戏](#)
- [Amazon GameLift Servers 论坛](#)

2024 年 11 月 12 日：Amazon GameLift Servers 推出了基于托管容器的多人游戏托管服务

更新了 SDK 版本：

- AWS SDK 1.11.445
- 服务器 SDK，版本 5.2.0（所有语言）

Amazon GameLift Servers 正式发布了适用于容器化游戏服务器工作负载的全新托管解决方案。在此版本中，游戏开发人员现在可以利用容器化带来的优势，包括一致、安全的环境、简化的部署流程和优化的资源利用率。

托管容器实例集使用 Amazon EC2 实例，这些实例由 Amazon GameLift Servers 代表您并根据您的配置进行管理。您可以为游戏构建自定义容器架构，并通过将容器映像存储在 Amazon Elastic Container Registry (Amazon ECR) 存储库中来提供容器映像。容器实例集仅适用于基于 Linux 的游戏服务器。游戏服务器必须与服务器 SDK 5.2.0 或更高版本集成。

使用托管容器实例集，您可以获得与托管 EC2 实例集相同的优势。这包括支持按需实例和竞价型实例类型、智能容量扩展、使用队列放置游戏会话以及对战机制。您还可以获得与其他实例集类型相同的指标，以及一些针对容器的新指标。容器实例集的其他功能包括：

- 与容器化工作负载的无服务器体验保持一致。每个容器运行一个游戏服务器进程，并将多个容器打包到每个实例集实例上，以实现最佳资源利用率。如果您更喜欢使用包含多个游戏服务器进程的容器，则可以使用 Amazon GameLift Servers Agent 进行自动主机管理。
- 简化了实例集创建流程。容器舰队的设计要求使用最少的部署配置设置和合理的值 suggested/default。您可以快速部署可正常运行的实例集，然后根据需要自定义个人设置。
- 容器架构的版本控制工具。现在，您可以更新容器组定义（类似于容器“任务”）、维护多个版本以及指定要部署到实例集的版本。
- 实例集更新工具。借助容器实例集，您无需再为发布游戏服务器版本更新而创建新实例集。相反，您现在可以更新容器映像并将更新部署到现有实例集。

您可以在任何 Amazon GameLift Servers 支持多地点舰队 AWS 区域的地方构建 Amazon GameLift Servers 集装箱舰队，也可以将集装箱舰队实例部署到任何支持的远程位置。有关更多详细信息，请参阅 [Amazon GameLift Servers 服务位置](#)。托管容器目前 AWS 在中国区域不可用。

了解更多：

- 博客文章：[Leverage fully managed containers to host multiplayer games at global scale on Amazon GameLift Servers](#)
- [托管式容器](#)概述，《Amazon GameLift Servers 开发人员指南》
- [容器在 Amazon GameLift Servers 中的工作原理](#)，《Amazon GameLift Servers 开发人员指南》
- [借助 Amazon GameLift Servers 托管容器进行托管的开发路线图](#)，《Amazon GameLift Servers 开发人员指南》
- [CreateContainerFleet](#)，Amazon GameLift Servers API 参考

2024 年 9 月 19 日：Amazon GameLift Servers 发布了适用于 Unreal Engine 的 C++ 服务器 SDK 和插件的更新

更新了 SDK 版本：

C++ 服务器 SDK，版本 5.1.3

- 新的日志记录功能。您现在可以访问 SDK 请求日志。
- 提高了 SDK 消息传输的可靠性。SDK 现在使用更稳健的重新连接机制，可以在网络中断或随机丢失消息的情况下进行恢复。

更新了插件版本：

适用于 Unreal Engine 的 Amazon GameLift Servers 插件，版本 1.1.2

- 已升级为支持最新版本的 C++ 服务器 SDK 5.1.3。
- 在适用于 Unreal Engine 的 Amazon GameLift Servers 插件中，当您浏览实例集的服务器生成包可执行文件时，您现在可以选择浏览所有文件。

适用于 Unreal 的 C++ 服务器 SDK 插件，版本 5.1.2

- 已升级为支持最新版本的 C++ 服务器 SDK 5.1.3。

了解更多：

- [将游戏与适用于 Unreal Engine 的 Amazon GameLift Servers 插件集成](#)，《Amazon GameLift Servers 开发人员指南》

- [Amazon GameLift Servers 插件和 SDK 下载](#)

2024 年 9 月 5 日：Amazon GameLift Servers 提高了实例集创建过程的可观测性

根据客户反馈，我们明确了用于创建托管的 EC2 实例集并使其准备好托管游戏会话的 Amazon GameLift Servers 工作流程。改进功能包括：

- 我们对实例集创建过程的每个阶段都进行了更具体、更准确的描述。这种提高的可见性让您可以更轻松、更快速地查明和解决问题。
- 构建和激活阶段将实例部署任务（构建）与启动游戏服务器进程和连接到 Amazon GameLift Servers 服务的任务（激活）更好地分开。此更改可让您更轻松地确定问题的可能原因。此外，您现在可以在实例集处于激活阶段时远程连接到它们。
- 两个新的实例集创建事件可传达游戏服务器安装脚本的成败。如果您的游戏服务器生成包包含安装脚本，Amazon GameLift Servers 会尝试运行该脚本并发出以下新事件之一：
 - FLEET_CREATION_COMPLETED_INSTALLER
 - FLEET_CREATION_FAILED_INSTALLER

了解更多：

- [调试 Amazon GameLift Servers 队组问题](#)，《Amazon GameLift Servers 开发人员指南》
- [事件数据类型](#)，《Amazon GameLift Servers API 参考》

2024 年 8 月 15 日：Amazon GameLift Servers 改善了控制台体验

根据客户反馈，我们对 [Amazon GameLift Servers 控制台](#) 体验进行了以下更新：

- 现在，您的页面显示首选项会自动保存给您的 AWS 账户用户，并在您返回页面时应用。利用显示首选项，您可以选择在表显示中包含哪些信息，例如在“实例集列表”页面上。使用表右上角的  图标可自定义显示首选项。
- 针对托管式 EC2 实例集的“创建实例集”工作流程已经简化，将实例集位置和实例类型的选择结合起来。您可以更轻松地为实例集找到合适的实例类型，即使您更改了位置选择也无妨。

了解更多：

- [创建Amazon GameLift Servers托管 EC2 舰队](#)，《Amazon GameLift Servers 开发人员指南》

2024 年 7 月 25 日：Amazon GameLift Servers增加对 AWS 尼日利亚本地区域的支持

借助 Amazon GameLift Servers 托管式托管，您现在可以在西非尼日利亚部署游戏服务器资源，并将游戏的覆盖范围扩展到整个非洲的玩家。使用 Local Zones 将游戏服务器放置在离玩家更近的地理位置，以减少延迟并显著改善游戏玩法。

要立即开始在尼日利亚托管游戏会话，请将新的尼日利亚本地区域作为偏远位置添加到新的或现有的多位置实例集。如果您的游戏使用 Amazon GameLift Servers FlexMatch，请更新对战队列中的实例集，以包含新的本地区域。利用多位置实例集，您可以直接管理每个位置的托管容量。

尼日利亚拉各斯本地区域的父 AWS 区域 区域是非洲（开普敦）区域 `af-south-1`，Amazon GameLift Servers该区域也支持作为远程位置。尼日利亚本地区域的名称是 `af-south-1-los-1`。

了解更多：

- [Amazon GameLift Servers 服务位置](#)，《Amazon GameLift Servers 开发人员指南》
- [更新实例集位置](#)，《Amazon GameLift Servers 开发人员指南》

2024 年 7 月 2 日：Amazon GameLift Servers 发布了新的控制台工具，用于查看玩家会话数据

Amazon GameLift Servers 控制台现在提供一个玩家会话查找工具，让您可以按游戏会话 ID、玩家会话 ID 或玩家 ID 检索玩家会话信息。使用 FlexMatch 对战的游戏会自动为每个对战的玩家生成玩家会话。对于所有其他游戏，玩家会话是一项可选功能。

您可以在 Amazon GameLift Servers 控制台的主导航中找到玩家会话查找工具。查看单个玩家会话或比较多个玩家会话的数据。您还可以在查看游戏会话详细信息页面时打开玩家会话数据。

了解更多：

- [Amazon GameLift Servers 控制台中的游戏和玩家会话](#)，《Amazon GameLift Servers 开发人员指南》

2024 年 4 月 24 日：Amazon GameLift Servers 发布了容器实例集预览版

Amazon GameLift Servers 现在提供容器实例集预览版，从而提高了可移植性、可扩展性、容错能力和敏捷性。

在容器实例集中，Amazon EC2 实例可托管一个或多个容器。这些容器包含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 SDKs 和软件包。当您将容器上传到私有 Amazon Elastic Container Registry 后，Amazon GameLift Servers 会使用容器填充您的实例集。

要在容器实例集中正常运行，您的游戏服务器必须在 Linux 中运行并与服务器 SDK 5.x 集成。在容器实例集中，您可以对托管资源进行微调控制，从而优化 CPU 单元和内存等资源的消耗。您还可以在一个容器中托管多个游戏服务器，以减少资源的使用。

在容器实例集中，您可以享受其他类型的实例集具有的诸多优势，例如按需型实例类型、扩展（自动和手动）、队列和对战。您还可以获得与其他实例集类型相同的指标，以及一些针对容器的新指标。容器实例集可让您在全球范围内覆盖以下位置区域的玩家：

- ap-northeast-1
- ap-northeast-2
- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2

要覆盖更多区域和本地区域，请创建多位置容器实例集。

了解更多：

- [使用 Amazon GameLift Servers 容器管理托管](#)，《Amazon GameLift Servers 开发人员指南》
- [CreateContainerGroupDefinition](#)，Amazon GameLift Servers API 参考

2024 年 2 月 13 日：Amazon GameLift Servers 推出对 SDKs 虚幻引擎 Amazon GameLift Servers 插件的改进并简化其安装

更新了 SDK 版本：

- Go 服务器 SDK，版本 5.1.0
- C# 服务器 SDK，版本 5.1.2
- C++ 服务器 SDK，版本 5.1.2

我们进行了以下改进：

- 通过增加在网络中断时自动重新连接的功能，提高了 SDK 的可靠性。
- [Go] 您现在可以在使用或不使用服务器参数的情况下调用 `InitSDK()`。在 Amazon GameLift Servers 托管的 EC2 实例集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 Servers Anywhere 实例集上的游戏服务器必须使用服务器参数调用 `InitSDK()`。

更新了插件版本：

- 适用于 Unreal Engine 的 Amazon GameLift Servers 插件，版本 1.1.0
- 适用于 Unity 的 Amazon GameLift Servers 插件，版本 2.1.0
- 适用于 Unreal 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：

- [适用于 Unreal Engine 的 Amazon GameLift Servers 插件] 更新了安装说明并简化了打包。此插件现在包含最新版本的适用于 Unreal 的 C++ 服务器 SDK。
- 升级了插件以支持最新版本的 Amazon GameLift Servers 服务器 SDK。

了解更多：

- [将游戏与适用于 Unreal Engine 的 Amazon GameLift Servers 插件集成](#)，《Amazon GameLift Servers 开发人员指南》
- [Amazon GameLift Servers 插件和 SDK 下载](#)

2023 年 12 月 14 日：Amazon GameLift Servers 增加了更新活动游戏会话的游戏属性功能

您已经能够在创建游戏会话时设置游戏属性，并搜索指定属性的游戏会话。现在，您还可以在活动游戏会话中添加和更新这些属性。

例如，您的玩家投票选出他们想玩的地图。您的游戏客户端调用 `UpdateGameSession` 以将 `GameProperty` 值修改为 `{"Key": "map", "Value": "jungle"}`。然后，您的游戏会为游戏会话中的玩家实施新地图。

游戏管理员还可以使用 `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 `ACTIVE` 值为 `Status` 且游戏属性为 `{"Key": "map", "Value": "desert"}` 的游戏会话。

了解更多：

- [the section called “集成游戏客户端功能”](#)，《Amazon GameLift Servers 开发人员指南》
- [GameProperty](#)，Amazon GameLift ServersAPI 参考
- [UpdateGameSession](#)，Amazon GameLift ServersAPI 参考
- [SearchGameSessions](#)，Amazon GameLift ServersAPI 参考

2023 年 11 月 21 日：Amazon GameLift Servers推出对 Terraform 和 Pulumi 等基础设施即代码工具的支持 AWS 云端控制 API

您现在可以使用基础设施即代码 (IaC) 工具管理整个 Amazon GameLift Servers 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 Cloud Control API 配置和配置所有Amazon GameLift Servers资源类型。您可以使用Amazon GameLift Servers APIs 或的 AWS CloudFormation 模板继续使用资源Amazon GameLift Servers。

有关通过 IaC 提供的 Amazon GameLift Servers 资源的详细信息，请参阅 [Amazon GameLift Servers 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 为开发者提供了一套标准的 APIs 创建、读取、更新、删除和列出资源 (CRUDL) 的工具，包括数百种 AWS 服务和多个第三方工具，例如 Terraform 和 Pulumi。

了解更多：

- [AWS CloudFormation](#)
- [AWS 云控制 API](#)

- [AWS CC Terraform 提供商](#)
- [Pulumi](#)

2023 年 11 月 16 日：Amazon GameLift Servers 更新了适用于 Unity 的独立插件

更新了 SDK 版本：适用于 Unity 的 Amazon GameLift Servers 插件，版本 2.0.0

适用于 Unity 的 Amazon GameLift Servers 插件提供了工具和工作流程，可简化您使用 Amazon GameLift Servers 启动和运行 Unity 游戏的步骤。Amazon GameLift Servers 是一项完全托管的服务，游戏开发人员可以使用它来管理和扩展基于会话的多人游戏的专用游戏服务器。

在此版本中，适用于 Unity 的插件已更新为使用最新的 Amazon GameLift Servers 功能，包括服务器 SDK 版本 5.x 和对使用 Amazon GameLift Servers Anywhere 进行本地测试的支持。该插件与 Unity 版本 Unity 2021.3 LTS 和 2022.3 LTS 兼容。

插件的主要功能包括：

- Unity 编辑器中的引导式 UI 工作流程，适用于以下场景：
 - 使用您的本地工作站作为主机，测试游戏与 Amazon GameLift Servers 的集成。此工作流程可帮助您为本地计算机设置 Amazon GameLift Servers Anywhere 实例集、启动游戏服务器和客户端的实例、通过 Amazon GameLift Servers 请求游戏会话，以及加入游戏。
 - 使用托管 EC2 和支持 AWS 资源，为您的集成游戏服务器部署云 Amazon GameLift Servers 托管解决方案。此工作流程可帮助您配置游戏以进行云托管，并提供三个部署方案：
 - 将游戏服务器部署到单个实例集。
 - 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 Servers 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 IDs 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，该插件可自动执行所需的服务器参数设置。
 - 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 Amazon GameLift Servers 服务之间，用于管理对 Amazon GameLift Servers 服务的所有直接调用。

- 集成测试的内容，包括一个简单示例多人游戏的资产和代码，这些资产和代码用于说明游戏服务器和游戏客户端集成。
- 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 Anywhere 和 Amazon EC2 队列）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 Servers 插件下载文件中。您可以按原样使用它们，也可以根据游戏进行自定义。

了解更多：

- [Amazon GameLift Servers 适用于 Unity 的插件（服务器 SDK 5.x）](#)，《Amazon GameLift Servers 开发人员指南》
- [从以下地址下载插件 GitHub](#)
- [关于 Amazon GameLift Servers 托管](#)
- [Amazon GameLift Servers 论坛](#)

2023 年 11 月 2 日：Amazon GameLift Servers 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 Servers 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。此更新将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 GameLift Servers 将您的游戏服务器生成包部署到每个 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 Servers 会在实例的整个生命周期内自动刷新临时凭证。

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 Servers 控制台实例集创建工作流程中。
- 使用新参数 `InstanceRoleCredentialsProvider` 调用服务 API 操作 `CreateFleet` 时。

- 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 Servers 开发者指南》，[与舰队中的其他 AWS 资源进行通信](#)
- [CreateFleet](#)，[InstanceRoleCredentialsProvider](#)，[Amazon GameLift Servers API 参考](#)
- [设置 IAM 服务角色](#)，《Amazon GameLift Servers 开发人员指南》

2023 年 9 月 28 日：Amazon GameLift Servers 发布了全新的 Unreal Engine 独立插件

更新了 SDK 版本：适用于 Unreal Engine 的 Amazon GameLift Servers 插件，版本 1.0.0

适用于 Unreal Engine 的 Amazon GameLift Servers 插件提供了工具和工作流程，可简化您使用 Amazon GameLift Servers 启动和运行游戏的步骤。Amazon GameLift Servers 是一项完全托管的服务，游戏开发人员可以使用它来管理和扩展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：
 - 使用您的本地工作站作为主机，测试游戏与 Amazon GameLift Servers 的集成。此工作流程有助于为本地计算机设置 Amazon GameLift Servers 实例集、启动游戏服务器和客户端的实例、通过 Amazon GameLift Servers 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方案：部署到单个实例集，部署到多个区域的一组竞价型实例集，或使用 FlexMatch 对战构建器部署到一组实例集。每种部署场景的解决方案都包括 Amazon GameLift Servers 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 Servers 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，包括提供唯一玩家的基本 Amazon Cognito 用户池 IDs、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 Servers 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 Anywhere 和 Amazon EC2 队列的部署方案，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 Servers 资源时使用这些模板。这些模板包含在 Amazon GameLift Servers 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：

- [Amazon GameLift Servers 虚幻引擎插件](#)，《Amazon GameLift Servers 开发人员指南》
- [从以下地址下载插件 GitHub](#)
- [关于 Amazon GameLift Servers 托管](#)
- [Amazon GameLift Servers 论坛](#)

2023 年 8 月 17 日：Amazon GameLift Servers 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

现在，Amazon GameLift Servers 您可以使用带有 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 Servers，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 处理器](#)：了解基于 Graviton 的 EC2 实例的好处和实际用途。
- [Getting started with Graviton](#)：大致了解基于 Graviton 的实例，并深入了解应用程序如何根据其操作系统、语言和运行时在这些实例上运行。

 Note

Graviton Arm 实例需要一个针对 Linux AMI 的服务器生成包。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：Amazon GameLift Servers 发布了服务器 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 Servers 服务器 SDK 提供了 C++、C# 和 Unreal 插件的更新，以及与 Unity 游戏引擎配合使用的新插件。游戏开发人员将 Amazon GameLift Servers 服务器 SDK 集成到他们部署在 Amazon GameLift Servers 上托管的游戏服务器中。

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 [Amazon GameLift Servers 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 Servers 托管的 EC2 实例集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 Servers Anywhere 实例集上的游戏服务器必须使用服务器参数调用 InitSDK()。
 - [C++、C#、Unreal] 服务器软件开发工具包调用改进了错误消息。
 - [C++ SDK] 为了缩短服务器 SDK 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 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 操作](#)
 - [适用于 Amazon GameLift Servers 的 C++ 服务器 SDK 5.x – 操作](#)
 - [适用于 Amazon GameLift Servers 的 C++ \(Unreal \) 服务器 SDK 5.x – 操作](#)
 - 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 Servers 集成到 Unity 项目中](#)
 - [将 Amazon GameLift Servers 集成到 Unreal Engine 项目中](#)
- 其他文档更新
 - 修订了 Amazon GameLift Servers 服务 API 操作文档 [GetInstanceAccess](#)，[GetComputeAccess](#) 并根据所使用的 Amazon GameLift Servers 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：Amazon GameLift Servers 增加了实例集硬件指标

现在，您可以跟踪 Amazon GameLift Servers 托管的 EC2 实例集的硬件性能指标。指标包括 CPU 利用率、网络流量和磁盘 read/write 活动的 EC2 实例指标。对于 Amazon GameLift Servers，这些指标描述了实例集所在区域的所有活动实例。您可以使用中的 Amazon CloudWatch 控制面板查看这些舰队硬件指标 AWS 管理控制台。您还可以在 Amazon GameLift Servers 控制台的实例集详情中查看它们。

了解更多：

- [Amazon GameLift Servers 使用 Amazon 进行监控 CloudWatch \(实例集指标 \)](#)，《Amazon GameLift Servers 开发人员指南》

2023 年 6 月 29 日：Amazon GameLift Servers 推出了对 Amazon Linux 2023 的支持

更新了 SDK 版本：S AWS DK 1.11.111

Amazon GameLift Servers 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 在 AL2 包括安全性在内的基础上提供了多项改进。除中国地区外，该操作系统在所有 AWS 区域 地区均可用。

2023 年 12 月，当亚马逊 Linux (AL1) 的支持终止时，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。对亚马逊 Linux 2 的支持将持续到 2026 年 6 月 30 日。

了解更多：

- [Amazon GameLift Servers Linux 服务器常见问题解答](#)
- [比较 Amazon Linux 2 和 Amazon Linux 2023](#)
- Amazon GameLift Servers 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：Amazon GameLift Servers FleetIQ 添加过滤器以排除耗尽实例上的游戏会话位置

更新了 SDK 版本：S AWS DK 1.11.87

如果您使用 Amazon GameLift Servers 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 Servers FleetIQ 在搜索和申领可用的游戏服务器时仅查看活动实例。

了解更多：

- 《Amazon GameLift Servers API 参考》中的 [ClaimGameServer](#)
- 《Amazon GameLift Servers FleetIQ 开发人员指南》中的 [FleetIQ 工作原理](#)

2023 年 5 月 16 日：Amazon GameLift Servers 支持为实例集添加成本分配标签

Amazon GameLift Servers 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。您可以为各个 Amazon GameLift Servers EC2 实例集资源分配成本分配标签，以跟踪您的实例集如何影响总体托管成本。

了解更多：

- [资源成本和利用率工具](#)
- [使用 AWS 成本分配标签](#)，《AWS Billing 用户指南》

2023 年 4 月 20 日：Amazon GameLift Servers 推出了对 Windows Server 2016 的支持

更新了 SDK 版本：S AWS DK 1.11.63

Amazon GameLift Servers 客户现在可以使用 Windows Server 2016 操作系统来托管其游戏服务器。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 Servers SDK 或 CLI 命令创建游戏服务器生成包时，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 Servers 控制台创建游戏服务器生成包时，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：

- Amazon GameLift Servers API 参考链接：
 - [CLI 命令 `upload-build`](#)
 - [CLI 命令 `create-build`](#)
 - [AWS SDK 操作 `CreateBuild`](#)
- [适用于 Windows 2012 的 Amazon GameLift Servers 常见问题解答](#)

2023 年 4 月 13 日：Amazon GameLift Servers 推出了适用于 Unreal 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

适用于 Unreal Engine 的 Amazon GameLift Servers 轻量级插件的最新版本现在基于 Amazon GameLift Servers 服务器 SDK 5.x。要开始将 Unreal Engine 环境与 Amazon GameLift Servers 集成，请参阅以下链接。

了解更多：

- [将 Amazon GameLift Servers 集成到 Unreal Engine 项目中](#)
- [借助服务器 SDK 将 Amazon GameLift Servers 添加到游戏服务器](#)
- [适用于 Amazon GameLift Servers 的 C++ 服务器 SDK 5.x – 操作](#)

2023 年 3 月 14 日：Amazon GameLift Servers 推出了全新控制台体验

新的 Amazon GameLift Servers 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 Servers 资源之间进行导航。
- Amazon GameLift Servers 登录页面 – 新的登录页面提供了实用文档的链接，显示了 Amazon GameLift Servers 的高级概述，并通过文档链接、常见问题和 AWS re:Post 提供支持。
- 改进了 Amazon CloudWatch Amazon GameLift Servers 指标 — Amazon GameLift Servers 控制台和您的控制 CloudWatch 面板现在均提供指标。此更新还包括性能、利用率和玩家会话的新指标。

了解更多：

- [使用 Amazon GameLift Servers 管理游戏托管资源](#)
- [构建 FlexMatch 对战构建器](#)

2023 年 2 月 14 日：Amazon GameLift Servers 现支持 Amazon SNS 主题的服务器端加密

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 对战事件](#)
- [静态加密](#)

2023 年 2 月 9 日：Amazon GameLift Servers 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 Amazon GameLift Servers 服务器 SDK 5.0.0。Unity 不支持 .NET 6。

了解更多：

- 在 [Amazon GameLift Servers 入门](#) 中下载最新版本的 Amazon GameLift Servers 服务器 SDK
- [适用于 Amazon GameLift Servers 的 C# 服务器 SDK 5.x – 操作](#)

2023 年 1 月 31 日：Amazon GameLift Servers 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 [Amazon GameLift Servers 入门](#) 中下载最新版本的 Amazon GameLift Servers 服务器 SDK
- [适用于 Amazon GameLift Servers 的 Go 服务器 SDK – 操作](#)

2022 年 12 月 1 日：Amazon GameLift Servers 推出了 Amazon GameLift Servers Anywhere 和 Amazon GameLift Servers 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

Amazon GameLift Servers Anywhere 使用您的游戏服务器资源托管 Amazon GameLift Servers 游戏服务器。您可以使用 Amazon GameLift Servers Anywhere 将自己的计算资源与 Amazon GameLift Servers 托管的 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 Servers Anywhere 来迭代测试您的游戏服务器，而无需在每次迭代时将生成包上传到 Amazon GameLift Servers。

要点：

- 全新 Amazon GameLift Servers Anywhere 实例集和计算类型
- 注册 Amazon GameLift Servers Anywhere 计算资源
- 改进了测试迭代周期

Amazon GameLift Servers 服务器 SDK 5.0.0 引入了对现有服务器 SDK 的改进和一种新的资源类型，即计算。服务器 SDK 5.0.0 支持 Amazon GameLift Servers 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 Servers 服务器 SDK 5.x](#)
- [地理位置](#)
- [为托管式实例集选择计算资源](#)
- [创建 Amazon GameLift Servers Anywhere 实例集](#)

2022 年 8 月 25 日：Amazon GameLift Servers 推出了对 Local Zones 的支持

更新了 SDK 版本：S AWS DK 1.9.333

Amazon GameLift Servers 现已在美国的八个 Local Zones 可用，因此您可以将实例集部署到离玩家更近的地方。通过将 Local Zones 添加到实例集中，您可以将所有托管的 Amazon GameLift Servers 功能与 Local Zones 一起使用。

Local Zones 将 AWS 资源和服务扩展到云边缘，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：

- [Amazon GameLift Servers Local Zones](#)
- [地理位置](#)
- [创建 Amazon GameLift Servers 托管 EC2 舰队](#)

2022 年 6 月 28 日：Amazon GameLift Servers 推出了全新的可选控制台体验

新的 Amazon GameLift Servers 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 Servers 资源之间进行导航。

- Amazon GameLift Servers 登录页面 – 新的登录页面提供了实用文档的链接，显示了 Amazon GameLift Servers 的高级概述，并通过文档链接、常见问题和 AWS re:Post 提供支持。
- 改进了 Amazon CloudWatch Amazon GameLift Servers 指标 — Amazon GameLift Servers 控制台和您的控制 CloudWatch 面板现在均提供指标。此更新还包括性能、利用率和玩家会话的新指标。

了解更多：

- [使用 Amazon GameLift Servers 管理游戏托管资源](#)
- [构建 FlexMatch 对战构建器](#)

2022 年 2 月 15 日：FlexMatch 增加了复合规则和其他改进

FlexMatch 用户现在可以访问以下特征：

- 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 Servers FlexMatch 开发人员指南》](#)：
 - [FlexMatch 规则类型](#)
 - [FlexMatch 属性表达式](#)
- [AttributeValue: SL](#)

2021 年 10 月 28 日：Amazon GameLift Servers 增加了对亚太地区（大阪）地区多区域队列的支持；Amazon GameLift Servers FleetIQ 增加了对 Graviton2 处理器的 AWS 支持

更新了 SDK 版本：S AWS DK [1.9.133](#)

Amazon GameLift Servers 现已在亚太地区（大阪）区域推出。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成本提高性能。

要点：

- Amazon GameLift Servers 现已在亚太地区（大阪）区域推出。
- 现在可以将 Amazon GameLift Servers 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 Servers 多区域实例集](#)
- [CreateGameServerGroup](#)
- [AWS 引力子处理器](#)

2021 年 9 月 20 日：Amazon GameLift Servers 发布了适用于 Unity 的插件

适用于 Unity 版本 1.0.0 的 Amazon GameLift Servers 插件包含库和原生用户界面，可以更轻松地访问 Amazon GameLift Servers 资源并将 Amazon GameLift Servers 集成到 Unity 游戏中。您可以使用适用于 Unity 的 Amazon GameLift Servers 插件来访问 Amazon GameLift Servers APIs 和部署常见游戏场景的 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 GameLift Servers Local 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 Amazon GameLift Servers 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 Spot Fleets，以及 FlexMatch

了解更多：

- [将游戏与适用于 Unity 的 Amazon GameLift Servers 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：Amazon GameLift Servers 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 SDK 更新，您现在可以将 IL2 CPP 集成到使用 RTS 客户端 SDK 的移动应用程序中，并遵循框架的最佳实践。您现在也可以为 Unreal 版本 4.26 构建 Amazon GameLift Servers 服务器 SDK。此更新包含与 Windows 或 Linux 游戏服务器集成的组件，包括 C++ 和 C# 版本的 Amazon GameLift Servers 服务器 SDK、Amazon GameLift Servers Local 以及 Unreal Engine 插件。

要点：

- 在 RTS 客户端 SDK 中增加了对 IL2 CPP 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 Servers 开发人员指南](#)》：
 - [DescribePlayerSessions\(\)](#)

- [GetInstanceCertificate\(\)](#)

2021 年 3 月 23 日 : Amazon GameLift Servers 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 Servers 开发人员指南》 :
 - [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](#)
- [Amazon GameLift Servers 论坛](#)

2021 年 3 月 16 日 : Amazon GameLift Servers 增加了多区域实例集, 六个新区域

更新了 SDK 版本 : S AWS DK [1.8.163](#)

Amazon GameLift Servers托管主机现已在 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)。

借助全新的 Amazon GameLift Servers 多位置实例集特征, 您现在可以设置一个实例集, 将游戏服务器托管在 20 个 Amazon GameLift Servers 支持的区域 (北京区域除外) 中的任意一个或全部区域。此特征旨在显著减少在全球范围内设置和维护 Amazon GameLift Servers 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-2 (法兰克福)、eu-central-1 (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 GameLift Servers 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 Servers 定价](#)。) CloudFormation 不久将提供对多地点舰队的支持。

Note

中国区域不提供多位置实例集。位于中国区域的 Amazon GameLift Servers 资源不能与其他 Amazon GameLift Servers 区域中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 GameLift Servers 将相同类型和配置的实例（包括生成包和运行时配置）部署到实例集的主区域和所有添加的位置。
- 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 Servers 控制台中获取托管指标，按实例集内所有位置汇总或按每个实例集位置细分。

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](#)、[DescribeFleetLocations](#)
 - 更新了舰队运营，增加了新的多地点支持：[CreateFleet](#)、[DescribeFleetCapacity](#)、[UpdateFleetCapacity](#)、[DescribeFleetInstances](#)、[DescribeInstances](#)、[StopFleetActions](#)、[StartFleetActions](#)
 - 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](#)、[SearchGameSessions](#)
- [《Amazon GameLift Servers 开发人员指南》](#)：

- [Amazon GameLift Servers 服务位置](#) (已更新)
- [托管资源自定义](#) (新)

[利用 Amazon GameLift Servers 扩展游戏托管容量](#) (已更新)

- [自定义游戏会话队列](#) (新)
- [Amazon GameLift Servers 控制台中的实例集详细信息](#) (已更新)
- [Amazon GameLift Servers 论坛](#)

2021 年 2 月 9 日 : Amazon GameLift Servers 延长了对 AMD 实例、独立版 FlexMatch 的支持

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 Servers 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.24xlarge
 - m5a.large、m5a.xlarge、m5a.2xlarge、m5a.4xlarge、m5a.8xlarge、m5a.12xlarge、m5a.16xlarge、m5a.24xlarge
 - r5a.large、r5a.xlarge、r5a.2xlarge、r5a.4xlarge、r5a.8xlarge、r5a.12xlarge、r5a.16xlarge、r5a.24xlarge

注意 : 的 AMD 实例 FleetIQ 目前不适用于中国 (北京 AWS) 区域。请参阅中国 [特征可用性和实施差异](#)。

- Amazon GameLift Servers 托管的游戏托管资源现在支持中国 (北京) 区域的 AMD 实例 , 由 Sinnet 运营。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、m5a.24xlarge
 - r5a.large、r5a.xlarge、r5a.2xlarge、r5a.4xlarge、r5a.8xlarge、r5a.12xlarge、r5a.16xlarge、r5a.24xlarge
- Amazon GameLift Servers FlexMatch 现在可以作为中国 (北京) 区域的独立对战解决方案使用 , 由 Sinnet 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 托管主机或非托管 Amazon GameLift Servers Amazon GameLift Servers 管解决方案的更多信息 , 请参阅 《[Amazon GameLift Servers FlexMatch 开发人员指南](#)》。
- 在为 Amazon GameLift Servers FlexMatch 设置事件通知时 , 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](#) , Amazon GameLift Servers API 参考
- 《Amazon GameLift Servers FlexMatch 开发者指南》中的 “[设置 FlexMatch 事件通知](#)”
- [介绍 Amazon SNS FIFO — First-in-first-out Pub/Sub 消息](#)、新闻博客 AWS

2020 年 12 月 22 日 : Amazon GameLift Servers 服务器 SDK 支持 Unreal Engine 4.25 和 Unity 2020

更新了 SDK 版本 : Amazon GameLift Servers 服务器 SDK 4.0.2 , Unreal 插件版本 3.3.3

最新版本的 Amazon GameLift Servers 服务器 SDK 包含以下更新组件 :

- 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 [Amazon GameLift Servers 入门](#) 中下载最新版本的 Amazon GameLift Servers 服务器 SDK。

2020 年 11 月 24 日 : Amazon GameLift Servers FlexMatch 现已适用于在任何地方托管的游戏

更新了 SDK 版本 : S AWS DK [1.8.95](#)

Amazon GameLift Servers FlexMatch 是一项针对多人游戏的可自定义对战服务。最初是为托管 Amazon GameLift Servers 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 Servers FleetIQ 在 Amazon EC2 上托管的游戏现在可以通过 FlexMatch 实现对战。

FlexMatch 提供了强大的对战算法和规则语言 , 可让您自由地自定义对战流程 , 以便根据关键玩家特征和报告的延迟将玩家匹配在一起。此外 , FlexMatch 还提供对战请求工作流程 , 支持多方玩家、玩家接受和匹配回填等特征。当您将 FlexMatch 与 Amazon GameLift Servers 托管式托管资源或 Amazon GameLift Servers Realtime 一起使用时 , 对战构建器会自动使用 Amazon GameLift Servers 来查找托管资源并为新形成的对战开始新的游戏会话。将 FlexMatch 用作独立服务时 , 对战构建器会将对战结果传回您的游戏 , 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 Amazon GameLift Servers 服务 API 的一部分 , 该服务包含在 AWS SDK 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 指示对战构建器是与 Amazon GameLift Servers 托管式托管一起使用还是作为独立的对战。
- 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 Amazon GameLift Servers 上线

更新了 SDK 版本：S AWS DK [1.8.95](#)

Amazon GameLift Servers 支持的 Amazon EC2 实例类型列表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，频率最高可达 3.3 GHz。AMD 实例兼容 x86；当前在 Amazon GameLift Servers 上运行的游戏无需更改即可部署到 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.24xlarge
- m5a.large、m5a.xlarge、m5a.2xlarge、m5a.4xlarge、m5a.8xlarge、m5a.12xlarge、m5a.16xlarge、m5a.24xlarge
- r5a.large、r5a.xlarge、r5a.2xlarge、r5a.4xlarge、r5a.8xlarge、r5a.12xlarge、r5a.16xlarge、r5a.24xlarge

了解更多：

- [Amazon 游戏科技博客](#)
- [Amazon GameLift Servers 实例定价](#)
- [采用 AMD EPYC 处理器的 Amazon EC2 实例](#)
- [Amazon GameLift Servers 论坛](#)

2020 年 11 月 11 日：Amazon GameLift Servers 服务器 SDK 版本更新

更新了 SDK 版本：Amazon GameLift Servers 服务器 SDK 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 [Amazon GameLift Servers 入门](#) 中下载最新版本的 Amazon GameLift Servers 服务器 SDK。

2020 年 11 月 5 日：新的 FlexMatch 算法自定义

FlexMatch 用户现在可以调整对战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。没有任何变化 Amazon GameLift Servers SDKs。

- 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：Amazon GameLift Servers 更新服务器 SDK

更新了 SDK 版本：Amazon GameLift Servers 服务器 SDK 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 [FProcess](#) 参数 添加了回调操作 OnUpdateGameSession 以支持对战回填。

在 [Amazon GameLift Servers 入门](#) 中下载最新版本的 Amazon GameLift Servers 服务器 SDK。

2020 年 8 月 27 日：Amazon GameLift Servers FleetIQ 用于在 Amazon EC2 上托管游戏（正式上市）

更新了 SDK 版本：S AWS DK [1.8.36](#)

在 Amazon EC2 上托管低成本、基于云的游戏的 Amazon GameLift Servers FleetIQ 解决方案现已正式上市。Amazon GameLift Servers FleetIQ 通过优化游戏托管的可行性，让开发人员能够直接在 Amazon EC2 竞价型实例上托管游戏服务器。游戏开发人员可以使用 Amazon GameLift Servers FleetIQ 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 Servers FleetIQ 解决方案的以下更新：

- 新的 API 操作 `DescribeGameServerInstances` 返回 Amazon GameLift Servers FleetIQ 游戏服务器组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：Amazon GameLift Servers 更新了适用于 Unity 和 Unreal Engine 的服务器 SDK

更新了 SDK 版本：Amazon GameLift Servers 服务器 SDK 4.0.0，Amazon GameLift Servers Local 1.0.5

最新版本的 Amazon GameLift Servers Server SDK 包含以下更新组件：

- 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。
- 更新了 Amazon GameLift Servers Local 版本 1.0.5，以测试使用 C# 服务器 SDK 版本 4.0.0 的集成。

在 [Amazon GameLift Servers 入门](#) 中下载最新版本的 Amazon GameLift Servers 服务器 SDK。

2020 年 4 月 2 日：推出 Amazon GameLift Servers FleetIQ 用于在 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 Servers FleetIQ 功能优化了将低成本 Spot 实例与游戏托管一起使用的可行性。此功能现在扩展到希望直接（而不是通过托管 Amazon GameLift Servers 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 上的@@ [博客文章](#) Amazon GameLift Servers FleetIQ

2019 年 12 月 19 日：改进了资源的 AWS Amazon GameLift Servers 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 Servers 资源。特别是，所有关键 Amazon GameLift Servers 资源（生成包、脚本、实例集、游戏会话队列、对战配置和对战规则集）现在都分配了 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 Servers 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 Amazon GameLift Servers 资源的标签时，请使用 Amazon GameLift Servers API 操作 `TagResource()`、`UntagResource()` 和 `ListTagsForResource()`。

了解更多：

- 《Amazon GameLift Servers API 参考》中的 [TagResource](#)
- 《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：中国（北京）区域的新 CloudFormation 模板和更新

更新了 SDK 版本：S AWS DK [1.7.210](#)

CloudFormation 的模板 Amazon GameLift Servers

Amazon GameLift Servers 现在可以通过创建和管理资源 CloudFormation。现有的 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。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 Servers 资源类型参考](#)
- 《Amazon GameLift Servers 开发人员指南》中的 [使用管理 Amazon GameLift Servers 托管资源 CloudFormation](#)

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。