AWS 白皮书

游戏行业视角

游戏行业视角: AWS 白皮书

Table of Contents

摘要和简介 i
镜头可用性 1
设计原则 2
场景 3
游戏托管可实现实时同步游戏 3
游戏服务器进程 4
基于会话的游戏服务器托管和无服务器后端 5
适用于低延迟游戏的多区域和混合架构 6
游戏后端 8
基于容器的游戏后端架构 8
基于无服务器的游戏后端架构 10
云游戏开发 (CGD) 13
云端游戏开发：CI/CD 13
云游戏开发：工作站 15
Gaming Analytics Pipeline 16
定义 19
游戏系统 20
游戏服务器 20
游戏客户端 22
消息收发 22
直播游戏操作（Live Ops） 23
卓越运营 24
设计原则 24
直播操作 25
GAMEOPS01-BP01 使用游戏目标和业务绩效指标来制定实时运营策略 25
账户结构 26
GAMEOPS02-BP01 采用多账户策略，将不同的游戏和应用程序隔离到自己的账户中 26
GAMEOPS02-BP02 使用资源标签组织基础架构资源 29
游戏部署 30
GAMEOPS03-BP01 在游戏中重复使用现有核心游戏系统和基础架构之前，请对其进行验证和测试 30
GAMEOPS03-BP02 在每个版本发布之前进行性能工程（或者至少对于主要版本） 31
GAMEOPS03-BP03 尽早并经常进行负载测试 32
GAMEOPS03-BP04 采用可最大限度地减少对玩家影响的部署策略 33

GAMEOPS03-BP05 需要预扩展基础架构才能支持峰值需求 36
健康监测 37
GAMESOPS04-BP01 控制游戏以检测和监视影响玩家的问题 38
负载测试 39
GAMEOPS05-BP01 选择合适的阶段、架构和负载测试框架来实现您的目标 39
持续优化 41
GAMEOPS06-BP01 监控关键游戏指标以识别玩家的趋势和模式，并使用这些信息来改进游戏 42
GAMEOPS06-BP02 随着游戏规则的变化更新和调整负载测试方法 43
资源 44
文档和博客 44
合作伙伴解决方案 45
白皮书 45
视频 45
培训材料 46
安全性 47
设计原则 48
安全基础知识 48
GAMESEC01-BP01 使用角色和联合访问权限（而不是账户根用户）对您的 AWS 环境执行操作 49
GAMESEC01-BP02 AWS Control Tower 用于在上快速设置多账户环境 AWS 49
GAMESEC01-BP03 使用针对特定工作职能量身定制的最低权限角色策略 50
GAMESEC01-BP04 使用角色和联合访问策略以及账户级别访问策略来授予对 AWS 资源的访问权限 51
GAMESEC01-BP05 使用中央身份提供商 52
持续的安全 53
GAMESEC02-BP01 使用随时可部署的模板进行标准安全实践 53
GAMESEC02-BP02 发生安全事件时使用自动补救技术 54
Identity and access management 55
GAMESEC03-BP01 确定识别和控制玩家对游戏环境和资源的访问权限的方法 56
GAMESEC03-BP02 对发送到游戏后端服务的请求进行身份验证 57
GAMESEC03-BP03 使用您的游戏后端服务验证玩家加入多人游戏的请求 58
GAMESEC03-BP04 要求使用强密码，对玩家用户账户实施严格的安全政策 59
GAMESEC03-BP05 为玩家提供在账户上设置多重身份验证 (MFA) 的选项 60
访问控制 60
GAMESEC04-BP01 限制授权客户和用户访问可下载内容 61

GAMESEC04-BP02 限制源站对授权内容分发网络的访问权限 (CDNs) 63
GAMESEC04-BP03 实施地理限制以限制未经授权的访问 63
GAMESEC04-BP04 使用数字版权管理 (DRM) 解决方案限制对内容的访问 64
检测 65
GAMESEC05-BP01 实施全面的数据收集策略来监控玩家行为 65
GAMESEC05-BP02 收集、存储和分析玩家使用日志，以检测不当行为 66
基础设施保护 67
GAMESEC06-BP01 使用工具检测和应对基础设施面临的威胁 67
GAMESEC06-BP02 使用人工智能和机器学习工具自动执行基础架构保护策略的各个方面 68
GAMESEC06-BP03 利用来自系统级日志的见解来持续改进您的基础架构保护策略 69
事件响应 70
GAMESEC07-BP01 实施事件响应计划以处理不良行为者和虐待行为 71
GAMESEC07-BP02 封禁与不良行为者相关的账户 71
应用程序安全性 72
GAMESEC08-BP01 在 CI/CD 管道的每个阶段都应用安全性 72
实现安全自动化 73
GAMESEC09-BP01 集成工具和自动化以缩短安全审查的平均时间 74
威胁建模 74
GAMESEC10-BP01 确定在整个应用程序开发生命周期中何时以及如何完成威胁建模练习 75
资源 76
可靠性 78
设计原则 78
基本原理 78
工作负载架构 79
GAMEREL01-BP01 跨多个可用区和区域分配游戏基础设施以提高弹性 79
变更管理 80
GAMEREL02-BP01 实施包含活跃玩家游戏会话状态的扩展策略 81
GAMEREL02-BP02 Support 支持在游戏中使用多种 EC2 实例类型 82
故障管理 82
GAMEREL03-BP01 监控游戏服务器中断情况，并使用数据改进托管架构，以实现可靠性目标 83
GAMEREL03-BP02 实现游戏功能的松散耦合以处理故障，同时最大限度地减少对玩家体验的影响 84
GAMEREL03-BP03 监控一段时间内的基础设施事件，以衡量对玩家行为的影响 85
资源 86
性能效率 88

设计原则 88
架构选择 89
GAMEPERF01-BP01 评估游戏服务器资源要求和可扩展性需求 89
GAMEPERF01-BP02 考虑扩展游戏服务器的运营开销 90
GAMEPERF01-BP03 评估与其他 AWS 服务、开发环境、目标 CPU 架构和功能的集成 91
区域选择 92
GAMEPERF02-BP01 选择靠近玩家的主区域 92
GAMEPERF02-BP02 设计一种方法，支持将对延迟敏感的游戏基础设施放在靠近玩家的地方以提高性能 93
迭代开发 95
GAMEPERF03-BP01 使用 Amazon GameLift Anywhere 和 GameLift测试工具包 95
GAMEPERF03-BP02 测试游戏服务器的性能和可扩展性 97
GAMEPERF03-BP03 优化 GameLift容器的资源利用率 98
计算和硬件 98
GAMEPERF04-BP01 监控游戏服务器进程以检测问题 99
GAMEPERF04-BP02 性能使用模拟和真实的游戏场景测试您的游戏服务器 100
计算选择 100
GAMEPERF05-BP01 跨多种计算类型对游戏性能进行基准测试 101
GAMEPERF05-BP02 将 non-latency-sensitive计算任务移至异步工作流程 102
数据管理 103
GAMEPERF06-BP01 集中日志收集和存储 103
GAMEPERF06-BP02 根据访问模式对游戏数据进行分类和存储 104
GAMEPERF06-BP03 启用高效的日志格式化和批处理 104
GAMEPERF06-BP04 实施日志轮换和保留策略 105
GAMEPERF06-BP05 使用监控和可视化工具 105
联网和内容分发 105
GAMEPERF07-BP01 为您的游戏定义网络延迟阈值 106
GAMEPERF07-BP02 为每种游戏模式和游戏托管区域运行单独的配对服务 106
GAMEPERF07-BP03 定期监控配对表现 106
GAMEPERF07-BP04 定期监控网络性能 107
GAMEPERF07-BP05 使用网络加速技术提高互联网性能 108
流程和文化 109
GAMEPERF08-BP01 通知玩家并将其纳入您的流程 109
GAMEPERF08-BP02 使解决方案选择与工程团队的技能和专业知识保持一致 110
资源 111
成本优化 113

设计原则 114
践行云财务管理 114
支出和使用情况意识 114
GAMECOST01-BP01 实现每个玩家、游戏功能和环境的成本归因 114
GAMECOST01-BP02 发现优化的机会 116
具有成本效益的资源 117
GAMECOST02-BP01 优化通过互联网传输数据的成本 117
GAMECOST02-BP02 优化每个游戏服务器实例上托管的游戏会话数量以优化成本 118
GAMECOST02-BP03 选择适当的计算定价选项以降低成本 119
数据传输成本 121
GAMECOST03-BP01 为用户生成的内容选择适当的存储类型以降低成本 122
GAMECOST03-BP02 优化游戏后端的数据库 123
管理需求和供应资源 124
随着时间的推移进行优化 124
资源 124
可持续性 126
设计原则 126
区域选择 126
符合需求 127
软件和架构 127
数据管理 127
GAMESUS01-BP01 使用适合用户内容、订阅者信息和游戏内购买模式的存储技术 127
GAMESUS01-BP02 使用生命周期策略或 TTL 过期时间删除不必要的游戏用户数据、日志文件或已弃用的资产 129
硬件和服务 130
GAMESUS02-BP01 为适当的计算工作负载选择托管服务 131
GAMESUS02-BP02 调整计算规模，仅在需要的地方部署 GPU 性能 132
资源 133
关键 AWS 服务 133
结论 134
贡献者 135
文档修订 137
AWS 词汇表 138
..... cxxxix

游戏行业视角 — Well-Archit AWS ected 框架

发布日期：2025 年 12 月 9 日 (文档修订)

Well AWS -Architected Framework 可帮助云架构师为其应用程序和工作负载构建安全、高性能、弹性和高效的基础架构。Well-Architected基于六大支柱（卓越运营、安全性、可靠性、性能效率、成本优化和可持续性），为客户 AWS 和合作伙伴提供了一种一致的方法，以评估架构、修复风险和实施可带来商业价值的设计。

从这个角度来看，我们专注于如何在中设计、部署和架构您的游戏工作负载 AWS 云。我们定义组件，探索常见的工作负载场景，并概述设计原则，以帮助您应用 Well-Architected Framework。我们建议您在开始设计架构时，先考虑 Well-Ar chitecte AWS d Framework 白皮书中的最佳实践和问题。本文档为游戏行业客户提供了补充性最佳实践。

根据我们与全球游戏行业开发商和发行商合作的经验，本视角列出了最佳实践，旨在解决在云端构建和运营游戏的独特特征。它提供了有关如何设计和运营环境的指导，以便针对全球玩家需求的波动进行成本优化和扩展。该镜头还为保护游戏基础设施和调整性能提供了指导，以提供积极的玩家体验。

本文档适用于担任技术职务的人员，例如首席技术官 (CTOs)、游戏工作室技术总监、架构师、开发人员和运营团队成员。阅读本文档后，您将了解设计游戏架构时要使用 AWS 的最佳实践和策略。

镜头可用性

定制镜头扩展了所提供的最佳实践指南 AWS Well-Architected Tool。 AWS WA Tool 允许您创建自己的自定义镜头，或者使用其他人创建的、与您共享的镜头。

要开始查看您的游戏工作量，请 AWS Well-Architected Tool 从公开的 WellAr chitecte AWS d 自定义镜头库中下载游戏行业镜头并将其导入。 GitHub

设计原则

Well-A AWS rchitected Framework 确定了以下一般设计原则，以促进在云端为游戏工作负载进行良好的设计：

- 了解玩家的行为和使用模式以发展游戏并帮助保护玩家：为了不断改进游戏并有效管理玩家体验，了解玩家与游戏本身以及与其他玩家的互动方式非常重要。这有助于你了解如何改进游戏、管理成本，以及如何监控和应对对玩家体验构成风险的未经授权的使用活动。
- 使用可简化游戏操作和提高开发速度的技术：优先采用能够提高速度并减少向玩家提供新功能和改进的运营开销的技术。游戏以命中率为导向，玩家需要考虑很多选择，因此，快速行动并适应变化对于游戏的成功至关重要。考虑一下您是否愿意操作自己的软件，还是更愿意采用 AWS 合作伙伴提供的类似托管服务 AWS，或者两者兼而有之。
- 优化架构以改善反映玩家实际体验的指标：随着时间的推移，随着时间的推移调整和演变架构，请考虑这些改进和变化将如何影响玩家体验。游戏工作负载应该能够承受并最大限度地减少故障的影响，从而阻止对游戏玩法的广泛干扰。应将彼此不严重依赖的游戏功能和系统分离，以减少故障的影响并隔离影响玩家的问题。
- 设计基础设施以满足玩家并发峰值并根据需要进行动态扩展：基础设施的设计应可扩展以满足玩家的需求。诸如玩家会话并发性和登录次数之类的指标可用于在系统过载之前抢先扩展。被动的系统利用率指标（例如 CPU 和内存消耗）可用于在系统过载后进行扩展。通过动态扩展基础架构，您可以降低游戏运营成本。
- 实施运行手册以改善游戏运营：应使用操作运行手册来持续管理重复的游戏操作任务。对于常见的游戏运营工作流程，例如调查和回应玩家报告、管理基础设施预扩展活动，为预期的大型活动（例如新赛季发布和游戏内容发布）做好准备，以及处理典型的游戏维护活动，应有运行手册。

场景

在本节中，我们将介绍游戏架构中常见的几种场景。每个场景都包括推动设计的共同特征和示例参考架构图。

场景

- 游戏托管可实现实时同步游戏
- 游戏后端
- 基于无服务器的游戏后端架构
- 云游戏开发 (CGD)
- Gaming Analytics Pipeline

游戏托管可实现实时同步游戏

实时同步游戏允许两个或两个以上的玩家同时参与游戏并进行互动，游戏状态由连接的玩家共享，从而创造尽可能接近实时的体验。同步游戏的示例包括第一人称射击游戏、大型多人在线游戏 (MMOG)、体育和动作游戏，或者必须连接两个或更多玩家才能近乎实时地分享游戏体验的在线游戏。

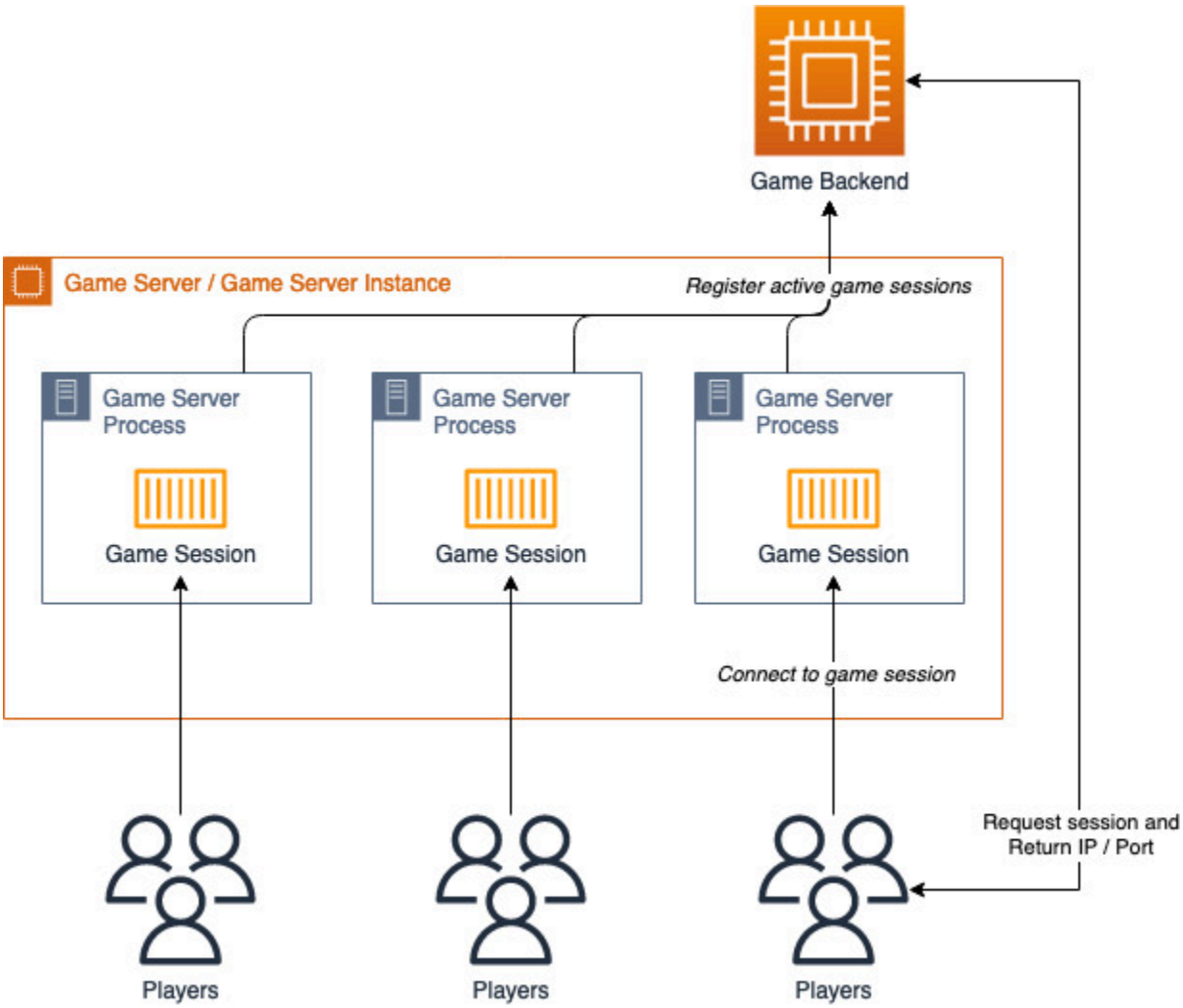
实时同步游戏架构的特点包括：

- 有些游戏可能通过在专用服务器上运行的游戏服务器进程作为游戏会话托管。有些游戏可能使用类似 P2P 的架构，这些架构使用轻量级的 NAT (STUN) 会话遍历实用程序，或者在 NAT (TURN) 服务器周围使用中继遍历。无论涉及哪种类型的服务器，游戏服务器都托管在多个数据中心和 AWS 区域全球范围内。
- 游戏客户端可以通过向游戏后端系统中托管的集中配对服务请求配对，或者从预定义的可用游戏服务器列表中选择匹配项来加入游戏会话。游戏客户端提供了 IP 地址和要连接的端口。
- 许多同步游戏都对延迟敏感，例如第一人称射击游戏和大型多人在线游戏。它们可能包括倒带和时间膨胀等算法，以最大限度地减少延迟影响，但也可能具有预定义的延迟容忍度，经过仔细测量和优化，以减少玩家在高延迟情况下有时可能出现的延迟体验。这种延迟信息是通过检测游戏客户端来确定的，以ping可用的游戏服务器 AWS 区域 来捕获诸如延迟、网络抖动和其他重要的游戏体验指标之类的指标。这些指标被发送到游戏后端系统中的中央指标收集服务，以便实时操作流可以监控游戏生命值。在配对过程中，游戏客户端在请求匹配时将其当前延迟数据作为请求参数之一，而配对服务在选择游戏服务器来托管玩家时，可以将该延迟数据用作变量之一。
- 通常，游戏玩法是通过混合协议进行的（例如游戏服务器使用更快的基于 UDP 的消息传递，以及使用 HTTPS 的配对、身份验证和其他客户端服务器流量）。

- 游戏服务器采用算法和设计来最大限度地减少客户端-服务器流量，例如转换流、增量和数据压缩。
- 游戏服务器经常成为恶意活动的目标，应使用诸如 DDo S 保护解决方案进行保护 AWS Shield Advanced。

游戏服务器进程

下图说明了典型的游戏服务器架构。它描述了游戏服务器实例和托管游戏会话的游戏服务器进程之间的逻辑关系。



逻辑游戏服务器架构

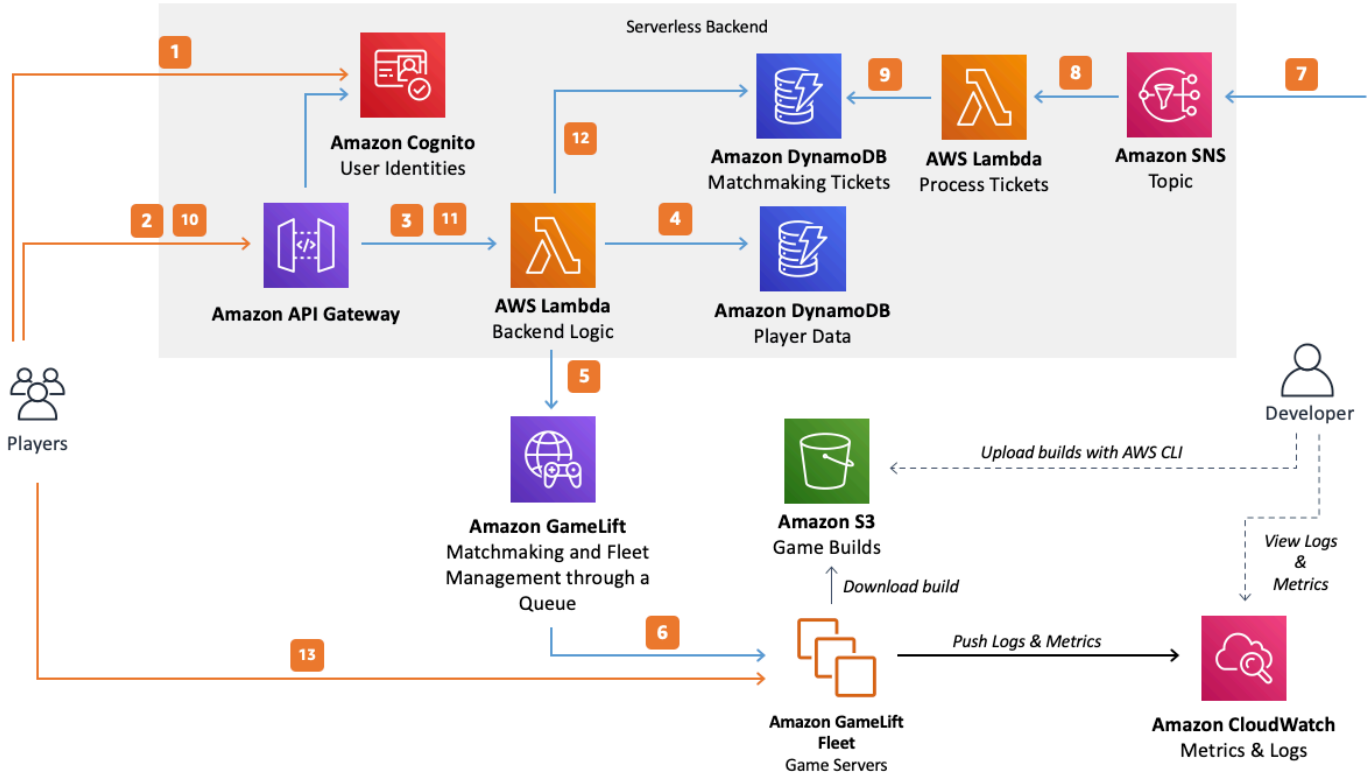
- Amazon EC2 实例用作游戏服务器，也称为游戏服务器实例。游戏服务器托管一个或多个游戏服务器进程，每个进程都在运行您的游戏服务器版本的副本。通常，一个游戏服务器实例上运行多个游戏服务器进程，以高效利用计算资源并降低成本。当游戏会话处于活动状态并准备好主持玩家会话时，其状态会随着游戏后端（通常是配对服务）进行更新，以便可以开始使用它来接待玩家。

- 游戏后端可以为玩家提供托管游戏会话的 IP 地址和服务器端口，以便他们可以连接玩游戏。

基于会话的游戏服务器托管和无服务器后端

在为游戏开发架构时，请考虑您需要的特性和功能以及您准备拥有的运营管理开销水平。为了在易操作性和灵活性之间取得最佳平衡，您可以使用云提供商提供的托管服务来构建游戏。托管服务使您可以控制开发和自定义自己的自定义游戏功能，同时还可以减轻部署和管理基础设施的负担。

托管基于会话的多人游戏需要服务器基础架构来托管游戏服务器进程，以及用于配对和会话管理的可扩展后端。以下参考架构展示了如何使用 Amazon 托 GameLift管主机和无服务器后端来管理基于会话的游戏。



Amazon 托 GameLift 管基于会话的游戏

该图描述了让玩家进入在托 GameLift 管游戏主机上运行的游戏的过程。它包括以下步骤：

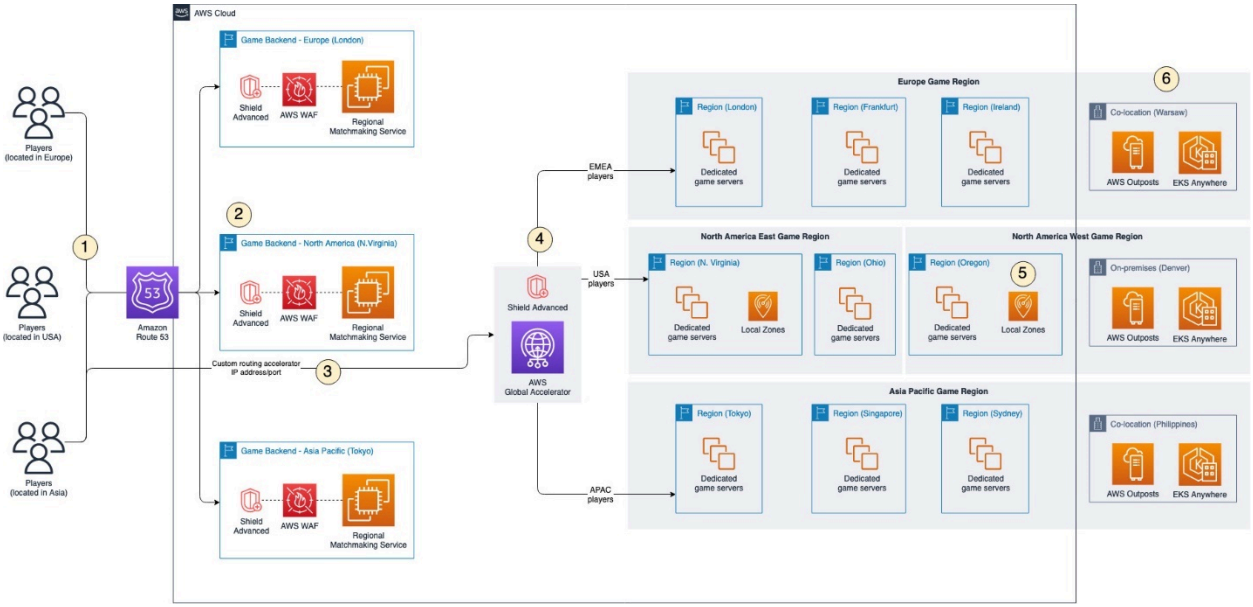
1. 游戏客户端向亚马逊 Cognito 身份池请求亚马逊 Cognito 身份。可以选择将其连接到外部身份提供商。
2. 游戏客户端接收临时访问凭证，并使用亚马逊 Cognito 凭证签署请求，通过 Amazon API Gateway 请求游戏会话。
3. API Gateway 调用一个函数。 AWS Lambda

4. Lambda 函数从亚马逊 DynamoDB 表中请求玩家数据。Amazon Cognito 身份用于安全地请求正确的玩家数据，因为经过身份验证的身份是在请求上下文数据中提供的。
5. Lambda 函数使用正确的玩家数据获取更多信息（例如玩家技能等级），通过 GameLift FlexMatch 配对请求配对。您可以使用基于 JSON 的配置文档定义 FlexMatch 配对配置。游戏客户端可以通过 ping不同区域的服务器端点来生成延迟指标，延迟数据可用于支持基于延迟的配对。
6. 将一组具有适当延迟的合适玩家与某个区域 FlexMatch 匹配后，它会通过 GameLift 队列请求游戏会话放置。队列包含具有一个或多个注册区域位置的舰队。
7. 在车队的某个地点进行会话时，会向 Amazon SNS 主题发送事件通知。
8. Lambda 函数将接收亚马逊 SNS 事件并对其进行处理。
9. 如果 Amazon SNS 消息是一个 MatchmakingSucceeded 事件，则 Lambda 函数会使用服务器端口和 IP 地址将结果写入 DynamoDB。 time-to-live(TTL) 值用于确保在不再需要配对票证时将其从 DynamoDB 中删除。
10. 游戏客户端向 API Gateway 发出签名请求，以在特定时间间隔内检查配对票证的状态。
11. API Gateway 调用 Lambda 函数来检查配对票证状态。
12. Lambda 函数会检查 DynamoDB 以确定票证是否成功。如果成功了，Lambda 函数会将 IP 地址、端口和玩家会话 ID 发回客户端。如果票证失败，Lambda 函数会发送一个响应，声明匹配尚未准备就绪。
13. 游戏客户端使用后端提供的端口和 IP 地址，使用 TCP 或 UDP 连接到游戏服务器。它将玩家会话 ID 发送到游戏服务器，然后游戏服务器使用 Amazon GameLift Server SDK 对其进行验证。

或者，您可以修改上述架构，将 API Gateway WebSockets 与 Amazon 配合使用 GameLift。在这种方法中，游戏客户端和游戏后端服务之间的通信使用WebSocket基于的实现进行。可以使用此实现，以便游戏后端 Lambda 函数通过 WebSocket 而不是实现轮询模型向游戏客户端启动服务器端消息。

适用于低延迟游戏的多区域和混合架构

本节介绍适用于低延迟游戏的多区域和混合架构。



通过网络加速和全球部署游戏服务器来减少延迟

1. 全球可用的游戏中的玩家可以来自任何地方。当玩家请求游戏会话或比赛时，他们的游戏客户端会向注册到 Amazon Route 53 的游戏后端服务发送请求。基于 Route 53 延迟的路由可用于将玩家路由到最近的可用游戏后端。
2. 游戏后端部署在 AWS 区域 最接近玩家群的多个地方。每个游戏后端都包含一个区域配对服务，该服务可以从整个游戏区域中查找游戏会话. 尽管玩家的配对请求由他们附近的区域配对服务机构处理，但如有必要，配对服务能够将玩家路由到另一个游戏区域的游戏会话。此操作可提高弹性和性能。此外，每个游戏后端服务都使用AWS WAF 和 AWS Shield Advanced 来提供第 7 层 Web 过滤和机器人控制，以及针对分发拒绝服务 (DDoS) 攻击的保护。您可以选择多种方式来构建游戏后端服务，例如无服务器、容器、 EC2实例，或者在自己的数据中心托管游戏后端服务。
3. 为了通过减少网络延迟和抖动来改善玩家体验，我们使用 AWS Global Accelerator部署了自定义路由加速器，该加速器使用全球网络自动优化从游戏客户端到游戏服务器的 AWS 流量路由。您可以将自定义路由加速器配置为将全球加速器侦听器端口映射到游戏服务器的 EC2 实例端口。游戏客户端连接到充当代理的全球加速器IP和端口，并确定性地将玩家路由到托管游戏会话的正确游戏服务器IP和端口。
4. 您的游戏包括对玩家友好的逻辑游戏区域，这些区域代表一系列地理位置相邻的游戏服务器托管位置，例如北美或亚太地区。为了减少延迟并扩大地理覆盖范围，可以组合使用不同的游戏服务器托管解决方案来改善玩家体验。尽可能优先使用，因为这些地点功能齐全，容量占用量最大。 AWS 区域
5. 使用 L AWS ocal Zones 将游戏服务器托管在服务不足的玩家所在的地理位置，在这些位置您没有现有托管设施，或者没有托管设施可用。 AWS 区域

6. 部署 AWS Outposts 到您现有的本地数据中心和托管提供商中，使用完全托管的机架和机架式服务器，在每个部署位置创建无缝的控制平面和管理体验。 AWS Outposts 如果您的本地环境中没有现有的服务器容量，也很有用。但是，如果您想要在自己的现有服务器基础设施上运行软件的混合实施，则应使用 Amazon EKS Anywhere，它允许您在自己的基础设施上创建和运行 Kubernetes 集群，并连接到亚马逊 EKS。这为本地的 Kubernetes 集群提供了一致的控制台视图。

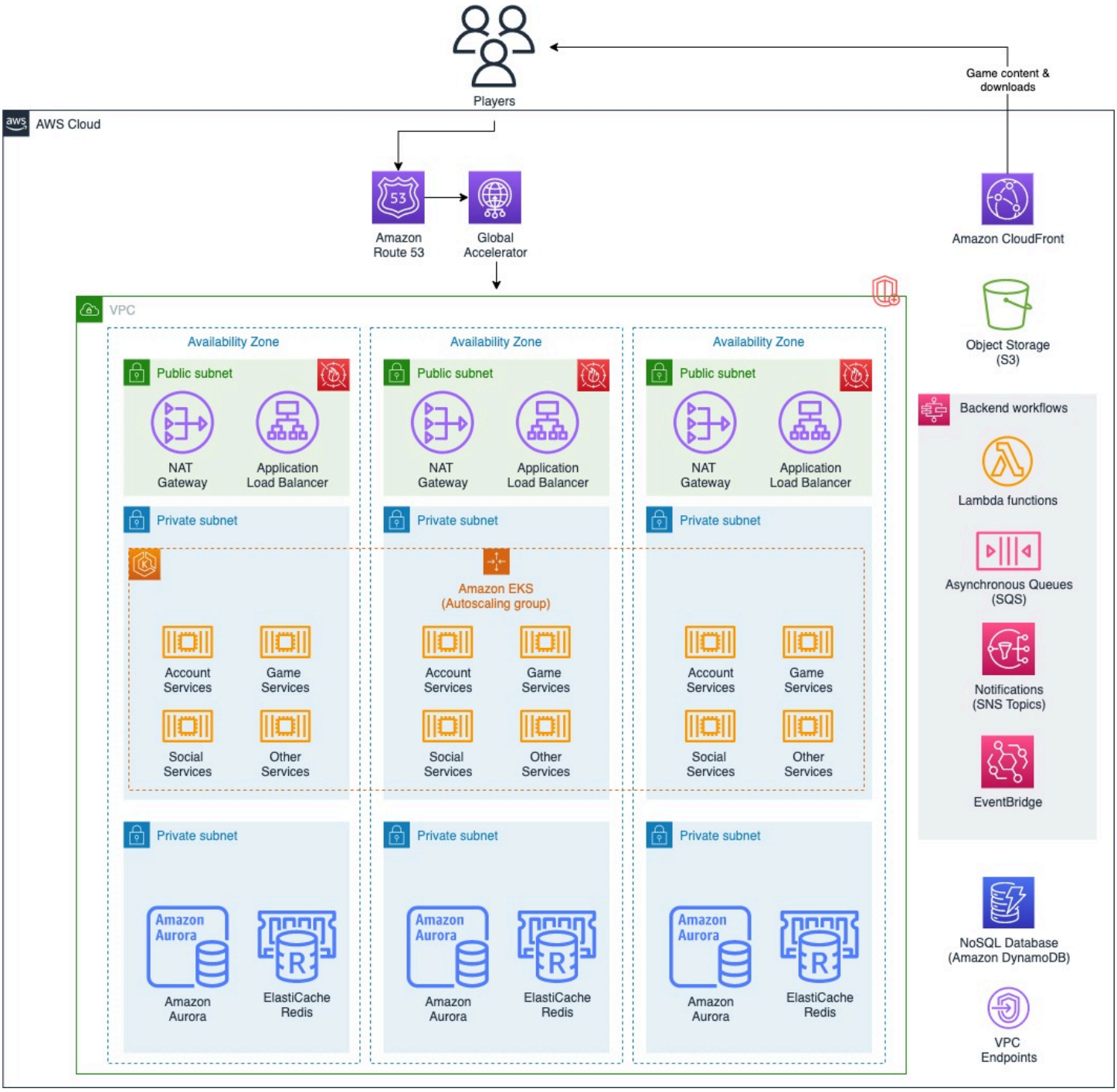
游戏后端

游戏后端用于管理游戏和玩家状态，并将社交和系统级功能集成到游戏中以支持游戏体验。玩家资料管理、物品和库存存储以及统计数据和排行榜都是游戏后端托管的服务示例。

游戏后端通常以 REST 形式构建 APIs ，客户端可以使用 HTTPS 进行访问。但是，其他方法也很常见 WebSockets ，例如为用例提供双向渠道，例如游戏内聊天和在线状态的客户端通知。游戏后端可以使用各种不同的部署架构进行部署，包括使用实例、容器或无服务器架构。

基于容器的游戏后端架构

本节概述了基于容器的游戏后端架构。



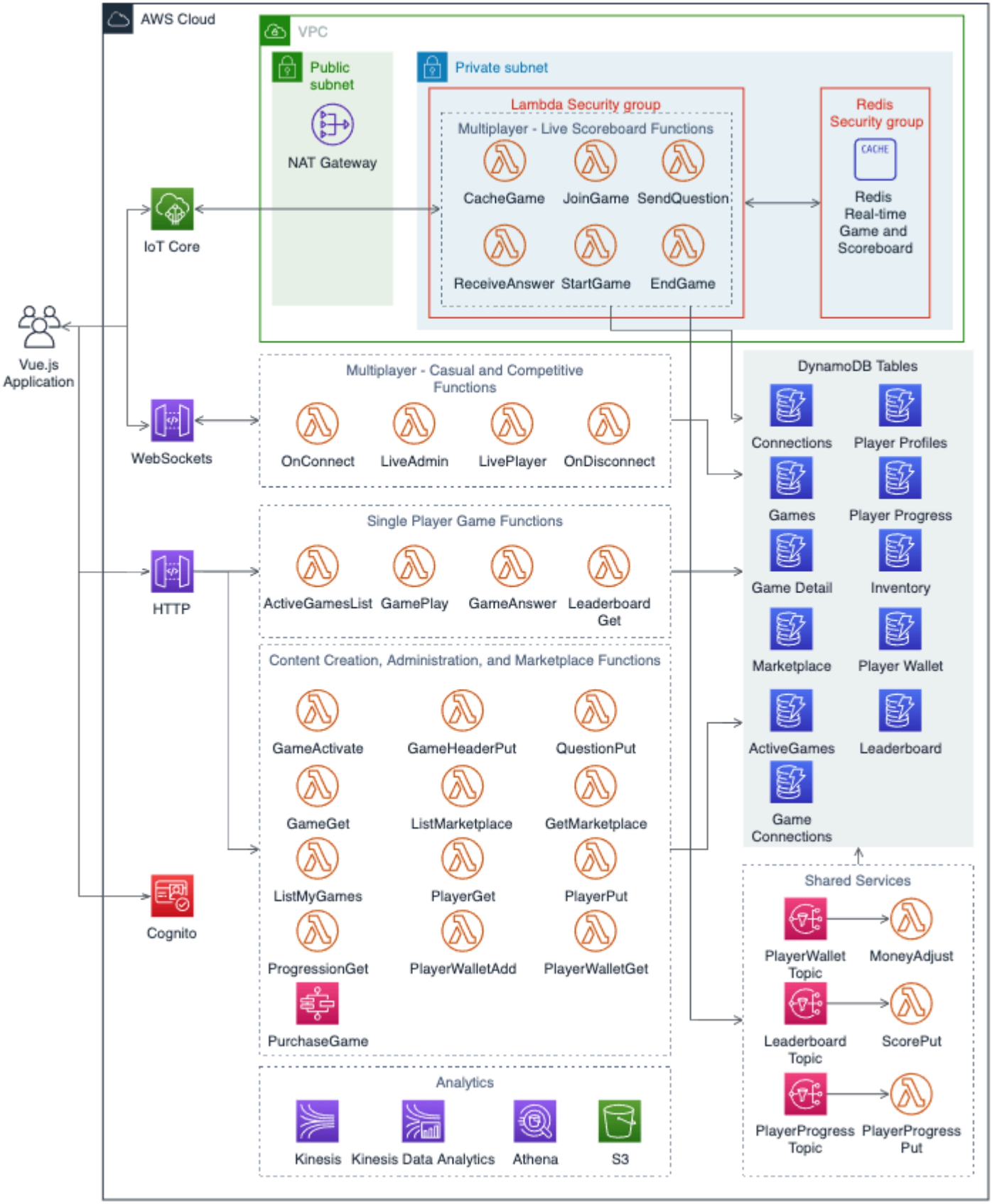
使用容器托管游戏后端

- 玩家使用游戏客户端软件访问游戏，该软件可以通过游戏系统、数字店面或直接从内容交付网络 (CDN)（例如亚马逊）下载分发给他们。 CloudFront CDNs在边缘位置提供缓存，以提高用户下载内容的性能。例如， CloudFront 可用于向玩家分发游戏客户端软件以及游戏资产和其他内容。

- AWS Global Accelerator 提供流量加速和可自定义的控件，用于将流量从玩家游戏客户端路由到您的负载均衡器，以及出于多区域和故障转移目的跨区域路由流量。在 Amazon Route 53 中配置了游戏后端 REST APIs 的自定义域名，以将流量路由到全球加速器终端节点。图中未显示，AWS Shield Advanced 可以为您的加速器和游戏后端提供额外的 DDo S 缓解措施。
- NAT Gateway 和 Application Load Balancer 部署到游戏后端使用的每个可用区的公有子网，以提供该区域的高可用性。AWS WAF 部署在 Application Load Balancer 上，用于提供第 7 层 Web 流量过滤。
- 游戏后端作为一组基于容器的单个微服务托管，这些微服务部署到分布在各个可用区的私有子网中的 Amazon EKS 集群中，以实现弹性。自动扩展会根据资源利用率动态调整服务和群集节点的容量，资源利用率通常与玩家需求相关。集群自动扩缩器会自动调整集群中的节点数量，而水平容器自动扩缩器会自动扩展部署到集群中的 Pod。
- 游戏和玩家数据存储在后端数据库和缓存中，这些数据库和缓存部署到跨可用区的私有子网中，并在主节点和副本节点之间进行复制。Amazon Aurora 是玩家资料、权利和游戏内购买等用例的热门选择，这些用例的查询要求可能更加复杂，并且可以从 MySQL 和 PostgreSQL 的关系数据建模功能中受益。Amazon ElastiCache 对于构建高性能排行榜、 pub/sub 消息传送以及缓存经常访问的数据以减少延迟和数据库负载非常有用。 Amazon DynamoDB 是一种完全托管的 NoSQL 数据存储，非常适合不可预测的访问模式，并且能够扩展到几乎无限的吞吐量，适用于玩家和游戏状态数据、会话数据、库存和物品存储等用例，或者您想要以最小的开销建立全球数据库的用例。
- 应使用异步处理工作流程来执行可在后台完成的工作，例如更新排行榜或发送好友请求。配置您的游戏后端以将此类工作推送到 Amazon SQS 队列中，以便随着游戏的增长进行扩展，或者考虑使用 Amazon SNS 主题在多个使用者应用程序队列之间分配工作以进行并行处理。使用 AWS Lambda 函数以事件驱动的方式执行处理，以降低计算基础设施成本和管理开销。对于持续时间较长或需要通过多个步骤进行任务协调的工作流程，可以考虑使用来协调整个工作流程。AWS Step Functions Amazon EventBridge 可用于启动响应 AWS 服务和自定义应用程序事件的函数。

基于无服务器的游戏后端架构

许多游戏开发者不想管理基础架构，而是更喜欢使用允许他们专注于软件的技术来开发游戏。在这种情况下，建议使用无服务器架构，因为它允许您更快地构建和发布功能，同时减少运营开销。无服务器架构是使用云服务设计的，这些服务可以根据需求动态扩展，而无需设置、管理和扩展服务器。以下参考架构说明了如何使用无服务器架构构建游戏。


AWS Cloud
VPC
Public subnet
NAT Gateway
Private subnet
Lambda Security group
Multiplayer - Live Scoreboard Functions
CacheGame
JoinGame
SendQuestion
ReceiveAnswer
StartGame
EndGame
Redis Security group
CACHE
Redis Real-time Game and Scoreboard
IoT Core
Vue.js Application
WebSockets
HTTP
Cognito
Multiplayer - Casual and Competitive Functions
OnConnect
LiveAdmin
LivePlayer
OnDisconnect
Single Player Game Functions
ActiveGamesList
GamePlay
GameAnswer
Leaderboard Get
Content Creation, Administration, and Marketplace Functions
GameActivate
GameHeaderPut
QuestionPut
GameGet
ListMarketplace
GetMarketplace
ListMyGames
PlayerGet
PlayerPut
ProgressionGet
PlayerWalletAdd
PlayerWalletGet
PurchaseGame
Analytics
Kinesis
Kinesis Data Analytics
Athena
S3
DynamoDB Tables
Connections
Player Profiles
Games
Player Progress
Game Detail
Inventory
Marketplace
Player Wallet
ActiveGames
Leaderboard
Game Connections
Shared Services
PlayerWallet Topic
MoneyAdjust
Leaderboard Topic
ScorePut
PlayerProgress Topic
PlayerProgress Put

基于无服务器的游戏后端参考架构

此参考架构说明了一款基于网络的问答游戏，该游戏提供单人游戏和多人游戏功能。

- 玩家身份验证：玩家使用 Amazon Cognito 进行身份验证，Amazon Cognito 提供安全的身份验证以及用于玩家身份管理的用户目录。
- 游戏逻辑作为无服务器函数：游戏功能和后端业务逻辑作为响应事件而启动的 AWS Lambda 函数运行，这可以降低成本，因为只有在函数运行时才需要付费。Lambda 使您能够灵活地使用自己选择的编程语言将每个游戏功能编写为单独的微服务。例如，如果您有使用 C# 构建 Unity 游戏的经验，则可以选择开发.NET Lambda 函数；或者如果您想在两者中为基于 Web 的游戏编写前端和后端，则可以选择开发 Node.js Lambda 函数。 JavaScript
- 用于存储游戏和玩家数据的 NoSQL 数据存储：使用 DynamoDB 存储玩家和游戏数据，因为它是专门为存储来自微服务的大量数据而设计的。如该架构所示，最佳做法是使用单独的数据存储来满足每个游戏功能的数据存储需求，这样您就可以直接独立监控和管理这些功能。如果您的团队中的功能或服务所有权发生变化，这也会造成分离界限。在此参考架构中，DynamoDB 表用于存储连接状态、游戏详情、玩家进度和排行榜信息等数据。
- 单人游戏玩法：单人游戏功能允许玩家执行诸如选择和玩游戏以及查看排行榜之类的操作。这些功能是作为 RESTful 后端服务实现的，由 Amazon API Gateway HTTP API 托管，该API调用相应的 Lambda 函数来获取和设置 DynamoDB 表中的数据。游戏完成后，后端还会向 Amazon SNS 主题发送通知，这些主题会异步启动 Lambda 函数以存储玩家的进度和统计信息。
- 多人游戏玩法：多人游戏功能要求玩家能够与游戏互动以进行交 point-to-point流，以及广播和接收来自其他联网玩家的更新。 WebSockets 实现适合在轻量级游戏（例如琐事）中进行 point-to-point 交流。玩家可以建立 WebSockets 与 Amazon API Gateway 的连接 WebSockets，Amazon API Gateway 负责管理连接，并且只有在有消息可供玩家发送或接收时才调用 Lambda 函数。对于需要在玩家之间进行 one-to-many通信的用例， AWS IoT Core 支持 WebSockets 通过 MQTT 进行消息传递，允许客户端订阅主题并根据收到的消息进行操作。在此架构中，ov WebSockets er MQTT 用于支持使用案例，例如直播游戏内直播更新和向联网玩家提问。作为替代方案 AWS IoT，您可以选择 Redis Pub/Sub 进行消息传送，或者如果您需要保留消息，则可以选择 Redis Streams。
- 使用支持 VPC 的 Lambda 函数访问私有子网中的资源：配置支持 VPC 的 Lambda 函数以访问您的 VPC 私有子网中的资源，例如 Amazon，它用于缩短实时排行榜 ElastiCache等低延迟数据集的查询时间。

有关更多信息，请参阅自定义游戏后端托管指南 AWS。

云游戏开发 (CGD)

云游戏开发 (CGD) 是指游戏开发生命周期构建、测试和开发游戏所需的基础设施和工具。游戏开发是用户之间的协作开发，在整个开发阶段，基础设施要求经常发生变化。

许多游戏开发者正在拥抱全球分布式和远程开发团队，这需要支持此类开发的技术。游戏开发者可以在其中托管全部或部分环境， AWS 并利用的全球可用性将资源放在 AWS 区域 离用户更近的地方，并通过根据需要扩展计算和存储，更具成本效益地管理他们的开发环境。

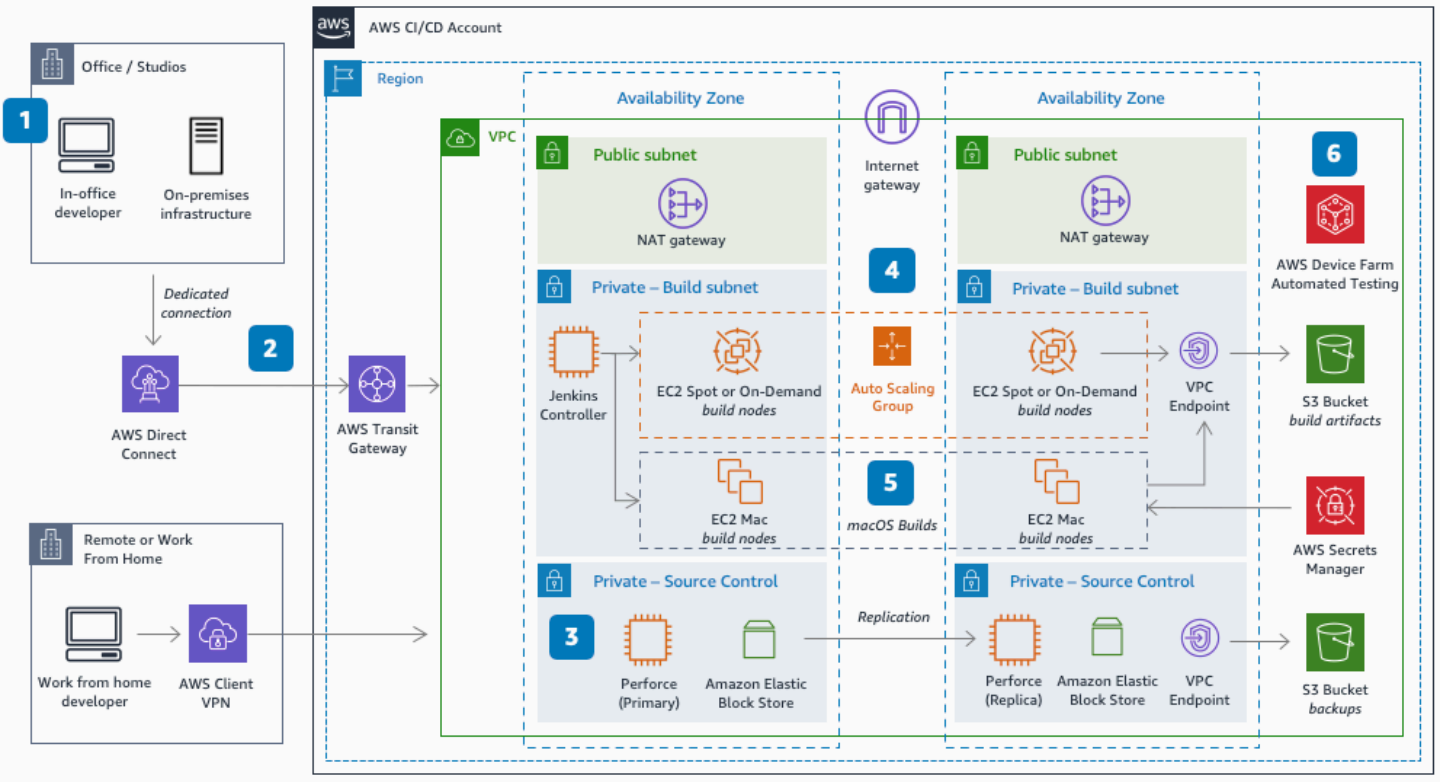
环境可能因游戏开发者的需求而异，但它们通常包括开发者工作站，供美术师、设计师、工程师、质量保证测试人员、承包商和其他人员执行工作。这些环境通常还包括一个由源代码存储库组成的构建场，供用户签入更改，以及用于构建、打包和测试已开发工件 CI/CD 的基础架构。

这些游戏制作架构具有以下特征：

- 用户应该能够通过网络浏览器或本地桌面客户端（例如 Amazon DCV）访问虚拟工作站，该客户端为他们提供了低延迟的流媒体会话，以访问他们在办公室或开发工作室的计算机上工作时可以访问的相同软件和工具。这些虚拟工作站（通常是基于云的服务器）应允许用户完全通过局域网或广域网在云环境中协作和处理项目。当用户不积极使用计算机时，应将他们的工作备份到耐用的云存储中，例如源代码控制存储库或文件系统，例如 Amazon Elastic File System (EFS) 和 Amazon FSx，并应关闭他们的计算机以降低成本。
- 源代码控制存储库（例如 Perforce）应在可用区之间或本地环境之间复制，备份存储在云存储（如 Amazon S3）中，在设计时具有高可用性。例如，基于云的 Perforce 服务器应包括托管在一个可用区中的主提交服务器，并复制到托管在同一区域另一个可用区的备用服务器。
- 游戏开发构建场资源的设计应具有自动扩展功能，以便根据需要配置计算资源，并应使用EC2竞价型实例来减少在扩展构建所需的服务器数量时产生的成本。

云端游戏开发：CI/CD

CI/CD 无论团队规模大小，在开发游戏时，基础设施都非常重要，可以缩短迭代时间，构建可靠性，提高部署效率，更好地控制开发和发布流程，从而为玩家提供高质量的游戏体验。游戏开发 CI/CD 管道通常由高度可用的源代码控制服务器和存储、用于运行构建的计算资源、用于执行自动测试的软件以及来自开发计算机的适当网络连接组成。以下参考架构演示了如何将游戏版本从远程或本地游戏开发环境卸载到， AWS 云 以帮助开发人员迁移或构建新的构建场。



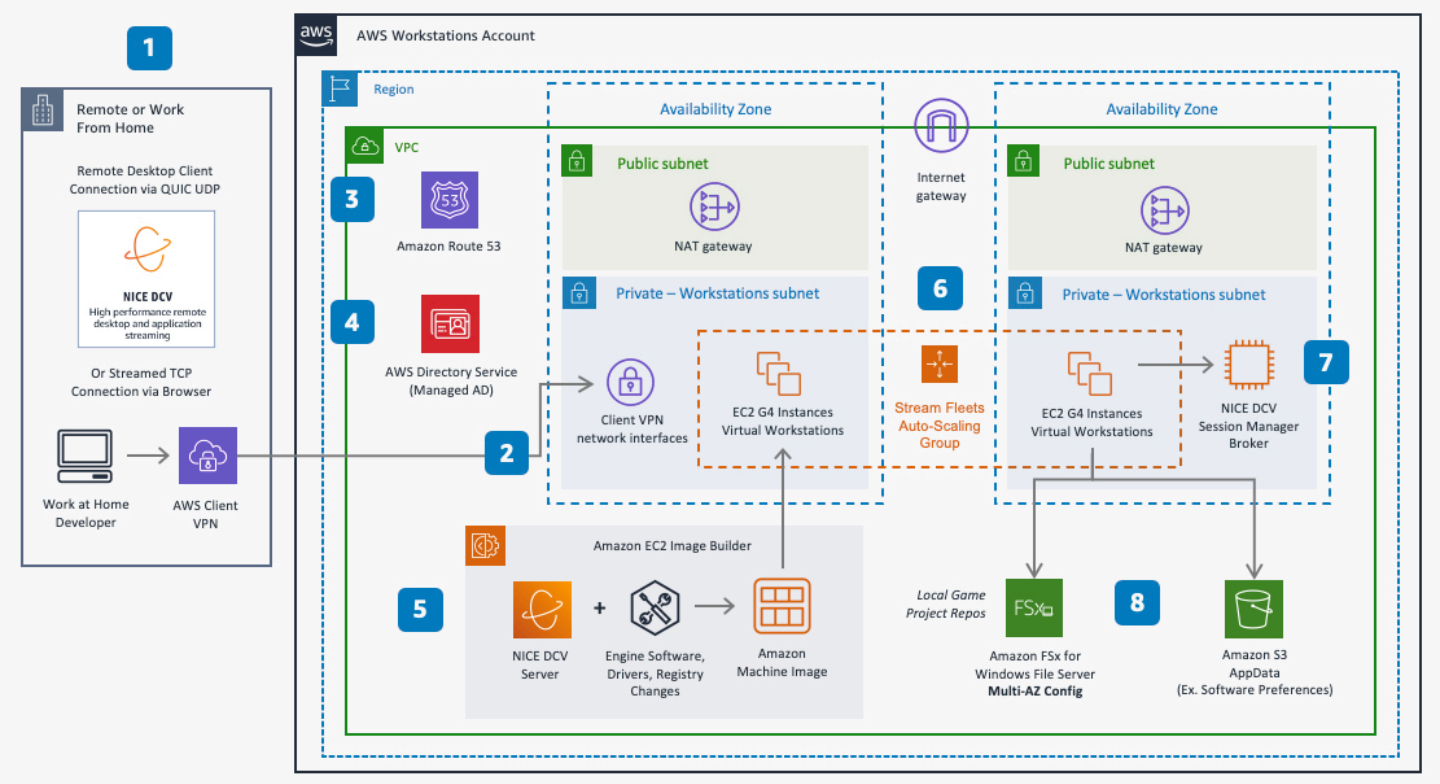
将游戏版本卸载到云端

1. AWS Direct Connect 为办公室内的开发人员提供低延迟、私密 AWS 的专用连接。远程开发者使用零信任技术 AWS Verified Access，例如虚拟专用网络 (VPN)，例如 AWS Client VPN。
2. AWS Transit Gateway 简化了本地网络之间 VPCs 和来自本地网络的连接的网络管理。
3. Perforce 管理由 Amazon EBS 存储支持的源代码和版本控制 (CI)，以实现快速访问的永久数据。Perforce Helix Core 在。AWS Marketplace
4. 当开发人员将更改推送到绑定到分支的 Perforce 时，提交会在 Jenkins 中开始构建 (CD)。Perforce 向詹金斯发起 POST 一个 JSON 有效负载。Jenkins 控制器调用引擎无头 C LI 命令，在临时的 Docker 节点（例如 Amazon Spot EC2 实例或亚马逊按需实例）上运行和并行化构建过程。 EC2 开发人员可以通过在负载均衡器后面使用两个 Jenkins 控制器（每个可用区各一个）来提高可用性。对于某些游戏引擎，开发者可能需要在其他子网中配置额外的许可基础架构，以便在每次运行并发版本时为构建环境提供许可证。
5. iOS 版本的 Xcode 部分已卸载到 Amazon EC2 Mac 实例，用于签署、构建和导出.IPA 文件，从而拆分流程并缩短构建时间。AWS Secrets Manager 包含配置文件、私钥和证书。
6. 构建项目将交付给 Amazon S3，后者会发送成功或失败的通知。 AWS Device Farm 支持对移动设备进行自动测试。

云游戏开发：工作站

游戏开发通常涉及分散在不同地点的团队远程办公，他们需要访问共享基础架构，并能够支持分散在不同地理位置的协作开发。这种游戏开发流程更加分散的趋势，包括使用工作 work-for-hire室和远程办公，需要实施强大的技术和工作流程，以提高生产力、共享专业知识和敏捷开发实践。

以下参考架构演示了如何使用 AWS Amazon DCV 协议托管远程游戏开发工作站。



使用 Amazon DCV 随时随地直播游戏开发

1. Amazon DCV 是一种支持 4K、60-FPS 流媒体的流媒体协议。使用浏览器的开发人员通过 TCP 连接进行连接，而桌面客户端可以通过端口 8443 使用 QUIC UDP 来提高性能。
2. 开发人员使用 AWS Client VPN 通过源网络地址转换 (SNAT) 安全地连接到工作站子网中的网络接口。
3. Amazon Route 53 为 VPC 中的资源提供私有 DNS 以及入站和出站 DNS 转发功能。
4. Directory Service 提供托管 Microsoft Active Directory，允许将本地游戏项目存储映射到个人用户。
5. 工作站是使用使用 Image Builder 构建的亚马逊系统映像 (AMI) 创建的。图像包括 Amazon DCV 服务器、开发者软件、注册表更改以及诸如 NVIDIA 游戏驱动程序或外设驱动程序之类的驱动程序。AWS Marketplace 包括常 AMIs 用于工作站。

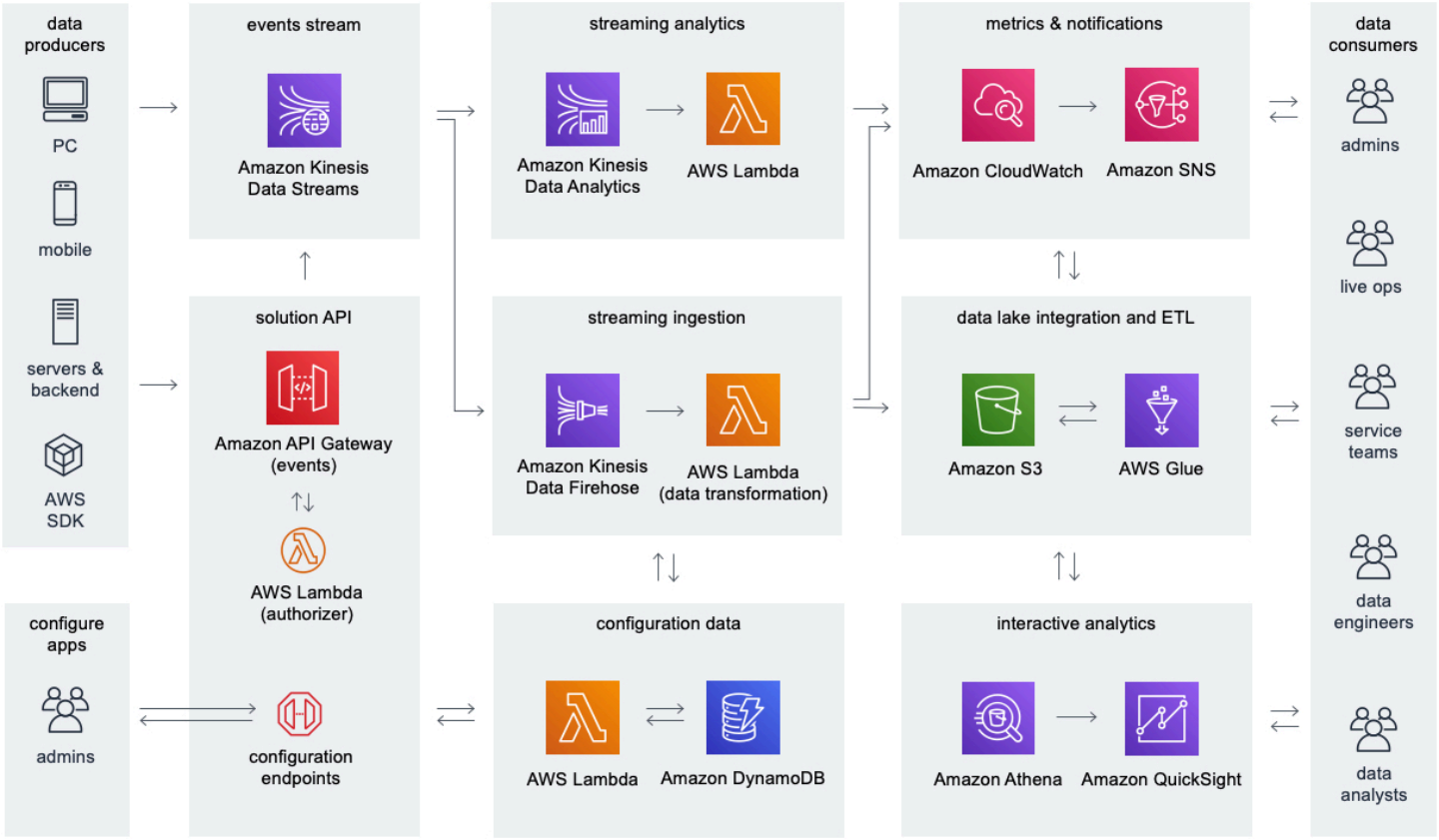
6. 工作站队列使用图形的 Amazon EC2 实例类型，这些类型可提供 GPUs Auto Scaling 组并使用 A EC2 uto Scaling 组进行扩展。
7. Amazon DCV 会话管理器代理支持对亚马逊 DCV 会话的管理。
8. 项目的本地文件存储托管在 Amazon f FSx or Windows 文件服务器中。开发人员通过从本地存储推送到源代码控制，承诺使用单独的 CI/CD 管道。

Gaming Analytics Pipeline

游戏开发者越来越多地寻找方法来更好地了解玩家的行为，这样他们就可以改善游戏体验，从而留住和扩大玩家群。游戏分析代表了理解和分析游戏和相关服务生成的数据所需的技术基础设施和流程。这通常需要使用可以支持此 end-to-end流程的分析管道架构，例如游戏分析管道解决方案的实现。

游戏分析架构具有以下特征：

- 数据源以 JSON 等通用格式发送数据，通常包括游戏服务器和游戏后端服务，以及游戏客户端，包括 PC、移动设备和游戏机。
- 游戏分析管道可自动执行整个工作流程，即摄取和存储原始数据并将其处理成可用的输出格式，以便数据使用者（例如最终用户和分析应用程序）可以对其进行高效、经济高效的分析。
- 游戏分析管道为摄取和处理大量实时数据提供支持，以便随着游戏的增长进行扩展。
- 为实时报告和批量报告用例提供支持。例如，实时运营团队通常使用实时仪表板和警报来监控游戏基础设施和玩家行为以检测问题。数据分析团队通常依靠必要的批量报告来了解一段时间内的趋势。



用于游戏遥测的无服务器游戏分析管道

游戏数据是从游戏客户端、游戏服务器和其他应用程序中提取的。流数据会被提取到 Amazon S3 中，用于数据湖集成和交互式分析。流式分析处理实时事件并生成指标。数据使用者分析 Amazon 中的指标数据 CloudWatch 和 Amazon S3 中的原始事件。

- 解决方案 API 和配置数据：使用 Amazon API Gateway 提供 REST API，用于管理您的游戏分析管道以及使用 Lambda 函数在亚马逊 DynamoDB 中存储配置数据。您可以在此 API 的基础上构建内部门户，也可以构建用于管理的自定义命令行界面。REST API 还提供服务器身份验证，用于从数据源提取游戏数据，并将遥测数据转发到 Amazon Kinesis Data Streams 进行实时处理和摄入存储。
- 事件流：Amazon Kinesis Data Streams 捕获游戏中的流数据，并允许亚马逊 Data Firehose 和适用于 Apache Flink 的亚马逊托管服务进行实时数据处理。
- 流式分析：适用于 Apache Flink 的托管服务分析来自 Kinesis Data Streams 的流式事件数据，并可以生成使用 CloudWatch Lambda 函数发布到的自定义指标和警报。
- 指标和通知：使用 Amazon CloudWatch 监控解决方案的指标、日志和警报。使用 Amazon SNS 向待命工程师和其他数据使用者发送通知。
- 流式提取：使用 Firehose 使用来自 Kines is Data Streams 的流式传输数据，并将其传输到 Amazon S3 中的数据湖，用于长期存储、转换以及与其他数据集成。

- 数据湖集成和 ETL：用 AWS Glue 于 ETL 处理工作流程，并在中组织元数据 AWS AWS Glue Data Catalog，这为数据湖提供了与灵活分析工具集成的基础。
- 交互式分析：最终用户可以使用 Amazon Athena 对存储在 Amazon S3 中的数据集执行临时交互式查询，Quick Suite 可用于构建仪表板。

有关分析管道的自动参考实现，请参阅 Gam e Analytics Pipeline，该管道可使用部署到您的账户 AWS CloudFormation。

定义

Well AWS -Architected Framework 基于六大支柱：卓越运营、安全性、可靠性、性能效率、成本优化和可持续性。 AWS 提供了多个核心组件，允许您为游戏工作负载设计 state-of-the-art架构。在本节中，我们将概述关键定义。

就本 paper 而言，游戏架构包括构建和运营游戏所需的后端技术基础架构。有些游戏可能没有社交、多人游戏或其他在线功能，也可能不需要使用本 paper 中描述的后端技术基础设施的某些方面。有关为支持游戏架构而经常部署的不同类型工作负载的详细讨论，请参阅场景。

AWS 云 基础设施是围绕区域和可用区构建的。

- 区域是世界上有多个可用区域的物理位置。
- 可用区由一个或多个离散的数据中心组成，每个数据中心都具有冗余电源、网络和连接，位于不同的设施中。

根据游戏的特点，您可能需要将游戏架构的某些组件部署到多个区域，例如提高玩家的性能，或者根据玩家的位置为他们提供定制的体验。

有许多不同的游戏类型，支持游戏所需的后端技术基础设施因正在开发的游戏类型而异。例如，流行的游戏类型可能包括第一人称射击游戏 (FPS)、角色扮演游戏 (RPG)、大型多人在线游戏 (MMOG)、大逃杀 (BR)、体育游戏、益智游戏等。还有不同的游戏互动模式会影响游戏的架构，例如回合制和同步游戏，它们具有不同的性能特征。

游戏是为在一个或多个游戏系统上玩而开发的，包括桌面、网络、移动设备、游戏机和较新的交互模式，例如增强现实 (AR)、虚拟现实 (VR) 和游戏流媒体解决方案。游戏通常支持跨系统玩法，这意味着玩家可以在其他系统上保存游戏进度并恢复游戏玩法，也可以在其他系统上启动与其他系统的玩家的游戏会话。

电子游戏盈利使游戏发行商能够使用不同的策略来创造收入，例如广告、数字和零售游戏购买、游戏内购买被称为微交易的可下载内容（DLC），以及通过所需的付费订阅才能玩游戏。游戏行业中一些最常见的关键绩效指标 (KPIs) 包括：

- 每日活跃用户 (DAU)
- 月活跃用户 (MAU)
- 并发用户 (CCU)
- 会话持续时间

- 每次安装成本 (CPI)
- 玩家生命周期价值 (LTV)
- 每位用户的平均收入 (ARPU)

游戏系统

视频游戏是为在游戏系统上玩而开发的，该系统提供客户端输入控件、图形、客户端软件（称为游戏客户端）和硬件，在某些情况下还提供系统专有功能来支持游戏玩法。

游戏系统通常分为以下几类：

- 游戏机：专为玩游戏而设计的专用娱乐系统，包括索尼、 PlayStation微软 Xbox 和 Nintendo Switch 等热门示例。游戏机通过在游戏系统提供商制造的主机硬件上安装物理或数字分发的游戏内容来提供玩游戏的能力。在此定义中，控制台可以是手持式或固定式，旨在用于家庭娱乐场景。
- 个人电脑 (PC)：使用安装在客户端计算机上的计算机软件玩的游戏，玩家可以对其进行自定义。出于这个原因，电脑游戏因其提供的灵活性和控制力而在玩家中很受欢迎。
- 网页：专为使用网络浏览器玩而设计的游戏，其优点通常是让玩家无论使用何种操作系统都能访问游戏。
- 手机：专为在手机（通常是智能手机操作系统）上玩而开发的游戏。手机游戏通常从数字应用商店下载并安装到手机上。

除了前面提到的系统外，还有一些新兴的系统仍然相对较新，并且还在不断增长，与更占主导地位的系统相比，其市场份额要小得多。此类游戏系统的示例包括增强现实、虚拟现实和游戏流媒体，有时也称为云游戏。

游戏直播涉及在云端渲染游戏玩法并流式传输到精简客户端（通常是浏览器）。游戏流媒体允许玩家玩完全远程托管的游戏，通常由游戏流媒体服务提供商在云端托管。在游戏直播中，玩家通过浏览器或云游戏服务提供商（游戏系统）提供的瘦客户端，连接到基于云的游戏。

游戏服务器

游戏服务器是游戏计算基础设施中最重要的方面之一。游戏服务器，有时也称为专用游戏服务器，用于开发多人游戏或需要服务器对游戏事件进行权威处理时。游戏服务器是游戏架构的中心，是核心逻辑的运行位置，包括管理玩家和游戏状态以及管理连接的游戏客户端和游戏服务器之间的交互。游戏服务器通常是游戏架构中对性能最敏感的方面之一，因为它负责处理来自玩家游戏客户端的输入，并将其适当

地实时分发给其他连接的玩家。性能不佳的游戏服务器会影响游戏体验的整体性能。因此，您应该优化游戏服务器性能并提供足够的容量，尤其是在游戏启动或游戏高峰期。

就本文档而言，游戏服务器或游戏服务器实例是指托管一个或多个游戏服务器进程的计算，例如虚拟机。游戏服务器进程代表您的游戏服务器版本中托管游戏会话的单个实例，这是您正在运行的游戏的实例，玩家可以通过玩家会话连接到该实例。出于这个原因，由于游戏会话与托管它的游戏服务器进程之间存在隐含的一对一关系，因此我们经常互换提及游戏服务器进程或游戏会话。在中 AWS，托管游戏服务器有多种计算选项，这些服务器通过弹性资源配置提供对基于云的可扩展容量的访问权限。

亚马逊 EC2 提供基于云的虚拟服务器（称为实例），支持多个版本的 Linux 和 Windows。您可以创建实例并像其他服务器或虚拟机一样直接管理它们。通常，将多个游戏服务器进程部署到一个实例以提高效率和降低成本。如果您希望最大限 EC2 度地控制计算基础设施，那么Amazon是游戏服务器的不错选择。

Amazon GameLift 为云端的专用游戏服务器托管提供了完全托管的解决方案，以及其他功能，例如与之配对 GameLift FlexMatch. GameLift在 Amazon 之上提供了一个抽象层， EC2 使游戏服务器管理变得简单，并且在大多数情况下都可用， AWS 区域 因此您可以使用竞价型实例将游戏服务器托管在靠近玩家的地方，从而减少延迟、实现高可用性并显著降低成本。虽然 GameLift 可以集成到现有的游戏后端中，但对于那些不想开发自己的游戏服务器管理和配对解决方案，更喜欢由游戏管理 AWS 并可以随着游戏发展而扩展的解决方案的游戏开发者来说，它特别有用。

亚马逊弹性容器服务 (Amazon ECS) Amazon ECS 是一项完全托管的容器编排服务，可用于运行基于 Docker 的容器。你也可以使用亚马逊 Elastic Kubernetes Service（亚马逊 EKS）来运行使用 Kubernetes 构建的基于 Docker 的容器。使用容器技术（例如 Amazon ECS 和 Amazon EKS 提供的容器技术）可以有效地将许多游戏服务器进程或其他游戏应用程序实例打包到一个 EC2 实例中，从而帮助您提高计算利用率。

容器的使用还可以通过使用开发人员在开发期间在本地计算机上使用的相同 Docker 镜像操作运行时来托管应用程序，从而提高开发人员的工作效率。您可以通过使用来进一步减少运营开销 AWS Fargate，这是一种用于运行容器的无服务器计算解决方案，与 Amazon EKS 和 Amazon ECS 兼容。Fargate 最适合您希望在容器中运行游戏服务器而不负责运行容器底层实例的用例。

您可以使用 AWS Outposts 在数据中心或本地设施中运行 AWS 基础设施和服务，从而使游戏能够在本地环境中运行，并 AWS 使用相同的基础设施来支持混合云采用策略。 AWS Local Zones AWS 区域是的扩展，它允许你的游戏服务器和其他对延迟敏感的工作负载更接近玩家或开发团队的运行。此外，为了减少游戏服务器的全球网络延迟，您可以使用 Gl AWS obal Accelerator 来提高游戏服务器的玩家流量的性能。

AWS Lambda 是一种无服务器计算服务，无需预置或管理服务器即可运行代码，这使得它非常适合异步游戏服务器用例，例如回合制游戏或具有轻量级计算需求、代码库较小以及可以使用无状态微服务

架构设计游戏功能的异步游戏服务器用例。请务必记住，Lambda 函数以事件驱动、每个请求为基础运行，而不是运行长时间运行的游戏服务器进程。Lambda 提供了本白皮书中选项的最多的运行时抽象，因为开发人员可以随时选择底层应用程序来托管他们的代码。

在选择游戏服务器托管方法时，请考虑各种要求，包括操作开销、旧代码库、性能要求和规模。 EC2 实例和容器是传统代码库的不错选择，因为它们只需最小的更改即可迁移到云端，而且您可以使用 EC2 实例来专用计算实例的资源，而容器可以简化管理和高利用率的实现。Serverless 函数提供了最高级别的抽象，你可以用它来定义只在响应事件时运行的代码，从而降低成本。

游戏客户端

游戏客户端代表玩家用于玩游戏的软件和硬件设备。游戏客户端提供软件，用于将玩家的输入转换为消息，然后发送到服务器进行处理，它负责处理来自服务器的传入响应并将图形等输出呈现给玩家。在实时联网的多人游戏中，游戏客户端通常在游戏会话期间与游戏服务器保持持续的网络连接，以减少网络延迟并最大限度地缩短处理时间。但是，游戏客户端也可以使用 REST 与游戏服务器或后端服务进行交互。

消息收发

游戏中通常有三种主要的消息类别：

- 针对特定用户或群组用户的@@ 玩家参与度消息，例如游戏邀请或推送通知
- 玩家之间的@@ 群组消息，例如游戏内聊天
- S ervice-to-service 消息，例如用于集成两个或多个应用程序的 JSON 消息

发送和接收这些类型消息的常见策略是使用发布者-订阅者和异步处理架构模式。 AWS 提供了多种服务，可以帮助您在游戏中实现消息传递。

- 亚马逊简单通知服务 (SNS) Simple Notification Service：用于使用架构模式在发布者和订阅者之间传送消息的托管服务。 pub/sub 发布者使用 API 向 Amazon SNS 发送消息，Amazon SNS 将消息异步传送到订阅应用程序，并且可以直接向支持一些最广泛使用的推送通知服务的移动客户端或台式机发送推送通知。Amazon SNS 可用于向客户发送推送通知以及 service-to-service消息传递用例。
- Amazon Simple Queue Service (SQS)：一种完全托管的队列服务，无论游戏服务器和游戏使用何种编程语言，都可以直接集成游戏服务器和您的游戏。许多游戏任务可以在后台解耦和处理，例如更新排行榜或数据库中的游戏时间值。这种方法是一种有效的方法，可以将游戏的各个部分分开，并将面向玩家的功能与后端处理独立扩展。

- 适用于 Apache Kafka 的 Apache Managed Streaming (MSK)：一项完全托管的服务，可使用流行的开源解决方案 Apache Kafka 简化数据流和生产者或消费者应用程序的构建。Kafka 通常用于摄取和处理实时流数据，也可用于发送消息。 service-to-service
- 亚马逊 ElastiCache （Redis OSS）：提供完全托管的内存数据存储，其中包括对 Redis 的热门 pub/sub 功能的支持，该功能通常用于开发聊天室应用程序和高性能消息传递。 service-to-serviceRedis 还支持列表和集合等丰富的数据类型，因此开发人员可以使用 Redis 进行高性能队列。
- Amazon Pinpoint：通过电子邮件、短信、语音和推送通知提供用户参与消息。例如，Amazon Pinpoint 可用于向玩家发送用户参与消息，邀请他们重返游戏，并可用于交易用例，例如支持多因素身份验证令牌、订单确认和密码重置电子邮件。

直播游戏操作（Live Ops）

直播游戏运营（Live Ops）是一种游戏管理和运营风格，它将游戏视为直播服务，并不断提供新功能、更新、促销、游戏内活动以及对已发布游戏的改进，以改善玩家社区的体验。

传统上，游戏是作为产品而不是服务交付的，新的内容和功能经常被整合到随后的版本或续集中，而不是包含在已发布的产品中。借助 Live Ops 的游戏管理方法，游戏运营团队可以发布游戏并通过实验、促销、游戏内活动和创新来维持一个参与度高的玩家社区，让玩家玩得开心。

尽管这种方法的好处是可以解锁新的玩家参与策略并提供经常性的收入来源，但它需要更多的运营专业知识。例如，要成功实施Live Ops策略，开发人员可能需要与云服务集成或运营自己的后端技术基础架构。他们还需要一种有效的方法来识别和应对游戏中或玩家社区中出现的、可能对玩家体验产生负面影响的问题。

卓越运营

卓越运营支柱侧重于大规模部署和运营基于云的游戏的最佳实践。重要的是要专注于卓越运营，以保持积极的玩家体验，并采取预防措施，为影响他们体验的问题做好准备并从中恢复过来。

聚焦领域

- 设计原则
- 直播操作
- 账户结构
- 游戏部署
- 健康监测
- 负载测试
- 持续优化
- 资源

设计原则

除了 Well-Architected Framework 白皮书中的设计原则外，以下设计原则还可以帮助您在构建和运营游戏方面实现卓越运营：

- 为游戏运营团队定义可衡量和可实现的目标，并在必要时进行调整：由于游戏的命中率驱动性质，很难提前确定游戏发布时会有多少玩家玩游戏，或者玩家对你正在进行的游戏运营有何期望。重要的是要与利益相关者一起设定雄心勃勃但可以实现的目标，并设计一种方法，在游戏开发团队优化玩家体验的同时，如果你的游戏超出预期，可以缩小规模。提前做好充分的准备和测试，以满足这些要求，并使您的业务和技术利益相关者在游戏运营的目标上保持一致。通过定义目标，游戏团队可以在规划、设计、配置、测试、部署和运营游戏后端基础设施的过程中在成本和性能之间取得适当的平衡。
- 使用运营手册来规划与游戏发布和特别活动相关的扩展活动：游戏运营团队应与业务利益相关者协调，对活动的预期玩家并发峰值预测进行建模，并进行主动规划，提前预先扩展基础设施容量。由于赛事期间玩家流量的波动性，事先的计划和预先扩展的活动应增强您现有的自动扩展系统，以提高您在赛事期间的成功机会，并验证您是否有足够的资源来提供积极的玩家体验。实施性能工程实践，以制定资源基准，并以数据为导向地了解系统容量，这将有助于指导预扩展活动和自动扩展配置。制定操作手册以保持流程的一致性。 这种提前规划和对玩家需求的响应对于直播服务游戏尤其重要，直播服务游戏必须保持可靠的性能和基础设施，才能在较长的时间内为活跃、参与的玩家群提供支持。

- 建立接收、调查和回应玩家支持请求的运营模式：游戏发布后，监控投诉和游戏问题的报告。实施适当的系统，以安全有效的方式与玩家互动，从而充分解决玩家问题，例如社区论坛、社交媒体、电子邮件、票务系统、呼叫中心或自动聊天机器人解决方案。 这对于直播服务游戏尤其重要，因为直播服务需要与玩家群进行持续的沟通，响应玩家的反馈以满足不断变化的需求，并在更长的生命周期内维持一个活跃的社区。

直播操作

GAMEOPS01：如何定义游戏的实时操作（Live Ops）策略？

与业务利益相关者协商，根据定义的目标和绩效指标，为您的游戏制定实时运营（Live Ops）策略。

最佳实践

- GAMEOPS01-BP01 使用游戏目标和业务绩效指标来制定实时运营策略

GAMEOPS01-BP01 使用游戏目标和业务绩效指标来制定实时运营策略

咨询业务利益相关者，例如游戏制作人和发行合作伙伴，以确定游戏的目标和绩效指标。这可以帮助您制定管理游戏的计划，包括定义维护时段、软件和基础架构更新计划以及系统可靠性和可恢复性目标。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

这些指标还可以帮助您确定应在游戏生命周期的哪个阶段加入实时操作Steam（Live Ops）来监控游戏生命值，收集直接的游戏反馈，并构建简化的自动化发布流程。例如，一款新游戏可能要等到达到一定的规模（以活跃玩家人数、收入或其他指标来衡量）后，才会成立专门的直播运营团队。一家成熟的游戏开发工作室可能已经拥有直播运营经验，也许是其他游戏的运营经验，因此他们只需要加入新游戏即可。

实施步骤

- 您可以定义游戏基础设施应能够有效支持的玩家并发量 (CCU) 以及每日和每月活跃用户（DAU 和 MAU）的目标、您的基础设施预算、财务目标和其他绩效目标，例如发布内容和功能以提高玩家参与度的频率。这些目标和指标有助于做出有关游戏设计、发布管理、可观察性和高效运营所需的支持的决策。

- 您的游戏的目标可能是每月至少发布一次新内容更新，并且在发布期间不停机。这些信息可帮助您定义发布部署策略，协调所需维护的时间安排，这些维护可能需要在整个月的其他时间停机，并有助于提高可用性 SLA。

账户结构

GAMEOPS02：你们是如何构 AWS 账户 造托管游戏环境的？

实施多账户策略，隔离不同的游戏环境，增强安全性、运营效率和可扩展性。 AWS Organizations 用于管理账户层次结构、对账户设置防护以及对已部署的资源实施标签策略和标记。

最佳实践

- GAMEOPS02-BP01 采用多账户策略，将不同的游戏和应用程序隔离到自己的账户中
- GAMEOPS02-BP02 使用资源标签组织基础架构资源

GAMEOPS02-BP01 采用多账户策略，将不同的游戏和应用程序隔离到自己的账户中

设计一个能够指导基础设施部署的客户结构，使其符合每个环境的安全、隔离和运营需求。通过限制对环境的访问并只允许在其中使用必需的 AWS 服务来隔离环境至关重要，因为生产环境处于封锁状态，而开发和测试环境则宽松，允许进行实验。强烈建议进一步隔离每个环境中的主要子系统，并将多个环境使用的公共服务 AWS 账户 单独托管和管理。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

采用多账户策略，将不同的环境（例如开发、测试、暂存、生产和共享服务）隔离给个人 AWS 账户，从而缩小事件的范围。 AWS 考虑 AWS Organizations 集中管理您的层次结构 AWS 账户 以进一步简化操作，并有选择地定义和应用账户级别和组织单位级（OU 级别）的策略。通过设计符合您的开发和生产工作流程需求的适当组织单位和 AWS 账户 结构，您可以优化成本并增强可扩展性。

- 采用多账户策略：隔离环境以缩小事件半径并简化操作。
- 使用 AWS Organizations：分层管理账户、应用策略并启用集中式治理。

- 规划可扩展性：设计精细的账户结构，为未来的增长实施成本节约措施。

实施步骤

中部署的游戏系统 AWS 应使用多个经过逻辑组织的帐户，以提供适当的隔离，从而缩小问题的爆发半径并随着游戏基础设施的扩展而简化操作。 AWS 账户 主机游戏基础架构通常分为以下逻辑环境：

- 开发人员使用游戏开发环境来开发游戏的软件和系统。
- 测试或质量保证 (QA) 环境用于执行集成测试、手动 QA 和其他必须执行的自动测试。
- 暂存或预生产环境用于托管已完成的软件，以便在发布到生产之前可以进行负载和烟雾测试。
- 直播或制作环境用于托管直播软件和基础架构，以及为玩家的制作流量提供服务。
- 共享服务或工具环境提供对许多不同团队使用的通用系统、软件和工具的访问权限。例如，中央自托管源代码控制存储库和游戏构建场可能托管在共享服务帐户中。
- 安全环境用于整合专注于云安全的团队使用的集中式日志和安全技术。

对于开启的游戏基础架构 AWS，建议为每个游戏环境（开发、测试、暂存和生产）创建单独的帐户，以及用于安全、日志记录和中央共享服务的帐户。

通常，管理有限数量的基础架构资源（通常为几百台或更少的服务器）的小型游戏开发工作室可能会 AWS 账户 为每种环境创建一个服务器（例如，一个制作账户、一个开发账户和一个暂存账户）。但是，随着游戏基础设施或团队规模的增长，这种简化的模型可能无法很好地扩展。

在设置这些环境时，请考虑许多 AWS 服务在特定区域内共享整个账户的资源和 API 级服务配额。在确定如何对账户进行逻辑组织时，必须考虑这一点。 AWS 账户 只会因使用部署到其中的服务而产生成本。因此，这提供了一种有效减少资源争用和服务配额的方法，尤其是在您的游戏不断发展以及越来越多的开发者需要访问权限来构建和管理资源的情况下。

根据我们与大型游戏开发工作室合作的经验，这些工作室通常运营着数千台服务器，数百名开发者访问资源，我们建议您设计一个更精细的账户结构，让支持您的游戏的各个应用程序拥有自己的开发、测试、暂存和制作帐户。由于规划和迁移实时系统的复杂性，在游戏启动后重新设计 AWS 多账户策略既困难又耗时，因此在确定正确的多账户结构时，请考虑未来的扩展需求。

您可以使用AWS Organizations设置层次结构和分组 AWS 账户，并定义组织单位（OUs），以便通过服务控制策略（ ）将常见的 OU 级别策略应用于它们。SCPs AWS Organizations 随着资源的增长和扩展，集中管理和治理您的环境。您可以通过编程方式创建新账户并分配资源，对账户进行分组以组织工作流程，将策略应用于账户或群组进行管理，并通过对账户使用单一付款方式来简化账单。此

外，Organizations 还与其他服务集成，因此您可以定义组织中各个账户的中央配置、安全机制、审计要求和资源共享。

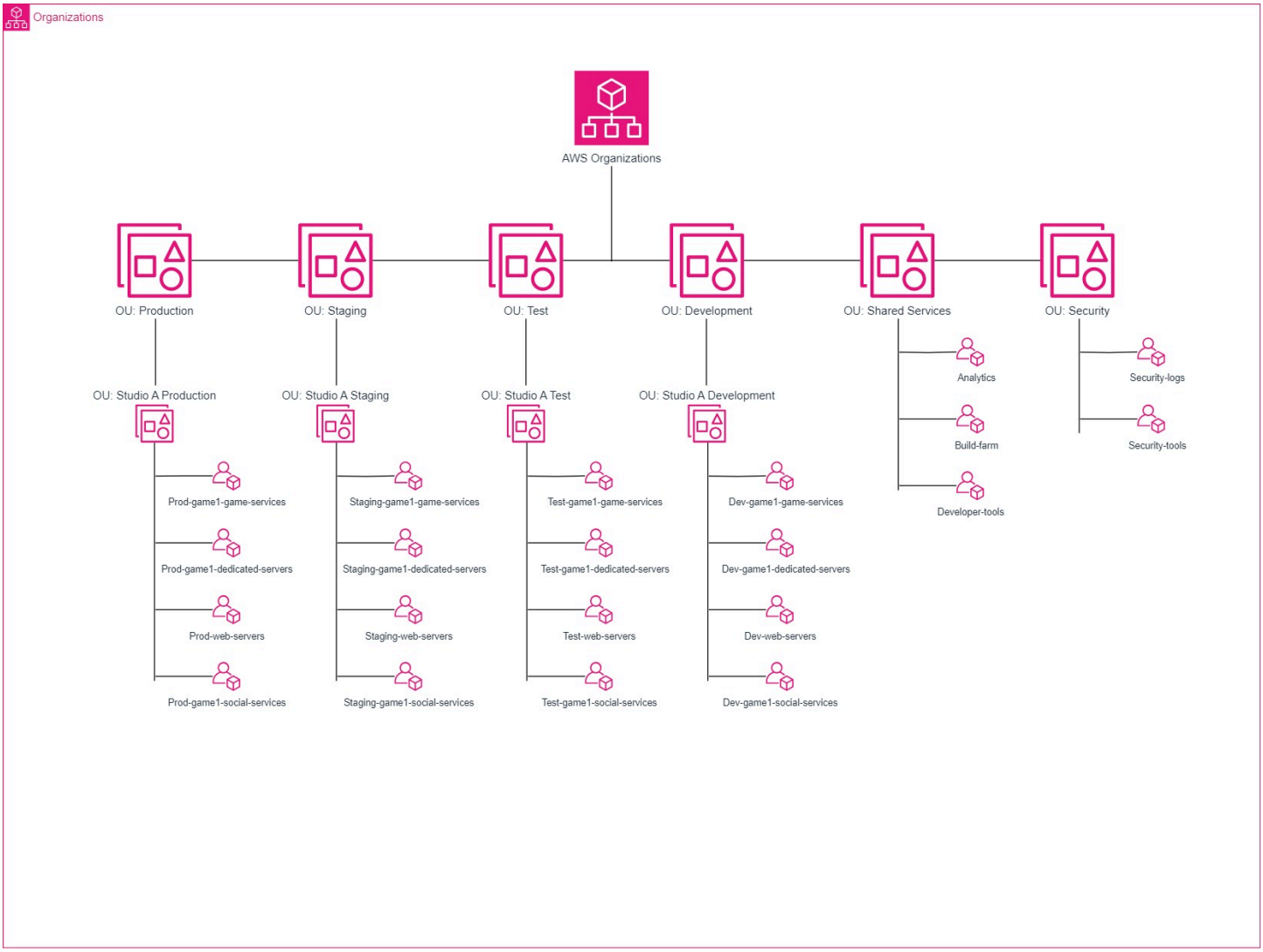
AWS Control Tower提供了一种设置和管理安全的多账户环境（称为 landing zon e）的简单方法。Control Tower 使用创建您的着陆区 AWS Organizations，带来持续的账户管理和治理，以及基于与成千上 AWS万客户迁移到云端的过程中合作的经验实施最佳实践。 AWS ConfigAWS Trusted Advisor、和AWS Security Hub CSPM是提供账户卫生状况的汇总或集中视图的服务。

这种隔离可帮助您为每个游戏环境设置自定义或个人权限和护栏。生产账户应具有必要的护栏、访问限制、监控和警报以及安全工具，而非生产账户可能不需要相同级别的护栏和权限。非生产环境可以自动化，以便在下班后关闭资源并节省成本。在这种精细度级别上进行账户分离可以直接监控每个支持游戏的环境的基础架构成本。

以下是游戏公司的多账户结构示例，该结构使用 AWS Organizations 和组织单位 (OUs) 在逻辑上 AWS 账户 分组为不同的环境和工作室。在此示例中， OUs 用于根据账户的环境，然后根据运营环境的工作室对账户进行分组。这演示了如何创建嵌套层次结构，以允许将单独的应用程序和游戏部署到其环境中各自的账户中（描绘为 OUs），如果您开发和运营多个游戏，这将非常有用。请参阅本支柱资源部分提供的文档和白皮书，了解在组织多账户策略时可以考虑的其他策略。

根据上面的讨论，下面的示例图假设游戏工作室（组织）的开发管道由 4 个阶段（开发、测试、试运行和制作）组成。对于给定的游戏 (game1)，每个环境 (OU) 都有单独 AWS 账户 的游戏服务、专用游戏服务器、社交服务和网络服务器。每个子系统中运行的资源 AWS 账户 都与相应的子系统相关。通常，使用这种开发管道的每款游戏都会为其复制这种或类似的结构 AWS 账户。

除了这些以游戏为中心的环境外 OUs，还有共享服务 OU 和安全 OU。这些 OUs 应该是整个组织的，而不是针对每个单独的游戏。这样，游戏就会消耗开发工具、数据和分析的共享服务，如本例所示。然后，将应用程序和系统日志发送到安全 OU 中的日志 AWS 账户 设置。



游戏环境的账户结构示例

GAMEOPS02-BP02 使用资源标签组织基础架构资源

要有效地管理和跟踪您的基础设施资源 AWS，请使用适当的资源标记和分组来识别每个资源的所有者、项目、应用程序、成本中心和其他数据。可以使用资源组将标记的资源组合在一起，这有助于提供运营支持。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

定义标记策略。典型的策略包括用于识别资源所有者的资源标签，例如团队名称或个人姓名、游戏、应用程序或项目的名称、工作室名称、环境（如开发或制作）以及资源的角色（例如数据库服务器、Web 服务器、专用游戏服务器、应用程序服务器或缓存服务器）。您可以添加其他标签来满足业

务和 IT 需求。 AWS Config还可以在资源创建和更新时强制执行标记策略。标签和资源组可从 AWS 管理控制台 AWS CLI、和 API 操作中获取。

实施步骤

- 标记资源以识别其所有者、项目、应用程序、成本中心和其他相关数据。
- 实施标签政策，包括所有者、项目、工作室、环境和资源角色的标签。
- AWS Config 用于强制执行标记策略，并通过 AWS 管理控制台 CLI 和 API 管理标签。

游戏部署

GAMEOPS03：你如何管理游戏部署？

通过彻底验证重复使用的组件、定期进行性能工程以及在整个开发生命周期中实施定期负载测试来管理游戏部署。

最佳实践

- GAMEOPS03-BP01 在游戏中重复使用现有核心游戏系统和基础架构之前，请对其进行验证和测试
- GAMEOPS03-BP02 在每个版本发布之前进行性能工程（或者至少对于主要版本）
- GAMEOPS03-BP03 尽早并经常进行负载测试
- GAMEOPS03-BP04 采用可最大限度地减少对玩家影响的部署策略
- GAMEOPS03-BP05 需要预扩展基础架构才能支持峰值需求

GAMEOPS03-BP01 在游戏中重复使用现有核心游戏系统和基础架构之前，请对其进行验证和测试

Organizations 倾向于重复使用以前游戏中的现有组件和源代码，以节省开发时间和成本。这些遗留组件和代码可能没有经过彻底的审查，也可能没有经过详细的集成测试，而是依赖于它们过去的性能。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

虽然重复使用有助于提高工作效率，但也可能带来将过去的性能和稳定性问题重新引入新项目的风险。因此，在重复使用以前游戏中的现有组件和源代码时，应进行严格的测试。

实施步骤

- 识别重复使用的代码和组件：对以前游戏中重复使用的源代码、库和组件进行分类。明确区分主动维护的代码和已弃用的代码
- 记录原始行为和已知问题：记录原始性能特征、功能限制以及与重复使用的组件相关的已知错误或生产事件。
- 进行全面的代码审查：对重复使用的组件进行详细的技术审查，尤其是那些过去存在问题或记录不佳的组件。
- 替换或重构高风险的传统组件：优先替换或更新有问题历史记录或不再可维护的旧组件，而不是依赖生产中的变通方法。
- 进行集成和兼容性测试：在新游戏系统的背景下验证重复使用的组件。验证它们与新模块、工具和的交互是否正常 APIs。

GAMEOPS03-BP02 在每个版本发布之前进行性能工程（或者至少对于主要版本）

性能工程是监控应用程序的多个关键运营指标的过程，以发现可以进一步提高应用程序性能的优化机会。这是一个迭代过程，从测试开始，然后优化代码、其依赖关系、关联进程、主机操作系统和底层基础架构。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

要对应用程序的性能进行更深入的分析，请在应用程序代码中集成应用程序性能监控 (APM) 或调试工具，该工具可以通过跟踪应用程序流程中的异常行为来隔离问题并缩短故障排除时间。APM 工具还能够识别执行缓慢的方法和外部操作。

AWS X-Ray协助开发人员进行性能工程活动，例如识别性能瓶颈以及分析和调试生产错误。您可以使用 X-Ray 来了解您的应用程序及其底层服务的性能，并确定和排除性能问题和错误的根本原因。通过多轮负载测试，在这些测试中，应用程序及其基础架构逐渐加载合成玩家流量，发现了各种系统瓶颈、应用程序错误、异常、操作系统问题以及其他在其他质量保证测试中可能没有发现的问题。

对于游戏发布、内容发布、促销和重大游戏内活动等关键事件，请使用 Countd AWS own，它根据游戏专家编写的剧本提供实施指导，以验证运营准备情况、降低潜在风险和规划容量需求。AWS Countdown 还提供高级支持选项，可提供增强的支持和工程师等选项，以优化您的基础架构。

实施步骤

- 性能工程包括评估和监控关键运行指标，以验证应用程序的代码、流程、操作系统和基础设施是否按预期运行。生产前审查还有助于定义不同模拟使用水平的基准性能。
- 使用 sar、top、vmstat、sysstat、netstat 和 Performance Monitor 等系统工具，发现和跟踪利用率、服务、I/O、流程等关键指标。
- 使用 APM 工具跟踪应用程序的性能和行为，例如 AWS X-Ray 隔离问题、识别瓶颈和调试生产错误。
- 对于游戏发布等关键事件，请订阅 Countd AWS own (IEM)，获取架构和运营指导、按需运营支持，以及识别风险和计划缓解措施。

GAMEOPS03-BP03 尽早并经常进行负载测试

负载测试是在系统上模拟真实流量以评估其可靠性和性能的过程。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

负载测试是制定资源性能基准和了解系统容量的关键因素，这可以指导财务预测、架构设计、资源分配、自动扩展配置和发布后的预扩展活动。其他优点包括：

- 优化的基础架构：资源可能过剩或配置不足。了解所需的资源可以降低成本，减少需要管理的基础架构。
- 可扩展性就绪：某些机制和功能可以让用户快速进入游戏。知道何时以及如何扩大规模可能是适当满足不断增长的需求和失去玩家之间的区别。使用负载测试结果来准备包含不同缩放级别的系统阈值、警报点和关键警报点的运行手册。
- 更高质量的代码：服务之间的串扰过多、未批处理的数据库调用、算法效率低下、内存泄漏和服务降级问题等问题有时更易于大规模识别。
- 行为验证：在测试中注入不同类型的失败可以验证系统的预期行为或发现需要纠正的错误处理问题。

理想情况下，开发人员应在整个开发过程中的多个阶段进行负载测试，因为每个阶段可以产生不同的好处：在早期阶段，他们可以指导架构决策和重构工作，同时进行更改更便宜、更简单。在每次冲刺或迭代结束时，他们都会使用最新的特性和功能来验证应用程序的性能。

在部署到生产环境之前，模拟实际预期使用模式的大规模负载测试可确认系统处理生产工作负载的能力。部署后，定期的负载测试会监控系统的性能，并确定随着时间的推移可能出现的变化或瓶颈。

要模拟玩家流量，您需要轻量级客户端或机器人来模拟游戏客户端流并与游戏后端进行交易，以模拟现实世界中的玩家行为。这些数据通常是通过游戏日志和由人工驱动的 QA 测试生成的数据来捕获的，以及通过现实世界的有限规模 alpha 或 beta 测试来捕获的，在这些测试中，真实玩家会被邀请玩游戏的抢先体验版本。

必须将系统的行为记录在操作手册中，以帮助排除将来可能出现的故障，并保留性能指标，以便将来的负载测试与之进行比较。还建议让人类 QA 人员在游戏进行负载测试时对其进行测试，因为他们可能会发现机器人无法识别且指标无法反映的问题。

AWS Fault Injection Service 是一项完全托管的服务，用于运行故障注入实验，可以直接提高应用程序的性能、可观察性和弹性。故障注入实验在混沌工程中被广泛应用，即一种实践做法，具体为：通过创建中断事件（如 CPU 或内存使用量的突然增加）在测试或生产环境中对应用程序施加压力，观察系统如何响应，并实施改进措施。故障注入实验可帮助团队创建所需的真实条件，以发现分布式系统中难以发现的隐藏错误、监控盲点和性能瓶颈。

实施步骤

- 使用 Kubernetes-Bases 游戏负载测试指南设置分布式负载测试环境。
- 使用提供的部署文件在 EKS 集群中自定义和部署 Locust 控制和工作容器，从而实现可扩展且可管理的负载生成。
- 在运行手册中记录负载测试期间的系统行为和指标，以帮助将来进行故障排除并建立性能基准。
- 使用故障注入实验来模拟现实世界的中断，发现系统性能、可观察性和弹性方面的隐藏问题。

GAMEOPS03-BP04 采用可最大限度地减少对玩家影响的部署策略

为您的游戏软件和基础设施制定部署策略，最大限度地减少导致玩家无法进入游戏的停机时间。虽然某些类型的更新可能需要在游戏客户端上安装新的更新，但要设计游戏时尽量减少或避免部署期间的停机时间。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

制定游戏部署策略时要考虑的最重要步骤之一是确定如何管理游戏基础架构。使用基础设施即代码 (IaC) 工具（例如 Hashicorp 的 Terraf AWS CloudFormationhttps://aws.amazon.com/cloudformation/ or m ）来管理您的游戏基础架构，以减少环境准备期间的人为错误。基础设施模板可以在自动化管道中部署和测试，从而在不同游戏环境的配置中保持一致性。

有几种部署策略可用于一款游戏：

滚动替换

滚动替代部署的主要目标是在不关闭游戏且不影响玩家的情况下执行发布。重要的是，要执行的升级或更改必须向后兼容，并且与系统的先前版本相似。

在此部署中，服务器实例被运行更新版本的实例以增量方式替换（替换或推出）。这种滚动替换可以通过几种不同的方式执行。例如，要实现对专用游戏服务器队列的滚动更新，一种典型的方法是创建一组新的 Auto Scaling EC2 实例，其中包含部署在这些服务器上的新游戏服务器构建版本，然后逐步将玩家路由到托管在这个新服务器队列上的游戏会话。如果需要关联的游戏客户端更新作为使用新游戏服务器版本的先决条件，则必须包括验证检查，以验证只有安装了此新游戏客户端更新的玩家才能进入这些游戏会话。

包含旧游戏服务器版本的服务器舰队（例如 EC2 Auto Scaling 群组）只有在以优雅的方式耗尽活跃的玩家会话后才会从服务中删除，通常是通过设置允许游戏运营团队自动执行此过程的个性化服务器指标。或者，为了减少基础设施的数量和进行滚动部署的时间，可以执行另一种方法，即从服务中移除现有生产实例，使用新的游戏服务器版本进行更新，然后将其放回生产队列中。这种方法减少了所需的基础设施数量，但也增加了风险，因为随着服务器的更换，可供玩家使用的实时游戏服务器数量会减少。

此模型还可用于对不托管游戏玩法的数据库、缓存和应用程序服务器等后端服务执行滚动部署。只要这些服务以高度可用的方式部署到多个集群实例，那么部署到这些服务的复杂性就应该低于部署到专用游戏服务器的复杂性。

蓝/绿部署

游戏中 blue/green 部署的主要目标是最大限度地减少停机时间，同时还允许在发现问题时安全地回滚到之前的部署。它适用于两个版本的游戏后端兼容并且可以同时为玩家提供服务的部署。

在 blue/green 部署策略中，设置了两个相同的环境（蓝色和绿色）。现有游戏版本被标记为蓝色，而作为部署目标的新游戏版本被标记为绿色。当绿色环境准备好迁移时，您可以将路由层配置为将流量转移到绿色环境，同时保持旧环境（蓝色）可用以防需要故障恢复。在这种情况下，路由更新可能需要更新配对服务以将其配置为开始向新队列发送游戏会话，或者对于游戏后端服务，可能需要更新服务的 Amazon Route 53 中的 DNS 记录，或者更改应用程序负载均衡器权重以将流量发送到新的目标组。

blue/green 部署策略的缺点之一是备用环境的固有成本，因为在执行部署时需要额外的基础架构。降低这种额外基础架构成本的一种选择是考虑采用一种 blue/green 部署变体，将新的游戏软件部署到已经部署到生产环境中的相同服务器上。在这种情况下，可以在现有的蓝色服务器进程的同时使用新软件启动新的绿色服务器进程，切换发生在服务器进程之间，而不是单独的物理基础架构之间。这种方法还可以通过消除等待新服务器在云端启动的麻烦，从而加快在大量基础架构上的游戏部署。有关此部署方法的最佳实践，请参阅上的 Blue/Green 部署。 AWS

金丝雀部署

Canary 部署对游戏开发者很有用，因为该策略可以应用于发布游戏的早期 alpha 或 beta 版本，或者发布游戏功能，例如新的游戏模式、地图，或者向生产中的受限或少数玩家发起挑战。这样的部署被称为金丝雀。该版本可能有额外的跟踪和报告，因此，当真实玩家玩该游戏或功能时，会收集他们的游戏玩法遥测并分析其异常和问题。

对于新功能，玩家不会持续收到有关此问题的通知，游戏遥测是用于确定玩家是否遇到问题以及是否应回滚版本的主要来源。同时，如果未发现重大问题，则可以将该功能进一步推广给更多玩家以获取更多数据。如果玩家收到通知，则可以要求他们定期提供有关其体验的反馈。理想情况下，此类测试活动将由现场操作团队进行协调。

作为一种策略，金丝雀部署也可以用于标准版本，以逐步向玩家提供新功能。与标准 blue/green 环境相比，一个潜在的优势是不需要全面的第二个环境。新的缩小规模环境的容量决定了将有多少玩家加入新功能。在添加更多玩家之前，必须适当调整容量。即使预计这种定制 blue/green 技术的成本将相对低于标准的蓝/绿，但据估计，它所产生的成本仍可能高于金丝雀部署的滚动替代技术。

在生产环境中只运行一个 canary，然后将其重点放在数据和反馈上。如果部署了多个 Canary，则会使生产中问题的故障排除和隔离变得复杂，并会损害所收集的数据集和反馈的质量。

金丝雀的一个变体是，通过定向部署运行一个或多个实验（通常是用户界面测试），其中一组游戏后端服务器提供一个版本的功能，另一组相同大小的服务器提供同一功能的另一个版本。没有为此创建任何额外或特殊的基础架构，只有选定的后端服务器才能收到这些更新。实验的结果是观察玩家对同一功能的每个版本的反应，确定总体上是否存在共识，并观察其可用性或功能是否存在问题。这样的战略实验也称为 A/B 测试，整个过程称为 A/B测试。完成这些实验后，将收集必要的测试数据，然后在用于测试的服务器上恢复到游戏后端系统的当前版本。

传统部署

在传统的部署方式中，在定期维护时段内，游戏会关闭，连接的玩家会被丢弃或耗尽，然后使用最新的代码版本更新游戏后端中的服务器实例。这种部署每次执行都会影响玩家，并且必须提前通知玩家。因此，这种模式对玩家的影响最大，应尽可能避免。

部署游戏更新后，可以在向玩家开放游戏之前对游戏进行烟雾测试，玩家将等待游戏重新开放。当玩家尝试在短时间内登录和玩游戏时，这可能会导致流量激增。因此，如果游戏的设计不是为了应对如此激增的流量，你可以选择逐步允许玩家分批重返游戏。

或者，您可以选择过度配置基础架构以维持开放的流量峰值，在游戏流量稳定下来之后，可以缩小资源规模。如有必要，可以在玩家人数最少的非高峰时段进行此类部署。频繁的定期维护以及延长的维护本

质上会带来玩家流失和潜在收入损失的风险。玩家还期望新版本发布后会发生变化，并且在停机一段时间后返回游戏后可能会失去对游戏的信任。

实施步骤

- 最大限度地减少停机时间：实施部署策略，减少停机时间并让玩家参与游戏。
- 基础设施即代码 (IaC)：使用诸如 AWS CloudFormation 或 Terraform 之类的工具来管理游戏基础设施并减少人为错误。
- 部署策略：使用滚动替换、蓝/绿和金丝雀部署的一种或组合来提供流畅的更新并减少对玩家的影响。

GAMEOPS03-BP05 需要预扩展基础架构才能支持峰值需求

在大型游戏活动之前扩展基础架构，确保您可以应对玩家需求的突然增长。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

除了新游戏发布外，直播游戏通常还会举办游戏内活动、促销活动、新内容和赛季发布，以此作为维持和提高玩家参与度的方法示例。在活动或促销期间，此类活动会吸引大量玩家流量。企业期望达到或超过赛事的预期目标，游戏基础设施必须维持和支持他们度过难关。

提前准备好基础架构，以便能够支持大型赛事中预期的玩家负荷。为了做好准备，游戏运营团队应与销售和营销领域的利益相关者协调，通过查看过去的玩家并发度、参与度指标和销售数据，估算即将举行的活动中将产生的预计需求。如果活动是针对新游戏的发布，则游戏运营团队应与这些利益相关者合作，对他们预期的规模做出切合实际的预测。虽然可能很难预测一款游戏会取得多大的成功，但重要的是每个人都要了解对成功的期望，这样才能扩展和测试基础架构以支持这些目标。

许多游戏选择分阶段发布，首先是向少数玩家开放游戏的软启动，然后在全面公开发布之前，在每个阶段有机地扩大玩家规模。在试发布期间，监控、识别、跟踪和解决问题，同时完善对公开发行的预测。

要正确估算基础架构需求，请在游戏发布之前，通过针对在生产环境或类似生产的暂存环境中运行的游戏后端运行的负载和性能测试来收集数据。应运行多轮这些测试，以模拟游戏的不同条件，并验证后端在大多数条件下是否可以承受负载。

为了实现这一目标，开发者可以编写游戏机器人，这些机器人可以遍历游戏中的各种工作流程并模拟不同的条件。这些测试应检查游戏后端的不同系统层，以便对每个层和组件进行测试并记录细节。使用从这些测试中收集的数据来制定游戏发布计划。

应尽可能通过使应用程序具有高可用性和容错性来识别和消除单点故障 (SPOF)。使用负载测试 SPOFs 通过模拟不同上游和下游层的故障以及验证游戏和其他组件的行为来进行识别。

除了为游戏发布、游戏内活动或促销准备准备所需的估计基础设施外，还可以将系统设置为自动按需扩展。定义、配置和监控缩放事件阈值，以允许游戏后端进行扩展以维持大量的玩家流量。对于可变流量，最好进行预配置，因为可能没有足够的时间进行横向扩展。在最初的游戏发布期间，可能需要手动扩展，这会比自动化系统扩展资源的速度快于预期。

开启后 AWS，组织应为他们在游戏后端使用的服务申请更高的服务配额。Service Quotas 是为账户设置的，目的是保护客户免于无意中建立或扩展超出预期的基础架构。当账户中运行的游戏达到该区域中配置的服务配额的上限时，该服务会限制超出预配置配额和突发配置的请求。油门可能会导致意外或意想不到的错误，并损害玩家的体验。监控、跟踪并定期检查游戏在生产中使用的服务的服务配额阈值，以避免限制。当使用量超过可容忍的服务配额阈值时，可以通过向控制台支持中心提出 Support Console Center、登录受影响的账户后或使用 Support API 来请求增加配额。

对于游戏发布、内容发布、促销和重大游戏内活动等关键事件，请使用 Countd AWS own。Countdown 根据游戏专家编写的行动手册提供实施指导，以提供运营准备、降低潜在风险和规划容量需求。AWS Countdown 还提供高级支持选项，可提供增强的支持和工程师等选项，以优化您的基础架构。

如果您要发布在亚马逊上托管的游戏 GameLift，请查看发布前的清单以做好准备。

实施步骤

- 提前扩展基础设施：提前为大型游戏活动做好基础设施准备，以应对玩家需求的突然增长。
- 估算需求：与销售和营销部门协调，使用过去的玩家数据和现实的预测来估算预计的需求。
- 负载测试和 SPOF 移除：进行多轮负载测试，以验证后端容量，识别单点故障，并正确配置自动扩展。

健康监测

GAMEOPS04：你如何监控游戏的生命值？

通过实施全面的工具来检测和跟踪影响玩家的问题，包括客户端活动日志、后端服务监控和错误报告，监控游戏运行状况。使用诸如 Amazon CloudWatch 和 AWS X-Ray之类的 AWS 工具组合以及第三方解决方案，帮助快速识别和解决问题。

最佳实践

- GAMESOPS04-BP01 控制游戏以检测和监视影响玩家的问题

GAMESOPS04-BP01 控制游戏以检测和监视影响玩家的问题

除了回应社交媒体和玩家报告的问题外，还要使用监控解决方案来检测和调查影响玩家的问题，从而检测和调查影响玩家的问题。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

任何测试都无法识别出游戏中的每个问题。游戏发布时通常会出现已知问题，这些问题计划在游戏的下一版本中逐步修复。已知和可重现的问题很容易解决和修复。为了帮助识别此类问题，游戏客户应在各个战略位置实施玩家活动跟踪、应用程序日志和报告，以帮助后端团队识别客户端问题。尽早发现此类问题的能力有助于游戏开发者在问题变得广泛之前对其进行故障排除和修复。跟踪代码报告的数据和日志不应包含个人身份信息 (PII)，并且应仅包含有助于调试的游戏特定元数据。

实施可观察性解决方案，用于检测和响应游戏崩溃或错误等问题。您可以使用 Amazon S CloudWatch ynthetics 创建可以监控面向玩家的后端游戏服务的运行状况的加那利群岛。您可以使用检测您的后端服务AWS X-Ray来跟踪分布式服务间的请求，并将您的自定义日志和指标发送到 Amazon CloudWatch。

第三方解决方案，例如 Backtrace.io 和 Sentry，是游戏中错误报告的流行解决方案。来自Ne w Relic、Splunk、Datadog 和Honeycomb .io等合作伙伴的应用程序性能监控（APM）解决方案也很受欢迎。

除了官方支持渠道外，游戏的直播运营团队和社区经理还应监控各种社交网络和频道，以检查玩家的反馈、投诉和错误报告。查看并尝试重现所有针对游戏的投诉，或将其发送给 QA 团队进行审查。如果问题可以重现，请将问题上报给游戏开发者进行故障排除和修复，以免影响更大的玩家群。

实施步骤

- 实施监控解决方案：使用监控工具检测影响玩家的问题并快速做出响应。
- 跟踪玩家活动和日志：使用工具让游戏客户端记录玩家活动和报告问题，并确认不包含任何个人身份信息 (PII)。
- 使用第三方和 AWS 工具：使用 X-Ray 等 CloudWatch工具和第三方解决方案进行错误报告和性能监控，并监控社交媒体以获取玩家反馈和错误报告。

负载测试

GAMEOPS05：在对游戏进行负载测试时应该考虑什么？

在对游戏进行负载测试时，请考虑适当的测试阶段、负载生成架构和测试框架，以有效评估系统性能和可扩展性。选择时机（早期开发、冲刺、预制作或部署后）、基础设施（、EKS、Fargate 或 Lambda）和测试工具（EC2JMeterLocust、Grafana K6 或 Gatling）的正确组合，以适应游戏的独特特征和开发目标。

最佳实践

- GAMEOPS05-BP01 选择合适的阶段、架构和负载测试框架来实现您的目标

GAMEOPS05-BP01 选择合适的阶段、架构和负载测试框架来实现您的目标

游戏负载测试的方法可能会有很大差异，具体取决于许多因素，包括游戏的开发过程所处的阶段、负载生成系统本身的架构以及负载测试框架的选择。无论是在早期阶段、迭代冲刺期间、生产部署之前还是部署后，何时进行测试的时间都将决定测试工作的目标和重点。负载生成基础设施的不同设计各有优缺点，负载测试框架的选择会极大地影响测试过程的功能、易用性和集成。通过深思熟虑地调整这些元素，开发团队可以根据游戏的独特特征量身定制负载测试方法，提取最有价值的性能见解，并为玩家提供流畅的体验。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

不同开发阶段的负载测试

在开发阶段的早期进行探索性负载测试可以验证底层系统架构。这有助于开发者在完成大量实施工作之前，就游戏的基础架构、数据库设计和网络拓扑做出明智的决策。负载测试可以识别风险并创建性能基准，从而有可能最大限度地减少开发生命周期后期对昂贵的返工和技术债务的需求。他们还可以促进团队对游戏性能要求的共同理解，从而改善协作和决策。 最终，初始阶段的负载测试为高性能、可扩展和弹性的游戏奠定了坚实的基础，有助于增强整体玩家体验。

在每次冲刺或迭代结束时，负载测试可以评估最新周期中引入的新功能、错误修复和其他更改对性能的影响。这种有针对性的方法使开发团队能够快速识别最新更新引入的回归或性能下降情况，使他们能够在这些问题进一步传播之前解决这些问题，并保持稳定的质量和性能水平。

在部署到生产环境之前，强大的负载测试可帮助团队验证系统是否有能力处理预期的实际流量和负载条件。他们可以发现生产基础架构中的可扩展性瓶颈或资源限制，并提供优化游戏性能的机会，从第一天起就创造流畅且响应迅速的用户体验。从发布前的负载测试中获得的见解可以降低发布当天的风险，并为正在进行的容量规划提供信息，这为游戏的长期可持续性和可扩展性奠定了基础。

对已经投入生产的游戏进行负载测试可以让团队监控游戏的性能，并识别随着时间的推移可能出现的性能下降或降级。这使他们能够在问题影响玩家体验和对用户留存率产生负面影响之前主动解决问题。此外，生产环境中的负载测试可以验证已实施的性能优化工作或基础架构扩展的有效性。即使游戏不断发展和成熟，该过程也能为玩家提供高质量、响应迅速且可扩展的游戏体验。

生成负载的架构

游戏负载测试的负载生成架构的设计可以采取多种形式，每种形式都有自己的优势和注意事项。

在最基本的层面上，可以将自我管理的 Amazon EC2 实例配置为充当负载生成器。 使用控制节点和工作节点的方法，您可以设置多个生成负载的实例，每个实例都运行自己的测试脚本，并由单个控制实例进行总体管理。该架构可以在不增加复杂性的前提下扩展并生成更多负载，但是这种亲身实践的方法需要团队来处理底层基础架构的配置、配置和管理。

要获得更具可扩展性和编排性的方法，您可以使用 A mazon EKS Kubernetes 集群在基于容器的负载代理队列中管理和分配负载测试工作负载。 Kubernetes 的自动扩展功能可用于处理产生负载的 Pod 的扩展，而团队则自己在托管 Pod 的集群中配置和管理底层 EC2 实例。

或者，的无服务器性质AWS Fargate可以抽象基础架构管理，同时仍提供必要的可扩展性和灵活性，从而加快和简化负载测试设置。对于已经存在本地、产生负载的 Kubernetes 集群但可能需要额外容量的混合解决方案，EK S Anywhere 可以将两个集群作为一个集群进行管理。 AWS 管理控制台

您也可以根据自己的要求和目标使用AWS Lambda函数。Lambda 函数的设置和扩展相对简单，无需预置和管理额外资源。由于与其他 AWS 服务的深度集成，它们还允许创建更复杂、更动态的测试场景。但是，Lambda 函数确实对并发函数和运行时间（15 分钟）有限制，这可能会限制可以实现的负载测试的规模和长度。冷启动延迟也会影响结果的准确性，而 Lambda 的资源限制可能不适合要求很高的负载测试工作负载。

希望使用预建解决方案的工作室可以在上使用分布式负载测试。 AWS此解决方案使用 Amazon ECS AWS Fargate 来部署容器，这些容器可以模拟成千上万的连接用户。您可以使用它以 IAC 方式 AWS CloudFormation快速启动负载测试基础架构。

负载测试框架

没有两个负载测试框架的构建方式相同。有些具有用于创建测试的直观图形界面，而另一些则完全基于命令行。一种工具可能灵活且性能出色，但需要花费时间和精力来配置和管理，而另一种工具可能是

无服务器的，但它可以创建和运行的测试有限。有些人喜欢大型社区和大量教程，但尚未在现场得到证实，这与其他可能在生产中经过实战考验但缺乏社区支持或文档的教程形成鲜明对比。选择能为您和您的团队取得适当平衡的框架。一些流行的选项是：

- Apache JMeter：流行的基于 Java 的开源负载测试框架，因为它具有强大的功能集和易用性。它能够模拟复杂的用户场景、广泛的支持协议、全面的报告和久经考验的跟踪记录， JMeter 是负载测试的可靠选择。
- Locust：基于事件驱动架构构建的现代分布式负载测试框架，既高性能，又节约资源。 测试是用 Python 编写的，允许灵活的测试场景利用数千个强大的第三方库，同时保持友好性和易读性。
- Grafana K6：强大的负载测试框架，将易用性与高级功能相结合。它支持分布式负载生成、灵活的脚本编写以及与 Grafana 的无缝集成以实现数据可视化，这使得 Grafana K6 成为一个有吸引力的选择。
- Gatling：开源负载测试框架，以其性能和可扩展性而闻名。其基于 Scala 的特定领域语言 (DSL) 允许开发人员创建简洁、可维护的负载测试脚本，其强大的报告和分析功能可提供被测系统的详细见解。

实施步骤

- 负载测试阶段：在不同的开发阶段（早期开发、冲刺、预生产和部署后）进行负载测试，以验证系统性能并发现问题。
- 负载生成架构：根据可扩展性需求、管理偏好和特定的测试要求选择适当的负载生成架构（EC2EKS、Fargate 或 Lambda）。
- 负载测试框架：选择一个在易用性、性能 JMeter、灵活性和社区支持之间取得平衡的负载测试框架（例如 Locust、Grafana K6 或 Gatling），以满足团队的需求。

持续优化

GAMEOPS06：随着时间的推移，你是如何优化游戏的？

通过监控关键指标和遥测数据来确定玩家趋势、系统性能和需要改进的领域，从而随着时间的推移优化您的游戏。根据这些见解不断更新您的游戏设计、基础设施和负载测试方法，同时适应新的技术和框架，以便随着游戏的发展提供最佳性能和玩家体验。

最佳实践

- GAMEOPS06-BP01 监控关键游戏指标以识别玩家的趋势和模式，并使用这些信息来改进游戏
- GAMEOPS06-BP02 随着游戏规则的变化更新和调整负载测试方法

GAMEOPS06-BP01 监控关键游戏指标以识别玩家的趋势和模式，并使用这些信息来改进游戏

除了游戏客户端系统使用情况、应用程序使用情况、异常和崩溃数据外，还可以捕获发送到游戏后端系统的游戏遥测数据。这些数据应代表玩家的活动，以便您可以了解玩家如何与游戏中的各种功能互动。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

根据其实现情况，游戏客户端可以在游戏世界中预定义的游戏功能或位置收集遥测数据。数据被发送到后端摄取服务进行处理。如果无法访问后端服务，则客户端可以在本地设备上存储数据，直到后端服务再次可用。游戏设计师使用这些遥测数据来查看玩家的游戏玩法，以及游戏中是否存在异常。

例如，可以从遥测数据中提取玩家的移动以及与地图中物品的互动，并将其绘制为玩家在设定的时间段内在游戏中活动的热图。这些数据可以帮助游戏设计师确定平衡游戏中各种元素的必要性，例如武器的力量、游戏中角色的力量或地图的复杂性。通常存储原始遥测数据，然后进行处理，以提取分析师可以可视化的分析。

游戏 AnalyticsPipeline解决方案的实现可帮助游戏开发者启动可扩展的无服务器数据管道，以提取、存储和分析游戏和服务生成的遥测数据。该解决方案支持以流媒体方式提取数据，允许用户在几分钟内从他们的游戏和其他应用程序中获得见解。

对于自定义游戏遥测数据摄取、存储、处理和分析， AWS 还提供许多用于大数据处理和分析的专业服务。

实施步骤

- 捕获游戏遥测数据：收集有关玩家活动、系统使用情况、异常和崩溃的数据，以了解玩家的互动并识别问题。
- 实现遥测收集：使用预定义的游戏功能或位置收集遥测数据并将其发送到后端服务，如果后端无法访问，则存储在本地。
- 使用 AWS 分析解决方案：使用诸如Game Analytics Pipeline之类的 AWS 服务进行可扩展的数据摄取、存储和分析，以及专业的大数据处理和分析服务。

GAMEOPS06-BP02 随着游戏规则的变化更新和调整负载测试方法

优化负载测试方法是一个持续的过程，应该随着游戏开发周期的变化而发展。随着游戏复杂性、用户群和功能集的增长，负载测试策略必须进行调整，以验证其是否准确地模拟了现实世界的条件并提供了可行的见解。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

请考虑以下事项：

缺少或过时的测试场景

在开发过程中向游戏添加新功能时，请创建并运行新的负载测试场景，以验证新功能的性能和可扩展性。同样，通常会对特性和功能进行重构以提高性能、解决玩家反馈或与新的设计目标保持一致，因此需要不断更新测试场景以跟上变化的步伐，并真正测试和反映系统的状态。

新的负载测试框架

出于各种原因，开发人员可能需要更改负载测试框架：

- 初始框架可能无法再充分模拟用户负载，也无法提供对系统性能的必要深入了解
- 新游戏功能可能需要为新协议或集成点提供负载测试支持 APIs
- 开发人员可能会想要更高级的功能，因为他们对负载测试过程越来越满意
- 优先选择更符合团队技术专长、编程语言或现有工具链的框架

通过仔细评估和随着时间的推移进行调整，开发者可以使负载测试过程与游戏不断变化的需求保持一致，并继续提供必要的见解来优化和改善整体用户体验。

优化成本

使用托管 AWS 服务的便捷性可能非常有益，尤其是在开发的早期阶段。这些服务抽象了底层基础设施管理，使团队能够快速设置解决方案，只专注于制定负载测试场景和分析结果。但是，使用托管服务通常会带来更高的成本，因为它们提供了额外的价值和便利性，例如配置、配置和维护基础架构，以及提供高可用性、扩展和监控功能。

随着团队的成熟以及对负载测试过程越来越自信和自信，有时候，自我管理基础架构可以提供额外的优化和成本节约。虽然这种亲身实践的方法会增加运营开销，但直接控制计算资源、配置、扩展行为和资

源利用率可以为微调和降低成本开辟新的机会。例如，对于团队来说，从 AWS Fargate 无服务器架构开始负载测试之旅，然后转向自我管理 Amazon EKS 集群中的底层节点可能是有意义的。

实施步骤

- 更新测试场景：持续创建和更新负载测试场景，以验证新功能和重构后的功能，并验证它们是否反映了游戏的当前状态。
- 评估负载测试框架：根据需要调整新框架，以模拟用户负载，支持新协议，并与团队的专业知识和工具链保持一致。
- 优化成本：为了便捷起见，从托管 AWS 服务开始，然后随着团队对负载测试过程的适应程度越来越高，可以考虑自我管理基础架构以节省成本。

资源

请参阅以下资源，详细了解我们与卓越运营相关的最佳实践。

文档和博客

- 游戏技术架构最佳实践
- 在 AWS pt.1 上管理你的游戏工作室
- 在 AWS pt 上管理你的游戏工作室 2
- 在 AWS pt 上管理你的游戏工作室 3
- 建立最佳实践 AWS 环境
- Control Tower 着陆区的多账户策略
- 游戏分析管道
- 利用游戏分析管道最大限度地提高您的游戏数据洞察力
- 利用 AWS Glue 亚马逊 Redshift Spectrum 获取玩家见解
- 如何在上设置 CI/CD 管道
- Good Job Games 如何通过构建管道加速 43 AWS %
- 为 Unity 移动应用程序实现构建管道
- 其他相关 CI/CD 博客
- 使用游戏服务器 CD Pipeline 博客 DevOps 让游戏变得简单
- Harmony Games 使用 AWS Cloud Development Kit (AWS CDK) ()AWS CDK部署了完全自定义的游戏后端

- GameLift为发布做准备
- 使用 Amazon Gamelift Anywhere 托管混合游戏服务器
- 使用 Amazon Gamelift Anywhere 和亚马逊 Gamelift Agent 加快游戏服务器的开发速度
- 如何使用 Amazon Gamelift 以每位玩家低于 1 美元的价格托管虚幻引擎游戏
- 新的解决方案指南，用于在上构建可扩展的跨平台游戏后端 AWS
- 在 Pragma 后端游戏引擎上对 100 万并发用户进行负载测试 AWS
- Code Wizards 是如何对英雄实验室的 Nakama 进行负载测试的 200 万并发玩家 AWS
- 组织单位的最佳实践
- AWS X-Ray
- AWS 倒计时
- AWS for Games 解决方案中心

合作伙伴解决方案

- New Relic
- Splunk APM
- 回溯.io
- 哨兵
- Datadog APM
- Honeycomb.i

白皮书

- 使用多个账户组织您的环境
- 可扩展游戏开发模式简介 AWS

视频

- YouTube系列：在上面建造游戏 AWS
- AWS 游戏版：Boss LEVEL 播客
- Re: Invent 2023： AWS 为前 1000 万用户扩大规模
- Re: Invent 2022：Riot Games 如何每天处理 20TB 的分析 AWS

- Re: Invent 2022：Riot Games 如何 AWS 构建治理报告引擎
- Re: Invent 2023：在上实现分布式设计模式 AWS
- Re: Invent 2023：使用《真人快打 1》将多人游戏扩展到数百万款
- Re: Invent 2022：Netflix 混沌工程的演变
- Re: Invent 2023：云治理的最佳实践
- Re: Invent 2023：在上创建多区域架构的最佳实践 AWS

培训材料

- 课程 — 游戏 AWS 入门第 1 部分

安全性

安全支柱包括保护信息、系统和资产的能力，同时通过风险评估和缓解来实现业务价值。由于全球知名度和大量玩家，游戏是漏洞利用者、黑客和其他寻求利用和滥用系统的方法的人的理想目标。如果没有坚实的安全基础，这通常会导致玩家体验令人失望，并增加游戏开发者的成本。

正如分担责任模型中所述，重要的是要了解安全的哪些方面是客户的责任，哪些方面是客户的责任，这样您才能做好保持强有力的安全态势的准备。 AWS 本支柱提供了最佳实践云安全指南，供您在云端开发和运营游戏时考虑。

在设计系统之前，必须制定一套安全最佳实践，其中包括访问控制。此外，您应该能够识别安全事件并保护您的系统和服务，同时通过数据保护来维护数据的机密性和完整性。您应该具备一个定义明确且经过实践的流程来响应安全事件。这些工具和技术之所以重要，是因为它们支持业务目标，例如防止财务损失或遵守监管义务。

客户示例

AnyCompany Games是一家虚构的游戏工作室，正在改善其安全状况。当有关于其直接应用的解释时，安全性可能很容易理解。 AnyCompany 本节使用游戏来说明支柱中描述的安全最佳实践

聚焦领域

- 设计原则
- 安全基础知识
- 持续的安全
- Identity and access management
- 访问控制
- 检测
- 基础设施保护
- 事件响应
- 应用程序安全性
- 实现安全自动化
- 威胁建模
- 资源

设计原则

除了 Well-Architected Framework 白皮书中安全支柱中的设计原则外，以下设计原则还可以增强云端游戏工作负载的安全性：

- 监控和审核玩家的使用行为：捕获和分析使用数据，以了解玩家如何与你的游戏和社交功能互动。通过分析这些数据，您可以检测并应对可能降低玩家体验的滥用和不当行为。

安全基础知识

GAMESEC01：如何实现游戏开发的安全基础知识？

游戏工作室需要一种独特的安全方法来保护开发环境和直播玩家服务。游戏工作室的强大 AWS 安全策略需要三个相互关联的组件：多账户结构、强身份验证和使用 IAM 策略的明确授权策略。多账户 AWS 结构使工作室能够将不同的游戏项目、开发阶段和工具环境分开。这使工作室可以更精细地控制诸如访问特定环境或服务之类的事情。启用强身份验证可以验证团队成员可以安全地访问开发资源，无论是在工作室工作还是远程工作，同时严格控制源代码、游戏版本和专有工具。Studios 还应制定明确的授权策略，以便使用 IAM 权限和角色的最低权限原则来授予权限。使用 IAM 角色在不同的开发团队角色之间分配权限，例如允许开发团队访问低级 AWS 服务，同时限制艺术家和设计师使用特定的资产管理和构建系统。这种专门的方法可以验证游戏工作室能否保护其知识产权，保持高效的开发工作流程，并安全地扩展团队，同时为开发人员提供适当的访问权限以快速迭代其项目。

最佳实践

- GAMESEC01-BP01 使用角色和联合访问权限（而不是账户根用户）对您的 AWS 环境执行操作
- GAMESEC01-BP02 AWS Control Tower 用于在上快速设置多账户环境 AWS
- GAMESEC01-BP03 使用针对特定工作职能量身定制的最低权限角色策略
- GAMESEC01-BP04 使用角色和联合访问策略以及账户级别访问策略来授予对 AWS 资源的访问权限
- GAMESEC01-BP05 使用中央身份提供商

GAMESEC01-BP01 使用角色和联合访问权限（而不是账户根用户）对您的 AWS 环境执行操作

首次创建时 AWS 账户，首先会有一个名为 root 用户的身份，该身份可使用与该帐户关联的电子邮件地址和密码进行访问。root 用户拥有对该账户中的 AWS 服务和资源的完全访问权限。在大多数情况下，应避免使用 root 用户执行 day-to-day任务。当需要根级访问时，请确认这是绝对必要的，并确认是否有额外的日志记录和护栏来跟踪其使用情况。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

在 AWS Organizations 配置中，每个账户仍有自己的根用户，但 day-to-day访问权限应改为通过 IAM 角色和 IAM Identity Center 用户进行管理。创建针对游戏生命周期阶段和团队量身定制的基于角色的访问权限。例如，直播运营团队可能需要管理游戏内活动的权限，而开发者则需要访问权限才能推送更新。与第三方服务或合作伙伴合作时，使用联合访问来实现安全的协作，而不会暴露敏感的基础架构。这种方法可以验证每个用户或合作伙伴只有他们需要的访问权限，同时维护游戏基础设施和玩家数据的安全性。

客户示例

AnyCompany 游戏在开发新游戏时实施了基于角色的访问控制。通过为不同的开发团队使用特定的 IAM 角色，他们可以避免使用共享证书。这种设置允许开发团队扮演核心游戏系统的角色，而内容团队的角色只能访问资产管理服务。

实施步骤

- 除非绝对必要，否则请勿在设置账户后使用 root 用户。创建账户，保护根用户，立即创建所需的管理 IAM 角色并将该角色分配给联合用户。
- 只有在需要执行数量有限且只有 root 用户才能执行的任务时，才使用 root 用户。这些任务的示例包括更改您的根用户电子邮件地址和更改 AWS 支持计划。

GAMESEC01-BP02 AWS Control Tower 用于在上快速设置多账户环境 AWS

如果你只用一个账号开始使用，你可能会发现 AWS 随着游戏开发过程的推进，你的游戏工作室正在从这个账户中发展出来。例如，使用单个 AWS 账户，您可能会开始达到服务限制，或者不同项目和工作负载的成本可能会变得更加复杂。为不同的游戏和环境创建不同的账户可以让团队尝试新功能，绕过服

务限制，保持安全态势和合规性。通过在中实施多账户策略 AWS，您可以从在多个账户中分配服务限额中受益，并深入了解您的 AWS 成本。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

一个常见的误解是，使用多个 AWS 账户 会自动变得更加混乱和耗时。相反，使用旨在促进多个账户管理的 AWS 服务可以帮助你的游戏工作室减少管理账户的时间。

您可以使用 AWS Control Tower 这项服务来安全地配置多账户 AWS 环境。如果您正在构建新 AWS 环境、开始旅程或完全不熟悉，建议使用 C AWS ontrol Tower AWS。在简短的设置过程中，您可以与管理账户和用户访问权限所涉及的其他 AWS 服务集成，例如 Service Catalog 和 AWS IAM Identity Center。 AWS Organizations

客户示例

AnyCompany 游戏最初是从单一游戏运行的 AWS 账户，当他们的一个游戏开发团队在一次关键的 Beta测试中达到 EC2 服务限制时，它们遇到了多个障碍。同时，他们开发另一款游戏的开发团队在自动测试管道的资源分配方面苦苦挣扎。当 AnyCompany Games无法在项目之间准确分清成本时，情况达到了临界点，这使得很难为每款游戏的开发制定预算。

AnyCompany 然后，游戏使用 AWS Control Tower实施了多账户策略。他们为每个游戏项目创建了单独的账户，具有不同的开发、质量保证和生产环境。这种账户级别的分离将每个项目的数据和资产隔离开来，因此开发一个游戏的团队无法访问或修改另一个游戏的资源。通过这种方式 AWS Organizations，他们建立了集中式计费结构，可以清楚地显示每款游戏的基础架构成本，还制定了全组织范围的访问策略。

实施步骤

- 使用 Cont AWS rol tower 设置自动化的多账户环境。
- 根据环境（例如开发、QA 和生产）整理账户。
- 使用 AWS IAM Identity Center 和 Service Catalog 来集中用户权限并简化跨账户的资源配置。

GAMESEC01-BP03 使用针对特定工作职能量身定制的最低权限角色策略

配置 IAM 策略是建立坚实安全基础的重要组成部分。在使用 IAM policy 设置权限时，请仅授予执行任务所需的许可。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。例如，质量保证团队需要在测试环境中进行更改的权限，但不应具有修改生产环境的能力。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

在探索工作负载或用例所需的权限时，您可以从托管策略等广泛权限入手。随着使用场景逐渐成熟，您可以努力减少授予许可，直至达到最低权限的标准。

实施步骤

- 遵循最低权限的做法，为用户和应用程序创建 IAM 角色。
- 使用 AWS托管策略快速提供广泛的访问权限，同时确定团队或应用程序执行任务所需的特定权限。
- Studios 还可以使用 IAM 访问分析器策略生成，根据 CloudTrail 事件生成自定义 IAM 策略，以识别 IAM 条目使用的操作和服务。
- 定期查看 IAM 策略并编辑过于宽松的策略。

GAMESEC01-BP04 使用角色和联合访问策略以及账户级别访问策略来授予对 AWS 资源的访问权限

新 AWS 用户通常仅在向他人授予访问权限时才使用 IAM 策略。但是，如果您正在使用 AWS Organizations，请考虑如何使用服务控制策略和 IAM 策略来授予工作室团队成员和承包商必要的访问权限级别。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

您可以创建 IAM 策略来允许或拒绝访问与之配合使用的 AWS 服务或 API 操作 AWS Identity and Access Management。它们只能应用于 IAM 身份，例如用户、群组或角色。例如，可以使用 IAM 策略为用户提供对 Amazon S3 的只读访问权限。

服务控制策略 (SCPs) 是您的护栏。 AWS 账户 SCP 不授予权限，而是用来限制个人成员账户对 AWS 服务的操作。例如，SCP 可以拒绝访问特定区域。 AWS 账户

采取行动时，将结合评估相关的 IAM 策略和 SCPs。接着上一个示例，如果角色尝试运行 EC2 实例，IAM 表示他们被允许（“允许” 对于 ec2:RunInstances），然后 SCPs 将决定他们选择的区域是否有效（允许 “us-east-1”，但 SCP 拒绝了 “us-west-1”）。

对 IAM 策略进行分层， SCPs 可以验证访问您的 AWS 资源的任何人只会获得他们所需的相应权限。考虑到您的 AWS 账户 和资源是否跨越多个区域，但并非游戏工作室中的每个人都需要访问所有区域，这一点尤其重要。

您可以定制 IAM 政策，向特定团队授予更新游戏配置、管理玩家数据、配置促销活动和审核用户生成的内容等内容的特定权限。同时，您可以使用 SCPs 来强制执行对游戏运营至关重要的组织范围的控制。这些措施可能包括将部署限制在游戏运营的经批准的区域，帮助防止未经授权访问敏感的玩家数据存储，强制执行合规要求，以及通过限制开发账户之间的服务使用来控制成本。

实施步骤

- 使用 IAM 策略管理个人用户、群组或角色的权限。
- 使用中的服务控制策略 (SCPs) AWS Organizations 来强制执行账户级权限。
- 结合 IAM 政策，仅 SCPs 向特定用户和账户授予所需的访问权限。

资源

- AWS IAM 中的策略和权限
- 服务控制策略
- 针对工作职能的AWS 托管式策略

GAMESEC01-BP05 使用中央身份提供商

中央身份提供者充当存储和管理用户证书、身份、权限和身份验证的单一来源。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

使用中央身份提供商来简化您的用户身份验证流程，实施一致的安全策略，并简化您 AWS 账户 和应用程序中的用户管理。采用集中式方法无需单独管理用户身份和证书，从而降低出现不一致、冗余和其他安全漏洞的风险。将用户身份和身份验证整合到一个地方，还可以提高整个 AWS 环境的可见性、控制和可审计性。

客户示例

AnyCompany 游戏在管理其快速扩展 AWS 的基础架构中的开发者访问权限方面面临重大挑战。他们的开发团队从50人增加到200人，涉及三个主要游戏。最初，每个项目团队都管理自己的 AWS 访问凭证，导致安全实践不一致，新开发人员入职延迟，偶尔还会发生安全事件。

该工作室 AWS 将 IAM Identity Center 作为其中央身份提供商，将用户管理整合到一个系统中。他们将其与现有的公司名录关联起来，使开发人员能够使用相同的公司凭据进行 AWS 访问。现在，开发人员使用他们现有的单一公司登录名来获得完成工作所需的 AWS 访问权限

实施步骤

- 考虑使用 AWS IAM 身份中心作为您的中央身份提供商。这为您提供了一致的访问管理 AWS 账户，为您的员工提供了单点登录身份验证，并简化了对 AWS 应用程序的用户访问审计。IAM Identity Center 还可以连接来自受支持的身份提供商的现有企业身份。

持续的安全

GAMESEC02：如何实现、维护和监控持续的安全最佳实践？

对于各行各业的企业而言，遵守最佳安全实践至关重要，尤其是在游戏行业。游戏行业依赖于培养和维持玩家的信任以及建立良好的声誉，即使是轻微的安全问题也可能很快削弱这种信心。

此外，游戏行业的全球性质要求游戏提供地区遵守管理数据保护、消费者隐私和安全的各种行业法规和标准。公平和安全的游戏玩法是另一个关键方面，它凸显了强有力的安全措施的重要性。作弊、黑客攻击和其他形式的游戏利用可能会破坏合法玩家的游戏体验，这使得强大的安全控制对于维护游戏的完整性和为参与者营造公平的竞争环境至关重要。

最佳实践

- GAMESEC02-BP01 使用随时可部署的模板进行标准安全实践
- GAMESEC02-BP02 发生安全事件时使用自动补救技术

GAMESEC02-BP01 使用随时可部署的模板进行标准安全实践

Ready-to-deploy模板提供了一种主动而敏捷的方式来评估您在云中的安全状况。预先配置的模板会评估您的云安全性，并及时实施必要的更改。这些模板包含各种技术和广泛接受的安全框架的广泛最佳实践。使用模板可以帮助游戏工作室保持一致的基础架构配置，尤其是在他们可能会扩展和添加更多内容 AWS 账户 以支持新的工作负载时。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

通过使用 AWS 服务和实施 ready-to-deploy模板，游戏开发者可以主动评估和加强其云安全态势，保护其知识产权，保护玩家数据，并通过定期安全评估和持续监控来快速识别和解决潜在的漏洞，从而营造安全的游戏环境。

客户示例

AnyCompany 游戏在准备在欧洲行业推出下一款游戏时面临着重大挑战。他们意识到他们现有的数据处理做法不符合 GDPR 的要求。他们求助于 AWS Security Hub CSPM AWS Config 和及其 ready-to-deploy模板来寻找解决方案。该团队实施了特定于GDPR的合规包 AWS Config，该合规包根据GDPR标准自动评估了其现有基础架构。最初的扫描发现了几个关键差距，例如数据保留政策不当以及对玩家数据存储位置的访问控制不足。使用模板的预定义规则， AnyCompany Games 迅速实施了必要的更改。此外，该模板提供的持续自动合规检查使这个小团队能够毫不费力地保持GDPR合规性，即使他们继续更新和扩展游戏范围。

实施步骤

- 使用标准安全实践的模板，例如托管规则和一致性包 AWS Config 以及中的 AWS Security Hub CSPM标准。
- 查看 Sec urity Hub CSPM 标准的详细信息，以确定哪些标准最符合游戏工作室的安全需求。

GAMESEC02-BP02 发生安全事件时使用自动补救技术

使用自动修复技术，游戏开发者可以主动保护和维护其游戏基础设施，并将安全事件可能产生的潜在影响降至最低。如果检测到安全问题，请使用运行手册来指导您应对这种情况。尽可能自动执行这些响应，以更快地修复问题并减少其影响。这减少了停机时间和游戏中断的可能性，从而改善了玩家的体验。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

为应对安全问题做好准备不仅可以保护玩家的体验，还可以满足各种合规和监管标准。此外，随着工作负载的扩大，使用自动安全响应可以扩展您的安全运营。 AWS 提供服务以帮助识别和自动响应这些事件。

客户示例

AnyCompany 游戏面临严重的安全事件，其中包含即将推出的游戏的未发布角色模型和纹理的 S3 存储桶在例行资产管道更新期间意外公开。自动安全系统在修改后的几分钟内就检测到了存储桶权限的更改。系统立即执行了修复操作手册：将存储桶恢复为私有状态，记录暴露窗口期间的访问尝试，通知安全团队，并创建权限变更的详细 CloudTrail 日志。

实施步骤

- 使用 AWS解决方案上的自动安全响应来实施自动化操作手册，这些操作手册定义了为响应中的安全事件而自动采取的操作。 AWS Security Hub CSPM

资源

- AWS for Games 博客 — 管理你的游戏工作室 AWS：第 1 部分
- AWS for Games 博客 — AWS 第 2 部分管理你的游戏工作室
- 向注册现有组织单位 AWS Control Tower
- AWS 账户 root 用户
- 需要 root 用户凭证的任务
- 开启自动安全响应 AWS

Identity and access management

GAMESEC03：如何管理玩家身份和访问管理？

开发游戏时，您必须确定如何为玩家提供对您的游戏和相关服务的访问权限。下一节探讨了设计注意事项，包括玩家身份验证、授权和多因素身份验证。

最佳实践

- GAMESEC03-BP01 确定识别和控制玩家对游戏环境和资源的访问权限的方法
- GAMESEC03-BP02 对发送到游戏后端服务的请求进行身份验证
- GAMESEC03-BP03 使用您的游戏后端服务验证玩家加入多人游戏的请求
- GAMESEC03-BP04 要求使用强密码，对玩家用户账户实施严格的安全政策
- GAMESEC03-BP05 为玩家提供在账户上设置多重身份验证 (MFA) 的选项

GAMESEC03-BP01 确定识别和控制玩家对游戏环境和资源的访问权限的方法

这一决定受您的玩家获取和盈利策略、玩家体验以及其他因素的影响，例如您的游戏发行合作伙伴可能提供的现有功能。例如，游戏可能需要购买，并要求玩家创建用户个人资料以将真实货币支付方式与其账户关联起来。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

或者，游戏可能希望通过取消在玩游戏之前创建用户帐户的必要性来减少首次体验玩家体验的进入门槛，从而提高玩家首次尝试游戏的机会。通常，游戏会为其游戏实现一种或多种玩家身份和访问管理方法的组合。

未经身份验证或匿名访问

在游戏不需要玩家在社交网络和游戏系统上创建新的用户帐户或与其身份关联的情况下，此访问级别非常有用。这是玩家开始玩游戏的最简单、最快捷的方式，在游戏开发者可能希望降低初始体验的进入门槛的手机游戏中特别有用。

在这种访问场景中，如果您想从游戏安装中识别使用情况，则可以对游戏客户端进行编程，以生成唯一标识符并将其存储到玩家的设备上。此唯一标识符用于在设备上的游戏会话中识别玩家，并允许分析报告一段时间内的使用情况。 之后，如果玩家选择创建账户，你可以将他们的新用户账户与他们之前生成的唯一标识符相关联。这将把他们的新玩家身份与他们的历史使用情况联系起来，其中可能包括统计数据和游戏成就。

如果玩家最终没有创建和关联帐户，则可以唯一识别该玩家用于与游戏互动的设备，但不会收集和存储有关该玩家的可恢复信息。因此，如果玩家损坏或丢失了设备，则先前存储的与该设备相关的数据也会丢失，并且可能无法恢复。

使用用户名和密码进行身份验证

游戏可能允许玩家使用存储在游戏后端的用户名和密码创建自己的用户帐户。 当游戏开发者与已经拥有开发者可以集成的现有玩家账户系统的游戏发行商合作时，可能会发生这种情况。或者，发布自己的游戏的开发者可能希望通过允许玩家创建一个用户帐户来访问他们发布的游戏，从而简化玩家体验。

与第三方社交网络和游戏系统的身份验证和账户关联

网络游戏和具有社交功能的游戏通常会提供第三方身份提供商联合以简化玩家体验。与其要求玩家创建用户名和密码组合进行身份验证，不如使用身份联合来允许玩家使用社交网络和游戏系统的第三方帐户

进行身份验证。此登录过程简化了玩家的登录和注册体验。它还为强制创建帐户提供了一种便捷的替代方案，并为玩家提供了一种轻松访问游戏的方法。

对于游戏开发者来说，联合登录流程可以简化玩家验证工作流程。它还可以提供一种更可靠的方式来管理用于个性化的玩家数据。这是因为你不需要要求玩家向你提供他们可能已经向第三方身份提供商提供的某些数据。此外，这些系统还提供与其他社交功能的集成，例如能够将玩家与朋友联系起来。

实施步骤

- 使用未经身份验证或匿名的访问权限，通过生成唯一的设备标识符来跟踪使用情况，并在以后启用账户关联，从而减少首次玩游戏者的障碍。
- 使用现有的玩家账户系统对专用用户账户实施用户名和密码身份验证，或者在游戏中创建统一的体验。
- 集成第三方身份提供商以进行联合身份验证，简化登录流程并允许访问社交功能和个性化数据。

GAMESEC03-BP02 对发送到游戏后端服务的请求进行身份验证

对发送到游戏后端服务的请求进行身份验证可能会阻止不需要的请求成功。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

您应提供身份验证服务供玩家登录，当玩家成功进行身份验证时，该服务应向游戏客户端返回安全的短期令牌，例如 JSON Web Token (JWT)。

这些代币可以包括包含玩家属性和其他相关元数据的声明断言。这些相关的元数据可用于后续请求，这些请求从游戏客户端发送到您的游戏后端，用于对请求进行身份验证，并在经过身份验证的玩家的上下文中对其进行授权。

您可以选择设计和构建自己的玩家身份验证系统（这需要持续改进和维护），也可以使用 Amazon Cogn ito 提供的可扩展且安全的用户注册、登录和访问控制功能。

Amazon Cognito 用户池包含用于身份验证和授权的用户目录。用户池提供用户池 APIs ，您可以将其集成到游戏中，用于注册、登录和密码重置工作流程，这些工作流程可以与第三方身份提供商集成。应用程序负载均衡器和 Amazon API Gateway 都提供与 Cognito 的集成，以集成用户身份验证，用于发送到使用这些服务托管的自定义游戏后端的请求。

如果您的游戏支持匿名访问且您无法对玩家进行身份验证，则在与游戏后端集成时，您可以使用客户端身份验证方法提供更安全的体验。如果您的游戏客户端直接使用 AWS 服务，则必须使用凭据对这些服

务的请求进行签名。要为未经身份验证的用户提供游戏客户端证书，您可以使用 AWS 软件开发工具包从 Amazon Cognito 身份池中检索短期证书，这些证书可用于签署您对服务的请求。 AWS 这些凭据可以从您的游戏客户端刷新。

除了直接从游戏客户端与 AWS SDK 集成外，您还可以使用支持自定义授权的 Amazon API Gatew ay 等服务来构建自己的游戏后端。通过设计自己的游戏后端服务，您可以使用自定义服务器端逻辑对请求进行权威控制。

有关为使用 Amazon 托管的游戏构建后端服务的更多信息 GameLift，请参阅设计您的游戏客户端服务。

客户示例

AnyCompany 游戏采用托管身份验证和授权方法，增强了下一款游戏的安全性。他们没有维护自定义的用户名和密码系统，而是使用 Amazon Cognito 用户池来处理玩家的注册和登录，使用身份池来支持在创建账户之前尝试训练模式的玩家进行匿名访问。他们还在游戏中实现了自定义授权逻辑，以识别在 Cognito 中定义的管理员角色，从而允许这些用户访问特殊的游戏内管理功能。

实施步骤

- 使用 Amazon Cognito 用户池使用安全令牌管理身份验证 JWTs，例如启用注册、登录和密码重置等功能。
- 从 Amazon Cognito 身份池中检索短期证书，供匿名用户安全地与服务交互 AWS 。
- 使用 Amazon API Gateway 实现自定义游戏后端，实现自定义服务器端身份验证逻辑。

GAMESEC03-BP03 使用您的游戏后端服务验证玩家加入多人游戏的请求

通常，在多人游戏中，玩家可以通过直接从可用会话列表中选择一个选项来加入游戏会话，或者他们将提交查找匹配项的请求。后一种方法要求游戏开发者负责找到符合条件的游戏会话，并将连接信息（通常是 IP 地址和端口号）提供给玩家的游戏客户端。实现方式可能因你正在开发的游戏类型而异，但无论如何，对玩家加入游戏的请求进行服务器端验证是一种安全最佳实践。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

例如，在基于会话的多人游戏中，玩家提出的加入游戏会话的请求应由游戏服务器软件与游戏后端配对服务进行验证，然后再授权他们连接到服务器。当玩家请求加入游戏会话时，游戏服务器应检查请求以

获取唯一标识符，例如玩家会话 ID 和服务器生成的票证，这些票证之前由您的游戏后端配对服务提供给游戏客户端。

启动与游戏服务器的连接后，您的服务器端软件可以使用此信息向配对服务验证玩家的连接请求是否有效，并确认该玩家没有加入之前在游戏会话中为其他玩家保留的位置。

对于托管在 Amazon 上的游戏 GameLift，请参阅与 Amazon GameLift 服务器的游戏 client/server 互动，查看如何实施此类服务器端验证的示例。

客户示例

在 AnyCompany Games的首次测试版发布期间，他们发现玩家通过直接连接到游戏服务器来绕过配对系统，这导致了严重的竞争诚信问题。当排名靠前的玩家发现他们可以与朋友共享服务器IP地址时，他们开始规避基于技能的配对系统，导致经验丰富的玩家加入新手比赛，为新玩家创造了令人沮丧的体验。 AnyCompany 游戏通过实现服务器端验证系统来回应，该系统为每个配对请求生成唯一的会话门票。该系统要求玩家 IDs 和配对请求门票，并针对其后端配对服务进行验证的连接尝试。

实施步骤

- 使用玩家会话 IDs 和服务器生成的门票等唯一标识符在服务器端验证玩家加入请求。
- 确认与配对服务的连接请求的有效性，以阻止未经授权的访问。
- 验证过程中，确认未经授权的玩家不会访问游戏会话中的预留位置。

GAMESEC03-BP04 要求使用强密码，对玩家用户账户实施严格的安全政策

如果游戏允许玩家使用密码创建用户帐户，则应要求玩家的密码遵守严格的政策。例如，Amazon Cognito 用户池使您能够为用户账户定义密码要求。建立强有力的密码政策可以保护你的玩家的账号不被社交工程和暴力攻击所取代。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

客户示例

AnyCompany 由于密码政策薄弱，游戏的热门游戏经历了一波账户劫持浪潮，因此面临着危机。使用 “password123” 等简单密码的玩家正在成为自动暴力攻击的受害者，导致物品丢失和游戏内货币受损。为了解决这个问题， AnyCompany Games改进了登录系统，要求以前不要使用密码，密码必须至少包含一个大写字母、一个数字、一个特殊字符和最少15个字符。

实施步骤

- 要求为玩家账户设置严格的密码策略以增强安全性。
- 使用 Amazon Cognito 用户池来定义和强制执行密码要求。

GAMESEC03-BP05 为玩家提供在账户上设置多重身份验证 (MFA) 的选项

玩家账户可以成为不良行为者的资产，尤其是在支持游戏内货币和购买的游戏中。由于玩家账户黑客攻击和社会工程学攻击的普遍存在，为玩家提供通过配置多因素身份验证 (MFA) 来增强其账户安全的选项。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

当玩家尝试使用MFA访问其账户时，临时密码会发送到他们的电子邮件地址、电话号码或专门构建的多因素身份验证移动应用程序。要成功进行身份验证，玩家必须在有限的时间内将密码输入登录系统。

MFA 还可用于帮助保护试图从新的地理位置进行身份验证的账户、被玩家支持标记为潜在恶意活动的账户，甚至是长时间未登录游戏的账户。

例如，Amazon Cognito 用户池可以在用户目录上配置多重身份验证。

实施步骤

- 启用多重身份验证 (MFA) 以增强玩家账户安全。
- 使用通过电子邮件、电话或 MFA 应用程序发送的临时代码来验证账户访问权限。
- 为新的地理位置、被举报的账户或长时间处于非活动状态的账户申请 MFA。

访问控制

GAMESEC04：如何阻止对游戏内容的未经授权的访问？

现代游戏包含大量内容，例如可下载内容 (DLC)，这是玩家参与度和游戏盈利的重要方面。玩家期望不断出现新角色、关卡和挑战，这要求游戏开发者跟上对新内容的持续需求，以留住玩家。内容的种类和

大小可能会有很大差异，具体取决于游戏的类型和玩游戏的设备。无论游戏采用何种系统，都要保护游戏内容免遭未经授权的访问。

最佳实践

- GAMESEC04-BP01 限制授权客户和用户访问可下载内容
- GAMESEC04-BP02 限制源站对授权内容分发网络的访问权限 (CDNs)
- GAMESEC04-BP03 实施地理限制以限制未经授权的访问
- GAMESEC04-BP04 使用数字版权管理 (DRM) 解决方案限制对内容的访问

GAMESEC04-BP01 限制授权客户和用户访问可下载内容

限制授权应用程序和客户端对游戏内容的访问。考虑将 Amazon S3 用作存储可下载游戏内容的经济高效且可扩展的来源，并使用 Amazon CloudFront 为玩家提供全球高性能的内容交付。这两项服务都提供了用于限制对存储数据的访问的内置机制，例如限制经过身份验证的用户的访问权限。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

授予访问存储在 Amazon S3 中的内容的权限

当您需要授予对存储在 S3 中的内容的访问权限时，需要考虑几个因素。默认情况下，只有创建了 AWS 账户 S3 存储桶的用户才能访问存储在其中的对象。要授予对内部应用程序的访问权限和管理存储在 Amazon S3 存储桶中的内容，请使用 AWS Identity and Access Management (IAM) 创建提供适当访问权限的策略。

IAM 角色可以与托管在 Amazon 等服务中的联合用户、系统或应用程序以及托管在 Amazon EC2 EKS 和 Amazon ECS 中的基于容器的应用程序相关联。 AWS Lambda例如，您可以使用 AWS SDK 或 AWS CLI 在 S3 存储桶中发布和管理游戏内容资产。为了支持此用例，您可以创建一个 IAM 角色，该角色具有相应的访问权限，可以将游戏内容读取和写入您的 S3 存储桶，并将其与托管您的软件和脚本的 EC2 实例关联起来。

可以为您的存储桶和特定对象定义基于资源的策略。S3 存储桶策略与 S3 存储桶相关联，可用于限制对该存储桶及其中的对象的访问，以及授予从其他账户访问您的 Amazon S3 资源的权限。例如，如果多个团队或不同的游戏开发工作室正在开发相同的游戏内容，并且需要对 Amazon S3 中集中托管的内容具有相同的访问权限，则可以使用 S3 存储桶策略来定义跨账户访问 S3 资源的权限。考虑使用 S3 接入点，它可以通过创建具有特定于每个应用程序或应用程序集的名称和权限的访问点来简化对共享数据访问的管理。Amazon S3 文档包含在 Amaz on S3 中进行访问控制的其他最佳实践。

Granting short-term access to your content

如果只需要在特定的有限时间内进行访问，请生成临时 URLs 内容，以授予对您的内容的短期访问权限。Amazon S3 支持生成预签名 URLs，允许对象拥有者授予对 Amazon S3 中对象的限时访问权限，而无需更新您的存储桶策略。 这样，获得访问权限的最终用户或应用程序无需拥有账户或 IAM 权限，而是使用预签名 URL 来访问内容。

这是一种通常用于各种游戏用例的最佳实践，例如授予授权玩家访问他们有权获得的可下载内容的权限，以及提供对限时游戏内容的临时访问权限。预签名还 URLs 可用于提供将内容上传到 S3 存储桶的临时权限。例如，您可以考虑使用预签名 URL 为玩家提供上传客户端日志的权限，以帮助您的支持团队对玩家支持案例进行故障排除。

使用内容分发网络提供对您的内容的访问权限

虽然出于开发和管理目的，您的应用程序、游戏开发人员、美术师和其他人员可能需要直接访问 S3 存储桶中的内容，但请使用内容分发网络提供对玩家或其他用户通过互联网公开可用的内容的访问权限。这种方法通过缓存经常访问的内容来提高下载性能并降低成本。亚马逊 CloudFront 可以在全球范围内分发您的内容，方法是缓存您的内容并将其交付到离玩家更近的地方，同时减少游戏下载来源（例如 Amazon S3）的负载。

建议不要直接从 S3 存储桶中提供您的公开内容，而是使用 CloudFront将这些内容保密，然后将其公之于众。 CloudFront 可以配置为要求玩家使用签名 URLs或签名 Cookie 访问您的私人内容（例如仅限付费玩家下载新游戏）。然后，您可以开发您的应用程序，以创建并分发给经过身份验证的用户，或者发送 URLs 为经过身份验证的用户设置签名 Cookie 的 set-cookie 标头。当您创建签名 URLs 或签名 Cookie 来控制对文件的访问时，您可以指定结束日期和时间，之后网址和 Cookie 将不再有效。

或者，您还可以指定可用于访问您的内容的计算机的 IP 地址或地址范围，如果您想限制对特定游戏开发工作室合作伙伴或承包商网络的访问，这将非常有用。如果您想提供对多个受限文件的访问权限，或者不想更改当前文件，请使用签名 Cookie URLs。如果您 URLs 想限制对单个文件的访问权限，或者您的用户正在使用不支持 Cookie 的客户端，请使用已签名。签名 URLs 优先于签名的 Cookie。

实施步骤

- 使用 IAM 角色和存储桶策略为内部应用程序、团队或跨账户场景授予对 S3 存储桶的适当访问权限。
- 生成预签名 URLs 以授予对 S3 对象的短期访问权限，适用于可下载内容或临时上传，例如客户端日志。
- 使用 CloudFront 带有签名 URLs 或 Cookie 的 Amazon，可以更安全地向经过身份验证的用户提供私有内容

GAMESEC04-BP02 限制源站对授权内容分发网络的访问权限 (CDNs)

阻止用户绕过您的内容分发网络，直接访问来自您的来源的内容，例如您的 Amazon S3 存储桶。只有经过授权的人才能访问您的来源，这一点很重要 CDNs，这样可以减少不必要地从源站提供内容所产生的数据传输成本。它还可以通过同一个入口点让公众访问您的原始内容来改善您的安全状况，您可以在其中部署边缘安全控制，例如 AWS WAF 第 7 层过滤、注入和检查与安全相关的 HTTP 请求参数以及分布式拒绝服务 (DDoS) 保护。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

要对 Amazon S3 源实施这些控制，您可以使用亚马逊 CloudFront 源访问身份 (OAI)，该身份验证对您的 S3 对象的请求是否来自您的 CloudFront 分配。 AWS WAF 与您的 CloudFront分配关联以提供第 7 层筛选。但是，如果您提供来自其他内容的内容 CDNs，则可以将 CDN 配置为在原始请求中插入一个或多个自定义 HTTP 标头，通过检查这些标头 AWS WAF 来验证传入流量是否来自您的授权 CDN 提供商。

当您的源托管在 Application Loa d Balancer (ALB) 后面时，这种方法还有助于防止用户绕过您的 CDN 提供商。 ALBs 可以与关联 AWS WAF 以获得第 7 层保护。您可以配置 AWS WAF 为插入自定义 HTTP 标头，ALB 将对其进行检查，以处理和检查流向负载均衡器的传入流量。 AWS WAF

客户示例

AnyCompany 游戏实施了原始访问限制，以帮助保护其游戏资产、可下载内容和补丁文件免受未经授权的直接访问，这可能会使玩家在没有适当身份验证的情况下绕过安全检查或获取优质内容。这种方法使他们能够通过集中点监控内容访问模式，从而可以直接识别可能表明存在协调攻击或未经授权的内容再分发的可疑下载行为。

实施步骤

- 使用 Amazon CloudFront 原始访问身份 (OAI) 限制对 S3 对象的直接访问
- AWS WAF 与 CloudFront 或 ALB 关联以提供第 7 层过滤并帮助防御 DDo S 攻击和恶意请求。
- 在 Cloudfront 中配置自定义 HTTP 标头，以验证传入流量是否来自授权来源。

GAMESEC04-BP03 实施地理限制以限制未经授权的访问

当玩家请求您的内容时，无论玩家身在何处，Amazon 都会从最近的边缘站点 CloudFront 提供所请求的内容。但是，在某些情况下，您可能需要限制世界特定地区的用户访问您的内容的方式。例如，您可

能有一个滚动游戏部署策略，可以分阶段发布内容 country-by-country，或者您可能必须遵守特定国家/地区的访问控制。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

您可以使用地理限制（也称为地理封锁）来阻止特定地理位置的玩家访问您通过 CloudFront 发行版分发的内容。此功能允许您限制对与发行版关联的文件的访问权限，并限制在国家/地区级别的访问权限。或者，您可以使用第三方地理定位服务来限制对与发行版关联的文件的子集的访问权限，或者以比国家/地区级别更精细的精度限制访问权限。

通过使用 CloudFront 地理限制，您可以允许玩家仅在已批准的国家/地区允许列表中的国家之一的情况下访问您的内容。如果您的玩家所在的国家/地区被列入禁止的国家/地区之一，您也可以阻止他们访问您的内容。如果收到来自被屏蔽的地理位置的请求，则 CloudFront 会向玩家返回 403 Forbidden HTTP 状态码。值得注意的是，这对于非敏感内容非常有效，不应用作个人身份信息或敏感游戏工件的独立保护。

实施步骤

- 根据国家级允许或拒绝列表，使用 CloudFront 地理限制来允许或拒绝内容访问。
- 对于来自被屏蔽的地理位置的请求，返回 403 禁止 HTTP 状态码。
- 避免仅仅依靠地理限制来保护敏感内容或 PII

GAMESEC04-BP04 使用数字版权管理 (DRM) 解决方案限制对内容的访问

考虑使用强大的加密工具（例如数字版权管理 (DRM) 解决方案）来限制对游戏内容的访问。这种类型的解决方案可用于加密您的私人内容并将解密密钥分发给授权玩家。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

如果您希望允许玩家尽早下载游戏内容，但又不希望他们在预定时间之前能够访问或玩游戏内容，则建议使用DRM解决方案。例如，在允许玩家预订游戏并将其游戏客户端配置为尽早自动开始下载加密文件的情况下，这种情况很常见。该策略用于验证游戏是否已下载并准备在游戏正式发布后随时可以玩。游戏发布后，玩家的游戏客户端可以向 DRM 后端解决方案请求解密密钥，这样它就可以解密之前下载的文件并开始玩游戏。

在授权玩家下载和安装游戏后，DRM 系统还用于阻止未经授权的重新分发和操纵游戏。DRM 系统需要与源站集成，以便交换加密密钥并授权玩家检索解密密钥。商业 DRM 提供商提供了一系列解决方案，这些解决方案具有针对不同设备的功能和支持。

实施步骤

- 使用 DRM 解决方案加密私人游戏内容，并将解密密钥分发给授权玩家。
- 为预购游戏启用加密文件的预下载，在发布时使用解密密钥解锁访问权限。
- 将 DRM 系统与源站集成，以管理加密密钥并阻止未经授权的内容再分发或操纵。

检测

GAMESEC05：如何监控和分析游戏中玩家的使用行为？

监控和分析玩家的使用行为对游戏工作室至关重要，因为它使您能够检测安全威胁、作弊行为和其他可能危及游戏完整性和玩家安全的滥用行为。通过跟踪异常进度、异常游戏内交易或可疑通信行为等模式，你可以在潜在的作弊者、欺诈账户或协调威胁严重影响玩家体验之前识别出它们。

最佳实践

- GAMESEC05-BP01 实施全面的数据收集策略来监控玩家行为
- GAMESEC05-BP02 收集、存储和分析玩家使用日志，以检测不当行为

GAMESEC05-BP01 实施全面的数据收集策略来监控玩家行为

要保持积极的玩家体验，请实施全面的数据收集和分析策略。捕获、存储和分析相关数据可以深入了解玩家如何与你的游戏功能以及彼此之间的互动。这种数据驱动的方法可以指导决策，提高玩家的参与度和留存率，优化盈利策略，并最终改善整体玩家体验。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

实施数据收集系统以捕获和记录相关的玩家行为，例如游戏会话、进度、成就、购买、与游戏元素的互动以及社交活动。收集服务器端数据，例如服务器负载、网络流量和错误日志，以监控技术性能并识别潜在问题。通过调查、论坛、支持票证和社交媒体渠道收集玩家反馈，以了解他们的体验和偏好。

存储游戏数据时，请建立集中式数据仓库或数据湖来存储和组织收集的数据，并实施数据清理、转换和聚合管道，为高效分析做好准备。

存储数据后，通过数据可视化工具对其进行分析，以获得诸如玩家留存率和流失率、获利策略以及功能使用情况等见解。

实施步骤

- 捕获和记录玩家操作、服务器端指标和反馈，以监控互动和技术性能。
- 使用像 Amazon Redshift 或 S3 数据湖这样的集中式数据仓库来存储、清理、转换和整理游戏数据以供分析。
- 使用 Amazon Quicksight 等可视化工具分析收集的数据，深入了解玩家留存率、盈利和功能使用情况。

GAMESEC05-BP02 收集、存储和分析玩家使用日志，以检测不当行为

使用您的游戏来收集日志，以了解玩家如何使用您的游戏功能以及他们如何与其他玩家互动。然后，您可以屏蔽可能降低玩家体验的未经授权的活动。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

使用诸如亚马逊日志或亚马逊 OpenSearch 服务之类的日志解决方案，或者通过 AWS 合作伙伴提供的解决方案（例如 Datadog、Sumo Logic、Ne w Rel i c、H oneycomb.io 或 Splunk）将结构化 CloudWatch 日志事件发送到游戏分析管道。整理这些玩家使用日志，使其可用于检测何时需要调查玩家的具体行为。

捕获数据后，可以考虑使用工具来检测不当使用行为。例如，如果您的游戏具有社交功能，例如游戏内玩家消息、语音聊天或在线论坛，请将这些玩家参与的日志保存为一种可以进行分析以进行审核的格式。

配置游戏的语音聊天功能，将录音导出到 Amazon S3，并使用 Amazon Transcribe 将音频语音转换为可以存储以供处理的文本格式。或者，您可以通过将游戏后端语音聊天服务直接与 Transcribe API 集成来执行实时流式转录，从而实时转录流式音频。审核团队可以手动审核内容，当内容采用标准格式后，您还可以使用 AWS AI/ML 服务自动进行审核。Amaz on Comprehend 可用于执行自然语言处理 (NLP)，从非结构化文本中发现信息，这样可以将对话分类和组织为相关主题，并识别亵渎等不当行为。

实施步骤

- 收集、存储和分析玩家使用日志。
- 使用人工智能和机器学习 AWS 服务，更有效地查看和深入了解您的玩家使用日志。

基础设施保护

请参阅 Well-Architected Framework 白皮书，了解适用于游戏工作负载的基础架构安全保护的最佳实践。

GAMESEC06：您如何监控和应对基础设施威胁？

监控和应对基础设施威胁对游戏工作室至关重要，因为他们的基础设施是支持数百万并发玩家、处理真钱交易以及存储宝贵的玩家数据和专有游戏内容的支柱。游戏工作室必须实施监控系统和事件响应流程，以保持玩家体验和业务运营的完整性。

最佳实践

- GAMESEC06-BP01 使用工具检测和应对基础设施面临的威胁
- GAMESEC06-BP02 使用人工智能和机器学习工具自动执行基础架构保护策略的各个方面
- GAMESEC06-BP03 利用来自系统级日志的见解来持续改进您的基础架构保护策略

GAMESEC06-BP01 使用工具检测和应对基础设施面临的威胁

要持续监控您 AWS 环境中的恶意活动和未经授权的行为，请考虑使用 Amazon GuardDuty。GuardDuty 通过监控您环境中的账户行为、网络活动和数据访问模式来识别威胁。它会分析多个数据源中的事件，例如 CloudTrail 事件日志、Amazon VPC 流日志和 DNS 日志，以发现潜在威胁。通过与 Amazon E CloudWatch vents 和 Lambda 集成， GuardDuty 警报可以自动转发给相关的安全团队进行进一步分析。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

AWS Security Hub CSPM全面了解您的安全状态， AWS 并根据安全行业标准和最佳实践检查您的环境。Security Hub CSPM 从各种 AWS 账户服务和支持的第三方合作伙伴产品中收集安全数据，分析您

的安全趋势并确定优先级最高的安全问题。亚马逊与 Security Hub CSPM 的 GuardDuty集成使您可以将调查结果从 GuardDuty 发送到 Security Hub CSPM。然后，Security Hub CSPM 可以将这些发现纳入其对您的安全态势的分析中。

不良行为者通常会雇用机器人来接管账户并在游戏中作弊。 WAF Bot Control 可让您查看和控制常见且普遍存在的机器人流量，这些流量可能会消耗多余资源、歪曲指标、导致停机或执行其他不良活动。

勒索软件是一种恶意代码，旨在未经授权访问系统和数据集，并对这些数据进行加密以阻止合法玩家的访问。在勒索软件将玩家锁定在系统之外并加密了他们的敏感数据之后，网络犯罪分子要求支付赎金，然后再提供解密密钥来解锁数据。Organizations 可能因恶意事件而完全关闭，从而造成巨额成本和业务生产力损失。有关最佳实践，请参阅保护您的 AWS 云 环境免受勒索软件的侵害，以增强在事件发生之前、期间和之后抵御勒索软件的能力。

您的游戏可能使玩家能够通过呼叫中心（例如 Amazon Connect）或使用Amazon Lex的聊天机器人联系玩家支持代理。Amazon Connect 支持监控实时对话和录制的对话。要分析玩家与使用 Amazon Lex 构建的玩家支持聊天机器人之间的互动，您可以将这些互动的对话日志存储在 Amazon Logs 中，这些 CloudWatch 日志可以导出到 Amazon S3 并按照前面所述进行分析。

最后，作为基础架构保护策略的一部分，进行渗透测试练习。无论您是在内部还是通过 AWS 合作伙伴进行这些评估，都要遵守渗透测试的AWS 客户支持政策。

实施步骤

- 使用 Amazon GuardDuty 监控账户行为、网络活动和数据访问模式中是否存在威胁，并与 Security Hub CSPM 集成以获得统一的安全视图。
- 实施 AWS WAF Bot Control 以帮助检测和缓解可能损害资源和玩家体验的机器人流量。
- 定期进行渗透测试练习，遵守 AWS 客户支持政策，以评估和加强您的安全状况。

GAMESEC06-BP02 使用人工智能和机器学习工具自动执行基础架构保护策略的各个方面

Amazon Lookout for Metrics 使用机器学习来自动检测和诊断您的业务和运营数据中的异常，并以更快的速度和准确度监控对您的业务最重要的指标。该服务还可以直接诊断异常的根本原因，例如收入突然下降、登录次数、交易量或留存率。它不需要游戏开发者具有机器学习经验即可进行设置，并且可以连接到流行的数据源，包括亚马逊 S3、亚马逊、亚马逊 RDS、Amazon Rds CloudWatch、Amazon Redshift 以及许多 SaaS 应用程序。例如，您可以将 Amazon Lookout for Metrics 与游戏分析管道和其他数据源集成，开始分析行为以检测异常情况。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

或者，您可以选择使用 Amazon A SageMaker I AI 构建、训练和托管自定义机器学习模型，以解决内容审核、毒性检测、作弊检测、欺诈检测等用例。

客户示例

AnyCompany 游戏使用 Amazon Lookout for Metrics 来自动检测服务器性能、玩家登录尝试次数或交易量中可能表明不良行为者威胁的异常模式。此外，他们还使用Amazon SageMaker AI开发了自定义机器学习模型，该模型可以持续分析网络流量模式和玩家行为，以帮助识别协调威胁，例如试图利用其虚拟经济的机器人网络。

这种自动化方法使他们的安全团队能够专注于调查和应对真正的威胁，而不是手动监控数千个指标，同时确保在新出现的威胁模式对游戏可用性或玩家安全产生重大影响之前对其进行检测和解决。

实施步骤

- 使用 Amazon Lookout for Metrics 来帮助自动检测和诊断关键业务和运营数据中的异常情况
- 将 Amazon Lookout for Metrics 与游戏分析管道、Amazon S3 等数据源集成， CloudWatch 或者监控收入、登录和留存率等指标。
- 使用 Amazon SageMaker AI 构建、训练和托管自定义机器学习模型，用于作弊检测、欺诈预防和内容审核等高级用例。

GAMESEC06-BP03 利用来自系统级日志的见解来持续改进您的基础架构保护策略

捕获和存储来自相关服务的系统级日志，例如 S3 服务器访问日志、CloudFront访问日志和 ALB 访问日志。这些日志可以存储在您账户的 S3 存储桶中，可用于将游戏中的玩家使用情况信息与系统级信息相关联，包括连接详细信息，例如 IP 地址、请求标头以及您可能在游戏后端配置的相关请求操作和过滤。您可以将这些日志发送到前面提到的相同日志解决方案，也可以使用 Amazon Athena 的 SQL 查询来分析这些日志，而无需将日志移出 Ama zon S3。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

适用于 S3 的 Access Analy zer 是一项监控存储桶访问策略的功能，确保这些策略仅提供对您的 Amazon S3 资源的预期访问权限。适用于 S3 的 Access Analyzer 会评估您的存储桶访问策略，并允许您发现并迅速修复可能存在意外访问权限的存储桶。

实施步骤

- 使用威胁检测和事件响应 AWS 服务，自动执行基础设施保护策略的各个方面。
- 通过用于人工智能和机器学习的系统级日志和 AWS 服务，深入了解您的基础设施保护。

数据保护

在开发和设计游戏时，请考虑你的工作室正在收集什么类型的数据，以及你决定如何保护这些数据。在这方面的安全方面需要探讨的主题包括：

- 您是如何选择对数据进行识别和分类的
- 您如何保护静态数据
- 您如何保护传输中的数据

没有专门针对Games Lens的数据保护最佳实践。有关安全数据保护的最佳实践，请参阅 Well-Architected Framework 白皮书。

事件响应

GAMESEC07：你们是如何制定和执行政策来应对玩家的不当行为和虐待行为的？

玩家的不当行为和虐待行为会严重影响玩家的体验。不良玩家行为会驱逐合法玩家，从而导致玩家留存率降低，游戏内购买收入减少，负面评论可能会损害游戏的声誉和未来的销量。

制定促进玩家采取积极行动的政策，并确定您将如何执行这些政策。

最佳实践

- GAMESEC07-BP01 实施事件响应计划以处理不良行为者和虐待行为

- GAMESEC07-BP02 封禁与不良行为者相关的账户

GAMESEC07-BP01 实施事件响应计划以处理不良行为者和虐待行为

制定应对游戏中的不良行为者和滥用行为的行动计划。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

考虑一些因素，例如何时暂时暂停或永久封禁玩家，以及暂时停用玩家的凭证需要多长时间。

客户示例

AnyCompany 游戏创建了一个分层的事件响应系统，在该系统中，诸如不当聊天消息之类的轻微违规行为会导致账户自动被暂停24小时，而更严重的违规行为（例如作弊或骚扰）会立即被暂停7天，并由人工版主进行强制审查。

此外， AnyCompany Games还制定了升级程序，根据该程序，累犯者面临的停职时间会逐渐延长。他们创建了上诉流程，允许被虚假举报的玩家对自动操作提出异议，同时通过身份验证要求来维护安全。

GAMESEC07-BP02 封禁与不良行为者相关的账户

如果任其缓解，游戏中的滥用行为可能会继续对他人的游戏体验产生负面影响，因此应尽快予以缓解。实施一项程序，对经证实违反您的服务条款的不良行为者实施禁令或其他形式的限制。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

通常，确定施加此类限制的条件的规则和评估过程将由组织内的玩家社区团队或信任与安全团队等人员决定。举报不良行为者后，运行预先确定的工作流程对已识别的玩家采取行动。

例如，AWS Step Functions和AWS Lambda函数可用于运行一个自动工作流程，该工作流程接受一批玩家账户作为输入。然后，该工作流程会更新名为 Bans 的 Amazon DynamoDB 表中的条目，其中可能包含有关玩家账户、封禁原因和持续时间的详细信息。

根据你的游戏和账户管理系统的设计以及你遇到的不良行为者滥用行为的类型，维护一个与账户管理系统分开的封禁记录系统。你可能不想从你的账户管理系统中关闭玩家的账户，而是选择简单地关闭他们玩你的游戏的权限。这在玩家的账户凭据被用来访问具有不同服务条款或政策的多个游戏时可能很有用。

实施步骤

- 制定和执行应对不良行为者的滥用行为的政策。
- 使用 AWS 服务自动应对不良行为者。

资源

- AWS 安全事件响应技术指南
- AWS Machine Learning 博客：使用 Amazon Rekognition Face Liveness 检测真实用户和真实用户并阻止不良行为者
- AWS 游戏解决方案：Community Health

应用程序安全性

GAMESEC08：你如何保护自己的 CI/CD 管道？

游戏开发 CI/CD 管道通常由高度可用的源代码控制服务器和存储、用于运行构建的计算资源、用于执行自动测试的软件以及来自开发计算机的适当网络连接组成。保护您的 CI/CD 管道对于保护敏感信息、保持代码完整性和维护可信版本非常重要。嵌入治理和护栏可以提高开发人员的灵活性，同时保持良好的安全实践。

由于游戏通常处理支付处理、存储个人信息并维护有价值的虚拟经济，因此开发过程中的安全漏洞可能会导致重大的财务损失、监管处罚和玩家信任的丧失。

通过整合保障措施，组织可以保持对软件交付过程的可见性和控制力，从而实现快速的事件响应，并促进安全编码实践的文化。

最佳实践

- GAMESEC08-BP01 在 CI/CD 管道的每个阶段都应用安全性

GAMESEC08-BP01 在 CI/CD 管道的每个阶段都应用安全性

诸如访问控制、职责分工和审计跟踪之类的护栏可防止未经授权的访问或恶意活动。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

您的人员、流程和技术也应确保管道的安全。最接近守则的人必须建立安全的编码实践，并确保他们遵守这些规范。持续对流程进行迭代，以验证整个管道的安全级别是否一致。最后，实施技术以验证最佳实践和流程没有被绕过。

客户示例

AnyCompany 游戏实施基于角色的访问控制，在这种控制中，只有高级开发者才能批准对其反作弊系统代码的更改，同时要求安全团队成员对处理玩家支付数据的组件进行强制性代码审查。

他们的 CI/CD 管道会自动运行威胁模型验证检查，确保玩家交易市场等新功能针对先前发现的攻击载体（例如物品重复漏洞利用或欺诈性交易尝试）进行测试。

实施步骤

- 根据最小权限原则为用户提供权限。
- AWS CloudTrail 用于审核管道中使用的服务之间发出的 API 调用。
- 使用预提交挂钩来验证代码是否遵循一般惯例和公司政策。

实现安全自动化

GAMESEC09：如何在 CI/CD 管道内实现安全自动化？

将安全措施集成到您的 CI/CD 管道中，以便在整个开发生命周期中保持稳健的安全态势。此过程具有许多与确保管道安全性相同的好处。拥有安全的 CI/CD 管道可以降低可能延迟游戏开发时间的安全事件的可能性。

为 CI/CD 管道提供安全性需要在开发周期的每个阶段实施安全最佳实践和工具。在管道中设置安全性还可以缩短安全审查的时间。

在 CI/CD 管道内实现安全自动化，对于验证每次更改代码时是否一致实施和测试安全控制措施尤为重要。实施适当的工具和自动化可以提供安全可靠的游戏。

最佳实践

- GAMESEC09-BP01 集成工具和自动化以缩短安全审查的平均时间

GAMESEC09-BP01 集成工具和自动化以缩短安全审查的平均时间

为了识别安全漏洞，组织可以使用各种不同的工具和服务，例如静态应用程序安全测试 (SAST) 和动态应用程序安全测试 (DAST)。SAST 是一种审查源代码和确定安全漏洞的方法。DAST 是一种测试代码的黑匣子方式，它无需查看源代码即可测试您的应用程序。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

组织可以使用的另一种工具是软件组成分析 (SCA)，它可以评估第三方或开源依赖关系的安全性。对于更手动的方法，可以在整个管道中实施安全的代码审查。

客户示例

AnyCompany 游戏使用 SAST 工具在开发过程中自动标记潜在的安全漏洞。他们还使用 DAST 工具来模拟针对正在运行的游戏版本的威胁，以验证安全控制是否按预期运行。此外， AnyCompanyGames 还在其开发过程中集成了依赖扫描工具，以自动识别第三方库和游戏引擎中的已知漏洞。

实施步骤

- 使用亚马逊 CodeGuru 作为 SAST 工具。
- 使用开源工具，例如 OWASP 依赖关系检查 SonarQube、或。 OWASPZap

资源

- 为开发人员提供安全保障

威胁建模

GAMESEC10：如何将威胁建模集成到组织的应用程序开发生命周期中？

威胁建模是识别应用程序面临的潜在威胁并确定其优先级，并确定可用于缓解这些威胁的解决方案的过程。随着游戏发展成为处理敏感用户数据和真钱交易的复杂互联系统，这种做法变得越来越重要。

将威胁建模整合为一项持续的练习，以支持游戏的安全，不仅在最初的设计阶段，而且在游戏的持续发展和演变中。

最佳实践

- GAMESEC10-BP01 确定在整个应用程序开发生命周期中何时以及如何完成威胁建模练习

GAMESEC10-BP01 确定在整个应用程序开发生命周期中何时以及如何完成威胁建模练习

没有一种最好的方法可以进行威胁建模。有关何时以及如何执行此操作的详细信息将根据您的游戏工作室的独特需求而有所不同。例如，根据工作室的规模，您的团队成员可能参与威胁建模过程的一个或多个方面。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

AWS 安全博客概述了在制定威胁建模策略时需要牢记的注意事项，例如：

- 你的哪些团队成员和角色应该参与威胁建模
- 如何确定要使用的工作流程工具
- 如何确定威胁建模各个方面的所有权
- 如何识别和评估工作负载设计中要使用的安全控制措施

客户示例

AnyCompany 游戏首先要对有价值的资产进行分类，例如玩家数据、游戏代码和算法、游戏内货币、用户生成的内容以及知识产权（例如未发布的内容或专有引擎）。他们考虑了不同类型的潜在不良行为者，例如寻求不公平优势的作弊者、试图窃取个人或财务数据的不良行为者以及试图破坏游戏玩法的恶意用户。

在整个开发过程中， AnyCompany Games 使用威胁模型来指导安全编码实践，并影响将重点放在高风险区域的测试策略。在游戏发布之前，他们会进行全面的威胁建模审查，以评估预期的玩家负荷和未经授权的访问尝试的准备情况，并准备事件响应程序。

实施步骤

- 在管道的每个阶段都安装护栏。 CI/CD
- 使用自动化和工具来提高应用程序安全审查的效率。
- 使用威胁建模作为提高应用程序安全性的过程。

资源

- AWS 安全博客：如何进行威胁建模
- NIST：以数据为中心的系统威胁建模指南
- 威胁建模是构建者的正确方法 — Skill Bu AWS ilder 虚拟自定进度培训
- 建筑商威胁建模 — AWS 研讨会

资源

请参阅以下资源，详细了解我们与安全相关的最佳实践。

相关文档：

- 常见的 Amazon Cognito 场景
- 使用已签名 URLs
- 使用渠道流从 Amazon Chime SDK 消息中删除消息中的亵渎内容和敏感内容
- Amazon 的安全 GameLift
- 使用 Amazon 保护内容传输 CloudFront
- 安全响应指南
- AWS DDoS 弹性最佳实践
- 保护您的 AWS 云 环境免受勒索软件的侵害

相关合作伙伴解决方案：

- Datadog
- Sumo Logic
- Splunk
- Honeycomb.i
- New Relic
- AWS Marketplace -DRM 解决方案

相关培训材料：

- 亚马逊 Cognito 入门

- 安全自定进度培训

可靠性

可靠性支柱包括系统能够从基础设施或服务中断中恢复、动态获取计算资源以满足需求，以及缓解配置错误或临时网络问题等中断。

聚焦领域

- 设计原则
- 基本原理
- 工作负载架构
- 变更管理
- 故障管理
- 资源

设计原则

除了 Well-Architecte AWS d Framework 白皮书中的设计原则外，以下是可以提高云端游戏工作负载可靠性的设计原则：

- 为满足业务预测所需的玩家并发峰值和系统可扩展性目标设定基准：在游戏发布之前和直播游戏运营期间，对峰值时预计的并发玩家数量进行估计，以确定系统可扩展性的目标目标，以满足这些预测。这有助于为游戏的可靠性创建基准。通过验证您的扩展系统是否可以正常管理活跃的玩家会话，定义扩展策略以自动适应需求的变化，而不会影响可用性。
- 衡量您的可靠性以及对玩家体验的影响：定义代表游戏健康状况的关键绩效指标 (KPIs)。监控基础设施和游戏功能的变化对可靠性的影响。

基本原理

为了实现可靠性，系统必须有精心规划的基础和适当的监控，以及处理需求或要求变化的机制。该系统的设计应能检测故障并自动自我修复。

没有专门针对Games Lens的基础最佳实践。有关适用于游戏工作负载的可靠性基础的最佳实践，请参阅 Well-Architected F r amework 白皮书。

工作负载架构

GAMEREL01：您的游戏架构是否利用了云的弹性？

AWS 基础设施是围绕区域和可用区构建的。 AWS 区域 提供多个物理分隔和隔离的可用区，这些可用区使用低延迟、高吞吐量和高度冗余的网络进行连接。这些结构可用于设计以可靠性目标为重点的工作负载。

最佳实践

- GAMEREL01-BP01 跨多个可用区和区域分配游戏基础设施以提高弹性

GAMEREL01-BP01 跨多个可用区和区域分配游戏基础设施以提高弹性

为了最大限度地减少局部基础设施损坏对玩家的影响，您应该将基础设施部署统一分配到足够的独立地点，以便能够承受意想不到的损失，同时仍有足够的容量来满足玩家的需求。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

在部署游戏基础设施时，建议将容量统一分配到一个区域的多个可用区，这样您就可以在不中断玩家体验的情况下承受一个或多个可用区域的中断。诸如 Web 应用程序之类的游戏后端服务应在多个可用区之间进行负载平衡，或者应使用托管服务（例如 AWS Lambda Amazon API Gateway）构建，这些服务通过设计提供区域高可用性。同样，维护缓存、数据库、消息队列和存储解决方案等状态的组件应设计为在多个可用区之间提供持久的数据持久性，这种持久性是在 Amazon S3、DynamoDB 和 Amazon SQS 等服务中设计的，也可以在其他服务中进行配置。

在设计游戏服务器托管架构以实现弹性时，请在中的可用区中统一部署游戏服务器队伍， AWS 区域 以最大限度地提高您对该区域可用计算容量的访问权限，并减少可用区损坏的影响范围。例如，您可以将 Amazon A EC2 uto Scaling 配置为使用可用区。如果 EC2实例运行状况不佳， EC2 Auto Scaling 可以替换该实例，也可以在一个或多个可用区不可用时将实例启动到其他可用区。

对于关键基础设施，例如身份验证，请配置在多个可用区中运行的最小数量的可行实例，如果其中一个可用区存在损害，则使用 auto scaling 来处理负载增加或容错问题。

将您的游戏基础设施部署到多个区域，以最大限度地提高可用性。跨区域灾难恢复功能（例如 Aurora 全球数据库和冗余基础设施）可以在主区域受损时提供服务连续性，只需在辅助区域中部署简单的

DNS 更改即可激活。虽然我们鼓励您的游戏后端服务实现高可用性，但此建议对您的游戏服务器尤其重要。

例如，在多人游戏中，游戏服务器的基础架构容量可能会超过其他服务的容量需求，因为游戏服务器用于为玩家托管游戏会话。许多游戏选择将玩家分成逻辑游戏区域（例如美国西部和东部）。为了简化玩家体验并便于使用全球基础设施来托管游戏，可以考虑将面向玩家的游戏区域的名称与实际托管游戏服务器的底层云提供商区域或数据中心位置以及托管支持该玩家游戏区域的游戏服务器实例的其他基础设施（例如本地区域或您自己的数据中心）分开。

在设计配对服务时，请部署多区域架构，并在各个区域之间部署单独的软件。将您的配对服务部署与托管游戏服务器实例的舰队分离，这样无论配对服务的哪个区域部署处理配对请求，您都可以将玩家路由到区域中的游戏服务器。

在配对实现中设计逻辑，以偏向于满足你的延迟和其他规则的游戏服务器区域，如果你的舰队容量不足或其他区域基础设施中断，可以回退到将玩家路由到其他区域。

实施步骤

- 在多个可用区之间统一分配游戏基础架构，以提供高可用性和弹性。
- 使用 Amazon S3 AWS Lambda、DynamoDB 和 SQS 等托管部署游戏后端服务和有状态组件，或者为自定义解决方案配置负载平衡和耐久性。
- 使用 Aurora 全球数据库和面向玩家的逻辑区域与底层物理位置分离的灾难恢复解决方案，为关键游戏服务和服务器实施多区域部署。

资源

- 静态稳定性
- Amazon GameLift 游戏会话队列的最佳实践
- Amazon GameLift 多区域舰队
- Aurora 全球灾难恢复数据库

变更管理

GAMEREL02：如何扩展有状态的游戏以适应需求的变化？

由于您的玩家需求会随着时间的推移而波动，因此您的游戏基础设施应该能够自适应扩展，以应对这些不断变化的需求。虽然很难提前预测游戏的受欢迎程度，但要设计一种架构方法，允许增加和移除基础设施容量，以适应玩家群体的波动。

最佳实践

- GAMEREL02-BP01 实施包含活跃玩家游戏会话状态的扩展策略
- GAMEREL02-BP02 Support 支持在游戏中使用多种 EC2 实例类型

GAMEREL02-BP01 实施包含活跃玩家游戏会话状态的扩展策略

实施一种自动扩展游戏基础架构的解决方案，其方式既要考虑到活跃连接的玩家会话的状态性质，又能在不中断游戏玩法的情况下优雅地处理扩展活动。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

在云端开发游戏的优势之一是，通过根据需要自动扩展服务器基础架构以满足需求，可以实现弹性。虽然可以使用 Amazon A ut EC2 o Scaling 策略、EK S 自动缩放或可扩展 Web 应用程序通常采用的类似技术来动态扩展无状态或异步游戏和后端服务，但游戏开发者通常需要一种更加定制的方法来扩展有状态或同步游戏，以帮助阻止对活跃玩家会话的中断。

实施步骤

- 对于有状态的游戏，生成可用于监控玩家会话状态和可用游戏服务器容量的自定义指标，这些指标可以 CloudWatch 作为自定义指标报告给 Amazon。使用应用程序监控功能进行锻炼，例如S CloudWatch ynthetics，可以检查游戏中是否存在简单的上下运行状况监测可能无法检测到的功能损害。
- 使用自定义指标，将游戏服务器扩展软件实现为使用 AWS Lambda 函数的无服务器应用程序，或者使用软件开发工具包进行 API 调用 AWS Fargate，更新托管游戏服务器版本的 A EC2 uto AWS Scaling 组的最小、最大和所需容量设置，从而管理专用游戏服务器实例的队列。
- 使用 Amazon GameLift 托管您的out-of-the-box游戏服务器，并使用游戏服务器的自动扩展功能为您管理扩展过程。

Amazon GameLift 的自动扩展功能可以识别活跃的玩家会话，并且可以将其配置为阻止主动托管玩家的游戏服务器实例的终止或缩小。有关更多信息，请参阅使用 Amazon 监控亚马逊 GameLift 服务器 CloudWatch。

GAMEREL02-BP02 Support 支持在游戏中使用多种 EC2 实例类型

使用 EC2 实例托管游戏时，或者如果您使用托管在 EC2 实例上的容器时 AWS 账户，请在托管策略中使用多种实例类型。通过使用多种实例类型，您可以增加游戏扩展时可以使用的计算选项的数量，以添加更多服务器来支持玩家增长，从而在首选实例类型不可用时提高可靠性。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

使用竞价型实例托管游戏时，请使用多种实例类型，因为竞价型实例的可用性会根据客户需求而波动。

在多个实例类型上测试您的游戏以满足您的成本和性能要求，并确定实例类型的优先顺序排名。Amazon A EC2 uto Scaling 支持使用多种实例类型和大小，也支持在配置中为每种实例类型分配权重，以便您可以对计算选项进行优先排序。

使用 Amazon 托管托 GameLift 管托管游戏时，请决定您的游戏需要什么类型的实例以及如何在这些实例上运行游戏服务器进程（使用运行时配置）。为队列选择资源时，请考虑多个因素，包括游戏操作系统、实例类型（计算硬件），以及是使用按需实例、竞价型实例还是两者兼而有之。Amazon 的托管费用 GameLift 主要取决于您使用的实例类型。有关更多信息，请参阅为托管队列选择计算资源。

实施步骤

- 在 EC2 或容器上托管游戏时，使用多种 EC2 实例类型来提高可靠性和扩展选项。
- 为 Amazon A EC2 uto Scaling 或 GameLift 队列配置按优先顺序排列的实例类型和权重，以优化成本和性能。
- 在各种实例类型上测试您的游戏，以验证性能是否符合要求，并相应地调整您的托管策略。

故障管理

GAMEREL03：在基础设施中断期间，如何保持游戏状态？

随着时间的推移，您的游戏基础设施会经历各种操作事件，因此游戏架构的设计应在基础设施事件期间保持玩家体验的连续性并保持游戏状态。要处理这些事件，请实施监控、正常关闭和状态持久化机制，以验证玩家的流畅游戏体验。

最佳实践

- GAMEREL03-BP01 监控游戏服务器中断情况，并使用数据改进托管架构，以实现可靠性目标
- GAMEREL03-BP02 实现游戏功能的松散耦合以处理故障，同时最大限度地减少对玩家体验的影响
- GAMEREL03-BP03 监控一段时间内的基础设施事件，以衡量对玩家行为的影响

GAMEREL03-BP01 监控游戏服务器中断情况，并使用数据改进托管架构，以实现可靠性目标

随着时间的推移，监控游戏服务器指标以及性能故障或降级（例如负载下的延迟增加）对玩家行为的影响，以便您可以调整游戏服务器托管策略以满足游戏的可靠性要求。如果要降级的游戏服务器基础设施影响玩家，则应立即将其从服务中移除，或者在服务器上没有托管活跃的玩家会话时主动更换。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

对于游戏以 REST 形式托管的场景 APIs，可以像传统的 Web 应用程序架构一样管理系统可靠性，在这种架构中，可以以分布式方式在多台服务器之间平衡流量，以降低服务器故障的风险。

对于实时同步游戏玩法，游戏会话通常托管在虚拟机或游戏服务器实例上运行的游戏服务器进程上，因为需要以高性能的方式维护游戏状态并复制到连接的游戏客户端。这种实现意味着玩家的体验与托管游戏会话的游戏服务器进程的性能和可靠性紧密相关。这种架构使得管理游戏服务器的可靠性比传统方法更加复杂。

要减轻游戏服务器故障的影响，请将游戏配置为持续对高度可用的缓存或数据库（例如亚马逊 ElastiCache (Redis OSS) 或 Ama zon MemoryDB ）执行异步更新玩家的游戏状态。如果服务器出现故障，则可以从外部数据存储中获取玩家上次保存的游戏状态，并在新的游戏服务器实例上恢复其会话。

但是，这种方法增加了管理这种外部状态的成本和复杂性，并且可能不适合快节奏或竞技游戏，在这些游戏中，状态变化非常频繁，而且规模如此之大，以至于即使引入高性能的内存缓存数据存储也会导致复制延迟太大，无法从中恢复会话。对于这种性质的游戏，最佳方法是接受服务器的损失，然后将玩家送回游戏大厅寻找其他会话，或者你可以自动将他们重定向到另一个游戏会话。

尽可能多地捕获有关导致服务器中断的原因的有用日志数据，以便日后可以调查问题。Amazon 为调试队列问题 GameLift 提供指导，并提供远程访问亚马逊 GameLift 队列实例的功能。

实施步骤

- 监控游戏服务器性能下降的指标，并在必要时移除或更换降级的服务器以保持可靠性。

- 使用 Amazon ElastiCache 或 MemoryDB 进行异步游戏状态更新，以便在可行的情况下在服务器出现故障后启用会话恢复。
- 利用诸如 Amazon 之类的工具进行队列监控和远程访问，捕获有关服务器中断的详细日志数据 GameLift 以进行调查和调试。

GAMEREL03-BP02 实现游戏功能的松散耦合以处理故障，同时最大限度地减少对玩家体验的影响

解耦组件是指设计服务器组件以使其尽可能独立运行的概念。游戏的某些方面很难脱钩，因为数据需要尽可能保持最新状态，才能为玩家提供良好的游戏体验。但是，许多组件和游戏任务可以分离。例如，排行榜和统计服务对游戏体验并不重要，对这些服务的读取和写入可以在游戏中异步执行。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

对游戏中检测到问题时可以自动禁用或由管理员禁用的功能实施优雅降级，并将依赖该功能的上游服务配置为能够优雅地处理故障。例如，如果游戏客户端中未正确加载特定的玩家数据，则应考虑这些数据是否对游戏体验至关重要。如果不是，请将游戏客户端配置为在不中断玩家体验的情况下优雅地处理此故障，并选择稍后在玩家重新访问屏幕时重试获取此数据。

使用超时、重试和退避等逻辑来处理错误和失败。超时可以防止系统长时间挂起。重试可以提供瞬时错误和随机错误的高可用性。

定义可以松散耦合到关键组件的非关键组件。松散耦合使系统更具弹性，因为一个组件的故障不会级联到其他组件。当游戏功能不需要与游戏服务器或后端建立状态连接时，您应该实现无状态协议以动态扩展并从暂时故障中恢复。使用 API 开发可以与无状态协议松散耦合的非关键组件。 HTTP/JSON 将来自游戏客户端的网络调用实现为异步且非阻塞的，以最大限度地减少性能缓慢的游戏功能或其他依赖服务对玩家的影响。

要通过松散耦合进一步提高弹性，请在可以异步处理的组件之间使用消息服务，例如队列、流媒体或基于主题的系统。此模型适用于不需要立即响应的交互或确认已注册请求就足够的情况。此解决方案涉及一个生成事件的组件和另一个使用事件的组件。这两个组件不会通过直接 point-to-point交互进行集成，而是通过中间层（例如耐用存储层或队列层）进行集成。这也有助于通过在处理失败时保留消息来提高系统的可靠性。

研究并选择适当的消息传递机制，因为各种消息服务具有不同的特征，例如排序和传送机制。将操作设计为等性，以便所选消息系统至少传送一次消息。举个例子，假设一个典型的游戏用例，在这个用例

中，你的游戏需要跟踪玩家的游戏时间、统计数据或其他相关数据，这可能会导致在玩家并发峰值时写入吞吐量很高的用例。

要实现可靠的架构，请考虑用例是否需要玩家所感知 read-after-write的一致性。通常，诸如此类的场景适用于异步处理，可以通过实现写入队列模式来实现，在这种模式中，请求会被提取到可扩展且耐用的消息队列（例如 Amazon SQS）中，并可以使用消费者服务（例如 Lambda 函数）批量插入到您的后端数据库中。这种方法比多个分布式组件之间的同步通信更可靠，这些组件包括玩家的游戏客户端、您的后端 Web 和应用程序服务器以及您的内部数据库系统。它还可以降低成本，因为无需扩展后端数据库即可满足峰值写入吞吐量，因为可以根据需要使用写入队列中的使用者处理来降低这种摄取速率。

实施步骤

- 将排行榜和统计服务等非关键组件与关键游戏功能分离，以实现异步操作并增强弹性。
- 使用超时、重试和退避逻辑对非关键功能实施优雅降级，并验证游戏客户端是否在不中断玩家体验的情况下处理故障。
- 使用像 Amazon SQS 这样的消息系统在组件之间进行异步通信，从而实现对高吞吐量用例的可扩展、持久和可靠处理。

资源

- 使用微服务架构构建高度可扩展且可靠的工作负载
- 使用 AWS 无服务器服务集成微服务
- 了解微服务的异步消息传递
- 可扩展游戏开发模式简介 AWS
- 实现优雅降级

GAMEREL03-BP03 监控一段时间内的基础设施事件，以衡量对玩家行为的影响

监控您的游戏服务器进程、游戏服务器实例指标和游戏体验指标，以确定问题的根本原因。除了监控 CPU 和内存外，您还可以设置对与 EC2 实例网络限制相关的网络指标的监控， packets-per-second以提醒您注意带宽超出或其他可能表明服务器资源配置不足的网络级问题。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

使用 CloudWatch Synthetics 检查关键路径应用程序功能以获得玩家体验，例如无法登录或其他影响服务的问题。对于使用 Amazon 托管的游戏服务器 GameLift，请考虑监控指标，例如：

- GameServerInterruptions 而且 InstanceInterruptions，这有助于了解竞价型实例可用性的限制如何影响使用 Spot 部署的游戏服务器。
- ServerProcessAbnormalTerminations，它可用于检测游戏服务器进程中的异常终止。

建议保留游戏服务器可靠性的历史指标数据。将这些历史数据用于报告目的，并将其与其他数据集相结合，以发现潜在的趋势，并评估游戏服务器问题可能对玩家行为的影响。

Amazon CloudWatch 不会无限期保留指标，而且指标的存储分辨率会随着时间的推移而提高，因此可以考虑将这些指标导出到经济实惠的长期存储中，例如 Amazon S3。您可以将CloudWatch指标流配置为自动将您的指标从CloudWatch区域传输到您自己的 S3 存储桶，这些指标可以长期存储在 S3 智能分层等存储层中，并最终使用 Amazon Glacier 进行存档。将您的指标放入 Amazon S3 后，就可以随时将这些指标与数据湖中的其他数据集合在一起，以便与 Amazon A thena 进行交互式查询。

实施步骤

- 使用 Amazon 和 Synthetic CloudWatch s 监控游戏服务器、实例和网络指标，包括带宽 CloudWatch 和 packet-per-second限制，进行关键路径功能检查。
- 跟踪 GameLift特定指标，例如GameServerInterruptions和 ServerProcessAbnormalTerminations，以评估竞价型实例可用性的影响并检测服务器异常终止。
- 将 CloudWatch 指标导出到 Amazon S3 以进行长期存储，使用 S3 智能分层或 Glacier 等经济实惠的分层，并使用 Amazon Athena 等工具分析趋势。

资源

- Amazon EC2 实例级网络性能指标揭示了新的见解
- CloudWatch指标流 — 实时向合作伙伴和您的应用程序发送 AWS 指标

资源

请参阅以下资源，详细了解我们与可靠性相关的最佳实践。

相关文档：

- 在以下位置练习持续集成和持续交付 AWS
- 自动缩放异步 Job 队列
- 设计您的 WorkloadService 架构
- 超时、重试和带抖动的退避
- Well-Architected 框架——可靠性支柱
- 为实现可靠的可扩展性而设计
- Amazon Builder 的图书馆
- 适用于多人游戏的大规模实时消息
- 可扩展游戏开发模式简介 AWS
- 在上运行容器化微服务 AWS
- 云端托管 Web 应用程序
- 构建可扩展且安全的多 vPC 网络基础架构

相关视频：

- re: Invent 2020：育碧：开发一款多平台多人游戏 AWS
- re: Invent 2018：Supercell-Scaling 手游
- re: Invent 2019：CAPCOM 如何使用容器、数据和机器学习制作有趣的游戏
- re: Invent 2018：在保持可用性的同时实现Riot Games的玩家账户全球化
- re: Invent 2020： GameLoft -零停机时间数据湖迁移深入探讨

相关培训：

- 将 Amazon GameLiftFleet IQ 用于游戏服务器
- 通过 Amazon 托管游戏服务器 EC2

性能效率

性能效率支柱侧重于有效利用计算资源来满足需求，并随着需求的变化和技术的演变而保持这种效率。

采用数据驱动的方法来选择高性能架构。收集有关架构的全面数据，从高级设计到资源类型的选择和配置。周期性地考虑您的架构选择，以利用不断变化的服务和解决方案集。指标有助于了解与预期绩效的偏差，以便您可以采取行动。数据驱动的方法有助于在架构中进行权衡取舍，从而提高性能、降低成本或改善开发人员体验。

聚焦领域

- 设计原则
- 架构选择
- 区域选择
- 迭代开发
- 计算和硬件
- 计算选择
- 数据管理
- 网络和内容分发
- 流程和文化
- 资源

设计原则

除了 Well-Architecte AWS d Framework 白皮书中的设计原则外，以下设计原则还可以提高游戏的性能效率：

- 从 end-to-end以下方面衡量游戏性能：重要的是要从玩家的角度来衡量性能。这意味着您应该衡量游戏客户端、游戏基础设施以及将玩家连接到基础设施的互联网连接的性能。这将有助于了解在架构中可以在哪些方面进行性能改进。
- 优化架构以改善反映玩家实际体验的指标：随着时间的推移，随着时间的推移调整和演变架构，请考虑这些改进和变化将如何影响玩家体验。游戏工作负载应该能够承受并最大限度地减少故障的影响，从而阻止对游戏玩法的广泛干扰。应将彼此不严重依赖的游戏功能和系统分离，以缩小故障的爆炸半径并隔离影响玩家的问题。

- 使用可简化游戏操作和提高开发速度的技术：优先采用可以提高开发者效率的技术。开发前期制作阶段的操作开销可能会分散人们对改善游戏玩法的注意力。通过利用我们 AWS 合作伙伴提供的 AWS 托管服务，可以减少工程设计，从而使游戏开发者能够专注于核心游戏循环和玩家体验。在整个游戏开发生命周期中，架构和性能要求可能会发生变化和演变，每个阶段都应考虑技术权衡。
- 设计基础设施以满足玩家并发峰值并根据需要进行动态扩展：基础设施的设计应可扩展以满足玩家的需求。诸如玩家会话并发性和登录次数之类的指标可用于在系统过载之前抢先扩展。被动的系统利用率指标（例如 CPU 和内存消耗）可用于在系统过载后进行扩展。通过动态扩展基础架构，您可以降低游戏运营成本。

架构选择

GAMEPERF01：如何为游戏服务器选择合适的托管选项？

为游戏服务器选择合适的托管选项是游戏服务器性能的基础。在进行生产架构时，决定使用 EC2实例、容器解决方案或完全托管服务是首先要做出的决定之一。每个托管选项在性能调整、扩展、运营和集成方面都有不同的功能和注意事项。

最佳实践

- GAMEPERF01-BP01 评估游戏服务器资源要求和可扩展性需求
- GAMEPERF01-BP02 考虑扩展游戏服务器的运营开销
- GAMEPERF01-BP03 评估与其他 AWS 服务、开发环境、目标 CPU 架构和功能的集成

GAMEPERF01-BP01 评估游戏服务器资源要求和可扩展性需求

根据您的可扩展性需求评估服务器需求，以验证您选择的托管选项既符合您的要求又能提供最佳性能。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

为游戏服务器选择合适的托管选项时，请考虑以下因素：

游戏服务器资源要求

评估游戏服务器进程的 CPU、内存、网络和存储需求，以确定游戏消耗的资源。不要忽视联网；每帧都需要 CPU 周期来接收玩家的操作、更新游戏状态并将其发送回给玩家。卸载数据包处理可以腾出

CPU 以用于核心游戏功能。网络是流畅且响应迅速的游戏玩法的基础，因此在过程的早期对其进行测试可以定义游戏的基准性能概况。

第一人称射击游戏的每秒动作可能很高，CPU 需要快速转移到网络上，这可能有利于计算优化的 C 系列实例，而每回合可能花费更多 CPU 周期处理的回合制策略游戏可能需要增加 R 系列实例的内存，以便在服务器上本地存储和更新游戏状态，然后再将其发送回给玩家。使用数据驱动的方法，例如利用率饱和度和错误 (USE) 方法，做出明智的架构选择。

可扩展性和弹性

评估每个托管选项在不影响性能的情况下扩展以满足玩家需求的速度和流畅程度。考虑游戏工作负载所需的自动化和灵活性水平，以便在高峰时段保持流畅的游戏体验。游戏服务器可以通过在同一实例上添加其他游戏服务器进程来提高利用率来快速扩展，在这种情况下，根据活跃用户数量的增加和正在玩的游戏，游戏后端的扩展速度可能会变慢。您的舰队应根据需求进行扩展，以最大限度地降低成本，同时最大限度地缩短玩家进入游戏的等待时间。查看 Amazon EC2 Spot Instance Advisor，深入了解游戏服务器队列具有成本效益的可用容量。

实施步骤

- 评估游戏服务器对 CPU、内存、网络和存储的资源需求，以选择合适的实例类型，同时考虑游戏特定的性能需求，例如 FPS 游戏的高网络吞吐量或回合制策略游戏的内存优化。
- 通过使用 USE 方法等框架分析性能数据，比较不同的托管选项，例如容器、实例、裸机和托管服务。利用这些见解对您的系统架构做出更好的决策。
- 设计队列以实现可扩展性和弹性，利用 EC2 Spot Instance Advisor 等工具来优化成本，同时促进快速扩展，以满足高峰时段玩家的需求。

GAMEPERF01-BP02 考虑扩展游戏服务器的运营开销

考虑与每个托管选项相关的管理和运营开销。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

运营开销

EC2 或容器上的自托管解决方案可以提供更多控制，但也需要更多的管理。ECS 或 EKS 等容器编排器可以缩短容器化服务器的启动时间，同时还会增加网络复杂性和维护编排开销。

例如，EKS 托管节点组可以自动配置游戏服务器和生命周期管理，但在终止节点时不考虑吊舱中断预算，如果您的游戏需要超过 15 分钟的终止时间才能安全完成游戏，则可能需要创建生命周期挂钩或考虑使用自定义控制器的自我管理节点来阻止游戏中断。

像 Amazon Game Lift 这样的托管服务可以处理大部分运营开销，但会降低对低级联网和安全配置特殊要求的可见性和控制力。选择游戏服务器解决方案需要在调整游戏服务器性能和扩展行为的自定义、控制和责任之间进行权衡。

实施步骤

- 评估托管选项的运营开销，在自托管解决方案（如 EC2 ECS 或 EKS）与 Amazon Game Lift 等托管服务之间平衡控制和管理工作。
- 使用 EKS 托管节点组实现自动化，但如果您的游戏服务器需要比默认终止时间更长的终止期，则可以实现生命周期挂钩或自定义控制器。
- 在选择游戏服务器解决方案时，权衡自定义、可见性和运营责任之间的权衡。

GAMEPERF01-BP03 评估与其他 AWS 服务、开发环境、目标 CPU 架构和功能的集成

评估每个托管选项与游戏所依赖的其他 AWS 服务（例如数据库、分析或内容交付服务）的集成程度。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

与其他 AWS 服务集成

服务之间的无缝集成提供了运营优势，例如改进的性能监控，以及在游戏组件、游戏服务器、游戏后端服务和可观察性解决方案之间高效的安全数据传输。

例如，协调直播游戏的流量变化可能很复杂。Amazon Route 53 将帮助您保持最新的 DNS 记录，从而简化协调的流量转移。 AWS Global Accelerator 流量拨号使您可以将一定比例的流量发送到另一个区域，并在维护期间保持游戏运行。

开发环境和工具

考虑每个架构选项所支持的开发工具、框架和环境。确认您选择的选项与您的游戏开发解决方案和编程语言一致，因为这可能会影响您的团队优化和维护游戏服务器性能的能力。在移动设备、主机和 PC 上交付游戏将增加工具和测试的复杂性。跨系统支持对于多游戏工作室尤其重要，在这些工作室中，集中式服务可以标准化跨游戏的开发最佳实践。

目标 CPU 架构和功能

考虑一下您的游戏引擎和游戏服务器进程的性能特征以及可用的 ARM 支持级别。评估您能否从基于ARM的Graviton或基于x86的处理器的更高性价比中受益。 AMD64 您需要使用 AES-NI 加密、AVX 或 Turbo Boost 等英特尔功能吗？ 查看专用主机类型以确定单插槽实例系列与多插槽实例系列。使用多插槽实例系列时，请考虑在游戏服务器进程中使用 NUMA 固定和 L3 缓存共享。使用 C 状态和 P 状态配置，通过调整频率时钟和降低睡眠水平，为游戏获得最佳性能。

实施步骤

- 选择与 ACM 等 AWS AWS Secrets Manager服务无缝集成的托管选项，以帮助简化性能监控、保护数据交付并减少手动操作任务。
- 验证您的托管选项与开发环境、框架和编程语言之间的兼容性，以有效优化和维护服务器性能。
- 评估 CPU 架构要求，利用 Graviton 获得性价比，利用 x86 获得 AES-NI、AVX 和 Turbo Boost 等特定功能，并通过 NUMA 固定和 C 状态/P 状态调整来优化服务器性能。

区域选择

GAMEPERF02：如何确定托管游戏基础设施的地理区域？

为游戏基础设施选择理想的位置可以提高玩家和后端的网络性能。考虑您的玩家群来自何处以及您的社区或服务器是如何构建的，这对于地理区域的长期增长和可持续性非常重要。部署分离的游戏服务器基础设施和后端服务可以提高整体运营效率，并通过使用多个区域、本地区域和前哨基地来托管游戏，从而提高灵活性。

最佳实践

- GAMEPERF02-BP01 选择靠近玩家的主区域
- GAMEPERF02-BP02 设计一种方法，支持将对延迟敏感的游戏基础设施放在靠近玩家的地方以提高性能

GAMEPERF02-BP01 选择靠近玩家的主区域

在首次发布游戏时，您应根据与业务利益相关者的讨论来确定在哪里部署基础架构，例如发布团队，他们将确定游戏预计在哪里向玩家开放，以及他们将发布前的营销和广告工作重点放在哪里。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

您的业务利益相关者还应该有刺激需求的机制，以更好地了解玩家的接受度和生存能力。例如，这些团队将拥有诸如游戏预订、营销活动和活动之类的机制，供玩家在发布前注册兴趣的公共电子邮件列表，以及其他方法来建立相关信号，以确定游戏在发布时可能拥有最多的玩家。该游戏还可能使用区域推出策略，包括游戏测试和软发布阶段来确定区域玩家的需求。

选择一个靠近您的玩家群和开发者并具有托管游戏所需的 AWS 服务和功能的主区域。主场 RSegion 将是游戏后端服务运行的地方，也可能运行游戏服务器。根据支持的服务、与边缘位置的连接、与故障转移区域的距离以及可用区域的数量来评估主区域。如果您使用的是本地区域，请考虑父区域有时位于不同的地理区域。举个例子：智利圣地亚哥本地区域 us-east-1-scl-1a 将弗吉尼亚北部 us-east-1 作为其母区域，尽管它在地理上更接近圣保罗 sa-east-1。

实施步骤

- 根据来自预发布活动（例如预订、营销活动和兴趣注册）的玩家需求信号，确定部署区域。
- 选择一个靠近主要玩家群和开发者的主区域，确保它支持所需的 AWS 服务、边缘位置和故障转移区域。
- 考虑到父区域在地理上可能与本地区域的位置不同，请仔细评估本地区域。

GAMEPERF02-BP02 设计一种方法，支持将对延迟敏感的游戏基础设施放在靠近玩家的地方以提高性能

将游戏服务器等延迟敏感型基础设施单独放置可以最大限度地减少长网络路由的影响。可重复的部署可以让你轻松维护多个地点，从而为玩家提供更高的性能。Ping 是游戏用户界面中浮出水面的常见指标，低 ping 可能是一种差异化能力。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

首次发布游戏时，您可能还没有足够的玩家群信息，无法充分了解在哪里部署最接近对玩游戏最感兴趣的玩家的基础架构。这是一个常见的挑战，您应该通过设计一种架构来为这种情况做好准备，该架构允许您快速调整主机放置策略，将服务器部署在离玩家更近的地方。游戏开发者通常会定期评估其游戏基础设施部署，以此作为发布后的反复分析，通过迭代方法逐步投资于改进。

最佳做法是使用 infrastructure-as-code模板（例如 AWS CloudFormation 或 Terraform by Hashicorp）来配置启动关键游戏服务所需的基础架构 VPCs，例如子网配置和依赖关系，以便您可以参考这些模板，在需要时快速对其进行自定义，然后将其部署到需要额外基础设施来支持玩家的位置。

您还应该确保了解如何发展当前的部署策略以允许将来的扩展。IaC 模板是可重复的，但不能替代网络规划。IPAM 可以管理你的。 VPCs子网规模、可用区域选择、IP 库存和跨账户可用区对齐。考虑网络很重要，更改后可能会对玩家造成干扰。部署在多个地理位置的游戏服务器将连接到您的游戏后端，而游戏后端托管在单个或多个主区域更为常见，可能需要额外的配置才能支持私有连接。应随着时间的推移不断评估这些注意事项，以便您可以随着游戏需求的变化或玩家要求的变化而更改游戏托管策略。

在确定游戏使用多少个游戏托管位置时，请考虑以下因素：

- 改善玩家体验质量：通过添加额外的游戏托管地点，你能在多大程度上改善玩家体验？ 这样做可以获得多少增量性能提升？ 您将如何衡量这种绩效改进？
- 优先考虑哪些玩家群体：如果您添加额外的游戏托管地点，可以改善多少玩家的体验？ 你会优先考虑哪些玩家群体或地理位置？
- 变化的下游影响：如果你改变游戏托管策略，这将如何影响玩家的配对等待时间？ 玩家池中的比赛规模、技能平衡或玩家人数能否适应游戏托管位置策略的变化？ 支持更多的地点可能会分散玩家群，增加成本和复杂性。

在确定在何处添加或移除游戏托管位置时，应评估所有这些注意事项。例如，你可以选择优先改善位于游戏体验效果最低的地理位置的玩家的体验，或者优先考虑那些表达最直言不讳的公众反馈的玩家。您也可以选择将玩家获利纳入优先事项，例如，将注意力集中在为游戏创造重要收入来源的地理位置的玩家的体验上，或者如果您引入性能改进，则有可能产生增量收入。

除了中的托管基础架构外 AWS 区域，您还可以使用 L ocal Z ones（的扩展）来托管游戏服务器和其他对延迟敏感的应用程序，例如离玩家更近的语音聊天服务器。 AWS 区域您也可以选择在 Local Zones 中运行游戏开发基础架构，以改善游戏开发团队的体验。例如，您可以使用 Local Zones 来解决使用案例，例如在离游戏开发者更近的地方托管自我管理的源代码控制服务器的副本，以及使用部署到开发工作室附近一个或多个本地区域的 Amazon EC2 实例、EBS 卷和亚马逊 FSx 文件系统的用户提供游戏开发虚拟工作站和内容存储，而无需在本地托管基础设施。

当 “区域” 或 “本地区域” 不适用于同一地理区域时，Outpost s 是一个不错的选择。 AWS 应考虑从您的数据中心到您的数据中心的连接，以使游戏服务器与后端系统保持可靠性。 AWS Outposts 而且 Outpost Servers 专为使用相同的服务 AWS 在您的数据中心中运行而设计，可 APIs 帮助您在任何地方运行游戏时创建一致的部署模型。可以将多个机架组合成一个逻辑前哨基地，并且可以共享 AWS 账户基础架构。硬件生命周期由管理 AWS ，交货时间可以短至 3 个月。

如果您使用容器开发游戏，并且希望灵活地采用混合部署架构，使用可部署在您自己的本地基础架构上的开源软件，则可以使用 ECS Anywhere 或 EKS Anywher e 作为或 Local Zones 的替代方案。 AWS Outposts 如果您使用亚马逊托管 GameLift；Amazon GameLift Anywhere 可用于在本地硬件上运行服务器构建，这可以加快开发过程，使您能够使用本地区域或将自己的金属注册为队列的一部分。

实施步骤

- 使用 AWS CloudFormation 或Terraform之类的 infrastructure-as-code工具进行可重复的部署，从而可以根据玩家需求快速自定义和扩展游戏托管位置。
- 评估玩家体验的改善、玩家群体的优先顺序以及下游影响，例如添加或移除游戏托管地点时的配对时间。
- 使用 L AWS ocal Zones、Outposts 或 ECS Anywhere、EKS Anywhere 或 GameLift Anywhere 等混合选项来优化延迟敏感型基础架构并支持不同的部署需求。

迭代开发

GAMEPERF03：如何使用 Amazon 提高迭代开发 GameLift 的性能效率？

Amazon GameLift 提供了在本地测试环境中对游戏进行开发和性能测试 end-to-end的工作流程。

最佳实践

- GAMEPERF03-BP01 使用 Amazon GameLift Anywhere 和 GameLift测试工具包
- GAMEPERF03-BP02 测试游戏服务器的性能和可扩展性
- GAMEPERF03-BP03 优化 GameLift容器的资源利用率

GAMEPERF03-BP01 使用 Amazon GameLift Anywhere 和 GameLift测试工具包

要通过迭代开发过程提高性能效率，请使用 Amazon GameLift Anywhere 和亚马逊 GameLift 测试工具包来建立全面的测试环境。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

这种方法允许快速迭代、高效的数据收集和详细的性能分析。关键步骤包括：

创建测试环境

使用 Amazon GameLift Anywhere 设置本地或基于云的测试环境。此设置无需将每个游戏服务器版本迭代上传到托管队列，从而缩短了激活时间。

整合 Amazon GameLift 测试工具包

将 Amazon GameLift 测试工具包纳入您的开发工作流程。该工具包提供脚本、工具和库，用于可视化 Amazon GameLift 基础设施、启动虚拟玩家以及使用 FlexMatch 模拟器迭代 FlexMatch 规则集。它简化了 Amazon GameLift 资源的集成和管理，使您可以自动执行常见任务并收集性能分析所需的数据。

快速构建和测试周期

使用新版本快速更新测试队列，启动测试队列，然后开始测试。这有助于 build-test-repeat缩短周期，使开发者能够验证游戏玩家体验的各个方面，包括多人游戏互动。

全面测试

测试您的游戏服务器与 Amazon GameLift 服务器 SDK、后端服务交互、配对配置和其他 GameLift 托管功能的集成。利用 GameLift 测试工具包自动测试并收集详细的性能指标，确保游戏组件无缝协作。

分析性能数据

使用 GameLift 测试工具包收集的数据来分析性能瓶颈并优化您的游戏服务器。该工具包有助于跟踪关键指标、识别问题并做出以数据为依据的决策，从而提高绩效效率。

通过将 Amazon GameLift Anywhere 和 GameLift 测试工具包整合到您的迭代开发流程中，您可以通过快速测试、全面的集成检查和详细的性能分析来显著提高性能效率。

实施步骤

- 使用 Amazon GameLift Anywhere 创建测试环境，从而缩短游戏服务器构建的激活时间并实现快速迭代。
- 集成 Amazon GameLift 测试工具包，在开发过程中自动执行测试任务、模拟玩家和验证 FlexMatch 配置。
- 使用 GameLift测试工具包收集和分析性能数据，以识别瓶颈、优化游戏服务器并提高性能效率。

GAMEPERF03-BP02 测试游戏服务器的性能和可扩展性

要测试游戏服务器的性能和可扩展性，请使用 Amazon GameLift 功能和测试工具包实施强大的 GameLift 测试框架。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

主要做法包括：

迭代测试

使用 Amazon GameLift Anywhere 队列创建基于云的托管环境，您可以在其中迭代构建和测试游戏组件。该环境应反映真实的托管条件，从而实现真实的性能和可扩展性测试。

游戏服务器集成测试

测试您的游戏服务器与 Amazon GameLift 服务器 SDK 的集成，包括使用 AWS CLI 或 GameLift 测试工具包启动新的游戏会话和跟踪游戏会话事件。这可以验证游戏服务器在 GameLift 环境中是否正常运行。

使用 GameLift 测试工具包自动测试并收集详细的性能指标。该工具包允许您可视化 GameLift 基础架构，启动虚拟玩家进行负载测试，以及使用 FlexMatch 模拟器迭代 FlexMatch 规则集。它对于扩展 ECS Fargate 任务特别有用，这些任务通过创建大量并发游戏会话来对服务器基础架构进行压力测试，从而模拟玩家会话。

可扩展性测试

尝试游戏会话队列设计、多地点队列、竞价和按需队列以及多种实例类型。测试游戏会话放置选项、延迟策略和队列优先级设置。配置容量扩展以满足玩家需求，并验证系统是否可以在不同条件下处理预期的负载。

实施步骤

- 使用 Amazon GameLift Anywhere 为迭代性能和可扩展性测试设置真实的测试环境。
- 测试游戏服务器与 GameLift 服务器 SDK 的集成，便于在 GameLift 环境中进行正确的会话管理和事件跟踪。
- 使用测试工具包进行可扩展性 GameLift 测试，模拟玩家负载，测试会话队列，并验证队列扩展、延迟策略和优先级设置。

GAMEPERF03-BP03 优化 GameLift容器的资源利用率

要优化 GameLift 集装箱的资源利用率，请有效地设计集装箱船队并设置精确的资源限制。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

主要指导方针包括：

- 容器组设计：将您的软件组织成容器组。主容器应将您的游戏服务器应用程序和 Amazon A GameLift gent 捆绑在一起。将 sidecar 容器用于其他软件，以管理依赖关系并设置容器特定的内存和 CPU 使用限制。
- 设置资源限制：为每个容器组确定所需的内存和 CPU 资源。为单个容器设置可选限制，以验证它们是否有预留资源，但如果有其他资源可用，也可以超过这些限制。这有助于防止资源争用和潜在的容器故障。
- 守护程序容器组：考虑将守护程序容器组用于不需要随主容器组扩展的后台或监视进程。这可以验证基本的后台任务是否得到有效处理，而不会影响主游戏服务器进程。

实施步骤

- 设计容器组，其中包含游戏服务器和 GameLift 代理的主容器，以及用于管理依赖关系的 sidecar，并具有特定的内存和 CPU 限制。
- 为每个容器组设置资源限制以保留所需的资源，同时允许控制资源使用以避免争用。
- 使用守护程序容器组执行后台或监控任务，确保它们在不影响主游戏服务器进程的情况下高效运行。

计算和硬件

GAMEPERF04：如何阻止游戏会话影响在同一游戏服务器实例上运行的玩家？

在游戏服务器运行后，无论资源利用率 AWS、底层计算或饱和度如何，您都需要监控其性能以提供优质的玩家体验。

最佳实践

- GAMEPERF04-BP01 监控游戏服务器进程以检测问题

- GAMEPERF04-BP02 性能使用模拟和真实的游戏场景测试您的游戏服务器

GAMEPERF04-BP01 监控游戏服务器进程以检测问题

您可以为每个实例运行多个游戏服务器进程，以有效利用游戏服务器实例上的资源。如果是，请设计您的架构，使托管游戏会话的单个游戏服务器进程不会对托管在同一实例上的其他游戏会话造成不利影响。使用指标来了解游戏位置和游戏模式类型如何影响游戏服务器实例的性能。结合低负载（大厅、商店或单人游戏教程）和高负载（排名、多人游戏或高技能游戏）流程，以避免游戏服务器实例的热点。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

通过收集有关 ping 时间和抖动、丢帧、API 响应时间、错误和成功完成游戏循环的遥测数据，通过客户端和服务器端指标监控玩家体验。将这些事件的时间戳与玩家支持问题和服务器日志相关联，以确定性能瓶颈。诸如 Dtrace、ftrace、uper f 和 e B PF 之类的工具可用于对系统性能进行深入调查和分析。

对游戏服务器实例可用的有限资源进行监控，以便在单个游戏服务器进程突破预先确定的资源预算阈值时生成警报。当突破阈值时，您可能需要将游戏服务器软件配置为将相关的系统和游戏服务器日志转储到持久存储中，例如中央日志解决方案，以便您的游戏服务器工程师可以调查这种行为。此外，您的游戏服务器实例应配置为报告该实例上运行的每个游戏服务器进程的指标，以便除了游戏服务器实例的总体指标外，您还可以监控这些单独的游戏服务器进程。

例如， GameLift 提供了用于监控游戏会话的指标，通过使用可在游戏服务器实例上配置的 A ma zon A CloudWatch gent 收集的自定义游戏特定指标和日志来增强这些指标。您的指标可以在其他工具中查看 CloudWatch 或导出到其他工具，例如 Amazon Managed Grafana，它与单点登录集成，使可能无法访问管理控制台的用户可以直接访问指标。请参阅以下使用 Amazon 管理日志和指标的最佳实践 GameLift，Amazon 还支持查看单个游戏会话日志。

实施步骤

- 每个实例运行多个游戏服务器进程，混合使用低负载和高负载游戏模式，以避免出现热点问题并验证资源利用率的平衡。
- 监控客户端和服务器端指标，例如 ping、抖动、丢帧和 API 响应时间，并将这些指标与服务器日志和玩家报告的问题关联起来，以识别瓶颈。
- 为每个游戏服务器进程配置资源监控，生成阈值违规警报，并将日志存储在耐用存储空间中，以便使用 Amazon Managed Grafana CloudWatch 等工具进行分析。

GAMEPERF04-BP02 性能使用模拟和真实的游戏场景测试您的游戏服务器

进行性能测试并评估各种游戏场景，以确定游戏服务器进程是否适当地处理固定资源的利用率，例如 EC2 实例内存、CPU 和网络带宽。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

使用可以反映玩家常见游戏路径和行为的机器人创建模拟游戏测试，可以确定游戏服务器在不同的使用场景下如何处理这个问题。例如，您可以实现一个解决方案，例如分布式负载测试 AWS，您可以对其进行自定义以运行游戏客户端模拟，或者在游戏客户端版本上进行自定义以生成游戏场景。运行内部游戏测试，并使用 QA 团队对游戏的各种功能进行压力测试，这样您就可以确信自己的游戏旨在实现最佳性能。 AWS Device Farm可用于在多种设备类型上对您的 iOS、Android 和浏览器游戏进行移动和网络测试。

实施步骤

- 使用模拟常见玩家行为的机器人进行性能测试，以评估不同场景下的游戏服务器资源利用率。
- 使用诸如分布式负载测试之类的解决方案 AWS 来自定义和模拟游戏场景以进行压力测试。
- 进行内部游戏测试，并使用诸如在各种设备上 AWS Device Farm 进行移动和浏览器游戏测试之类的工具。

计算选择

GAMEPERF05：如何为游戏选择合适的计算解决方案？

计算性能因实例大小和系列而异。使用来自不同容量池的多个计算选项是有益的。制定机队组成策略，该策略优先考虑性能，但要包括足够的多样性，以避免容量不足的错误。

最佳实践

- GAMEPERF05-BP01 跨多种计算类型对游戏性能进行基准测试
- GAMEPERF05-BP02 将 non-latency-sensitive计算任务移至异步工作流程

GAMEPERF05-BP01 跨多种计算类型对游戏性能进行基准测试

对于游戏服务器工作负载，没有单一的方法可以确定托管游戏服务器的最佳计算解决方案。对游戏服务器进行基准测试的常见策略是从计算优化 EC2 的 “c” 实例开始，因为该实例系列为计算密集型的工作负载提供了高性能。或者，如果您的游戏需要大量内存来实现特定功能，则内存优化的实例可能最合适。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

如果您的工作负载占用了大量网络资源，请考虑实施网络优化的实例（通常在实例名称中使用 “n” 表示），避免使用可突发实例类型 “t”，因为积分用完后性能会降低。游戏对延迟和丢包很敏感，因此建议使用 EC2 增强的联网来帮助提高游戏服务器的网络性能。增强联网使用单根 I/O 虚拟化 (SR-IOV) 为支持的实例类型提供高性能联网功能。SR-IOV 是一种设备虚拟化方法，与传统的虚拟化网络接口相比，它提供了更高的 I/O 性能和更低的 CPU 利用率。增强联网可以提高带宽，提高每秒数据包数 (PPS) 性能，并不断降低实例间的延迟。使用 Elastic Network Adapter 增强联网适用于最新的 EC2 实例类型，定期更新以受益于新实例的性能增强和 AWS Nitro 虚拟机管理程序的改进，这一点很重要。

如果您的游戏在多个 EC2 实例类型上的表现相似，则应考虑使用多种实例类型来托管游戏服务器。监控一段时间内的性能，并在托管足够多的制作游戏会话以确定性能趋势后进行进一步优化。请记住，当您在游戏中添加需要不同资源分配的新功能时，您的计算需求可能会发生变化。您可以将 EC2 Auto Scaling 组配置为使用多种实例类型，也可以使用单独的 Auto Scaling 组来托管运行不同实例类型的游戏服务器实例，这样可以更轻松地管理指标的关联和聚合。

评估您的游戏在不同类型的处理器上的表现，例如基于英特尔的实例、基于 AMD 的实例和基于 ARM 的 Graviton 实例。虚幻引擎5.1.1或更高版本可以为Graviton编译游戏服务器，并可以提高游戏的性价比。在每个系列中执行不同规模的扫描和饱和度测试，以确定利用率和性能一致的最佳位置。

当使用容器和 Lambda 函数托管游戏时，您还应该对游戏性能的影响进行基准测试。对于不需要长寿命游戏服务器进程的用例，例如异步游戏和游戏后端服务，可以考虑在 Lambda 中使用无服务器架构，这样可以简化游戏运营团队的管理和操作，并允许您在全球范围内更快地将游戏部署到许多人。AWS 区域有关无服务器最佳实践，请参阅无服务器应用程序视角——WellArchitected Framework。

实施步骤

- 在计算优化的 “c” 实例上对游戏服务器进行基准测试，用于 CPU 密集型工作负载，内存优化型实例用于内存繁重的任务，在网络优化的 “n” 实例上对游戏服务器进行基准测试，以实现高网络吞吐量。
- 在支持的实例上使用带有弹性网络适配器 (ENA) 的增强联网，以提高网络性能、减少延迟并提高数据包处理速率。

- 评估和测试多种实例类型、处理器（英特尔、AMD、Graviton）以及容器或 Lambda 托管选项，随着游戏功能的发展调整计算解决方案。

有关更多信息，请参阅为您的全球游戏服务器选择正确的计算策略。

GAMEPERF05-BP02 将 non-latency-sensitive计算任务移至异步工作流程

在优化游戏性能时，请务必记住，只有客户端和游戏后端之间的某些交互必须以同步方式执行。您应该从玩家体验的角度考虑每项功能，并确定某些互动是否需要同步通信（这些通信会阻塞且资源密集），或者这些功能是否可以异步实现。在实现网络呼叫时，请使用异步、非阻塞的方法。此外，还应将游戏后端配置为通过将任务卸载到队列并在可能的情况下优先考虑对客户端的快速响应，从而以高效的方式执行工作。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

例如，在玩家会话结束时更新排行榜可以异步实现，这样客户端就无需等待排行榜更新完成。取而代之的是，在游戏客户端上异步实现这一点，并考虑设计您的后端服务以将这些类型的操作推送到队列中，例如 Amazon SQS。使用此架构，将您的后端配置为接受请求，将其排队到有助于持久存储消息以进行异步处理的 SQS 中，并立即回复客户端。排行榜更新完成后，后端可以向游戏客户端发送更新，以便更新玩家对排行榜的视图。

或者，玩家只需访问游戏的排行榜屏幕即可检索最新数据，这可以向您的后端发出网络请求，以从缓存中检索最新数据。

实施步骤

- 确定客户端-后端交互是否需要同步通信；尽可能采用异步、非阻塞的方法来优化资源使用。
- 使用 Amazon SQS 来卸载排行榜更新等非关键任务。
- 允许客户端异步获取更新的数据，例如按需或通过后台更新检索最新的排行榜数据。

资源

- 了解微服务的异步消息传递
- Lambda-使用服务集成和异步处理

数据管理

GAMEPERF06：如何高效管理和分析游戏服务器日志并存储不同类型的游戏数据以获得最佳性能？

游戏可以包含玩家数据、游戏日志和服务器日志，这些数据应尽可能相互分离。集中日志摄取和生命周期管理可以让你的游戏团队深入了解游戏和服务器上正在发生的事情，从而使你的游戏团队受益。

最佳实践

- GAMEPERF06-BP01 集中日志收集和存储
- GAMEPERF06-BP02 根据访问模式对游戏数据进行分类和存储
- GAMEPERF06-BP03 启用高效的日志格式化和批处理
- GAMEPERF06-BP04 实施日志轮换和保留策略
- GAMEPERF06-BP05 使用监控和可视化工具

GAMEPERF06-BP01 集中日志收集和存储

实施集中式日志收集和存储解决方案，以从游戏服务器实例收集日志，然后 GameLift.

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

使用 Amazon CloudWatch Logs 等服务收集、监控和存储来自游戏服务器和 GameLift 实例的日志数据。 CloudWatch Logs 为日志管理提供了一种可扩展且完全托管的解决方案，便于在不影响游戏服务器性能的情况下高效存储和检索日志数据。如果您正在运行 CloudWatch Logs 代理，请考虑各种安装类型和配置选项，例如批处理大小、缓冲持续时间，以最大限度地减少对游戏服务器的影响。将游戏服务器实例视为临时实例，并尽可能减少对本地化日志的依赖。制定实施日志最佳实践的集中政策。

实施步骤

- 使用 Amazon Lo CloudWatch gs 收集、监控和存储来自游戏服务器实例的日志数据 GameLift，从而促进集中和可扩展的日志管理。

GAMEPERF06-BP02 根据访问模式对游戏数据进行分类和存储

根据访问模式和存储要求将游戏数据归类为不同的类型。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

常见类别包括玩家数据、游戏存档、永久世界存储和分析数据。

实施步骤

为每种数据类型使用适当的存储解决方案，以优化性能和成本效益：

- 玩家数据：使用 Amazon DynamoDB（一个快速且可扩展的 NoSQL 数据库）来存储玩家资料、偏好和进度数据。DynamoDB 的低延迟访问和自动扩展功能为玩家数据提供了高效的检索和更新。
- 游戏存档：使用 Amazon S3 存储游戏存档和检查点。S3 为存储大量游戏存档数据提供了很高的耐久性和可扩展性。可以考虑使用 S3 传输加速或 Amazon CloudFront 来更快地上传和下载游戏存档。
- 永久世界存储：对于具有永久世界状态或共享游戏数据的游戏，可以考虑使用亚马逊 DynamoDB、Amazon 或 Amazon MemoryDB。 ElastiCache ElastiCache 而且 MemoryDB 提供内存中的键值存储，而 DynamoDB 是支持固态硬盘的 NoSQL 数据库。这些服务提供了对存储数据的快速访问，从而缩短了游戏服务器进程保存游戏状态所需的时间，从而提高了整体流程性能。
- 分析数据：使用适用于 Apache Kafka 的 Apache Managed Streams 或 Kinesis Data Streams 的亚马逊托管流来摄取来自游戏数据制作者的数据流。适用于 Apache Flink 的亚马逊托管服务可用于实时转换和分析，然后发送到 Amazon Data Firehose 进行处理并交付到后端数据湖、仓库和分析服务。Game Analytics Pipeline 指南 AWS说明了这些服务如何协同工作以提供近乎实时的批量分析。

GAMEPERF06-BP03 启用高效的日志格式化和批处理

将您的游戏服务器进程配置为以结构化和可解析的格式（例如 JSON）生成日志。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

实施日志批处理技术，以最大限度地减少日志数据从游戏服务器传输到集中式日志存储的频率。批处理日志可以减少网络开销并提高游戏服务器性能。使用详细或调试级别的日志作为例外而不是默认日志，因为它们可能会导致性能和成本损失，应尽可能避免这种损失。

GAMEPERF06-BP04 实施日志轮换和保留策略

制定日志轮换和保留策略，以管理日志数据的增长并优化存储利用率。

在未建立这种最佳实践的情况下暴露的风险等级：低

实施指导

将游戏服务器配置为根据大小或时间间隔自动轮换日志。在 Amazon Logs 中定义 CloudWatch 日志保留策略，自动存档或删除主动分析或故障排除不再需要的旧日志数据。

GAMEPERF06-BP05 使用监控和可视化工具

使用监控和可视化工具深入了解您的游戏服务器性能并确定优化机会。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

使用 Amazon CloudWatch 监控关键指标并为主动通知设置警报。利用适用于 Prometheus 的亚马逊托管服务和 Amazon Managed Grafana 等工具，从游戏服务器和基础设施中收集、查询和可视化指标。创建信息仪表板以跟踪性能、识别瓶颈并进行数据驱动的优化。

网络和内容分发

GAMEPERF07：如何设计配对服务以优化性能？

玩家技能、互联网服务提供商 (ISP) 的素质和玩家人口分布作为性能调整的一个维度需要考虑。可以将游戏会话放置在战略位置优越的服务器上，以创造公平的竞争环境并举办公平的游戏。

最佳实践

- GAMEPERF07-BP01 为您的游戏定义网络延迟阈值
- GAMEPERF07-BP02 为每种游戏模式和游戏托管区域运行单独的配对服务
- GAMEPERF07-BP03 定期监控配对表现
- GAMEPERF07-BP04 定期监控网络性能
- GAMEPERF07-BP05 使用网络加速技术提高互联网性能

GAMEPERF07-BP01 为您的游戏定义网络延迟阈值

开发多人游戏时，请确认您的游戏基础设施不会给玩家带来不必要的延迟。如果你的游戏对网络延迟很敏感，那么你应该在配对逻辑中设置延迟阈值，以便优先将玩家置于托管在附近游戏服务器位置或 AWS 区域 符合你理想玩家体验目标的游戏服务器会话上。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

在许多对延迟敏感的游戏中，通常会让游戏客户端 ping 游戏的每个基础设施位置，以收集性能数据，例如网络延迟、抖动和数据包丢失，并将这些数据报告给指标收集后端以便进行分析。将玩家匹配到游戏会话中时，您可以将游戏配置为将游戏客户端感知到的网络延迟纳入游戏服务器基础架构，作为配对服务逻辑中使用的输入之一。

GAMEPERF07-BP02 为每种游戏模式和游戏托管区域运行单独的配对服务

如果您的游戏提供多种游戏模式供玩家选择，则应将每种模式的配对系统分开，以便您可以根据每种游戏模式的独特要求独立调整其性能并减少资源争用。每种游戏模式都可能对可接受的延迟、比赛大小以及其他针对游戏的自定义配对逻辑有独特的要求。它们还可能会吸引不同类型的玩家。将每种游戏模式的配对服务作为单独的软件部署来运行，这样您就可以独立进行性能测试和操作游戏模式。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

例如，您可以将它们作为每种游戏模式的单独的 Lambda 函数运行，也可以将它们作为单独的基于容器的服务部署进行操作。

将配对服务部署到游戏服务器位置附近的多个区域。玩家流量将走多条路线，因此，配对服务必须保持多条路径的 up-to-date延迟特征， ISPs 以提高低延迟游戏会话放置的效率。 GameLift FlexMatch 为媒人选择区域提供了更多指导，并包括将您的媒人与多区域游戏会话队列集成的功能。

GAMEPERF07-BP03 定期监控配对表现

为玩家优化游戏性能的最引人注目的方法之一是缩短他们进入游戏会话之前必须等待的时间。漫长的等待时间可能会导致玩家失去兴趣并导致流失，因此在设计配对解决方案时务必考虑这一点。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

在为游戏设计配对配置时，请创建规则来确定用于形成匹配的条件。你应该考虑这些规则对系统性能的影响，尤其是玩家的等待时间。在部署对配对实现的更改（例如添加新的配对条件或过滤器）之前，请事先对此进行测试，或者考虑将此更改作为金丝雀或 A/B 测试逐步发布给一小部分样本玩家，以收集绩效指标，然后再将此更改引入整个玩家群体。

配置您的配对服务以生成详细日志，以了解适用于每个配对请求的条件或规则。这有助于审查并在必要时调整配对的实施。

例如，亚马逊 GameLift FlexMatch提供完全托管的配对服务，该服务可用作独立服务，由您自己托管游戏服务器，也可以与托管在亚马逊上的游戏服务器一起使用 GameLift。 FlexMatch 可以向 Amazon 生成事件通知 EventBridge，请参阅设置 FlexMatch 事件通知。使用亚马逊简单通知服务 (Amazon SNS) Simple Notification Service 接收 JSON 格式的配对数据，这样您就可以自动处理和存储这些信息以进行分析，从而提高配对性能。

设置指标以跟踪您的配对服务需要多长时间才能为玩家找到合适的游戏会话。定期查看配对持续时间指标，并将这些时间与玩家的行为和社区情绪相关联。使用这些数据来制定合适的配对超时阈值，这些阈值可以包含在配对规则配置中。

例如，Amaz GameLift FlexMatch on 支持定义配对请求超时以及创建配对规则，允许随着时间的推移而放松要求。此功能允许您创建配对，该配对可以进行调整，以便在难以找到匹配项时可以直接创建匹配项并让玩家进入游戏会话。

GAMEPERF07-BP04 定期监控网络性能

对于竞技游戏来说，拥有稳定的玩家体验非常重要。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

对于更大的玩家群来说，可靠地保持50毫秒的游戏比一个玩家的ping为10毫秒而另一个玩家的ping为70毫秒的比赛更公平、更有趣。互联网服务提供商的路由更改可能会影响部分玩家群体，您的配对系统需要进行调整。Amazon CloudWatch 网络监控可帮助确定问题出在您的游戏还是玩家互联网提供商身上。

实施步骤

- 使用 Amazon Cloudwatch 网络监控来跟踪网络性能并识别路由问题。

- 使用 VPC 流日志来识别异常流量模式或丢弃的数据包，这可能表明网络拥塞、ISP 问题或影响玩家延迟的配置错误。

GAMEPERF07-BP05 使用网络加速技术提高互联网性能

除了将对延迟敏感的游戏基础设施放在离玩家更近的地方，您还可以通过优化游戏的网络性能来改善玩家体验。 AWS 使用 BGP 协议来影响互联网路由，从而使用互联网服务提供商提供的通往边境网络的最快路径。如果您运营自己的网络，并且需要对路由行为和 BGP 广告进行更多的控制和观察，则可以使用私有对等互连或 Direct Connect 将流量从互联网路由到正在运行的游戏。 AWS

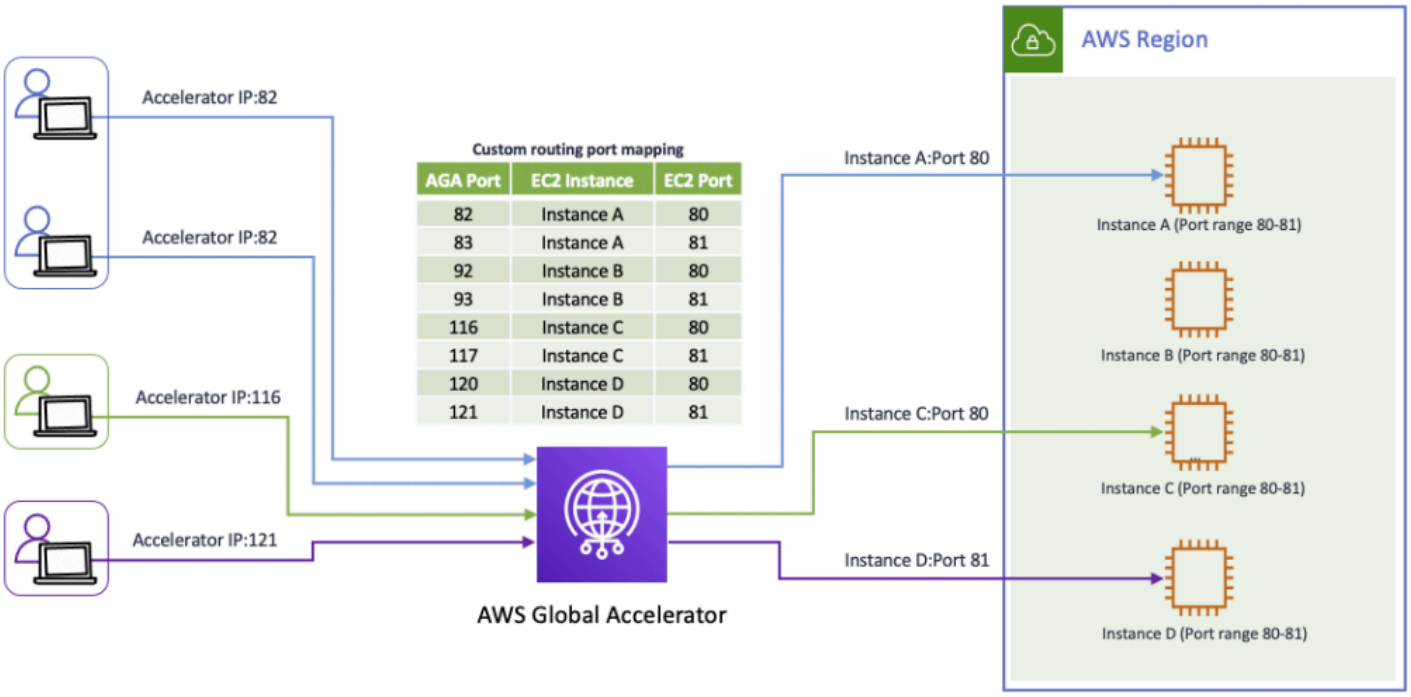
在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

考虑以下参考架构，以支持改进的互联网性能和响应能力。

使用全球加速器增强游戏网络性能

对于完全托管的网络路由解决方案，AWS Global Accelerat or 使用 AWS 全球网络来提高应用程序的网络性能，全球网络可用于加速游戏流量、语音聊天和实时消息流量，以及其他对延迟敏感的应用程序，同时为游戏服务器提供快速故障转移。Global Accelerat or 自定义路由加速器可以与您的配对服务集成，从而使用静态 anycast IP 地址和端口为多个玩家提供确定性路由到同一个游戏会话。



您的游戏开发团队可能分布在世界各地，需要对共享内容或资产的高效访问权限。为了提高存储在 Amazon S3 存储桶中的共享内容的性能，您可以使用 S3 跨区域复制设置跨区域数据的双向复制，以便用户可以访问离他们更近的存储桶中的数据。要简化这种访问模式，请使用 S3 多区域接入点，该接入点使用全球加速器加速通过全球网络向 S3 发出的请求。

有关更多信息，请参阅利用 AWS 全球加速器和Amazon GameLift FleetiQ改善玩家体验。

实施步骤

- 使用 AWS Global Accelerator 帮助提高游戏流量、语音聊天和实时消息传递的网络性能，同时促进快速故障转移到游戏服务器。
- 配置 Global Accelerator 自定义路由加速器以与您的配对服务集成，从而使用静态任播将玩家确定性路由到游戏会话。 IPs
- 启用 S3 跨区域复制，为分布式游戏开发团队跨区域复制共享内容。
- 使用 S3 多区域接入点加速全球分布式用户通过 AWS 全球网络访问 S3 数据。

流程和文化

GAMEPERF08：如何使游戏的性能标准与玩家和开发者的期望保持一致？

了解你的玩家和开发者是提高性能效率的最重要方面之一。交付一款操作开销低的高性能游戏是向玩家和开发者展示你关心他们的体验并能让你的游戏和工作室脱颖而出的最佳方式之一。

最佳实践

- GAMEPERF08-BP01 通知玩家并将其纳入您的流程
- GAMEPERF08-BP02 使解决方案选择与工程团队的技能和专业知识保持一致

GAMEPERF08-BP01 通知玩家并将其纳入您的流程

提供在游戏中显示延迟、每秒帧数和丢弃的数据包等指标的选项。通过面向玩家的通信（例如状态页面）来解决基础设施问题和维护停机时间。通过包括开发者博客在内的玩家通讯来庆祝新的游戏地点，并设定对预期玩家体验改善的期望。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

包括玩家

提供一个简单的诊断提交流程，收集相关文件并将其附加到游戏客户端的玩家支持票证中。启用支持论坛，玩家可以在其中互相帮助，参与改善游戏体验

考虑权衡取舍与玩家期望

为了成本效益而移动后端系统可能不会引起玩家的注意，但是移动游戏服务器可能会改变 ping 时间。在扩大和减少游戏托管地点的理由时，对玩家保持一致和公平。

玩家社区和地区将有自己的特征，这可能会影响您对游戏的期望。例如，韩国拥有世界上最快的互联网，人们对游戏玩法的期望是个位数的延迟，这推动了竞争激烈的游戏。与主机和 PC 会话游戏相比，移动设备上的休闲游戏创造了不同的性能特征和使用模式。

登录和游说是体验的一部分，即使服务器因维护而处于离线状态，也应该感觉反应灵敏。突袭之夜计划或在大厅闲逛是玩家体验的一部分，在选择重点区域以提高性能效率时要考虑这一点很重要。玩家可能会让你的游戏客户端保持几个月的打开状态，有时他们可能只是偶尔登录来阅读补丁说明。作为工程流程和文化的一部分，Live Ops 游戏需要将整个玩家体验牢记在心。

实施步骤

- 提供游戏内指标，例如延迟、FPS 和丢包，并通过状态页面和面向玩家的更新来传达基础设施问题和维护计划。
- 在游戏客户端中实现诊断转储和提交功能，并创建一个支持论坛，以促进社区驱动的故障排除和改进。
- 根据玩家社区的期望量身定制性能优化，例如竞争区域的低延迟，或者为休闲和长时间游戏玩家提供响应式 login/lobby 体验。
- 设计 Live Ops 工作流程以考虑整个玩家体验，从活跃的游戏玩法到闲置的客户端行为，促进无缝互动。

GAMEPERF08-BP02 使解决方案选择与工程团队的技能和专业知识保持一致

在选择托管选项时，请评估您的团队在管理和优化游戏服务器性能方面的技能和专业知识。像 EC2 和容器这样的自托管解决方案需要更多基础架构管理、性能调整和扩展方面的知识。如果你的团队缺乏这些技能，那么像这样的托管服务 GameLift 可能更合适，因为它可以消除许多复杂性，让你的团队能够专注于游戏特定的优化。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

通过评估这些因素并对不同的托管选项进行性能测试，您可以选择最合适的解决方案，既能满足游戏的特定要求，又能优化性能效率。

资源

详细了解我们与绩效效率相关的最佳实践。

相关文档：

- AWS 架构中心
- 性能效率支柱 — WellArchit AWS ected 框架
- 将您的本地存储模式与 AWS 存储服务进行比较
- 实例存储实例的临时块存 EC2 储
- 使用 CloudWatch 代理收集指标、日志和跟踪
- CloudWatch代理人
- 如何在我的 EC2实例上开启和配置增强联网？
- 利用 AWS 全球加速器和亚马逊 GameLift FleetiQ 改善玩家体验
- Riot Games 技术博客：Valorant 的可扩展性和负载测试
- 采用混合解决方案的超大规模在线游戏 AWS
- 利用率饱和度与误差 (USE) 方法
- 亚马逊 EC2 Spot 实例
- 借助 Amazon 实现大规模绩效 ElastiCache
- 使用 Redis 的数据库缓存策略
- Amazon Virtual Private Cloud Connectivity Options
- 最佳实践设计模式：优化 Amazon S3 性能

相关基准：

- 基准亚马逊 EBS 交易量
- Amazon EC@ 实例网络带宽

相关工具：

- 虚幻引擎：测试和优化你的内容
- Unity 分析器
- 开放式 3D 引擎 (O3DE) 质量体系
- 监控 Amazon GameLift 服务器
- Amazon GameLift 测试工具包

相关视频：

- AWS re: Invent 2019：[重复 2] 亚马逊 EC2 基金会 (-R2) CMP211
- AWS re: Invent 2019：为下一代亚马逊提供动力 EC2：深入研究 Nitro 系统 (03-R2) CMP3
- 亚马逊 GameLift FleetiQ 入门 — 在线技术讲座 AWS
- Riot Games 的 Zach Blitz 谈用 AWS 它来改善游戏
- AWS re: Invent 2023- AWS Graviton：工作负载的最佳性价比 () AWS CMP334

相关培训：

- 游戏服务器托管在 AWS
- 将 Amazon GameLiftFleet IQ 用于游戏服务器
- 游戏 AWS 入门 — 第一部分
- 游戏服务器托管在 AWS
- AWS re: Invent 2023 — AWS Graviton：工作负载的最佳性价比 () AWS CMP334

成本优化

成本优化支柱包括在系统的整个生命周期中对其进行持续的完善和改进。这个过程从最初的概念验证设计到生产工作负载的持续运行。通过采用本 paper 中概述的实践，您可以构建和运营具有成本意识的系统，从而以最低的价格实现预期的业务成果。实施这些成本优化实践可以让您的企业最大限度地提高云投资的价值。

游戏是独特的创意项目，必须争夺玩家的注意力和游戏时间。在发布之前，游戏开发者通常对他们的游戏的受欢迎程度或持续时间缺乏清晰的了解。根据游戏的盈利策略、业务优先级和生命周期阶段，开发者在评估成本优化决策时需要权衡取舍。

例如，在一款备受期待的新游戏的发布前阶段，重点通常放在 speed-to-market功能开发和性能上。当务之急是验证基础设施能否扩展以满足玩家的高峰需求。相反，如果游戏不成功或开发速度放缓，则重点可能会转移到尽可能降低成本上，以便继续为现有玩家运营游戏。

许多游戏开发者还会同时操作多个游戏，这需要额外的考虑。基础设施、软件和员工等资源可以在多个直播游戏中共享，这样一款游戏的损失就可以用另一款游戏的利润来抵消。在这种情况下，专注于成本优化可以改善整个游戏组合的财务状况。

鉴于游戏的独特商业模式、规模和不可预测性，以下关键问题可以指导成本优化决策：

- 如何衡量每个玩家、系统和游戏功能的基础设施成本？
- 在我的游戏当前生命周期阶段，成本优化和玩家体验之间的平衡是什么？
- 如何使用正确的 AWS 资源定价模式最大限度地提高投资回报率？

应用这些最佳实践并提出正确的问题可以帮助游戏开发者构建和运营具有成本意识的系统，在实现业务成果的同时最大限度地降低成本。

聚焦领域

- 设计原则
- 践行云财务管理
- 支出和使用情况意识
- 具有成本效益的资源
- 数据传输成本
- 管理需求和供应资源
- 随着时间的推移进行优化

- 资源

设计原则

除了 Well-Architected Framework 的成本优化支柱中的设计原则外，以下设计原则还优化了在云端运行游戏工作负载的成本。

- 衡量每个玩家、系统和游戏功能的基础设施成本：了解并跟踪跨游戏系统的特定玩家体验和功能所需的基础设施成本。这可以确定架构中可能需要成本优化的领域。
- 评估成本优化与玩家体验之间的权衡：评估您的游戏处于哪个阶段，以确定正确的重点——玩家体验还是成本优化。通常，一旦游戏达到临界质量并且玩家数量稳定下来，就该专注于优化运营成本了。关键是要在提供出色的玩家体验和以最具成本效益的方式运行游戏基础设施之间取得平衡。实施这些设计原则可以最大限度地提高游戏投资的回报。

践行云财务管理

没有专门针对游戏视角的云财务管理最佳实践。有关云财务管理的指导，请参阅 Well-Architected Framework 成本优化支柱。

支出和使用情况意识

GAMECOST01：如何衡量游戏环境的成本？

了解每位玩家、游戏功能和环境的费用，以便随着玩家数量的变化以及功能的添加和改进，您可以管理和预测支出。考虑以下最佳实践来管理不同游戏环境的成本。

最佳实践

- GAMECOST01-BP01 实现每个玩家、游戏功能和环境的成本归因
- GAMECOST01-BP02 发现优化的机会

GAMECOST01-BP01 实现每个玩家、游戏功能和环境的成本归因

游戏服务器的成本归因通常比游戏后端服务更容易执行，因为游戏服务器通常经过优化，能够为每个实例托管特定数量的并发玩家，这些玩家可以按实例的运行成本进行摊销。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

对于游戏后端服务，建议将游戏的组件分离成不同的功能，这些功能可以作为单独的逻辑或物理资源进行管理，以便直接分析成本。

例如，尽管实现单个单体应用程序来托管游戏后端服务可能看起来很简单，但由于资源的计算、网络和存储成本是跨功能共享的，因此在添加更多功能时，这种模式使得很难推断出每位玩家和游戏功能的总成本。考虑为游戏后端服务采用无服务器架构，包括用于计算的 Amazon API Gateway 和 AWS Lambda /或计算、Amazon SQS 和 Amazon SNS AWS Fargate 用于消息传递、Amazon S3 用于对象存储、Amazon DynamoDB 用于数据库存储。这些服务只是产品中的几个示例，这些产品提供的定价基于使用量，主要由请求量驱动，因此可以精细地可视化成本。可以将单个资源（例如 Lambda 函数、Fargate 服务、DynamoDB 表和 S3 存储桶）与成本分配标签相关联，这样您就可以将这些服务的成本归因于游戏功能名称，这样您就可以直接了解每项服务的成本。

还建议单独管理每个游戏开发环境，这样您就可以为不同的环境分配成本。通常，游戏开发者将为开发、测试、暂存和生产环境管理不同的环境，如本游戏行业视角的运营支柱所述。每个环境通常都有不同的可扩展性、性能和使用要求，可能由不同的团队管理。要控制成本，请组织这些环境，以便您可以正确监控和归因每个环境的成本。

有关更多信息，请参阅以下文档：

- 开发一款可扩展的无服务器多人游戏
- 带有 WebSockets基于后端的独立游戏会话服务器
- 带有无服务器后端的独立游戏会话服务器

实施步骤

- 使用无服务器或容器化架构（例如 Amazon API Gate AWS Lambda way）将游戏后端服务分解 AWS Fargate 为不同的功能，并启用每个功能的精细成本归因。
- 将成本分配标签应用于单个资源（例如 Lambda 函数、DynamoDB 表和 S3 存储桶），将成本与特定的游戏功能关联起来，从而更好地进行成本分析。
- 管理单独的开发、测试、试运行和生产环境，独立组织和监控其成本，以适应可扩展性和使用要求。

GAMECOST01-BP02 发现优化的机会

游戏开发者和发行商可以利用 AWS FinOps 实践来帮助优化云成本，更好地了解他们的云支出。通过这样做，游戏制作人可以将维护玩家基础设施所需的平均成本与游戏带来的财务业绩保持一致。

在未建立这种最佳实践的情况下暴露的风险等级：低

实施指导

AWS 为云财务管理提供了即用型解决方案指南，用于管理和优化您的云服务费用。该功能包括精细的可见性以及成本和使用情况分析，以支持支出仪表板、优化、支出限制、退款以及异常检测和响应等主题的决策。Cloud Financial Management 的解决方案指南包括预算和预测功能，为你的工作负载提供了一个经过定义的、成本优化的架构，这样你就可以选择正确的定价模型并归因于与团队相关的资源成本。这可以在您的环境和资源中激活跟踪、通知和成本优化技术。您可以集中管理支出信息，并根据需要为关键利益相关者提供访问权限，以实现有针对性的可见性并支持决策。

另一个关键 FinOps 工具是成本优化中心，它可以集中查看和中的成本优化建议 AWS 账户 和机会 AWS 区域，以便您可以从 AWS 支出中获得最大收益。 您可以使用成本优化中心在和中识别、筛选和汇总 AWS 成本优化建议 AWS 区域。 AWS 账户 它针对资源大小优化、闲置资源删除、节省计划和预留实例提出建议。使用单一控制面板，您无需前往多个 AWS 产品来识别成本优化机会。

如果您的游戏团队使用的是共享的 MyA AWS 账户 pplications in AWS 管理控制台 Hom e，则可用于查看单个工作负载的应用程序资源成本。这种精细的视图使您能够确定游戏基础设施中的特定成本趋势，从而使您能够就资源分配和优化做出明智的决策。

此外，使用 Data Exports 定期查看账单和成本管理AWS 数据，发现隐藏的成本节约机会。这份详细的报告提供了您的云支出的全面明细，使您可以确定超支领域、未利用的资源以及利用更具成本效益的服务或定价模式的机会。

通过接受 FinOps 原则并利用提供的工具 AWS，游戏开发者和发行商可以最有效地利用其云资源，最终提高他们的利润，腾出资金用于进一步的游戏开发和创新。

实施步骤

- 使用 AWS 云 财务管理工具获得精细和详细的可见性、支出仪表板、异常检测和成本归因，从而有效地优化和跟踪云支出。
- 使用成本优化中心集中管理跨地区的 AWS 账户 合理规模、Savings Plans 和预留实例建议。
- 定期使用数据导出等 MyApplication 查看 AWS 账单数据， AWS 以帮助分析特定工作负载的成本、发现节省机会并优化资源分配。

具有成本效益的资源

GAMECOST02：您是如何为游戏服务器选择合适的计算解决方案的？

与其他类型的工作负载相比，游戏工作负载最独特的方面之一是游戏服务器。游戏服务器对玩家体验至关重要，因为玩家通过游戏客户端连接到游戏服务器来玩游戏会话。

游戏服务器也是运营多人游戏的最大成本驱动因素之一。因此，优化游戏服务器计算基础设施的使用方式以降低成本非常重要。

最佳实践

- GAMECOST02-BP01 优化通过互联网传输数据的成本
- GAMECOST02-BP02 优化每个游戏服务器实例上托管的游戏会话数量以优化成本
- GAMECOST02-BP03 选择适当的计算定价选项以降低成本

GAMECOST02-BP01 优化通过互联网传输数据的成本

虽然 AWS 主要收取从您的 AWS 资源到互联网的出站（出口）数据传输费用，但游戏公司可能会面临与通过 AWS Direct Connect 或 AWS 网关负载均衡器传输数据相关的高昂成本，这些负载均衡器可能会对入站（入口）和出站数据收费。实施可降低将数据从游戏 AWS 后端传输到玩家的总体成本的解决方案，重点是最大限度地减少 AWS 资源的出站费用，并评估通过 AWS 连接服务管理入口和出站费用的选项。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

使用 Amazon CloudFront 降低内容交付和面向公众的 Web 应用程序的成本。

存储在云中的游戏内容和资产通常存储在 Amazon S3 中，然后直接从 S3 或从托管在亚马逊的网络服务器发送到游戏客户端 EC2 ，这些服务器从 Amazon S3 检索内容并将其交付给客户。要降低内容下载的数据传输成本，可以考虑 CloudFront 在云存储前使用亚马逊向用户交付内容。

使用 CloudFront 可以降低数据传输成本，因为从区域交付内容的成本要 CloudFront points-of-presence低于直接从区域传送内容，并且 CloudFront 不会对 AWS基于来源的来源（例如亚马逊 EC2

和Amazon S3）收取源检索费。如果您的内容是静态的，并且不经常更改，则可以使用 CloudFront 将这些数据缓存到离最终用户更近的地方，这样可以进一步降低成本。

CloudFront 即使不使用缓存，也可以提高面向公众的面向公众的 Web 应用程序和服务的成本效率，因为通过网络路由流量可以降低服务器和客户端之间的数据传输成本。 AWS

亚马逊 CloudWatch可用于监控您的亚马逊 CloudFront使用情况。对于使用多个内容分发网络 (CDNs) 的用例，Amazon O CloudFront rigin Shield 可以提供额外的缓存层，以整合和减少来自不同提供商的原始请求数量。

要了解您的游戏网络流量，您可以启用 VPC 流日志和 Amazon CloudWatch Internet Monitor 以 end-to-end查看玩家或游戏后端连接。这种方法可以找出高数据传输成本的原因，并进行架构更改以优化数据传输支出。

实施步骤

- CloudFront 在 Amazon S3 或 EC2基于 Amazon S3 的内容来源之前使用 Amazon，通过利用较低的交付成本 CloudFront points-of-presence 和取消源检索费用，降低数据传输成本。
- 启用 VPC 流日志和 Amazon CloudWatch Internet Monitor 来分析网络流量并识别架构变更以优化数据传输成本。
- 使用多个 CloudFront Origin Shield 可以整合和减少源站请求 CDNs ，从而提高成本效益。

有关内容交付的更多最佳实践，请参阅游戏内容交付白皮书。

GAMECOST02-BP02 优化每个游戏服务器实例上托管的游戏会话数量以优化成本

优化每个服务器实例托管的游戏会话数量，以提高计算利用率并降低计算基础设施成本。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

为了优化成本，游戏开发者应最大限度地提高托管在同一物理或虚拟服务器上的游戏会话数量，也称为游戏服务器的打包密度。这是通过增加可以同时托管在实例上的游戏服务器进程数量来实现的。

单个游戏服务器进程通常不应要求使用 EC2 实例上的全部可用资源。这是降低游戏计算成本的最重要方法之一，需要使用能够通过不同端口在 EC2 实例上生成和管理多个服务器进程的软件。

例如，Amazon GameLift 对每个实例的游戏服务器进程的最大数量设定了配额，您应该努力利用该配额来降低托管成本。有关每个实例最大游戏 GameLift 服务器进程的当前配额的详细信息，请参阅 Amazon Servers 终端节点和配额。

作为在虚拟机（例如 EC2 实例）上部署游戏服务器进程的替代方案，游戏开发人员越来越普遍地使用容器编排解决方案将游戏服务器作为基于容器的应用程序运行。游戏开发者可以在亚马逊 E KS 上使用亚马逊弹性容器服务 (Amazon ECS) 或游戏服务器托管指南，使用 Agones 和 Open Match。另一种选择是 Game Server Hosting on AWS Fargate，这是一款可与 ECS 和 EKS 配合使用的无服务器计算引擎，使您无需管理底层基础设施即可专注于游戏。

容器解决方案提供作业调度功能，可以根据资源要求和您指定的其他放置逻辑，自动在集群中找到可用的容器实例来托管您的游戏服务器容器。但是，重要的是要考虑如何以不干扰活跃玩家会话的方式管理缩放和玩家放置行为。

实施步骤

- 使用单独的端口和进程管理软件，在每个 EC2 实例上运行多个游戏服务器进程，从而提高打包密度。
- 使用 Amazon GameLift 或 ECS、EKS 等容器解决方案，或者 AWS Fargate 高效管理游戏服务器流程并降低基础设施成本。
- 持续监控资源利用率，在不影响玩家体验的情况下提高包装密度并保持成本效益。

GAMECOST02-BP03 选择适当的计算定价选项以降低成本

针对各种实例类型和计算选项对游戏服务器软件进行性能测试，以确定哪个选项对您的游戏来说最具成本效益。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

除了为您的工作负载高效利用正确的 EC2 实例类型外，还要考虑哪种可用的计算定价选项最适合您的成本优化目标。有多种定价选项可供选择，包括按需实例、竞价型实例、预留实例和 Savings Plans。

S@@ av ings Plans (SPs) 通过承诺使用量来提供计算折扣，非常适合无法预测 1 年或 3 年期的预期使用量的场景。它们提供诸如预留实例之类的折扣，并且可以灵活地将这些折扣应用于各个区域、实例系列、操作系统、租赁中。它们也可以应用于 AWS Fargate谁可以是休闲游戏的游戏服务器托管选项，或者 AWS Lambda 是不需要游戏服务器的回合制游戏的绝佳选择。有关更多信息，请参阅构建可扩展的无服务器多人游戏。

Savings Plans 是在游戏发布期间推出的，旨在节省游戏服务器工作负载的成本，这些工作负载会导致游戏向受众发布时 EC2 实例支出。Savings Plans 也可以在游戏发布后推出，当游戏运营团队在游戏长时间投入生产后对玩家流量有了更好的了解。

由于Savings Plans提供了区域灵活性，因此它们特别适合优化游戏服务器支出，以应对跨地区使用不可预测的游戏。

例如，如果您的每日玩家使用模式需要至少 20 台服务器来支持您的玩家群，但定期需要多达 40 台服务器，则可以考虑购买 Savings Plan 套餐以满足 20 台服务器的基准，因为这种使用需求是可预测且一致的，并且可以最大限度地利用您购买的使用承诺。

最大限度地提高 Savings Plans 的利用率，并通过其他购买选项为不可预测的游戏服务器使用量峰值提供更大的灵活性，例如按需实例和竞价型实例，从而实现最佳的节省。

竞价型实例非常适合运行游戏服务器，因为它们提供最大的计算折扣，不需要使用承诺，而且它们为不可预测的尖峰工作负载类型提供了灵活性。但是，竞价型实例可以中断，因此它们最适合游戏会话持续时间较短的游戏服务器工作负载或中断容忍度较高的情况。

有关在带有竞 EC2 价型实例的 Amazon EKS 上使用 Kubernetes 运行游戏服务器的指南的更多信息，请参阅如何 EC2 使用 Aurora Serverless 在 Spot 上运行大型多人游戏。

使用 Amazon EC2 Spot 实例来确定中断几率最小的池，与按需费率相比，这些池可以最大限度地节省开支。

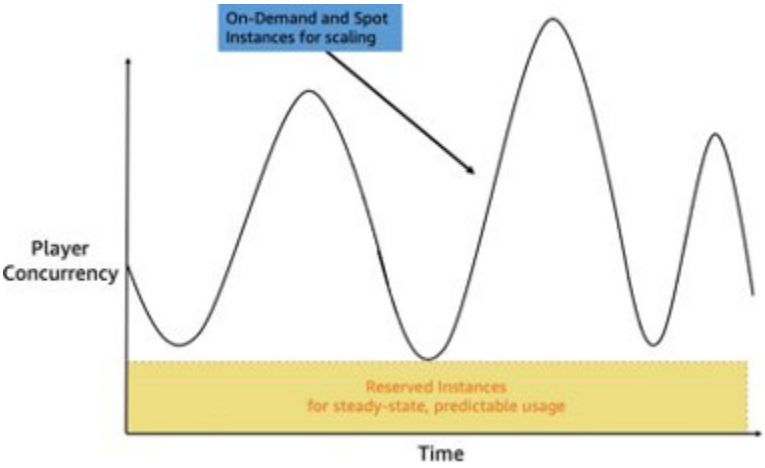
使用 Spot 时，还建议在多个 EC2 实例类型和可用区中运行游戏服务器工作负载， AWS 区域 以分散容量使用并降低中断风险。

考虑将竞价型实例与按需实例结合使用，以最大限度地减少潜在中断对活跃游戏会话的影响，并使用容量优化的分配策略来进一步降低中断风险。

有关其他最佳实践，请参阅 Amazon EC2 Spot 的最佳实践。Auto Scaling 中的容量重新平衡以取代存在风险的竞价型实例，可用于在竞价型实例面临中断风险增加时主动监控和增加额外容量。

Amazon GameLift FleetiQ 与竞价型实例集成，可优化低成本竞价型实例的使用，同时降低中断风险。如果您使用托管游戏 GameLift，请查看有关选择计算资源的 GameLift 文档。有关更多信息，请参阅为托管队列选择计算资源。

下图提供了一个示例，说明如何为游戏服务器工作负载使用多种计算定价选项：



托管具有多种 EC2定价选项的游戏服务器

在图中，玩家的并发度会随着时间的推移而波动，这使得管理利用率和实现成本优化变得困难。为了应对这种波动，可以考虑混合采用不同的计算定价选项，使用 Savings Plans EC2 来满足最低使用要求，同时依靠 EC2 按需实例和 EC2 竞价型实例来满足玩家的需求。

实施步骤

- 使用 Savings Plans 获得可预测的基准使用量，将它们与竞价型和按需实例结合使用，在使用量高峰期间实现灵活性和成本优化。
- 将竞价型实例用于会话持续时间短或中断容限较高的游戏服务器，在实例类型和可用区之间实现多样化以最大限度地降低风险。
- 实施 EC2 Spot Instances Advisor、容量再平衡和 Fle GameLift etiQ 等工具，以优化竞价型实例的使用并主动管理中断。

数据传输成本

GAMECOST03：您如何优化游戏基础设施的数据传输成本？

游戏可以通过互联网在玩家的游戏客户端设备和游戏基础设施之间传输大量数据，以提供游戏体验，也可以在游戏基础设施的各个组件之间传输大量数据。

例如，当玩家将游戏内容更新下载到游戏客户端、将游戏进度状态保存到云端、与朋友进行实时多人游戏会话，以及您的游戏基础设施在区域和可用区域之间传输数据时，就会发生数据传输。了解游戏工作负载中数据传输的位置非常重要，这样才能优化架构选择，从而降低数据传输成本。

要优化游戏工作负载的数据传输成本，请考虑以下最佳实践：

最佳实践

- GAMECOST03-BP01 为用户生成的内容选择适当的存储类型以降低成本
- GAMECOST03-BP02 优化游戏后端的数据库

GAMECOST03-BP01 为用户生成的内容选择适当的存储类型以降低成本

游戏中生成和存储的每种数据都有其独特的特征，在为工作负载确定合适的存储解决方案时，应考虑这些特征。

在未建立这种最佳实践的情况下暴露的风险等级：低

实施指导

使用 Amazon S3 对象生命周期管理将对象数据存储在最具成本效益的存储类别中。Amazon S3 提供多种存储类别和对象生命周期管理，因此可以直接设置简单而精细的策略，在存储层之间自动传输数据以降低成本。与其在默认情况下简单地将数据存储在 S3 标准存储类中，不如考虑设置生命周期配置，以便随着时间的推移在各层之间自动转移数据，或者使用 S3 Intelligent-Tiering 存储类来存储未知或不断变化的访问模式。

或者，S3 Intelligent-Tiering 可以经济高效地自动在层之间转移数据，建议将其作为默认存储类别，因为它无需手动设置生命周期策略即可实现成本优化，现在是小型和短寿命对象的最佳选择。有关更多信息，请参阅 Amazon S3 智能分层 — 针对短期和小型对象的改进成本优化。

Amazon S3 的常见用例包括游戏资产存储、静态内容、游戏日志、数据湖存储和备份。对于需要文件系统的用例，例如在开发过程中将共享文件系统连接到工作站，可以考虑使用 Amazon Elastic File System (Amazon EFS)，它提供不同的存储类别，并且在您添加和删除文件时自动增长和缩小，无需管理基础架构。

Amazon S3 One Zon e-IA 是存储与游戏内会话、配对或其他可以根据需要重新创建的临时信息相关的临时数据的理想存储选项。这种类型的游戏数据不需要跨多个可用区域的冗余（AZs）。这种成本较低的存储类别非常适合用于分析或调试的玩家操作、游戏事件和其他遥测数据的记录。

使用 S3 Express One Zone 处理此类游戏数据的关键成本优化优势是，与标准 S3 存储类别相比，可节省大量成本，存储成本最多可降低 20%。这对于拥有大量数据、不需要与任务关键型应用程序数据相同的耐久性和可用性的游戏尤其有利。通过利用 S3 One Zone，游戏开发者和发行商可以在不影响整体玩家体验的情况下优化云存储成本。

实施步骤

- 配置 Amazon S3 生命周期策略以在存储类之间转移数据，或者使用 S3 智能分层作为默认设置，在访问模式不断变化的情况下自动优化成本。
- 使用 S3 One Zone-Infrequent Access 来处理瞬态游戏会话数据，例如遥测和配对记录，可以在保持足够的可用性的同时将存储成本降低多达 20%。
- 为了满足开发期间的共享文件系统需求，可以使用 Amazon EFS 通过弹性容量和多种存储类别来简化存储管理。

GAMECOST03-BP02 优化游戏后端的数据库

游戏严重依赖数据库来存储各种关键数据，从玩家资料和库存到游戏中的微交易和进度指标。数据库在管理游戏的社交方面也起着至关重要的作用，例如创建和维护玩家群组、聚会以及执行审核政策。随着游戏玩家群的增长，相关的数据库成本将不可避免地增加，以适应不断增长的数据和使用需求。

在未建立这种最佳实践的情况下暴露的风险等级：中

实施指导

对于在 Amazon Aurora 上运行的游戏后端，可以采用多种成本优化策略。一项关键建议是根据使用模式自动缩放只读副本，动态地向上或向下扩展副本的数量以应对流量波动。这意味着您要为真正需要的资源付费。另一种优化策略是将用于游戏分析的只读副本替换为导出到 Amazon S3 的数据库快照，因为 S3 存储服务通常比预配置的 Aurora 数据库实例更实惠。有关更多信息，请参阅将数据库快照数据导出到适用于 Amazon RDS 的 Amazon S3。

探索将 Amazon Aurora 预留数据库实例用于核心数据库实例并过渡到 A urora Serverless 配置，还可以通过提供更高的灵活性和对资源利用率的精细控制，从而节省大量的长期成本。

同样，对于使用 Amazon DynamoDB 的游戏后端，采用 DynamoDB 按需容量模式可能是一个有效的选择，特别是对于新的或不可预测的工作负载，因为它允许您仅为消耗的资源付费，而无需过度配置。随着时间的推移，随着游戏流量模式变得更加稳定和可预测，您可以过渡到 D ynamoDB 预置容量模式，该模式可以通过更好的容量规划来节省成本。在 DynamoDB 表上激活自动缩放功能是另一项关键优化，它允许该服务根据流量波动动态调整预配置容量。在游戏启动之前，在开发环境中测试游戏的数据结构，以查找和删除不必要的本地二级索引 (LSIs) 和全局二级索引 (GSIs)。这可以为游戏数据存储和运营节省大量成本。从游戏后端代码中删除效率低下的扫描操作，转而使用更具针对性的查询，购买 Amazon DynamoDB 预留容量，以及利用带有触发器的 Dy namoDB Streams 来处理游戏后端事件，可以进一步优化您的 DynamoDB AWS Lambda 成本。有关更多信息，请参阅在 DynamoDB 中查询和扫描数据的最佳实践。

通过为 Amazon Aurora 和 DynamoDB 实施这些成本优化策略，游戏开发者和发行商可以显著减少其游戏后端数据库支出。

实施步骤

- 使用 Aurora 只读副本自动缩放和将数据库快照导出到 Amazon S3，以经济高效的方式处理波动的流量和分析需求。
- 优化 DynamoDB 成本，方法是从新工作负载的按需容量开始，过渡到具有可预测流量自动缩放功能的预配置容量，以及移除未使用的和。 LSIs GSIs
- 避免低效的扫描操作，转而使用定向查询，使用预留实例或预留容量，并将 DynamoDB St AWS Lambda reams 与事件一起使用。

管理需求和供应资源

没有专门针对Games Lens的管理需求和供应资源的最佳实践。

有关管理需求和供应资源的更多信息，请参阅成本优化支柱——Well-Architecte AWS d Framework。

随着时间的推移进行优化

没有专门针对 Games Lens 的长期优化最佳实践。

有关随时间推移优化成本的更多信息，请参阅成本优化支柱——Well-Architecte AWS d Framework。

资源

要详细了解成本优化的最佳实践，请参阅以下资源：

相关文档：

- 如何降低 Amazon VPC 中我的 NAT 网关的数据传输费用？
- 推出适用于 Agones 的 Amazon GameLift FleetiQ 适配器
- 如何在我的 Amazon VPC 中找到 NAT 网关流量的最大贡献者？
- 为您的全球游戏服务器选择正确的计算策略
- AWS Well-Architected Labs — 具有成本效益的资源
- Amazon VPC CNI 插件提高了每个节点的容量限制

- 成本优化的架构最佳实践
- 使用 Amazon A SageMaker I RL 和 Amazon EKS 缩短玩家等待时间并调整计算分配的规模
- AWS Compute Optimizer
- Electronic Arts 使用 Amazon S3 智能分层和 Amazon Glacier 优化存储成本和运营
- 通过将 Windows 工作负载迁移到 Linux 来逃避不友好的许可惯例
- Overview of Data Transfer Costs for Common Architectures
- AWS 与 Kubecost 合作为 EKS 客户提供成本监控
- 奠定基础：设置您的环境以实现成本优化
- Amazon EC2 竞价型实例概述

可持续性

可持续发展支柱提供设计原则、运营指导、最佳实践和改进计划，以帮助您实现 AWS 工作负载的可持续发展目标。

你可以在《可持续发展支柱——Well-Architected Framework》 AWS 白皮书中找到实施指南。

聚焦领域

- 设计原则
- 区域选择
- 符合需求
- 软件和架构
- 数据管理
- 硬件和服务
- 资源

设计原则

在世界不同地区，游戏行业的可持续发展正在以不同的速度发展。随着北美大片地区的可持续电力以及欧盟和英国的可持续发展要求，建筑师有多种方法可以在不久的将来实现更环保的工作量。Well-Architected 的可持续发展支柱可用于针对各种工作负载实现这些衡量标准。

本节镜头描述了可用于游戏工作负载的几种最佳实践。

- 为游戏用户数据选择适当的存储类型。
- 请注意数据生命周期策略，这些策略将删除重复数据并清除工作负载中不需要的数据。
- 选择性地调整计算资源的部署规模。
- 使用无服务器处理简短和事务性流程。

区域选择

没有专门针对游戏镜头的区域选择最佳实践。欲了解更多详情，请参阅可持续发展支柱——Well-Archit AWS ected 框架。

符合需求

对于Games Lens特有的最佳实践，目前尚不一致。欲了解更多详情，请参阅可持续发展支柱——Well-Archit AWS ected 框架。

软件和架构

没有专门针对 Games Lens 的软件和架构最佳实践。欲了解更多详情，请参阅可持续发展支柱——Well-Archit AWS ected 框架。

数据管理

GAMESUS01：如何在游戏系统中管理用户和游戏数据？

制定数据生命周期策略，通过仅保留关键的历史数据来优化数据存储和相关性。

最佳实践

- GAMESUS01-BP01 使用适合用户内容、订阅者信息和游戏内购买模式的存储技术
- GAMESUS01-BP02 使用生命周期策略或 TTL 过期时间删除不必要的游戏用户数据、日志文件或已弃用的资产

GAMESUS01-BP01 使用适合用户内容、订阅者信息和游戏内购买模式的存储技术

您应按类型、保留需求和访问频率对数据进行分类。这使您能够为游戏或后端服务生成的各种数据类型选择最优化的存储解决方案。快速变化的数据应存储在键值或内存数据库服务中。事务数据应存储在关系数据库服务中。大型文件、游戏资产或用户生成的内容应存储在对象存储服务中。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

游戏会生成和消耗各种各样的数据类型，需要针对访问频率、延迟和成本进行优化的存储解决方案。应使用标签对存储的数据进行分类，以区分可以删除或需要长期存储的数据。

以下服务适用于各种游戏用例：

Amazon Aurora（兼容 MySQL 和 PostgreSQL）提供高可用性、低延迟和自动扩展，使其成为处理大量交易数据的绝佳选择，例如玩家账户管理和身份验证、游戏内经济、排行榜和玩家排名、游戏状态持久性、事件和活动管理以及多区域和高可用性部署。

Amazon DynamoDB 是一个完全托管的 NoSQL 数据库，以其低延迟、高吞吐量和无缝可扩展性而闻名，非常适合处理实时玩家数据、会话管理、库存、游戏内经济、实时多人游戏状态、配对、事件记录和面向全球受众的扩展。

Amazon DocumentDB（与 MongoDB 兼容）提供可扩展、低延迟的面向文档的数据库服务，非常适合存储灵活的半结构化数据，例如库存系统、玩家资料和自定义数据、游戏世界和程序生成的内容、社交和玩家互动、分析和行为跟踪以及游戏内元数据和配置。

Amazon ElastiCache 支持 Redis 或 Memcached 的内存缓存，提供快速的数据访问和更短的响应时间，这对于实时多人游戏至关重要，因为在实时多人游戏中，速度和性能对于流畅的用户体验至关重要。 ElastiCache 在游戏中用于实时排行榜、会话管理、缓存游戏元数据、游戏内聊天和消息传递、配对、实时分析和遥测，以及针对高流量事件进行扩展。

Amazon Simple Storage Service (S3) 可用于存储游戏资产、视频、图片、文本日志文件等对象。S3 是一项对象存储服务，可提供业界领先的可扩展性、数据可用性、安全性和性能。

如果它提供支持频繁和不频繁的数据访问的多种存储类别，以及经济实惠的存档存储。对于开发过程中经常访问的数据，Studio 应将对象存储在 S3 标准中，以实现低延迟和高吞吐量性能。对于经常从热变冷（反之亦然）的数据，工作室应研究 S3 智能分层。智能分层监控数据的访问模式，并自动将数据移动到最具成本效益的访问层。

对于需要高吞吐量、低延迟且能在单个可用区内运行的工作室，请使用 S3 Express One Zone。与 S3 标准相比，这可以将数据复制到单个可用区，并且可以提高数据访问速度。 为了满足历史数据的深度存档需求，亚马逊还提供了 Amaz on Glacier。Amazon Glacier 存储类专为数据存档而构建，可为您提供高性能、灵活检索和低成本的云端存储。

Amazon Elastic Block Store 可用于存储游戏服务器的二进制文件、可执行文件以及游戏服务器或资产存储库运行所需的配置。您应该快照并删除未连接到 EC2 实例的未使用卷。这可以减轻您产生的存储费用，同时降低不需要的服务和硬件的使用量。

实施步骤

- 按类型、保留需求和访问频率对游戏数据进行分类，对数据进行标记以区分短期和长期存储需求。
- 使用 Amazon Aurora 存储交易数据，使用 DynamoDB 获取实时玩家数据，使用 DocumentDB 处理半结构化数据， ElastiCache 并使用低延迟缓存时间关键型游戏信息。

- 将游戏资产、日志和用户生成的内容存储在 Amazon S3 中，根据访问模式和存档需求选择适当的存储类别（例如，智能分层、One Zone 和 Glacier），并使用 EBS 处理游戏服务器二进制文件和配置，并定期管理快照。

GAMESUS01-BP02 使用生命周期策略或 TTL 过期时间删除不必要的游戏用户数据、日志文件或已弃用的资产

您可以使用标签和数据类型来创建生命周期策略，也可以使用 TTL 将数据移至存档存储或从服务中完全移除。这可能包括临时配置、过期的存档内容以及不再需要的历史日志。大多数服务都支持标记。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

对于存储在 S3 中的数据，您可以使用生命周期策略将数据移动到不频繁访问和存档存储层。在 S3 生命周期配置中，您可以定义用于将对象从一个存储类转换为另一个存储类的规则，以节省存储成本。如果您不了解对象的访问模式或访问模式不断变化，则可将对象转换为 S3 Intelligent-Tiering 存储类，以自动实现成本节省。

Amazon S3 支持用于在存储类之间进行转换的瀑布模型，如下图所示。

可以向 S3 生命周期配置中添加转换操作，来指示 Amazon S3 在对象生命周期结束时删除对象。当对象根据其生命周期配置达到其生命周期结束时，Amazon S3 会根据存储桶所处的 S3 版本控制状态采取过期操作：

- 非版本控制存储桶：Amazon S3 将对象排队等候移除，然后异步移除该对象，从而永久移除该对象。
- 启用版本控制的存储桶：如果当前对象版本不是删除标记，则 Amazon S3 会添加带有唯一版本 ID 的删除标记。这会使当前对象版本变为非当前版本，并使删除标记变为当前版本。
- 已暂停版本控制的存储桶：Amazon S3 会创建一个版本 ID 为空的删除标记。此删除标记将对象版本替换为版本层次结构中的空版本 ID，从而有效地删除对象。
- 当您向存储桶添加生命周期配置时，配置规则将应用到现有对象以及您在以后添加的对象。例如，如果您今天添加了一条生命周期配置规则，其过期操作会导致具有特定前缀的对象在创建 30 天后过期，那么 Amazon S3 将排队移除已存在超过 30 天且具有指定前缀的现有对象。

DynamoDB 的生存时间（TTL）是一种经济实惠的方法，用于删除不再相关的项目。通过 TTL，您可以定义每个项目的过期时间戳，指示何时不再需要某个项目。DynamoDB 会在项目过期时间到期后的几天内自动将其删除，而不会消耗写入吞吐量。

- 要使用 TTL，请先在表上启用它，然后定义一个特定的属性来存储 TTL 过期时间戳。时间戳必须用 Unix 纪元时间格式以秒为单位进行存储。每次创建或更新项目时，您都可以计算过期时间并将其保存在 TTL 属性中。
- 具有有效、已过期 TTL 属性的项目通常会在过期后的几天内被系统删除。您仍然可以更新待删除的过期项目，包括更改或删除其 TTL 属性。更新过期项目时，我们建议您使用条件表达式来确保该项目随后未被删除。使用筛选表达式从扫描和查询结果中删除过期的项目。
- 已删除项目的工作原理与通过典型删除操作删除的项目类似。删除后，项目会以服务删除而不是用户删除的形式进入 DynamoDB Streams，并像其他删除操作一样从本地二级索引和全局二级索引中删除。

使用 f ElastiCache or Redis，您可以通过使用缓存密钥 TTLs 或缓存密钥过期来控制缓存数据的新鲜度。 在设定的时间过去后，密钥将从缓存中删除，并且需要访问原始数据存储以及访问更新的数据。

- 两个原则决定了适宜应用 TTLs 的缓存模式和要实现的缓存模式的类型。首先，了解基础数据的变化率很重要。其次，重要的是要评估将过时的数据返回到应用程序而不是更新的对应数据的风险。
- 对于经常更改的动态数据，您可能需要应用较低的值 TTLs ，使数据的过期速度与主数据库的变化率相匹配。这降低了返回过时数据的风险，同时仍为卸载数据库请求提供了缓冲区。
- 同样重要的是要认识到，即使你只缓存数据几分钟或几秒钟而不是更长的持续时间，适当应用 TTLs 于缓存的密钥也可以提高性能，让玩家在游戏中获得更好的整体体验。

实施步骤

- 使用 Amazon S3 生命周期策略将对象过渡到不频繁访问或存档层，并配置过期操作以根据生命周期规则删除不必要的对象。
- 在 DynamoDB 表中启用 Time to Live (TTL) 可在不消耗写入吞吐量的情况下自动删除过期项目，以 Unix 纪元时间定义过期时间戳。
- 根据数据更改率和过时数据的风险承受能力 TTLs 为 ElastiCache 密钥设置适当的设置，从而提高缓存数据的新鲜度并改善玩家体验。

硬件和服务

GAMESUS02：如何在游戏后端管理计算资源的使用？

Studios 应制定一种混合使用不同计算类型、托管服务和储蓄计划的计算策略，以优化您的使用情况。您还应该优化游戏服务器和后端服务打包到计算实例的方式，以减少不需要的资源数量。

最佳实践

- GAMESUS02-BP01 为适当的计算工作负载选择托管服务
- GAMESUS02-BP02 调整计算规模，仅在需要的地方部署 GPU 性能

GAMESUS02-BP01 为适当的计算工作负载选择托管服务

设计游戏后端服务，使用托管服务处理事件驱动或高度可变的流量工作负载。由于多租户控制平面，托管服务将基础设施的管理转移到多个用户， AWS 并将环境影响分散到多个用户。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

AWS AWS Fargate （容器）和 Amazon Gamelift（游戏服务器编排）之类 AWS Lambda的服务可以运行代码、容器或编排游戏服务器，而无需管理底层基础架构。这些服务会根据玩家的需求自动扩展，您只需为消耗的资源付费。由于底层基础设施是代表您管理的，因此您可以只专注于游戏和后端服务的需求。

您可以使用 AWS Lambda 运行代码而无需预置或管理服务器。Lambda 在高可用性计算基础设施上运行您的代码，并管理计算资源，包括服务器和操作系统维护、容量预置和自动扩展以及日志记录。使用 Lambda，您需要使用 Lambda 支持的语言运行时之一来提供代码。Lambda 可用于处理游戏事件、玩家身份验证、游戏内购买处理和配对请求。Lambda 会根据事件的数量自动扩展，并且可以处理意想不到的流量峰值。

AWS Fargate是一款适用于容器的无服务器计算引擎，可与亚马逊弹性容器服务 (ECS) 和亚马逊弹性 Kubernetes Service (EK S) 配合使用。 AWS Fargate 通过减少配置和管理服务器的需求，可以直接专注于构建应用程序，允许您指定每个应用程序的资源并为其付费，并通过设计上的应用程序隔离来提高安全性。Fargate 非常适合处理玩家档案、状态管理和配对的后端服务。

Amazon GameLift 是一项托管服务，用于为基于会话的多人游戏部署、操作和扩展专用游戏服务器。您可以在短短几分钟内将第一台游戏服务器部署到云端，从而在前期软件开发方面节省多达数千小时的工程时间，并降低经常导致开发人员从设计中削减多人游戏功能的技术风险。

实施步骤

- 利用其自动扩展和无服务器管理， AWS Lambda 用于事件驱动的工作负载，例如处理游戏事件、玩家身份验证、游戏内购买和配对请求。

- AWS Fargate 使用 ECS 或 EKS 部署后端服务，例如玩家档案、状态管理和配对，从而取消服务器管理并改善应用程序隔离。
- 使用 Amazon GameLift 为基于会话的多人游戏部署和扩展专用游戏服务器，从而减少开发时间和操作复杂性。

GAMESUS02-BP02 调整计算规模，仅在需要的地方部署 GPU 性能

设计您的游戏服务器和后端，以有效利用计算资源。过度配置计算会导致不必要的成本，并最大限度地减少闲置或未充分利用的资源量。GPU 实例应该用于支持特定的开发工作，例如在虚幻引擎中重建 HLOD，或者如果你的游戏服务器在设计上需要它们。这大大减少了工作负载对环境的影响和成本。

在未建立这种最佳实践的情况下暴露的风险等级：高

实施指导

您应该优化游戏服务器和后端服务，使其使用多种 EC2 实例类型和所需的最少实例数。这会增加可用实例的数量，以满足您在开发期间或游戏发布期间的需求。您还应将实例类型与要部署的特定工作负载相匹配。计算优化型实例支持各种用例，包括游戏服务器和配对等后端服务。内存优化型实例旨在为在内存中处理大型数据集的工作负载提供快速性能。根据需要使用 GPU 实例以满足高性能要求，但不能用于一般计算任务。如果可以，请设计您的服务或游戏服务器，使其在带有 AWS Graviton 实例的 ARM 上运行。Graviton 是目前可用的最高性能至节能的实例类型。 AWS与 x86 实例类型相比，它们还提高了性能和成本。

使用机器学习AWS Compute Optimizer来分析历史利用率指标，使用机器学习来确定最佳 AWS 资源配置，例如亚马逊弹性计算云 (EC2) 实例类型、亚马逊弹性块存储 (EBS) Block Store 卷配置、亚马逊弹性容器服务 (ECS) 服务的任务大小 AWS Fargate、商业软件许可证、 AWS Lambda 函数内存大小和亚马逊关系数据库服务 (RDS) Database Service 数据库实例类。Compute Optimizer 提供了一套控制台体验，通过为您的 AWS 工作负载推荐最佳 AWS 资源来降低成本并提高工作负载性能。 APIs

实施步骤

- 将计算资源与特定工作负载相匹配，为游戏服务器使用计算优化型实例，针对大型数据集使用内存优化型实例，仅将 GPU 实例用于诸如 HLOD 重建或依赖于 GPU 的游戏服务器之类的任务。
- 与 x86 实例相比，尽可能通过部署 AWS Graviton 实例来优化计算利用率，从而提高能效、提高性能并节省成本。
- AWS Compute Optimizer 用于分析历史利用率并为 EC2 AWS ECS 和 Amazon RDS 工作负载推荐最高效的配置 AWS Lambda，以降低成本并提高性能。

资源

请参阅以下资源，详细了解我们与可持续发展相关的最佳实践。

- 联合国：游戏行业聚焦对地球的威胁
- 媒介：游戏开发中的环境可持续性：负责任地玩游戏创造更绿色的未来

关键 AWS 服务

- Amazon Aurora
- Amazon DynamoDB
- Amazon DocumentDB（与 MongoDB 兼容）
- Amazon ElastiCache
- Amazon S3
- Amazon S3 存储类
- Amazon S3 智能分层存储类别
- 亚马逊 S3 Express One Zone 存储类别
- Amazon Elastic Block Store
- AWS Lambda
- Amazon Elastic Container Service
- Amazon Elastic Kubernetes Service
- Amazon GameLift
- AWS Compute Optimizer

结论

游戏旨在为全球玩家提供娱乐体验，其使用特征通常是不可预测且可变的。《游戏行业视角》描述了通常构成游戏架构的常见场景类型，并提供了一系列问题和最佳实践，供您在云端构建和运营游戏时考虑。通过将此框架应用于您的游戏架构，您可以在云端构建可靠、安全、高效且经济实惠的游戏。

贡献者

以下个人参与了本文档的编撰：

- 亚当·哈特菲尔德，Amazon Web Services 高级解决方案架构师
- Brady Webb，亚马逊 Web Services 技术客户经理
- Bruce Ross — 高级解决方案架构师、Well-Architected 镜头负责人，亚马逊 Web Services
- Caleb Cecil，助理解决方案架构师，亚马逊 Web Services
- Carlos Perez，云优化成功 SA，亚马逊 Web Services
- Chase Herrington，ESL 技术客户经理，亚马逊 Web Services。
- Chris Blackwell，Amazon Web Services 高级解决方案架构师
- Corey Ouderkirk，Amazon Web Services 高级技术客户经理
- 德里克·比亚维森西奥，原型设计 SA，亚马逊 Web Services
- Erik Ynigo Becerril，Amazon Web Services 高级解决方案架构师
- Grzegorz Ochmanski，Amazon Web Services 高级解决方案架构师
- Hadrian Baron，Amazon Web Services 高级技术客户经理
- Ian Armbruster，Amazon Web Services 高级客户解决方案经理
- Jed Obray，Amazon Web Services 高级技术客户经理
- Khurram Khokhar，Amazon Web Services 高级技术客户经理
- Kyle Somers，Amazon Web Services 解决方案架构高级经理
- Madhuri Srinivasan，高级技术作家，Well-Architected，亚马逊 Web Services
- Matthew Wygant，高级TPM指南，Well-Architected，Amazon Web Services
- Nataliya Godunok，云优化成功 SA，Amazon Web Services
- Nirav Doshi，亚马逊 Web Services 首席解决方案架构师
- Olivia Liddell，Amazon Web Services 解决方案架构师
- 兰迪·詹姆斯，亚马逊 Web Services 首席技术客户经理
- Reou Ando，游戏解决方案架构师，亚马逊 Web Services
- Richard Raseley，Amazon Web Services 高级技术客户经理
- Amazon Web Services 高级解决方案架构师 Sam Patzer
- Scott Selinger，Amazon Web Services 高级解决方案架构师
- 肖恩·艾伦，Amazon Web Services 高级解决方案架构师

- Serge Poueme，Amazon Web Services 高级解决方案架构师
- Stewart Matzek，Amazon Web Services Well-Architected 高级技术撰稿人
- Trenton Potgieter，高级解决方案架构师AI/ML/Analytics，亚马逊 Web Services

文档修订

如需获取有关该白皮书更新的通知，请订阅 RSS 信息源。

变更	说明	日期
新镜头版本	使用新的最佳实践指南更新了整个镜头。	2025 年 12 月 9 日
初次发布	白皮书首次发布。	2021 年 11 月 19 日

AWS 词汇表

有关最新 AWS 术语，请参阅《AWS 词汇表 参考资料》中的AWS 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。